Adaptive monitoring and control framework in Application Service Management environment

University of London, Birkbeck
Department of Computer Science and Information Systems

Adaptive Monitoring and Control Framework in Application Service Management Environment

Tomasz D. Sikora

A Thesis Submitted to
Birkbeck, University of London
in Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

Birkbeck
UNIVERSITY OF LONDON

December 2021, London, United Kingdom

Abstract

The economics of data centres and cloud computing services have pushed hardware and software requirements to the limits, leaving only very small performance overhead before systems get into saturation. For Application Service Management–ASM, this carries the growing risk of impacting the execution time of various processes, resulting in reduced Quality of Services–QoS. In order to deliver a stable service at times of great demand for computational power, enterprise data centres and cloud providers must implement fast and robust control mechanisms that are capable of adapting to changing operating conditions while satisfying Service–Level Agreements–SLAs. In ASM practice, there are normally two methods for dealing with increased load. The first approach increases computational power by typically allocating additional machines, which must be available, waiting idle, to deal with high demand situations. The second approach releases load by terminating incoming actions that are considered less important to new activity demand patterns, throttling, or rescheduling jobs. Adaptive/automatic termination of processes, tasks or actions is not normally considered by most modern cloud platforms, or operating systems. However, it is administrators' common practice to manually end, or stop, tasks or actions at any level of the system, such as at the level of a node, function, or process, or kill a long session that is executing on a database server. In this context, adaptive monitoring and control of actions termination remains a significantly underutilised subject of ASM and deserves further consideration. That may be eminently suitable for systems with harsh execution time SLAs, such as real–time systems, high requirements of QoS standards, or systems running under conditions of hard pressure on power supplies, systems running under variable priority, or constraints set up by the green computing paradigm.

In this vein, the thesis investigates adaptive monitoring and actuation, and the relevant control challenges. It contributes a new framework and methods for identifying relevant dimensions in Enterprise Systems and selecting features, creating causal models, and decomposing metrics signals that can be exploited for more efficient action termination. These methods are integrated in adaptive control emulators and actuators powered by neural networks. The controllers are used to adjust the operation of the system to navigate to better conditions in environments with established SLA and QoS improvement goals. The improvements can be seen from both systems performance and economics perspectives. The behaviour of the proposed control framework is evaluated using complex load and service agreements scenarios of systems compatible with the requirements of on–

premises, elastic compute cloud deployments, server–less computing, and micro–services architectures.

Acknowledgements

I would like to express my special appreciation and thanks to my supervisor Professor George D. Magoulas, who throughout the study, in good times and bad, was always encouraging me to pursue my dreams. His academic and life related advice guided me and helped tremendously in this process. I would like to thank you Professor for all your help in my research and for allowing me to review even the strangest ideas to be validated in the field. I would also like to thank my second supervisor Dr George Roussos and the Committee who kept me on track, for insightful comments, and hard questions.

A special thanks to my family. It is difficult to express in words how grateful I am to my beloved wife, Ania, not only for all of the sacrifices that you have made on my behalf, but also for motivating me in the conduct of the study. Thank you for supporting me every day and night. And above all, I cannot thank you enough for your belief in the final success and encouragement throughout this experience.

Finally, I would like to thank my parents, Stanisława and Stefan, for everything, beginning from giving birth to me, through all the support in my life, for all their love and help in things of great and small importance.

I dedicate this work to my beloved daughters Agatka and Adusia; you are such good girls always cheering your parents up. I hope you will grow strong, in good health, be wise and have a good live.

Wisdom and knowledge will be the stability of thy times.

Isaiah 33:6

“Nemo nisi per amicitiam cognoscitur - No one learns except by friendship”

...one must like a subject in order to study it

Declaration

This thesis is the result of my own work, except where explicitly acknowledged in the text.

Tomasz D. Sikora, December 2021 ....

Contents

List of Tables

List of Figures

Listings

Chapter 1

Introduction

"Nemo nisi per amicitiam cognoscitur - No one learns except by friendship"
...one must like a subject in order to study it

"If you can't explain it simply, you don't understand it well enough."
— Albert Einstein (1879 – 1955)

This chapter introduces the problem and presents the aims and objectives of the research. It elaborates on the motivation and rationale of the approach used, and outlines the structure of the thesis and its contributions.

1.1 Background

In recent years, a trend has been building that is propelling Enterprise Information Systems (EISs) towards more distributed, heterogeneous, service–oriented, and therefore increasingly complex infrastructures. This trend is based on the fact that modern systems cover larger and larger functional areas, and as a result, they need to become more scalable and reliable. Moreover, they should be able to run independently and be integrated with a growing number of diverse tools operating in an organisational environment. The integration with existing on–premises deployed environments and the use of cloud computing services allow the enterprise to expand towards bigger computational ecosystems, confirming the overall distributed computing trend.

Almost every large organisation has an enterprise system that is typically offering high quality of service, deals with large volumes of data, and is capable of supporting the organisation's business processes. Rapid changes in the market generate a constant need for modifications, or, sometimes, redesign and re–implementation of significant parts of the systems. Very often, such organisations release newer versions following Agile (Fowler and Highsmith, 2001; Martin, 2002) and Lean (Poppendieck and Poppendieck, 2003) software engineering principles to modify iteratively existing implementations of their internal business processes in a continuous improvement exercise. A far more frequent cycle of modification is implemented with the use of system configuration. The application level parameters tuning and control cycle is depicted in Figure 1.1 in terms of six stages: monitoring, processing target Key Performance Indicators (KPIs), data collection, evaluations and control actuation on a system, where the response is monitored in a full context of hardware and software dynamics.

Because of the frequency of changes, the need for high quality of service, and the sensitivity of data, implemented processes and algorithms used, the responsibility for maintenance and development is often taken by dedicated teams working for the organisation. System operators and developers work closely together to solve any EISs technical issues and deploy a system that accommodate business needs and evolving requirements.

This phenomenal growth in the use of distributed enterprise solutions in the last 25 years has created a need for better monitoring and audit mechanisms. Monitoring can be either tightly integrated in the system (built–in) or deployed as a predefined component (agent–based), very often using standardised monitoring and controlling protocols. In recent years, isolated loosely coupled monitoring and control components have gained more attention, especially in systems following the Service Oriented Architecture (SOA) (Perrey and Lycett, 2003; Sharma et al, 2011) and microservice (Lewis and Fowler, 2014; Dragoni et al, 2017a) paradigms, and we have been observing a new emerging discipline – Application Service Management (ASM), which focuses on monitoring and managing the performance and quality of service in complex computational systems. ASM typically consists a collection of tools and processes supporting detection, diagnosis and reporting facilities helping to supervise and control the service quality of complex business transactions in order to ensure that they meet agreed levels. Furthermore, cloud providers introduce ASM systems specific for their technical stack. The topic of ASM will be elaborated in more detail in Section 2.3.

When it comes to monitoring and auditing, metrics should ideally be gathered from all the significant elements of an existing distributed systems infrastructure and from all tiers of their architecture. A holistic collection process starts with metrics from the physical tier (hardware), then the virtualisation and container tiers, the operating system metrics, and the service tier (database, application servers, web containers, messaging systems, etc.) metrics; it ends with the application tier calculating metrics such as active users count, front–end execution count/time, and application external/internal interface execution count/time. In this context, multidimensional cross–correlation of the various metrics (especially in the time domain) deserves special consideration because it may allow to explore a sequence of changes associated with a specific event that took place in the system. This issue will be discussed further in Section 4.2 which examines the monitoring aspects of the proposed approach.

Critical to the holistic monitoring process is gathering data about the final performance of functionalities exposed to users and other systems. The agreed and declared, in terms of technical definitions, level of commitment between service providers and clients are typically set in a form of Service Level Agreements (SLAs) (Jin et al, 2002; Bianco et al, 2008; Patel et al, 2009). The agreement may contain agreed values or ranges for expected execution time, level of system throughput or error rate. To quantitatively measure Quality of Service (QoS), several related aspects of the service are often considered, such as availability, execution delay, first to interactive, throughput, transmission delay, etc. The measures may differ depending on the type of the service. SLAs together with QoS goals formulate a type of control guidance; more about how this is expressed in a systematic manner is discussed in Sections 2.2–2.5.

Currently, in EISs, one of the most important Non–functional Requirements (NFRs) is scalability. Common practice is to keep a stable level of service (if the system is highly scalable) by adding an increasing number of hardware resources, which is expected to increase the computational power of a working solution. Adding more computational power just to remove any potential risk of saturating a crucial system resource can be expensive and may not be always necessary. Short or mid–term load fluctuations can be managed and mitigated at a lower cost by an adaptive management according to the expected QoS goals. Further chapters of this thesis will be focusing more on discussing this problem, possible solutions, control approaches, and profitability of providing the services in enterprise environments. The discussion will consider such aspects as cost of operation (including energy, hardware, ease of integration, human effort) and effectiveness in the

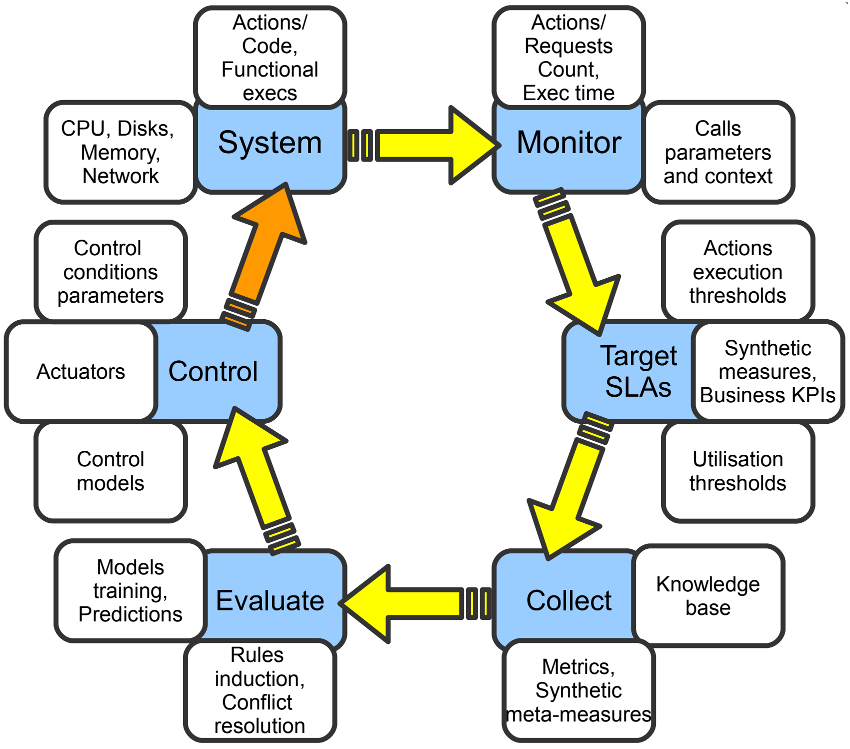

Actions/ Code, Functional execs
Actions/ Requests Count, Exec time
CPU, Disks, Memory, Network
System
Monitor
Calls parameters and context
Control conditions parameters
Actions execution thresholds
Actuators
Control
Target SLAs
Synthetic measures, Business KPIs
Control models
Utilisation thresholds
Models training, Predictions
Evaluate
Collect
Knowledge base
Rules induction, Conflict resolution
Metrics, Synthetic meta-measures

Figure 1.1: ASM control process life–cycle (starting from the top clockwise): (a) monitoring of resources available and application activity, (b) SLA processing, (c) metrics persistence, (d) evaluating control and rule generation, (e) applying control to (f) the system.

context of client exposed SLAs. Moreover, this can be seen as another factor increasing profitability in an enterprise.

Although it is very common and powerful, managing hardware resources via scalability strategies cannot be considered as a general control approach for meeting SLAs targets. Hence, adding/removing resources as a way to optimise performance and energy consumption objectives will not be the focus of this work. There are already several methods available in the literature for that purpose; in general there is strong reliance on the scalability requirement and as a result lack of universality.

The main motivation for the research presented in this thesis is a need for a methodology and software framework that could be applied in order to ease the work of system operators and developers by introducing and exploiting elements of knowledge engineering to adaptive controllers operating in ASM environments. Operators of complex systems having the responsibility of providing a balanced and stable level of service that is appropriate for the expected run–time responses struggle with controlling and constantly changing too many parameters. The proposed approach allows, to a certain degree, system maintenance to be moved towards adaptive controllers, so organisations can use limited system resources much more efficiently. System operators and developers using their experience and intuition together with a tool–set, such as the one developed in this research, can then focus on such aspects as: analysis on a higher level of abstraction, training processes of adaptive controllers, extracting characteristics of not yet detected metrics patterns. This could greatly enhance early identification of problematic events and allow better planning of the future use of the system.

1.2 The Research Motivation

The core aim of the research is to introduce a conceptual model of an ASM self–adapting controlling system of non–acquisitive actuation methods such as action termination and job re–scheduling. The ideation and the architecture are validated by prototype implementation of a control framework tested under use–cases derived from enterprise systems practice.

To this end, the thesis designs an SLA–driven adaptive controller that can be applied to on–premises and cloud applications. The controller actuators can be weaved–into the service layers of the systems and can apply control inputs to the run–time situation by terminating or rescheduling execution of the requested actions (functionalities).

The thesis explores the control problem, and discusses possible solutions and actuation approaches. The emphasis is on investigations of the performance and profitability of providing services in an enterprise environment. The control evaluation considers such aspects as cost of operation (including energy, hardware, ease of integration, or human effort) and effectiveness in the context of client exposed SLAs.

Lastly, the thesis introduces a tool–set that could be utilised in different applications and may support system operators and developers' work by offering software for analyses performed at a higher level of abstraction, definitions of training strategies for adaptive controllers consolidating service management, and further extraction of hidden patterns of metrics. This could greatly enhance early identification of problematic events and better plan future use of the system.

1.3 Objectives

The following thesis objectives were defined in order to achieve the aim:

1. Review the context of the ASM field and the control challenges, and discuss the requirements for an approach that will ease the work of systems operators and developers — Chapters 1 and 2 tackle this objective.

2. Define the control problem and derive an appropriate formulation — Chapter 3 explains the problem statement and declares mathematical nomenclature for the control apparatus to be followed further.

3. Investigate the scope of adaptive monitoring and actuation that could be applied to the ASM control problem, explore means of formal definition of objective functions that include operational and financial constraints in the domain of functionalities, execution time and economics of the enterprise environments — Chapter 4 lists those areas with special attention to SLAs as a declaration of QoS and economic targets.

4. Explore methods of data engineering, data pre–processing, time–series data management, knowledge mining (Mörchen, 2006; Esling and Agon, 2012) and feature extraction approaches to boost the performance of the applied control and to lower the dimensionality of the problem in order to make it understandable by human operators — Chapters 5 and 6 tackle those aspects.

5. Conduct a feasibility study and establish a scale of providing a flexible and extensible software framework that would be deployable into systems to support an ASM suite — this has been done in Chapter 7 of the thesis.

6. Design, implement and test an adaptive controller that is able to actuate the services' work and be applied into enterprise systems operating in on–premises, cloud based environments, or to support cloud providers — Chapters 7 and 8 explain the design and testing of such a controller.

7. Investigate ways to scale and test the approach so that it is able to be applied to modern cloud computing, specifically Software–as–a–Service (SaaS) and Function–as–a–Service (FaaS) models, to optimise system resource consumption and enhance

financial performance — Chapter 8 introduces the reader to this investigation in the context of complex SLA scenarios and explains the controller tests, whilst Chapter 9 elaborates further on wider industry standards.

1.4 Methodology

Modern data centres employ in practice much more computational power and effectively energy (Koomey, 2008; Koomey et al, 2011) than is needed to support a service. Planned overhead is used to ensure that the required level of service is consistently achieved. Notwithstanding the simplicity and robustness of this approach, it introduces clear inefficiencies. Securing extra server power in reserve, ready to be brought into operation in case of a high demand, is an expensive consequence of high inertia of the reserve systems.

Hence, although this strategy is considered an effective approach to secure some strongly required NFRs, such as resiliency, scalability and reliability (Zhang and Lin, 2010), it leads to substantial additional costs.

Therefore, the first priority of the research is to focus on service work actuation instead of task scheduling, and placements of execution rather than virtualisation or scalability.

Consequently, the thesis combines in an innovative way several research approaches to achieve its aim, namely experimental research, build research and simulation research. The experimental methodology uses experiments designed to test a hypothesis. It involves setup, record keeping, test–bed setup design, and experiment results reporting. The build methodology encompasses designing an artefact – a prototype system – reusing components, choosing an adequate programming language, and considering testing all the time. Lastly, the simulation methodology uses computer simulations to address questions difficult to answer in the real world. The thesis also includes several studies which usually employ several computational methods to evaluate their behaviour. The purpose is not necessarily to produce an absolute ranking but to identify performance differences and trade–offs.

Figure 1.2 depicts the iterative nature of the research methodology adopted in the thesis, which contains stages of implementation, test data collection and analysis for presentation, use–cases exploration for results evaluation, modification of properties of test–bed and the release of the prototype for the next round of experimentation.

The stages are presented extensively in later chapters, but a short introduction can be found in Section 1.5.

In order to meet the thesis objectives, several problem areas have to be considered in the research, development and testing. The diagram presented in Figure 1.3 outlines research areas involved in the thesis, main methods designed to tackle the goals and make a link with the relevant chapters, as described below.

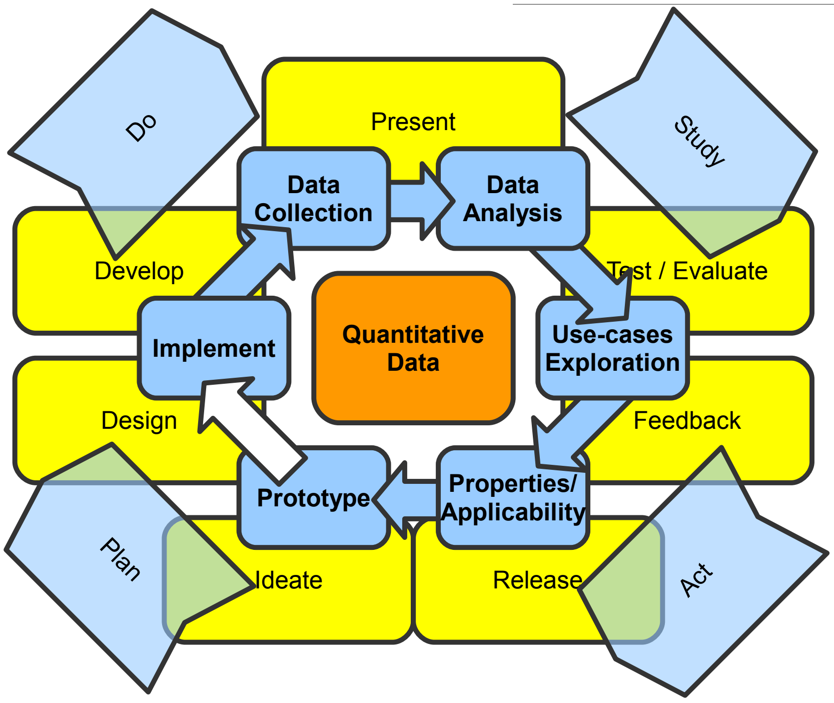


Figure 1.2: Methodology and iterative nature of the experimental research.

1.5 Structure of the Thesis

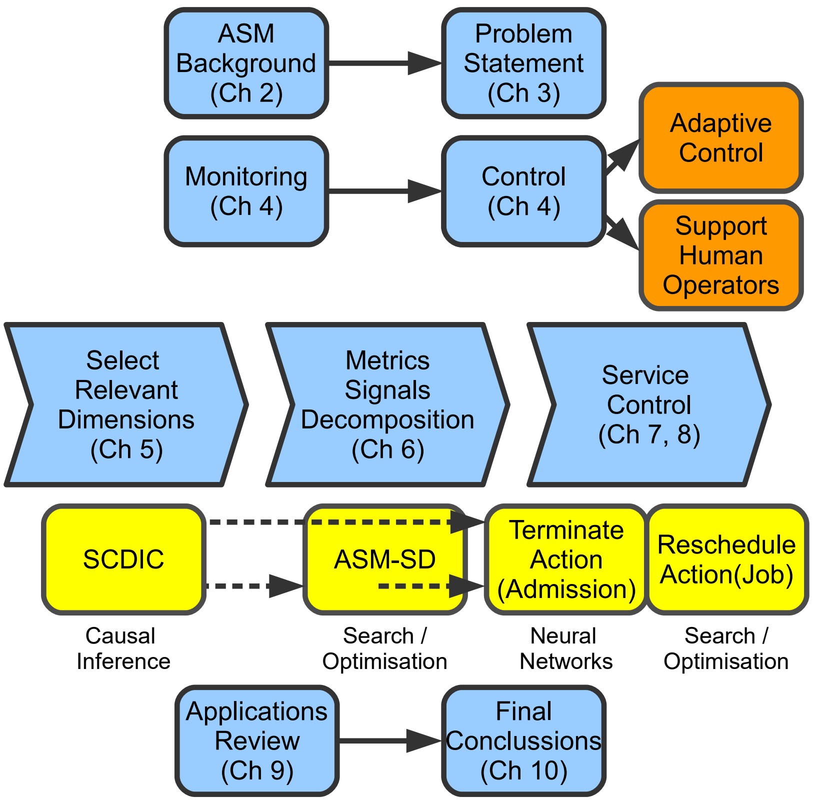


Figure 1.3: Dissertation Structure; Problem areas and Chapters (blue), Methods (yellow) and Goals (orange).

This section explains how the rest of the thesis is organised, referring to specific methods introduced in the various chapters (see also Figure 1.3).

Chapter 2: Background — Discusses the research background leading to the design and implementation work that has been done and the main approaches used. It provides literature review and discusses important concepts in context of ASM which are used in the rest of the thesis, such as monitoring, performance optimisation, adaptive control, service level agreements, etc.

Chapter 3: Problem Statement — This chapter defines the details of the problem that the research tackles and provides a description of the framework, as a conceptual model, at a higher level of abstraction so further chapters can refer to it providing lower–level details, e.g. monitoring, control approach and tested adaptive methods that are specific to a given area of each sub–problem.

Chapter 4: Monitoring and Control Objectives — This chapter focuses on the metrics of complex enterprise systems whose elements offer a basis for assessing control actions that should be taken to optimise the system work according to the specified directions, as well as introduces gathering and collection mechanisms.

Chapter 5: Dimension Relevance — With the use of analysis of time–series metrics, this chapter describes feature selection methods that enable the search for similarities in the signal space, selection of the most relevant dimensions in an enterprise system, and easier control in the reduced space, ultimately reducing maintenance effort. The approach is further used to create causal models of the controlled enterprise system, significantly simplifying evaluation of the defined elements of utilisation dependencies.

Chapter 6: Metric Signal Decomposition — This chapter illustrates the problem of missing data in the extraction of the ASM signal feature and introduces a method to tackle this issue; population–based metaheuristics are applied to solve the problem of activity and resource signal deconvolution as a method of unmixing the activity as input to the system from observed utilisation of resources.

Chapter 7: Neural Adaptive Control — Introduces a learning controller framework for adaptive control in ASM environments and explores its potential. Control actions are executed in the context of the current system state, which is again monitored and stored extending the enterprise system states repository, a database that forms a type of a knowledge base, giving views on the correctness or failure of frequently evaluated control actions. This incremental learning leads to evolving controller behaviour in a particular situation by taking into account consequences of earlier actions in those or similar situations. Experiments have been conducted with use of a real application that has been instrumented with termination actuators and monitored accordingly.

Chapter 8: Advanced Methods for Adaptive Control in ASM Environment — This chapter presents other adaptive control methods that have been tested with use of a modelled system, applying ‘grey-models’ and reduced dimensionality spaces.

Experiments have been conducted with use of modelled applications, which have been instrumented with termination actuators and job re–scheduling control. The simulations contain far more complex spaces/scenarios of SLA conditions.

Chapter 9: Potential Applications — A review of the research findings in the background of the market and industry needs. It addresses questions on the applicability of this work and discusses open problems and limitations.

Chapter 10: Conclusions and Future Work — This is the final chapter that concludes and summarises the main findings of the research. It discusses the main contributions to the research and provides insights for future work.

1.6 Contribution of the Thesis

This section presents the list of main contributions with reference to the thesis chapters.

Main contribution (Chapters 7, 8, 9)

A model for an adaptive controller framework is proposed that in contrast to the approaches found in literature, where a model–free system of any kind is present in the main approach (Chapter 7) and the control is performed with use of incorporated knowledge–base as the main enterprise system metrics and control actions store (Sikora and Magoulas, 2012, 2013b). This allows the controllers to persist all the metrics and SLA values before any control is applied, and then to re–evaluate the applied control actions in line with evolving system characteristics and retrain actuators with new control rules on–line. Such evaluation of control actions does not require setting directly control actions according to defined and delivered SLAs. Also in terms of SLAs, this approach provides a flexible way of defining functions, allowing the selection and combination of all metrics present in the knowledge base.

The controller uses neural termination actuators weaved in the application under control, providing an approach that, to the best of author's knowledge, has not been found in the ASM literature under SLAs setup per function so far.

Other contributions

– Chapters 4, 7, 8, 9: An investigation of the use of termination actuator to reduce utilisation of the resources. To the best of author's knowledge this is a technique that hasn't been used before in ASM systems, especially considering that this type of actuator is weaved in the application runtime and is governed by a control block relying on double Neural Network (NN); see more details in (Sikora and Magoulas, 2013b).

– Chapter 5: Dimensionality reduction is necessary in complex ASM systems, where thousands of dimensions may be present. The proposed Similarity, Consequence, Interference, and Clarity (SCIC) and Similarity, Dependence, Consequence, Interference, and Clarity (SDCIC) (Sikora and Magoulas, 2014a) algorithms help to solve the problem of dimensionality reduction. Such investigations of causality models have not yet been found in the ASM literature. To improve the methods further ASM signal deconvolution techniques have been studied as a search problem– an approach that has not been encountered in the available literature yet.

– Chapters 4, 5, 7, 8: The rule–based similarity criteria approaches allowing adaptive control evaluation in complex scenarios have been proposed and tested in a context of QoS/SLA and resources bounds requirements that can be contradicting with the decision making states. Applying conflicts resolution strategies to formulate effective interventions to the system following the established causal chains, under control feedback loops, is important addition to the wider problem space. This kind of rules generation has not yet been found in research reports published in the ASM field.

A complete list of the author's publications linked to the research area can be found in the Appendix A.

Chapter 2

Literature Review

"Imagination is more important than knowledge.
Knowledge is limited. Imagination encircles the world."
"What Life Means to Einstein" (1924)
— Albert Einstein (1879 — 1955)

"Houses are built to live in, not to look on;
therefore, let use be preferred before uniformity,
except where both may be had."
— Francis Bacon (1561 -– 1626)

This chapter reviews research and practice relevant to the design and implementation work of the thesis, as well as key approaches used in the research area. It introduces the nomenclature of the thesis and defines key concepts, such as monitoring, performance optimisation, adaptive control, service level agreements, which are used in the rest of the thesis.

2.1 System Design and Performance Optimisation

NFRs gathering is a key element of modern Software Delivery Life–Cycle (SDLC) (Chung et al, 2012). Performance, security, durability, robustness, scalability, integrability, stability, auditability, compliance with technical and law standards, portability, modifiability, operability are just a few of the most notable requirements taken into account while designing a new system (Glinz, 2007; Chung and do Prado Leite, 2009). The first five are of paramount importance in terms of resourcing, deployment and chiefly are shaping the architecture of the solution.

Going into the details of software design and modern system architecture paradigms is far outside of the scope of the thesis; however, it is key to highlight that adaptability of the design may often be in conflict with enterprise or functional goals and may even introduce considerable costs. Load characteristics are a result of a superposition of system design, allocated resources, deployment, technologies, database, computational parallelisation, data size, system capacity, etc. Changing any of the factors of a productionised system can be proved to be a difficult task.

Performance engineering has become a separate discipline and best practices (Sydor, 2010; Sturm et al, 2017; Heger et al, 2017) are widely available to both development and operation teams. Validating the systems NFRs under specific constraints for expected throughput, latency, or more explicitly SLAs requirements, in the context of hardware[1] usage, energy considerations (Beloglazov et al, 2011) and software resources[2] , as well as maintaining system models (Brebner, 2016) throughout the full SDLC is essential.

2.1.1 Software Development

Most modern software development methodologies employ an iterative process of Project Delivery Life–Cycle (PDLC), allowing teams to introduce value–added changes that benefit the whole technical and business ecosystem (Fowler and Highsmith, 2001; Poppendieck and Poppendieck, 2003). The software increments should be released iteratively, in stages, phases or milestones, and may do so following the "release early, release often" philosophy (Raymond, 1999). Hence rigour must be upheld to ensure quality performance when not only programming principles (Feathers, 2004; Beck, 2007; Henney, 2010), but also functional and regression testing are of immense importance (Kaner et al, 1999; Becker, 2002; Myers et al, 2004; Martin, 2009; Crispin and Gregory, 2009; Collins et al, 2012).

Test Driven Development (TDD) enforces the introduction of test automation, see Figure 2.1, as a core part of coding (Becker, 2002; Collins et al, 2012). It adopts integration tests, system tests, acceptance tests (Shahin et al, 2017) of the application's presentation layer, User Interface (UI)/User eXperience (UX), Core Web Vitals (CWV)[3], application facades via APIs, and end–to–end tests checking the entire journey of the user, see following sections and Figure 2.14. That offers an effective way to ensure that performance under load meets acceptance criteria by measuring execution times and resource allocation of a system that is running according to a system specification and a set of deployment standards (see Figures 2.2—2.3). In this setting, it is imperative to organise processes for Continuous Integration (CI) and Continuous Delivery (CD) using load and stress tests

[1]To name few as a basis: CPU and memory utilisations, disks state with amount of I/O read/writes, etc.

[2]The list of specific metrics could be very long, yet to name several groups for a better context: Relational Database Management System (RDBMS), Non–SQL, Non–Relational Database Management System (NoSQL), Enterprise Service Bus (ESB), Java Message Service (JMS) queue lengths, Web Service (WS) calls to Application Programming Interfaces (APIs), Java Virtual Machine (JVM) Java Management Extensions (JMX), etc.)

[3]CWV can be seen as SLA enforced by search–engine vendors, so joint with Search Engine Optimisation (SEO) targets.

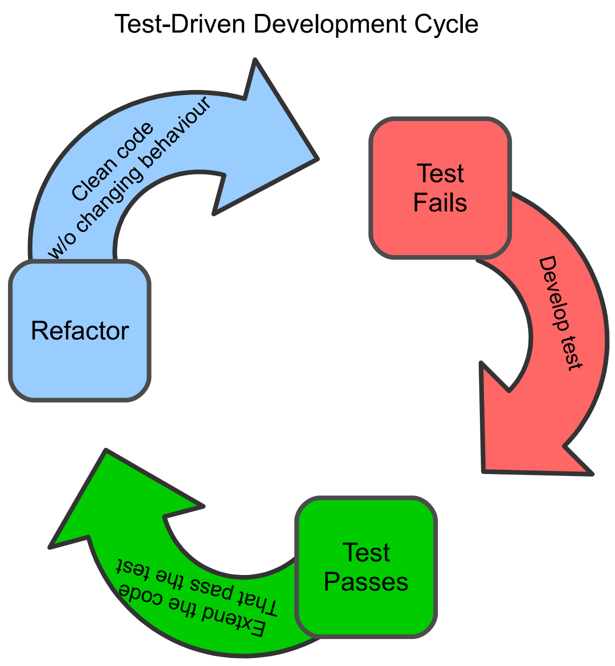

Test-Driven Development Cycle
Test Fails
Develop test
Test Passes
Extend the code That pass the test
Refactor
Clean code w/o changing behaviour

Figure 2.1: TDD Lifecycle.

that replicate load conditions, or extreme performance load, in order to gather system performance characteristics. This issue will be discussed further in another section of this chapter in the context of application SLA.

Nowadays, when development teams grow day by day, finding the right approach for scaling a team is essential for organisations. Scaling introduces a series of challenges with respect to knowledge management and the practices adopted by facilitators in large teams (Larman and Vodde, 2016; Knaster and Leffingwell, 2017; Leffingwell, 2018). Recently, the tendency is to combine development and operations teams in order to enable them to work as closely as possible in self–organising sub–teams (Dingsøyr and Moe, 2014; Dikert et al, 2016; Almeida et al, 2019). Inter–team coordination in such squads facilitates monitoring of development and production environments and enables testing of NFRs far–more effectively, ensuring adherence to service levels, and applications. This aspect, however, deserves separate attention in the context of ASM environments and will be discussed in the next section.

2.1.2 Operations

Since functional and non-functional system changes are frequently applied in the organisational infrastructure and refactoring (Roberts, 1999; Fowler, 2018; Fritzsch et al, 2018) is an iterative process of development, integrated as an internal phase of SDLC, software performance must be verified as apart of the regression suite. Moreover, many refactoring objectives are driven purely by performance improvements targets. In an ASM environment, changes to performance characteristics should be considered from the moment a new version is released not only by development teams, but, even more importantly, by

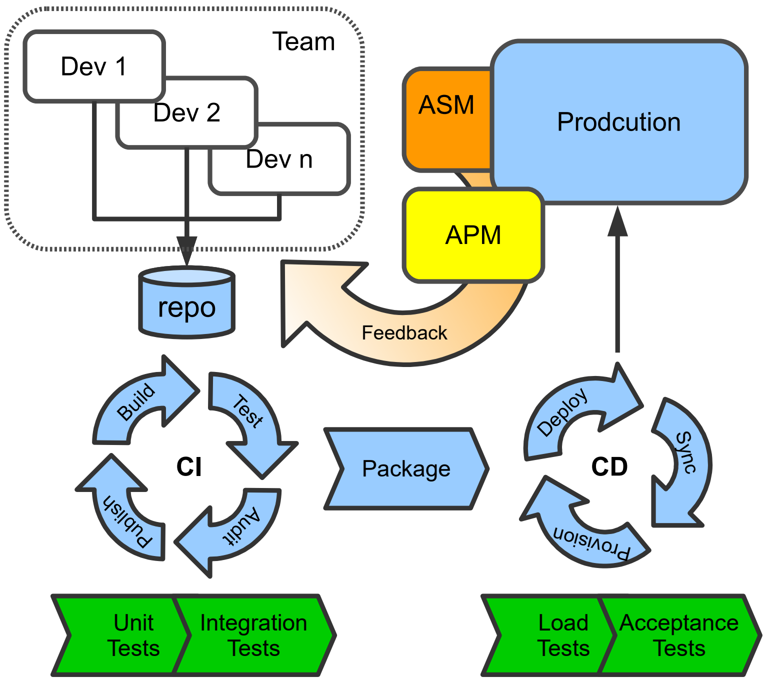


Figure 2.2: Continuous Integration and Continuous Delivery.

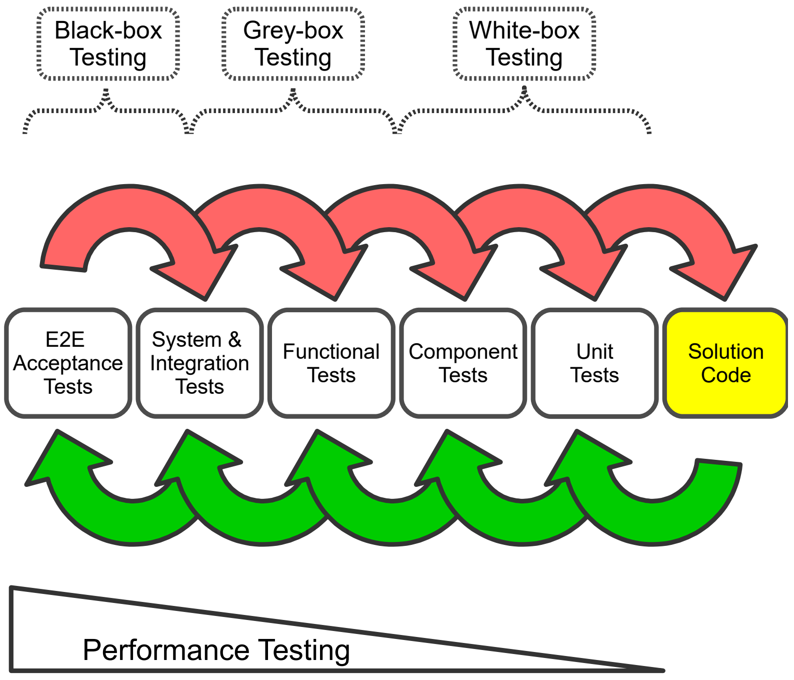


Figure 2.3: TDD life cycle and sequence of acceptance criteria checks in the context of performance testing.

operations crews– see Figure 2.4.

System operators inform their decision making about the tuning of specific services by taking into account collected data and suggest solutions to the development team. As a result, more and more new applications are designed in a way that allows higher real–time auditability and parametrisation of dynamic environments. Tuning activities can also be performed on aspects of critical software components, such as databases, application persistence layers and built–in caches, web containers, application servers, messaging queue brokers, external caching services, etc.

However, any decisions to tune sensitive parameters have to be made under conditions of high visibility of the executing processes of an ASM environment. Almost every change in a working ASM solution, under conditions of limited resources, may impact negatively other areas of the ASM environment, instead of bringing the benefits that they were originally anticipated. Although testing procedures in staging environments can reveal potential problems after deployment, it may still be difficult to identify all issues and pinpoint the exact cause and location due to limitations in the accurate representation of the live–system load.

Constant comparisons of technical and non–technical production metrics, performed before and after any changes are implemented, provide an outlook for the correctness of the operators' decisions and activities. That is the main reason why, in such environments, a continuous monitoring approach as a proactive measure is absolutely necessary, as will be discussed later in Sections 2.4 and 2.3.

Figure 2.4 illustrates the various dimensions of the performance optimisation process in the context of ASM environments. Where, starting from the top, it includes as inputs: (a) NFR, (b) infrastructure, architecture and cost considerations, and (c) effective business usage, load and SLAs. Whereas, the outputs are: (a) operation activities performed by Admins, DevOps, and Application Support teams, (b) redesign, refactoring and other development team activities, and finally, in the bottom centre, (c) automated adaptive control actions that are the result of ASM search space optimisation conducted in the context of the collaboration with the operations and development teams.

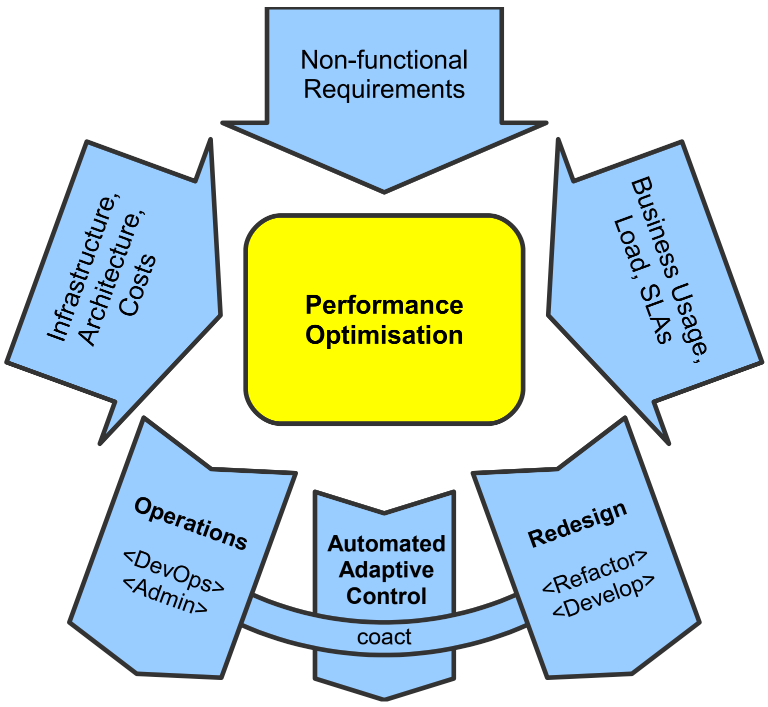

Non-functional Requirements
Infrastructure, Architecture, Costs
Business Usage, Load, SLAs
Performance Optimisation
Operations
<DevOps>
<Admin>
Automated Adaptive Control
Redesign
<Refactor>
<Develop>
coact

Figure 2.4: Performance optimisation in an ASM Environment where operations, development teams and adaptive control agents co-exist and collaborate.

2.2 Application Service Level Agreement

SLA is a form of agreement between a service provider and its clients that documents what services the provider will support, and declares the standards of execution the provider is required to meet. Since SLAs represent what is expected from a client–side service characteristics, they become a critical element defining the direction of optimisation, which can be done manually and/or automatically. In the thesis, the SLA, as a concept, is the cornerstone of the proposed adaptive control framework for ASM. Chapters 7 and 8 cover SLA definition in more detail and present how this is used in the context of the proposed framework.

However, it is worth noticing at this stage that although SLA for an acceptable level of service per action request is usually defined with the use of a single dimension, e.g. latency, more complex agreements, which use other metrics (i.e. executions count in time, amount of services employed, availability and so on) or combinations of them, can be used. This leads to the formulation of complex functions that makes application–level control extremely challenging. For example, performance and SLAs optimisation becomes particularly demanding when systems utilise serverless computing and SaaS environments, where an extra level of deployment abstractions and reduced control of the allocated resources may introduce additional degradation in the performance of the services.

Also, the global nature of cloud computing many times requires SLAs to be defined across many jurisdictions and legal requirements. Consequently, it becomes increasingly important to introduce standardisation guidelines (Van Der-Wees et al, 2014).

The subject of managing QoS by introducing software frameworks that use SLAs as agreements for life-cycles, including creation, expiration, and monitoring of agreement states, has been thoroughly researched (Keller and Ludwig, 2003; Theilmann et al, 2010). Complete protocols introducing rewards and also penalties as forms of the consequences of not meeting or violating agreed specified or guaranteed service standards have been formulated (Andrieux et al, 2007). More recently, Ziegler (2017) reviewed in detail applicability of SLAs in cloud computing. Concepts of SLAs are important for API, exposed WS and UI where good performance needs are defined for UX.

2.3 Application Service Management

ASM is a discipline which focuses on monitoring and managing performance and QoS in complex enterprise systems. In the context of the dynamic development of cloud computing there is increasing support of ASM offered by cloud service providers. Advances in this field laid the foundations for the Platform–as–a–Service (PaaS) environments and Serverless computing paradigms.

Generally speaking, the use of application service management is common for complex, multi–tier transactional applications, where holistic runtime information is delivered to the operations end–user from most of the critical layers of the instrumented systems. ASM software suites can be deployed to on–premises, in–cloud and hybrid environments. Cloud providers offer Cloud Application Management (CAM) software whose functionality is reflecting specifics of available services and their supported features (Kolb and Röck, 2016).

In cloud management services delivering PaaS, a need for a common approach has emerged in the form of Cloud Application Management for Platforms (CAMP). This has been introduced in order to implement an instrumentation that can contribute to a wide range of administration tasks, such as configuration, log auditing, monitoring, and patching of applications deployed on the platform (Carlson et al, 2012). There is also an important niche of ASM, not related to a cloud vendor, that can be applied to hybrid cloud or on premises deployment and container orchestration – see App Dynamics[4], DataDog[5], Moogsoft[6] or Dynatrace[7] systems (Prasad and Rich, 2018; Hoorn and Siegl, 2018; Pourmajidi et al, 2018; Noor et al, 2019). Moreover, often encountered in hybrid environments, after introducing wider virtualisation (hypervisor–based) technologies, like Docker, Docker Swarm or Kubernetes, or Serverless computing deployments, the adoption of ASM is growing since complex systems are disproportionately impacted by the introduction of the above techniques into existing systems integration and architectures. In this context ASM offers

[4]https://www.appdynamics.com/
[5]https://www.datadoghq.com/
[6]https://www.moogsoft.com/
[7]https://www.dynatrace.com/

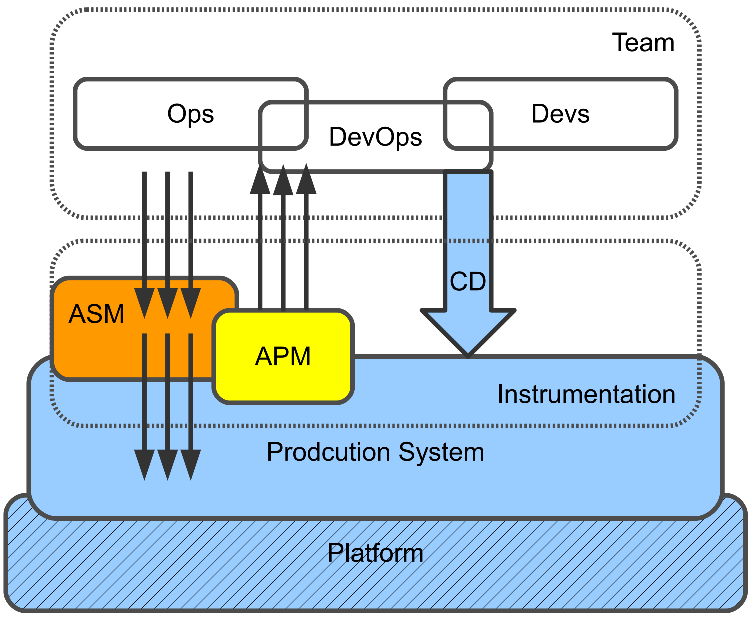


Figure 2.5: ASM, service adaptive control, SLA monitoring, continuous artefacts deployment (CD) and performance optimisation in a context of systems, platform and user groups.

visibility of key components, runtime status information, configuration interfaces, and in effect, allows deeper application dependency analyses.

Management software integrated within the deployment plan enables development teams, operators, admins and DevOps to control the environments in a far more comprehensive manner (Bass et al, 2015; Sussna, 2015; Kim et al, 2016) , taking into account SLAs, business–value targets, platform–costs and even energy–awareness considerations. ASM allows systems tuning and modification of vital application parameters, in larger distributed deployment and cloud systems, not only through faster automation for rolling deployment of new software and an enhanced flexibility for spinning off instances, but also through enabling run–time modifications that control application artefacts.

It is projected that the use of automation and adaptation methods that do not require human attendance is going to continue growing in ASM. Gartner envisages that IT Operations Analytics (ITOA), Algorithmic IT Operations, Artificial Intelligence for IT Operations (AIOps) (Prasad and Rich, 2018) and Machine Learning Operations (MLOps) will have an increasingly important role in historical and Root–Cause Analysis (RCA), complex event and anomaly detection, correlation and contextualisation for general performance analysis of modern complex systems. The subject area has been studied recently (Masood and Hashmi, 2019; Mormul and Stach, 2020; Levin et al, 2019) as an evolution from traditional systems management to Smart DevOps or further to AIOps. Figure 2.5 illustrates how ASM suites for adaptive service control and Application Performance Management (APM) systems are placed in the enterprise software management process, as explained in more detail in the following section.

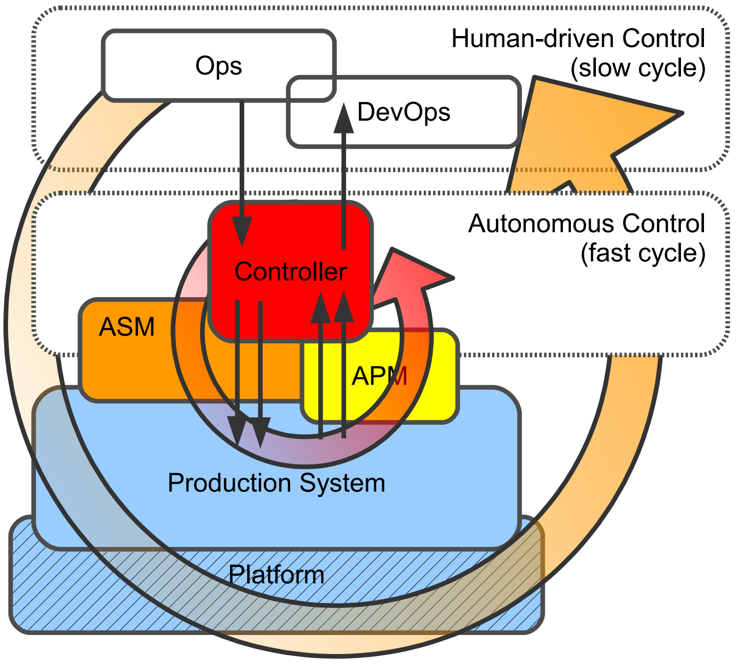

Ops
Human-driven Control (slow cycle)
DevOps
Autonomous Control (fast cycle)
Controller
ASM
APM
Production System
Platform

Figure 2.6: ASM human–driven and autonomous adaptive control of services.

An ASM controller needs to react adaptively to changing system conditions in order to optimise a defined set of targets, i.e. SLAs, as objective functions set under Application Response Measurement (ARM). The controller can be governed either by a human manually, e.g. though scripted automation, or by autonomous agents. The less human oriented control and more autonomous decisions are made, without the need of external interventions, the faster the feedback loop the regulator system is exposed to gets, and the better the conditions for the general performance of steering. Figure 2.6 depicts the combination of human–driven and autonomous control in presence of ASM, APM and controller components in a context of feedback–loops size and delays. The stronger the footprint of the control to application, system and platform, the greater the change to the complex system and more important the need for larger feedback–loop involving human operator. In this thesis autonomous control is understood as a process that involves minimal external human assistance even under significant uncertainties in the ASM environment for extended period of time. Furthermore the autonomous control should exhibit adaptation (Åström, 1983) with a high degree of autonomy, even with the power of self governance, and abilities to compensate for failures without the need of external intervention (Antsaklis et al, 1991).

2.4 Application Performance Management

APM is a software designed to help developers, DevOps, operations personnel, and administrators to ensure that the applications users work with meet functional and non–functional performance standards and support expected quality of user experience, often given as per SLA defined.

APM is a part of ASM field in a sense it tackles widely speaking performance monitoring while ASM is focused more on business service delivery.

Typical features of an APM tool–set include:

- user experience monitoring,
- business transaction monitoring and SLAs tracking,
- performance data and response time monitoring,
- infrastructure monitoring,
- online metrics data traversing, dice and slice or other deep–dive techniques into captured data,
- reporting, dash boarding and alerting capabilities.

As of now APM suites introduce truly widely adopted set of techniques that are instrumental to identify elusive performance problems. The market of APM tools is very dynamic and it is beyond the scope of this thesis to analyse their development, which is naturally expected to change significantly in coming years with the developments in the cloud computing area[8]. Nevertheless, we provide below a list that includes most notable open–source but also proprietary products and their setups currently seen.

- Open source: Nagios, Zabbix, Munin, Monit, Pinpoint, Apache SkyWalking, Stagemonitor, Scouter, (platform diven) JavaMelody (JVM), App Metrics (Node.JS, IBM), Codespeed (Python), GoAppMonitor (Golang).
- Proprietary, enterprise scale: App Dynamics, Dynatrace, AWS CloudWatch, IBM Cloud APM, Microsoft Azure App Insights, New Relic, Datadog, Epsagon, Moogsoft (for AIOps), Solarwinds, Exoprise, Instrumental, Riverbed, Oracle Application Performance Monitoring, Tingyun.
- Active monitoring: DataDog, Monitis, Site24, Cula.io, Pingdom.com.
- AIOps: Moogsoft AIOps, Splunk's IT Service Intelligence (ITSI) tool, BMC's TrueSight platform, Cisco's Crosswork Situation Manager.

2.5 Service Control of Applications, Systems and Platforms

There are many layers where the control of a service or an application may be executed as part of a wider system of integrated actuator services. The most notable for administration

[8]The trend will be highly impacted and interconnected with the advantages of APM suites offered by cloud computing providers.

actions (human–oriented), and scripted control (automated, but human–driven) are listed below:

- hardware,
- operating level (administration),
- database level (tuning, administration, avoidance of backups impacts),
- application level (application support and administration interfaces),
- batch jobs scheduling (find and set up execution time for best use of resources and business needs),
- load balancing (round robin, least connections, least time, random with a draw of one, power of two random choices),
- resources allocation (cloud formation, spin–off thresholds),
- cached objects memory tuning and control.

Issues of database control for resilient data persistence under scalability and data sharding or partitioning require a far different approach and will not be considered in the thesis. The thesis focuses on application functional level load control rather than dealing with indexing, transactions performance, distributed transaction, eliminating single points of failure, shared–nothing architectures, or eventual consistency challenges faced under high scalability requirements.

Along this direction, the main emphasis of the research described in the thesis was put on application–level adaptive control, of automatic and human–coordinated actuation of:

- requests termination (admission control, on action/request, threshold oriented),
- micro–scheduling,
- jobs scheduling.

Operating System (OS)–level task control or Virtual Machine (VM) scaling have not been researched as part of this work, although there are aspects of the research described in the thesis that could be applied to:

- OS level schedulers,
- hypervisors,
- create a joint control with servers / nodes allocation (horizontal and vertical scaling),
- cloud infrastructure and serverless computing.

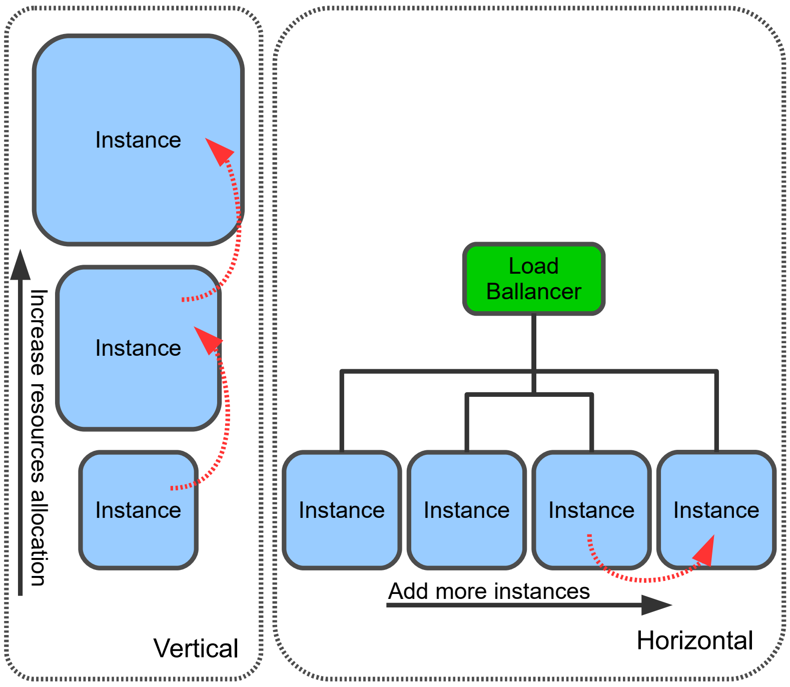


Figure 2.7: Vertical and Horizontal Scaling.

Further details of the proposed approach will be presented in the rest of the thesis. Chapter 3 will provide mathematical formulations, whilst Chapter 4 shall focus more on technical details of the control methods, in particular Sections 4.6 and 4.7 contain further discussion of the aspects delineated above. Chapter 9 discusses possible applications in more detail[9].

2.5.1 Resources Allocation: Scaling Horizontally and Vertically

As already mentioned in Chapter 1, scalability as an NFR, closely related to resilience, needs to be present at the very first phases of a design when a new system is proposed. Generally speaking there are two dimensions of scaling. Scalability can be categorised as vertical or horizontal (Lehrig et al, 2015):

- Vertical (up/down) — Add more or remove resources to a system or a single machine. Typically involving change of allocation of memory, CPU, or storage. This can be achieved far easier in virtualised environments present on premises or in the cloud. This type of scaling is not focusing on resilience since the architecture still may contain single point of failure.
- Horizontal (out/in) — Add or remove machines or nodes in a cluster setup or a distributed environment for a software system (Dutreilh et al, 2010; Mendoza and Amistadi, 2018). In this vein, a controller spins up or removes entire servers or nodes to align with load or resilience requirements. Linear horizontal scaling may offer far

[9]In particular one can observe Figure 9.6 that demonstrates comparison of various control techniques in a background of service reactivity as a function of system load.

bigger computational power than large supercomputers. The price of such a setup is typically lower than substituting it for a large machine since commodity hardware is normally used. This approach typically requires a load balancer or some other routing mechanism that can redistribute the load according to the method used, i.e. random, round–robin, per action or load type, etc.

The bigger and more complex systems are, the higher the need to decompose the application into smaller services. The more microservices powering the system, the lower the granularity of the service and thus the higher the need to orchestrate the independent components. Therefore, apparently, as a remedy to the growing complexity, systems become even wider and contain many isolate run–time services, impacting the easiness of human control (Lewis and Fowler, 2014; Dragoni et al, 2017a; Zimmermann, 2017).

Although advances to increase computation power with higher–clock–rate have been pushing processors manufacturers to implement computation routines into hardware and have, indeed, led to performance increase during the last five decades, the "ILP wall" and "power wall" have created barriers to that approach. Thus, the popular version of Moore's Law (Moore et al, 1965) – increasing performance with each generation of technology – is now going to be led by (a) programmers dealing with more and more parallel and distributed application, and (b) cloud computing service providers (Hennessy and Patterson, 2011, p. 55) who will be only amplifying needs of that control domain[10].

Growing importance are gaining cost–related factors; especially energy consumption which gets more and more important as well as difficult to optimise for bigger computations centres at high scale[11].

2.5.2 Scheduling and Micro–scheduling

The system is capable to schedule resource intensive processing for a time when the system's expected load allows that operation. Critical for the operation of this scheme is knowledge about the impact on the resources that are deployed based on the execution functionality and the time of the processing for a given set of parameters. The optimisation process can be driven by the SLA (Wu et al, 2012), energy consumption targets (Mehdi et al, 2011; Cui et al, 2017), or time–oriented requirements for the execution of critical tasks.

Micro–scheduling is in essence the same technique but for actions that SLA require far faster response, i.e. UI web requests or web service calls. Figure 2.8 depicts the control considerations in this scenario. Chapter 8 discusses this issue in more detail.

[10]The interested reader can find more about this direction of work in the appendix,

[11]A perspective on these issues is reported in the appendix, Sections B.1.3–B.1.5.

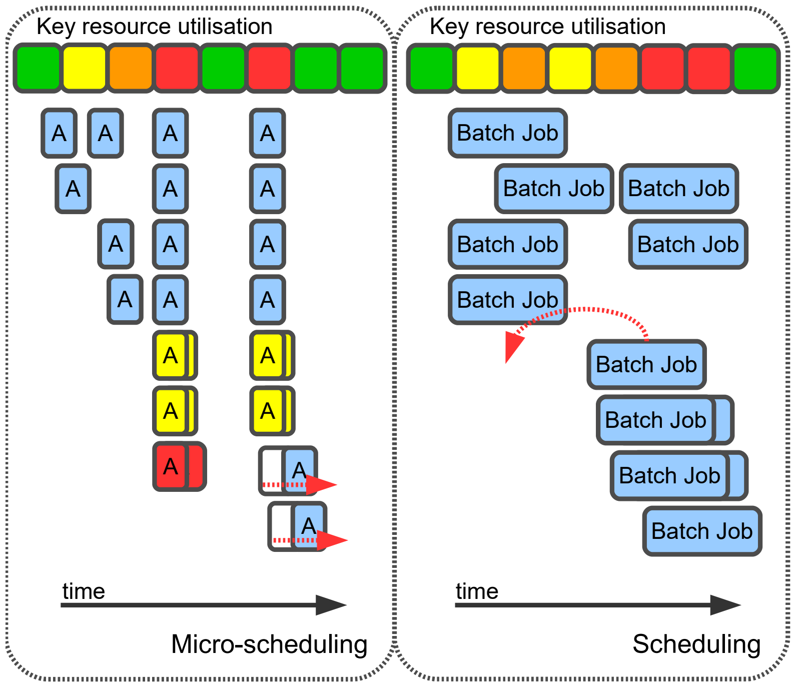


Figure 2.8: Micro-scheduling of requests and batch job scheduling.

2.5.3 Admission Control and Requests Termination

In order to release resources required to compute incoming requests for services processing, a controller can terminate the request, or discontinue processing operation that take longer than expected. Figure 2.9 illustrates the control in such use–cases.

This approach is especially effective in situations where there is instantaneous raise of services requests. It can be also seen as very supportive in securing other control mechanisms of wider healing impact that require more time to apply the control, i.e. to spin–off new instances or reshuffle VMs and add more memory to application.

Requests or actions termination can be done via admission control interfaces present in ASM suites instrumenting platform or by applications exposing such functions. Suspension of further action execution can be far easier in a context of FaaS or serverless computing since the function framework is already present and can be integrated fairly easily with controllers.

Chapters 7 and 8 tackle the problem in more details, whilst discussion of applications of such a control approach is provided in Section 9.5. The most promising potential of the method seems to be though when laying in systems that have real–time requirements. Such situation requires special attention and has been explained more in the following section.

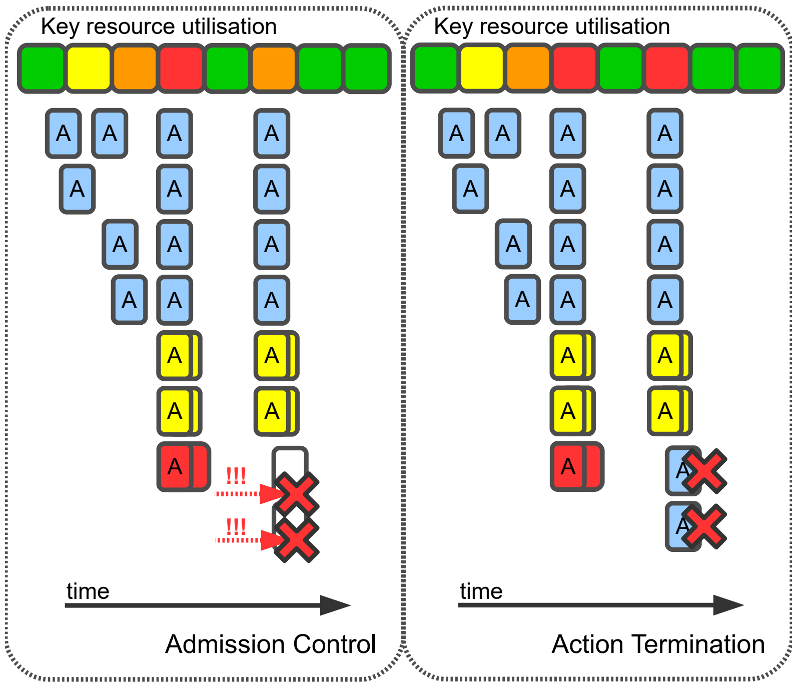


Figure 2.9: Admission Control and Action Request Termination.

2.5.4 Real–time SLA and Requests Control

In practice, real–time SLA processing fails if execution is not completed within a specified timeframe, depending on the action type. Deadlines must always be met, regardless of system load or functionality. Attempting to optimise resources allocation in a system experiencing higher load is a typical control consideration (Buyya et al, 2011; Al-Dahoud et al, 2016; Hwang et al, 2016; Tran et al, 2017). Alternative group of methods in contrast to scaling that do not require additional resources of non–acquisitive control are actions termination together with scheduling control, as it will be elaborated in Chapters 7 and 8.

2.6 Control of Application Services

The control of application services and systems has been the subject of substantial focus in the last two decades. Parekh et al (2002) were researching the area of controllers using classical control theory and standard statistical models in the background of SLA, where two distinctive steps system identification and the controller design are present. More pragmatic approaches were studied in (Abdelzaher and Lu, 2000; Abdelzaher et al, 2002; Zhang et al, 2002b,a; Lu et al, 2002; Abdelzaher et al, 2003; Lu et al, 2003), where application services performance control schemes with service differentiation using classical feedback control theory were proposed to meet established SLA targets and overload protection. Hellerstein and Parekh et al. introduced a comprehensive application of standard control theory to describe the dynamics of computing systems and apply system identification to the ASM field (Hellerstein et al, 2004), highlighting engineering challenges and control problems (Hellerstein, 2004). Fuzzy control and neuro–fuzzy control were proposed in (Hellerstein et al, 2004) as promising for adaptive control in the ASM field.

Since then many different methods of control and various techniques for actuators in ASM have been proposed e.g., adaptive admission control as dynamic service overload management protection (Welsh and Culler, 2002), event queues for response times of complex Internet services (Welsh and Culler, 2003), autonomic buffer pool configuration in a DB level (Powley et al, 2005), model approximation and predictors with SLA specification for different operation modes (Abrahao et al, 2006), observed output latency and a gain controller for adjusting the number of servers (Bodık et al, 2009).

The SLA as a concept based on QoS has been widely adopted in the business (Park et al, 2001). SLA definitions are often used in cloud and grid computing control (Vaquero et al, 2008; Buyya et al, 2009b; Patel et al, 2009; Stantchev and Schröpfer, 2009). Thus, more recent works in this area tend to focus on performance control in more distributed environments, e.g. virtualised environments performance management (Xiong et al, 2010), virtualised server loopback control of CPU allocation to multiple applications components to meet response time targets (Wang et al, 2009). Cloud based solutions with QoS agreements and energy–aware services were researched by Boniface et al (2010), Goudarzi and Pedram (2011), Zhang et al (2012), and Sun et al (2011), where multi-dimensional QoS cloud resource scheduling with use of immune clonal was researched. An interesting work by Emeakaroha et al (2010) presents cloud environment control with the use of a mapping from monitored low–level metrics to high–level SLA parameters. Power management by an on–line adaptive ASM system was researched in (Kandasamy et al, 2004; Kusic et al, 2009), and energy saving objectives and effective cooling in data centres in (Chen et al, 2010; Bertoncini et al, 2011).

Bigus over twenty five years ago applied NNs and techniques from control systems theory to system performance tuning (Bigus, 1994). Data were collected to train neural network performance models. For example, NNs were trained on–line to adjust memory allocations in order to meet declared performance objectives. Bigus and Hellerstein extended that approach in (Bigus et al, 2000) and proposed a framework for adjusting resources allocation, where NNs were used as classifiers and predictors. In contrast to these works, Sikora and Magoulas (2013b) used gathered system performance measures, such as devices utilisation, queue lengths, paging rates and execution times, to build a knowledge base. This approach takes advantages of an isolated metric store that can be continuously extended and analysed by a control rule generation engine with minimal performance footprint for the system under control. Moreover this control engine can terminate actions, whilst in previous attempts no control based on actions termination was incorporated.

Currently there are many out–of–the–shelf APM and ASM suites (Kowall and Cappelli, 2012), mainly focusing on network and operating system resources monitoring. Even though manual ASM administration is widely used together with APM tools, autonomous control in ASM environments still needs substantial research to be conducted. This is a dynamically growing domain; however, certain aspects still seem not to draw enough attention. For example, despite recent progress in the control theory deployment to design the controllers in the ASM field (Parekh et al, 2002), the use of neural networks and their ability to approximate multidimensional functions defining control actions in system states remains under explored.

2.7 Adaptive Control Approaches Used

An objective of the thesis is to develop a governing approach for controlling computing systems by exposed interfaces under ASM orchestration to support set up objective functions, where a control action is applied in an optimal manner with minimal delay and ensuring the system stability is protected. Hence, it would be useful to present some key works that exploited concepts from the adaptive control field(Åström, 1983; Bechlioulis and Rovithakis, 2008).

Based on earlier works (Lu et al, 2000) in the adaptive control field, Lu et al (2002) used adaptive control to manage resources of the proxy server caches to provide proportional differentiation on content hit rate. Adaptive control theory allows to deal with instabilities caused by system responses variations in a way where an adaptive controller can modify its behaviour in response to changes in the dynamics of the process of the application services and the character of the disturbances (Åström, 1983), with adjustable parameters, but also with mechanisms to adjust the parameters (Åström and Wittenmark, 2013).

Several neural adaptive control schemes have been developed so far, these schemes have been used to tackle general problem of nonlinear systems (Polycarpou, 1996) also under uncertain Multi–Input Multi–Output (MIMO) systems (Ge and Wang, 2004) and time–delays (Ge et al, 2004).

In this context, adaptive control in the field of ASM systems can be applied to SLAs variants (Emeakaroha et al, 2010), to update the controller, so that it works in the regions of newly established states; for example by updating a knowledge base (Bertoncini et al, 2011) or conducting repeated model updates by NNs training (Sikora and Magoulas, 2013b) or as a self–tuning queuing model (Liu et al, 2006), where the introduction of the adaptive controller helps adjusting, to some extent, the model to the system dynamics.

The range of available approaches spreads from white–box (Section 2.7.1), where maximum knowledge about the system is present a priori, to black–box (Section 2.7.2), which rely on ongoing collected data that can be used to build and maintain a knowledge about systems responses on experienced inputs. Figure 2.10 presents an outline of the approaches and key characteristics that can support their utilisation. As explained in Section 1.4, this issue is further elaborated in Chapters 7 and 8 (cf. with Figure 1.3 illustrating dissertation structure and the different approaches applied in the study).

There are many general control theory optimisation methods applied to cloud computing, i.e. with use of feedback control for adaptive VMs management (Li et al, 2009) and hardware resources management, (Lim et al, 2009; Manvi and Shyam, 2014), with data–driven and model based predictive controller (Xia, 2015), with self adaptation targets (Farokhi et al, 2015; Xia, 2015), with standard subgradient methods for distributed rate limiting (Raghavan et al, 2007), using rule–based scaling regulators control (Vaquero et al, 2011), and with revenue (Zhang et al, 2011) or SLA driven targets (Serrano et al, 2013; Wu et al, 2012).

Tables 2.1 and 2.2 present a comparison of 50 most notable existing studies in the area of service management and control that can be applicable to ASM. The comparison s split to two parts in order to present the wider scope of the references and to provide comprehensive literature background for further definition of the control challenge as part of the problem statement in Chapter 3. Certainly there are much more works in the area of resources allocation and scheduling rather than admission control and request termination. One can note that with time and with growing complexity of the systems, given in quantity of monitoring and actuations, more and more research concentrate on the control approaches that do not relay explicitly on the control theory support. Furthermore, the adaptive control is delivered not only by the control parameters update or existing model tuning but more by model extension or rebuild. Also, while the complexity raises more recent works tend to focus on pragmatic control performance evaluation in a context of holistic targets, also self– management, monitoring, optimisation and using more complex models (Gill et al, 2020; Tuli et al, 2022; Gill et al, 2022) that overlap with an area of autonomic computing (Kephart and Chess, 2003; Computing et al, 2006; Huebscher and McCann, 2008; Brun et al, 2009) which emphasises control systems can achieve the specified the targets on their own, completely without any intervention of a human operator, which is not a core consideration of this thesis.

2.7.1 White–box Approach

Enterprise class system architectures can vary substantially depending on functionality, deployment approaches, interconnections to other system, and other NFRs. There can be many different components and software tiers present, which consist internal and externals services calls, according to implemented routines. Typically the system can host hundreds to thousands exposed system actions (functions), and be described by hundreds of software and hardware resources depending on deployment architecture for high–availability, scalability and redundancy (Haines, 2006).

In fact a system code base, deployment scripts, cloud formation metadata, and defined control interfaces all contribute to a complete and final system structure, that is in essence a very precise definition of the systems as a whole. That is a required foundation of causal inference methods (Pearl et al, 2009; Pearl and Mackenzie, 2018), which can be applied to find elements of the system that have the biggest impact on the final SLA targets and in this way maximise control performance, as described in Chapter 5.

Table 2.1: Comparison of existing studies in a field of service management and control published before 2008.

No.	Work	Resources Provisioning	Scheduling or Micro–scheduling	Admission Control	Multiclass control	Observable areas[a]	QoS or SLA Aware	Direct overload prev., Security	Energy Awarea	ASM aware	Real-time or Deadline sensitive	Control theory support	Control model[b]	Adaptive Control	Experiments[c]	Cloud related
1	(Bigus, 1994)	y[d]	n	n	n	f	n	n	n	n	n	n	N	y	r	n
2	(Abdelzaher and Bhatti, 1999)	n	n	y	y	f	y	y	n	n	n	y	D	y	r	n
3	(Lu et al, 2000)	n	y[e]	n	n	f	y	n	n	n	y	y	D	y	r	n
4	(Keller and Kar, 2000)	n	n	n	y	–	n	n	n	y	n	n	–	n	r	n
5	(Abdelzaher and Lu, 2000)	n	n	y	n	f	n	y	n	n	y	y[j]	R[f]	y	r	n
6	(Park et al, 2001)	y	n	y	y	f	y	n	n	n	y	n	–	n	r	n
7	(Zhang et al, 2002a)	y[g]	n	n	n	f	y	n	n	n	n	n	Q	n	c	n
8	(Zhang et al, 2002b)	y[h]	y	y	y	m[i]	y	n	n	n	y	y[j]	D	y[k]	c	n
9	(Lu et al, 2002)	y[l]	n	y[m]	y	f	y	n	n	n	–	y[j]	D	y	r	n
10	(Parekh et al, 2002)	n	n	y		f	y	n	n	n	y	y[j]	D,S	y	r	n
11	(Abdelzaher et al, 2002)	y	n	y[n]	y	m[o,i]	y	y	n	n[p]	n	y[j]	D	n	r	n
12	(Welsh and Culler, 2002, 2003)	n	y[q]	y	y	f	y	y	n	n	n	n	S	y	r	n
13	(Abdelzaher et al, 2003)	y	n	y		f	y	y	n	n	n	y[j]	D	y	r	n
14	(Lu et al, 2003)	n	y	y	y	f	y	y	n	n	y	y	D,Q	y	r	n
15	(Hellerstein, 2004)	y[r]	n	n	n	f	y	y	n	n	n	y[j]	Q	y	r	n
16	(Kandasamy et al, 2004)	y[s]	n	n	n	f	y	n	n	n	n	y[j]	D[t]	y	r	n
17	(Powley et al, 2005)	y[u]	n	n	n	f	n	n	n	n	n	n	R	y	r	n
18	(Abrahao et al, 2006)	y[v]	n	y	y	f	y[w]	y	n	n	y	n	R,Q	y	r	n
19	(Liu et al, 2006)	n	n	y	n	f	y	y	n	n	n	y	Q	y[x]	r	n
20	(Raghavan et al, 2007)	y[y]	n	y[z]	y	f	y	n	n	n	n	n	R	n	s	y
21	(Padala et al, 2007a)	y[v,e]	n	n	y	m	y	y	n	n	n	y	D,S	y	r	n

[a]quantity of dimensions under monitoring: f = few, nm = not many, m = many
[b]D = Difference equations, S = Statistical, Q = Queue model (predictor), N = NN, R = Routine
[c]r = real, s = simulation, c = concept only
[d]operating system resources
[e]extendend FC-EDF (Feedback Control Earliest Deadline First) scheduling algorithm
[f]service dependencies information
[g]allocationg number of machines at each application tier to achieve horizontal scaling
[h]low level application resources allocation
[i]internal monitors of CPU, I/O, bandwidth
[j]standard system identification than controller design
[k]adaptive control by regulating controller configuration and tuning
[l]server-level cache resources
[m]proxy server access control
[n]web server access admission control
[o]workloads in requests in time, latencies, machines allocated
[p]control to meet the individual response time and throughput guarantees
[q]staged event-driven architecture
[r]multiple software properties of multi-tiered eCommerce system
[s]change processor frequencies
[t]also using near future predictictor by a few discret steps
[u]automatic sizing of buffer pools in DBMS
[v]VM sharing, capacity management
[w]SLAs definitions with costs – targets are derived from penalties and rewards.
[x]adaptive control by the model update
[y]network bandwidth control with standard subgradient methods for distributed rate limiting
[z]distributed rate limiting can be seen as a distributed admission control method

Table 2.2: Comparison of existing studies in a field of service management and control published after 2008. (Note, some of the footnotes may be declared in Table 2.1)

No.	Work	Resources Provisioning	Scheduling or Micro–scheduling	Admission Control	Multiclass control	Observable areas[a]	QoS or SLA Aware	Direct overload prev., Security	Energy Aware	ASM aware	Real-time or Deadline sensitive	Control theory support	Control model[b]	Adaptive Control	Experiments[c]	Cloud related
22	(Wang and Chen, 2008)	y^{d}	n	n	n	f	n	y	y	n	n	y^{j}	D	y	r	n
23	(Kusic et al, 2009)	y^{ve}	n	n	y	nm^{o}	y	n	y	n	n	n	S	n	r	n
24	(Bodık et al, 2009)	y^{g}	n	n	n	nm‘	y^{w}	y	n	n	n	n	S^{f}	y	r	y
25	(Lim et al, 2009)	y^{g}	y	y	y	m^{k}	n	n	n	n	n	y	D	y	r	y
26	(Li et al, 2009)	y^{v,e}	n	n	n	nm	y	n	n	n	n	y	D	y	c	y
25	(Boniface et al, 2010)	y	n	n	y	f	y	n	n	y	y	n	–	n	c	y
26	(Chen et al, 2010)	y^{v,e}	n	n	y	m^{h}	y	y	y	n	y	n	Q	n	r	n
27	(Emeakaroha et al, 2010)	y^{i}	n	n	n	m	y	n	n	n	n	n	R	y	r	y
28	(Xiong et al, 2010)	y^{v,e}	n	n	y	m^{h,o}	n	n	n	n	n	n	D	n	r	y
29	(Ardagna et al, 2010)	y^{v,e}	n	y	y	m^{h}	y^{w}	n	y	n	n	n	Q	n	r	n
30	(Bertoncini et al, 2011)	y^{v,e}	y	n	y	m^{h}	y	n	y	y	y	n	S	y	c	y
31	(Mao and Humphrey, 2011)	y^{v,e,j}	y	n	y	m^{k}	n	y	n	n	y	n	Q,R	n	r	y
32	(Wu et al, 2012)	y	y	y	y	m^{k}	y	y	n	n	n	n	R	n	r	y
33	(Beloglazov and Buyya, 2012b)	y	n	n	y	m^{k}	y	y	y	n	n	n	S,R	y	s	y
34	(Lin et al, 2012)	y	n	n	n	nm	y	n	y	n	n	n	R	n	r	y
35	(Sikora and Magoulas, 2013b)	n	n	y	y	m^{k}	y	y	n	y	n	n	N	y	r	n
36	(Herbst et al, 2013)	y	n	n	y	m^{k}	y	n	y	n	n	n	S	y^{l}	r	n
37	(Xia, 2015)	y	n	n	y	m^{k}	n	n	y	n	n	y	D,S	y	c	y
38	(Beloglazov and Buyya, 2015)	y^{v,e}	n	n	y	m^{k}	n	y	y	n	y	n	R	n	r	y
39	(Serrano et al, 2016)	y^{v,e}	n	y	y	m^{k}	y	n	y	n	n	n	R	n	r	y
40	(Singh et al, 2017)	y^{v,e}	n	n	n	m^{k}	y	n	n	n	n	n	R	n	r	y
41	(Alsarhan et al, 2017)	y	n	y	y	m^{k}	y	n	n	n	n	y	RL	y	r	y
42	(Xue and Jarvis, 2018)	n	y	y	y	nm	y	y	n	n	n	n	R	n	r	y
43	(Gill and Buyya, 2018)	y	n	y		f	y	y	n	y	n	n	S^{m}	y	r	y
44	(Gill and Buyya, 2019)	y^{v,e}	n	y	y	m^{k}	y	n			y	n		y	r	y
45	(Gill et al, 2020)	y^{v,e}	n	n		m^{h}	y	y	y	y	n	n	N	n	s	y
46	(Tuli et al, 2021)	y	n	y	y		y	y	y	y	n	n	n	y		y
47	(Singh et al, 2021a)	n	y	n	n	f	y	n	n	n	n	n	–	–	r	y
48	(Sikora and Magoulas, 2021)	n	n	y	y	m^{k}	y	y	y	y	y	y	N	y	s	y
49	(Tuli et al, 2022)	y	y	y	y	m^{h}	y	y	y	y	n	n	N	n	y	y
50	(Sriraghavendra et al, 2022)	y^{n}	n	n	y	f	y	y	n	n	y	n	R	n	s	y

[a]quantity of dimensions under monitoring: f = few, nm = not many, m = many

[b]D = Difference equations, S = Statistical, Q = Queue model (predictor), N = NN, R = Routine

[c]r = real, s = simulation, c = concept only

[d]CPU throttling actuation that manages the power consumption by Dynamic Frequency Scaling

[e]CPU share

[f]statistical model extended by curve fitting using smoothing splines and LOESS

[g]an approach relying on clean decoupling of cloud– and client– side control, where application control is factored out of the cloud platform and left to the client controllers

[h]APM and thermal sensors

[i]autonomic SLA management

[j]dynamically allocating/deallocating Vms and scheduling tasks on the most cost-efficient instances

[k]APM, requests counts and latencies

[l]adaptive control by the model update

[m]SNORT intrusion detector and SVM classifier

[n]dynamically determine the service placement in the fog environment

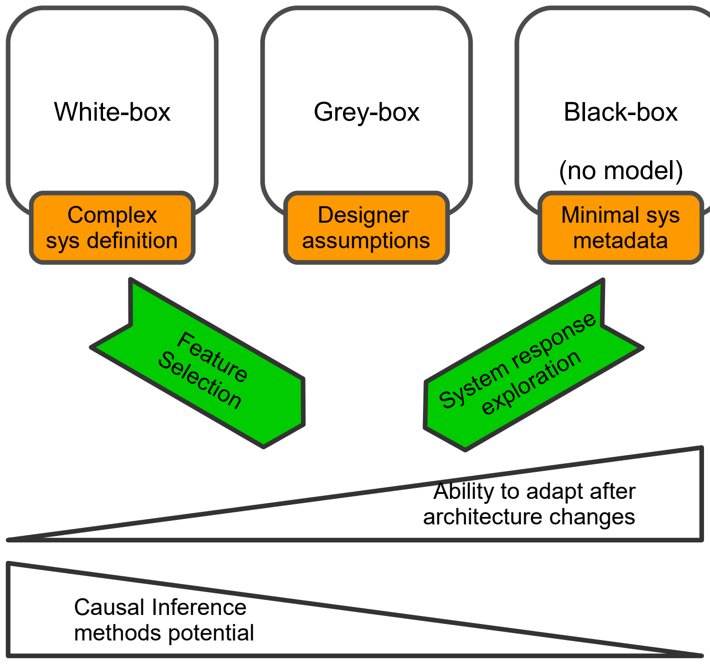


Figure 2.10: Adaptive Control, White–box, Grey–box and Black–box approaches.

Nevertheless, many characteristics may change in time due to load changes, parameters used, further development of functional modifications, or other technical interventions (Grinshpan, 2012), as described in Chapter 3. Moreover, systems code base and meta–description of system interconnections are complex and it is often too difficult to infer an analytical solution, mathematical equation or the model, of a system's performance responses. Thus, building a model of the entire system introduces plethora of practical challenges.

2.7.2 Model–free Approach

Many crucial run–time properties can be considered as unknown at the beginning of the system use, and cannot be derived based on a description of architecture and the code–base. Thus we treat those as a black–box and analyse the system with an approach where the behaviour of the stimulus/response will be accounted for, to infer the unknown box (Sjöberg et al, 1995; Juditsky et al, 1995) Building a model that would fully describe all complexities and non–linearities of the real system is very difficult. However, it is possible to observe the system and the applied control actions and infer some knowledge about its hidden characteristics.

A model free approach makes no assumption on software structure or architecture used. The system under control is treated as a back–box where only input and output variables are observed. The control mechanisms actively search the best performing control rules under given conditions (Sikora and Magoulas, 2013b), as described in Chapters 7 and 8. Although nonlinear system identification is a well developed discipline it has not found many applications in complex enterprise systems control in recent years.

2.7.3 Dimensionality Reduction

To apply efficient, constructive and more stable control, knowledge about the system run–time characteristics must be established, as each change can potentially impact other elements of the system or resource consumption, and in effect impact the actions execution time. A typical enterprise ASM control suite can consider hundreds–to–thousands of distinct actions, tens–to–hundreds of resources, and tens of SLAs used as both key performance indicators and revenue recognition. Due to non–linearities in such systems autonomous control has to be applied against a background of multidimensional space, which is difficult to control/regulate withour reducing its dimensionality (Fradkov et al, 1999; Gasparyan, 2008). The smaller the number of dimensions used, the more rigid the control can be (Pérez et al, 2004). Thus the right selection of the most relevant dimensions for the autonomous ASM control is crucial. In this vein, it is imperative to identify which elements (especially controllable actions) have the biggest impact on SLA (indirectly through resources utilisation), as discussed in Chapters 5 and 6.

One of the challenges of the thesis was to reduce the dimensionality by finding the most relevant dimensions. This task is mainly complicated by the fact that many non–linearities arise when a system is facing saturation (Hellerstein et al, 2004). Finally a new method was proposed, which is a combination of similarity search and interference check in multidimensional temporal dataset (Sikora and Magoulas, 2013a), and an extension to this research direction a further improvements to causal dependency based methods have been described (Sikora and Magoulas, 2014a).

2.7.4 Grey–box Approach to Features Selection

In order to limit the dimensionality and identify the system a model can be built using observed processes characteristics. This can be achieved by inferring dependencies and generating a structure of causality between architectural elements under monitoring and control.

To do so a controller designer needs to find a dependency measure which can be used to investigate dimensions relevance for better, more focused and stable control, where relevance criteria can consider the similarity between metrics sequences and the strength of dependence of utilisation metrics. Dimensionality reduction can be achieved by applying feature selection techniques (Guyon and Elisseeff, 2003), so only the most dependent of controllable actions and therefore relevant for the control are selected.

Although the control approach introduced in the thesis, presented in Chapter 5 and 6, does not employ an analytic model of the system, it exploits cause–and–effect dependencies between action requests, resources utilisation and SLA. Effectively the general causality direction here is quite well set (Pearl, 2000). Consequently, events or related changes in the system can be investigated through similarity of metrics changes in time. Even though

SLAs are clearly defined and directly related to parameters used, the non–linear nature of EIS may react surprisingly unstable especially under high load. Therefore, a method for finding causality strength by matching metrics variations can give system operators information to set up or activate the actuators in areas where resources and SLA have been identified as having high causality, so the control can have the strongest effect.

2.7.5 Queuing models and ASM modelling

Queuing models offer a very efficient way of aggregating structure of complex systems by statistical parameters that define context of responses on requests. Such an approach also offers the flexibility of being easily redefined based on new data coming from the system after control actions are applied.

Simplicity, elegance of the queuing models approach and very good approximation of the multi–tiered computational systems dynamics was found by many researchers in the past (Hellerstein, 2004,?; Chen et al, 2010; Ardagna et al, 2010).

Moreover, queuing models give a widely adopted framework for a complete, synthetically modelled, integration of system run-time, monitoring and control components in a single environment that supports fast, repetitive and rigorous way of running experiments. That allows full experiments to take much shorter time for a controller to establish stable control, which was the biggest impediment of earlier research reported in (Sikora and Magoulas, 2013b); this will be elaborated further in Chapter 8.

2.7.6 Predictive Model Control

Collecting more data and control characteristics allows to build models that offer predictive methods (Lu et al, 2003; Kandasamy et al, 2004; Tuli et al, 2021) and better targeted control, as will be discussed in Section 10.3.2.

Based on data collected during normal EIS work and control interventions rules–based, NNs, networks of queue models, Directed Acyclic Graphs (DAGs) of Bayesian networks or causal graphs can be utilised as a source of anticipated systems states, as reported in Chapters 7 and 8. Lastly, a discussion of non–acquisitive control methods applications in this context is included in Chapter 9.

2.8 Cloud Services Management and Function–driven Services

Currently, data centres employ much more computational power, machines and energy that is needed to provide a service so as to ensure consistent provision of the required service levels. This approach is driven by the need to hold extra servers in reserve, ready to be brought into operation in case of high demand, and it is a ramification of the effect of high switching on/off inertia of the reserve systems (Mao and Humphrey, 2012). Although this strategy appears to be inefficient, it is considered as an effective approach to secure some strongly required NFR; such as resiliency, scalability and reliability (Zhang and Lin, 2010).

The cloud computing paradigm tackles this problem with a promise to allocate computational power more efficiently by allowing many clients to reuse the same hardware available. To this end, cloud auto–scaling has been researched thoroughly, both from practical and theoretical perspectives, as a promising approach to monitor applications and make automatically adjustments based on resources availability, considerations about response times, and time and budget constraints (Mao et al, 2010; Dutreilh et al, 2010)

Figure 2.11 outlines the four main architectures, and deployment approaches used in cloud computing. Recently, the move from a monolithic application architecture to a SOA, which emphasises on the modularity of components, offered a way to keep big and complex systems maintainable (Fowler, 2018). This approach resulted in further developments in application design considering it as a collection of fine–grained modules with loosely coupled services, called microservices, where the communication protocols are lightweight (Lewis and Fowler, 2014). Such approach caused further development into serverless computing (Castro et al, 2017; Baldini et al, 2017).

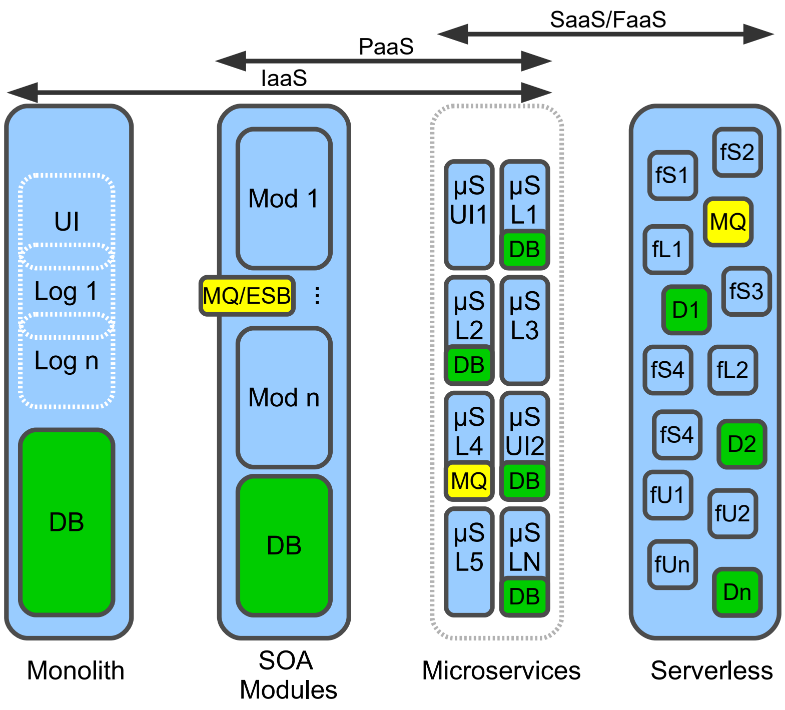


Figure 2.11: Monolithic, SOA/modules, microservice and serverless deployment.

Infrastructure–as–a–Service (IaaS) is a type of cloud computing service that offers an infrastructure covered with a visualisation layer, allowing resources to be used much more efficiently, e.g. the Amazon Web Services Elastic Computing Cloud (AWS EC2), or the Microsoft Azure. Still servers are substantially underutilised as clients are often choosing to secure more computational resources than are really necessary for the sake of simplified dynamical allocation, or for dealing with changing conditions when running their scripts. Effectively, clients decide to pay more for the resources allocated to mitigate the risk of reaching saturation and simplified deployment. Striking a balance often depends on the technical maturity of clients' systems and the requirements of the specific load profile [12].

Another promising alternative called SaaS can be more efficient, since the service provider deals with the costs of hardware (or IaaS/PaaS if applied), system performance, and the energy footprint under a given monetisation approach to produce an acceptable level of service given shorter execution times. To this end, providers offer a dedicated, but limited, feature set targeting specific problems that the clients demand for.

Although more and more flexible platforms become available, offering clients services to deploy their own routines for other customers (e.g. Salesforce Customer Relationship Management (CRM), Content as a Service, Disaster Recovery as a Service, Backup as a Service, or Blockchain as a Service) is a service–type with higher scale of customisation that goes smoothly into another modern cloud services category discussed next.

PaaS type of cloud computing relies on offering a software platform, reducing the complexity exposed to clients, such as deployment of servers, orchestration of sub–services, infrastructure auto–scaling, etc. It offers more flexibility and customisation to clients and allows running bespoke code for their customers more extensively. PaaS is a well stabilised but still growing area in cloud computing services and application platforms are expanding rapidly, e.g., Heroku, Cloud Foundry, Engine Yard, AWS Elastic Beanstalk, Amazon Elastic Container Service (Amazon ECS), Google App Engine, IBM Bluemix, Jelastic are only some examples.

Following this trend, there has been more work towards abandoning the use of servers level details, hiding servers behind layers of abstraction, and providing the platform of serverless computing, also called FaaS. A code realising functionality is deployed into a more abstract service, wrapping and not exposing the details of the servers provisioning, by applying more extensively containers and virtualisation for fast servers spin–up (Hendrickson et al, 2016). Such an approach to computational power provisioning offers a model that seems to be one of the more promising today with many vendors providing general FaaS, such as Google Cloud Functions, IBM OpenWhisk, AWS Lambda, and Microsoft Azure Functions (Kiran et al, 2015; Baldini et al, 2017). These frameworks give the required flexibility to deploy custom code, yet leaving the complexity associated with the deployments and the resources utilisation to the cloud provider. They vastly offer event–driven asynchronous calls processing. Apart of the high scalability introduced by design, the approach may bring significant costs saving as opposed to traditional on–premises or IaaS, as depicted on Figure 2.12, especially for not heavily used functionalities of entire systems or their parts: subsystems, components, services and functions.

[12]The risks of economical inefficiency increase further when the frequency of activity is high. Of course, the unpredictability of the load has a key impact on the service economics too.

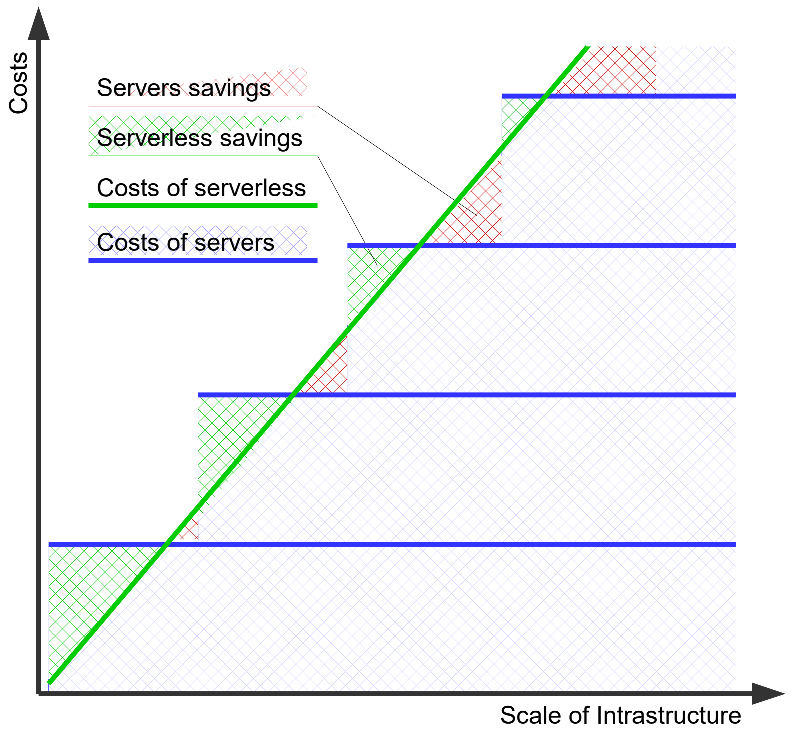


Figure 2.12: Cost and energy efficiency of traditional server–based deployments on–premises or under IaaS model vs. serverless and FaaS.

Cloud computing and private to enterprises (on–premisses) data centres are required to improve their general performance and electricity intensity, in order to minimise the staggering effect of the expected 25% annual growth rate for Global Data Centre IP Traffic. As reported by Andrae and Edler (2015), the prediction for the computing power consumption follows an exponential trend and all data centres are projected to use around 7% (even up to 13%) of global electricity in 2030 compared to 1% in 2010. Furthermore, one of the biggest reasons causing high energy consumption in cloud data centres is that resources with low utilisation still consume a sizeable amount of energy (Beloglazov, 2013). The average resource utilisation in most of the data centres is lower than 30%, and the energy consumption of idle resources is more than 70% of energy at peak–time (Barroso and Hölzle, 2007; Barroso et al, 2013).

The cloud computing developments discussed above target the efficiency of power and infrastructure use, with the ultimate aim to fully utilise the entire computational power available. However, the closer we get to that point, the higher the probability of facing risks related to fulfilled capacity and overused infrastructure. These types of scenarios generate much harder problems for scheduling, more difficult control situations, much longer processing times, and in effect can have an impact on SLA defined between service provider and clients, as described in Section 2.2. Violated SLAs may cause higher fines as contractual penalties to be paid.

ASM tackles control aspects of the operating domain trying to balance cost and SLAs. This thesis considers that these objectives can be reduced to monetisation, where all resources, usage (services calls and load profile) and SLAs can be aggregated down to cost and revenue calculations, as formulated in Chapter 3.

This also relates to advances in Power–Aware Computing, focusing to maximise the computation efficiency as a function of energy consumed (Fan et al, 2007; Kim et al, 2009; Berl et al, 2010; Kim et al, 2011) by relying on dynamic resources allocation (Beloglazov et al, 2012; Lee and Zomaya, 2012) or virtual machines placement (Beloglazov and Buyya, 2010a; Moreno et al, 2013; Duolikun et al, 2017), but may also be facilitated by application action level control (Sikora and Magoulas, 2015). Moreover, this approach can be beneficial in the context of Energy Proportional Computing, where the main issue is to deal with the high static power that is caused by the fact that a computer consumes significant energy when it is idle but also to control the power to operate under load, which is not linear (Barroso and Hölzle, 2007). Low dynamic range and poor efficiency at low to medium utilisation (low energy proportionality) is generated when there is high static power relative to the maximum loaded power (Barroso et al, 2013). Naturally, cloud providers consider upper ranges for resources utilisation in an attempt to avoid reaching saturation levels; that is an area where termination control can provide a good countermeasure, as elaborated in Chapters 7 and 8.

Other areas that can potentially benefit from this approach are new architectural paradigms, such as containerisation, microservices, and distributed computing (García-Valls et al, 2014). This new tendency adopts the idea of decomposing bigger systems into constellations of smaller, loosely coupled applications. This enables better targeting of specific needs and offers fit for purpose, yet autonomous, applications that are easier to deploy and scale up. Such an approach allows to reach operational points close to saturation but still deliver a good, healthy service without SLAs violation. This naturally offers opportunities for application level control like micro–scheduling and action–termination

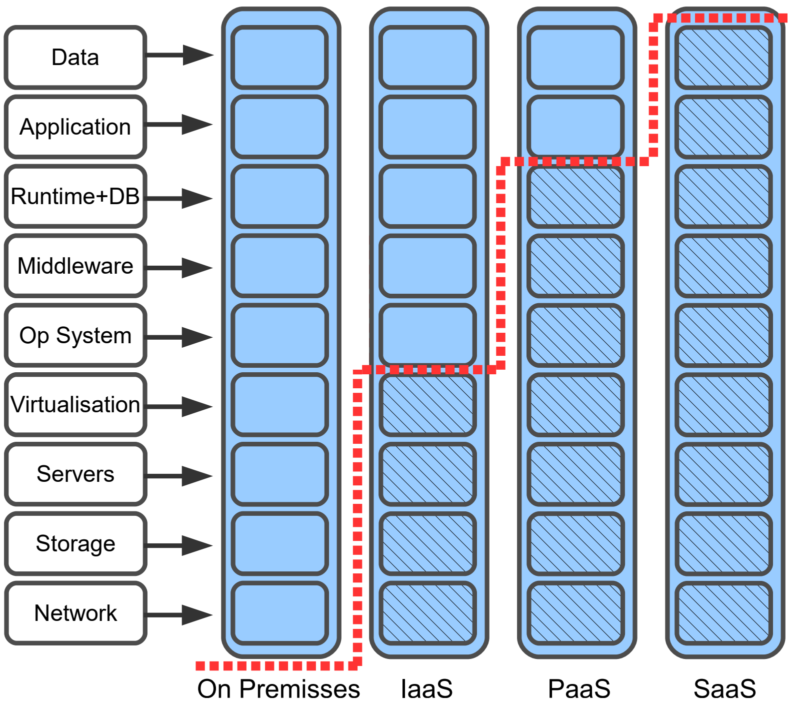


Figure 2.13: Areas of responsibilities between client and provider of service type.

aspects relevant to the proposed approach.

Scheduling is another technique worth mentioning to achieve higher performance for applications running on the cloud (Phan et al, 2011). Unfortunately, to the best of author's knowledge, most of the existing energy–aware scheduling algorithms do not sufficiently consider real–time task allocation and energy saving simultaneously on the cloud. Moreover, real–time cloud computing challenges require additional attention in that respect since tight execution–time coordination is a key aspect of SLAs (Zhu et al, 2014). Consequently, in many cases SLAs targets are achieved by allocating much more resources than needed, which is in a strong conflict to power–level economics, and this should be considered in the context of cloud resources optimisation (Bi et al, 2015).

Currently, cloud computing does not offer real–time effective promises- existing hypervisors provide no guarantee on latency, and there is lacking of service level agreements on latency applied. Still these requirements, although not in place at this stage, are going to be more and more present due to increasing demand from latency-sensitive applications, e.g., cloud gaming, communication, streaming, where simple cloud outsourced resources allocation, with calculated headroom, is not adequate (Liu et al, 2010; Duy et al, 2010; Zhu et al, 2014). More details about termination control in microservices and real–time systems and the cloud are presented in Chapters 4 and 9.

Resources, system and services activity monitoring provide an outlook of the enterprise system operation. Nevertheless, gathering measurements and making inferences from the data remain challenging in APM and ASM fields, since usage and performance metrics should be acquired from all significant elements of the enterprise systems and from all tiers of its architecture (Haines, 2006; Sydor, 2010; Grinshpan, 2012), as discussed in Sections 3.5 and 3.6.

Sikora and Magoulas (2013b) introduced a learning controller framework for adaptive control in application service management environments and explored its potential in simple scenarios. This framework is extended, it is equipped with re–training capabilities and is incorporated into a model–based test–bed, where the collection of results is more efficient allowing the execution of long test runs involving real–systems experimentation. This time the emphasis is placed on more complex, business–oriented scenarios where SLAs are defined as functions of not only call counts, as in author's previous work, but also take into account execution time, and convert directly to revenue, as per the defined monetisation model of SaaS, or PaaS systems, as detailed in Chapters 7 and 8.

2.9 Software Development Industry Practices

Of course the system must be designed and implemented accordingly to support performance level control and correctness of SLAs later in production. Currently adopted

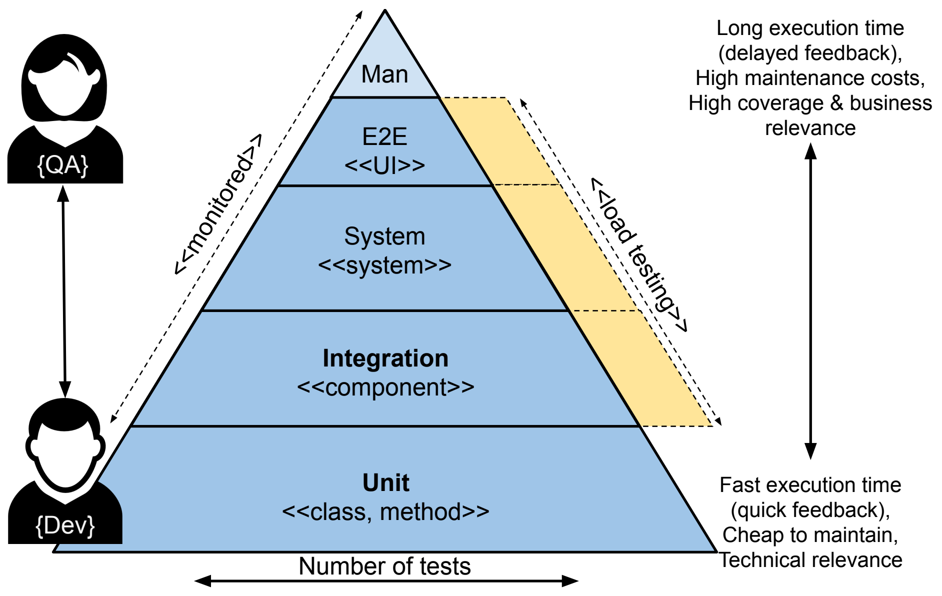

Man
E2E
<<UI>>
System
<<system>>
Integration
<<component>>
Unit
<<class, method>>
{QA}
{Dev}
<<monitored>>
<<load testing>>
Long execution time (delayed feedback), High maintenance costs, High coverage & business relevance
Fast execution time (quick feedback), Cheap to maintain, Technical relevance
Number of tests

Figure 2.14: End to end testing approach triangle. From top: full end–to–end manual user journey test, end–to–end UI driven automated testing, system tests, integration tests and unit tests.

software development methodologies and implementation techniques broadly tackle the performance engineering process.

Recent practices rely more and more on full automation trying to limit manual test plans to bare minimum. Approaches inspired by TDD and Behaviour Driven Development (BDD) stimulate designing tests automation as an internal part of SDLC. The testing triangle presented in Figure 2.14 (Crispin and Gregory, 2009) illustrates the levels of the different testing approaches that contribute to the overall testing strategy of a system. The top level gives the deepest business insights mapping full sequence of user interactions. A lower more detailed level of acceptance testing is also present, starting from the UI and the system wide test to integration, and ending on unit functionality level.

CI (Fowler and Foemmel, 2006; Duvall et al, 2007) helps to test as early as possibles, ideally after each build, whilst CD (Humble and Kim, 2018), as a wider concept, promotes fast and fully automated release process that pushes the changes as early and as often as possible, allowing to collect the feedback after each new version adoption.

With use of CI and TDD methods development teams test early and monitor everything which reduces performance blind–spots. Figure 2.15 presents the depth of performance testing approaches in the context of system layers. Top levels of automation can be a source of automated load tests tackling performance checks of all components that support functionality, like databases access, web–services, integration with other systems, and replication of end–to–end user journeys. Performance testing in a type of load, stress and soak tests, is a vital element of regression testing, executed before each release, but also verifies that the given software configuration and hardware resources provide acceptable foundations for the particular SLAs.

Rigorous functionality and performance testing prevent situations that could lead to rad-

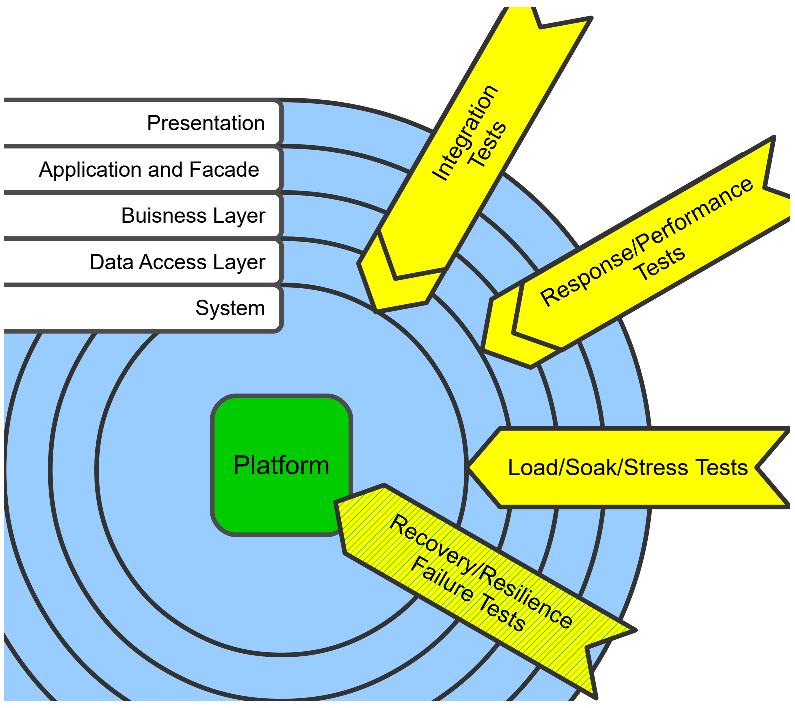


Figure 2.15: Performance testing of system layers.

ical or, most importantly from CD perspective, gradual performance degradation, risks of equipment bottlenecks, even power and energy consideration, and as a consequence create software bloat situations, larger resources use under IaaS, or high costs of service provisioning under FaaS.

Adaptive control methods are well aligned with targets of high frequency release cycles. The real–user performance monitoring, production systems logs and audit data can be great sources for new performance testing used prior to future releases.

The highlighted area has a great importance for an ASM control suite design, where operators, DevOps and development teams interact with gathered EIS data and collaborate in order to achieve higher system performance in each of the competence niches, and provide a more holistic approach to improve the overall performance of the enterprise, as discussed in Chapter 9.

2.10 Summary and Contribution of the Chapter

This chapter discussed various areas of modern software development and systems deployment, which form critical part of the background for the research rationale of application–level adaptive controllers that can be used in ASM environment. Outlining the wider picture is necessary to place the design targets of ASM control suites. In this context, along with development of modern cloud computing, virtualisation, containerisation, microservices, serverless computing advances, the adaptive control can introduce further support in the systems performance and SLAs.

The concepts, methods and the review presented in this chapter contribute to the thesis as they help positioning the work in the wider area, setting up the scene for methodological choices and design and implementation work that are described in the rest of the chapters.

Chapter 3

Problem Statement

"Nihil est in intellectu quod non prius in sensu."
"Nothing is in the intellect that was not first in the senses."
— Thomas Aquinas (1225 -- 1274)

"We are to admit no more causes of natural things than such as are both true and sufficient to explain their appearances."
"Rules of Reasoning in Philosophy"
— Sir Isaac Newton (1643 — 1727)

This chapter describes the statement of the problem tackled in this work and introduces a framework, as a conceptual model, in the field of ASM, describing it at a high–level of abstraction. Henceforth the following chapters will refer to the problem statement and discuss more practical and lower–level aspects, e.g. specifics of the monitoring and control approach and the adaptivity methods.

3.1 SLA–Driven Services Management

There is a tight relation between financial performance and the ability to provide a service under a contracted SLA function f_{SLA}, as described in Section 3.4. Financial performance P is defined as a composition of revenue R and costs C, and can be denoted as

$$P = \sum(R - C) \sim f_{SLA};\ R, C \in \mathbb{R}_+ \cup \{0\}\ . \tag{3.1}$$

The main objective of the problem being formulated is to optimise the service of the system, i.e. maximise effectiveness by reducing costs included in the SLA, in the background of load, resources usage, and performance characteristics of the service. All those aspects are time–variant, thus the adaptivity of the control process plays a key role.

An SLA, as a set of service–level objectives to be met by the parties, can be represented by a set of formal expressions that have an *if...then* structure. The antecedent, *if* part, contains conditions and the consequent, *then* part, contains actions. In the thesis, this definition considers the scope of possible actions as undertaking profit/costs related decisions, in line with what the parties agreed to do when the conditions are met. Such an approach allows to transform *if...then* structures into a function of action execution time and incorporate it into a more complex function, where costs and profits of the same argument are considered (see Equation 3.1 and, below, Equation 3.8). SLAs as functions of action execution time are normally exhibiting non–increasing monotonicity; an example is in Table 7.1 and Figure 8.4, described in later chapters, which give some instances of SLA used.

3.2 ASM Controller Work and Services Optimisation

The main challenge of the ASM controller is to adaptively react on changing system conditions in order to optimise a set of objective functions defined by SLAs. End–user behaviour, internal system actions/events and all scalar resources measures can be modelled by equations as formulated below.

An ASM control system running within an enterprise class system can be formulated in the context of dynamic systems. Given a system of functions: $\mathbf{C} : \mathbb{R}^{n+1} \to \mathbb{R}^{n+1}, n = k + m$ in time domain, where k is the number of actions and m is the number of system resources. All run-time characteristics of the system are defined by a set of resources and input/output actions - the current values define the *system state*.

Let's assume that $\mathbf{r}$ is the set of all system *resources*, which fully describe the system conditions. Main focus will be set on those resources which are measurable:

$$r = \{r_1, ..., r_i, ..., r_m\}, r \subseteq \mathbf{r}, \tag{3.2}$$

where the i–th resource utilisation is a scalar function, denoted as ρ, of other measurable resources and actions in time:

$$r_i(t) = \rho(r, a, t), r_i(t) \in [0, \infty]. \tag{3.3}$$

Only some measurable system resources can be controllable $r_c \subseteq r$. There are *synthetic resources* (abstract resources), which can be derived/calculated on the basis of other system parameters (resources and actions functions) $r_s \subseteq r; r_s \cap r_c = \emptyset : r_s(t) = \rho_s(r(t), a(t))$, therefore cannot be controlled directly, but can be used in later processing for control rules induction.

Suppose $\mathbf{a}$ is the set of all system *actions*, where we focus on only measurable actions based on the assumption that not all actions can be monitored (it should be noted that intrusiveness aspects are discussed in Chapter 4):

$$a = \{a_1, ..., a_i, ..., a_k\}, a \subseteq \mathbf{a}, \tag{3.4}$$

where the i–th action execution is a function, denoted as α, of resources utilised, by executed actions, that were triggered by i–th input event signal $\gamma_i(t)$, with use of i–th parameters vector $\mathbf{P_i}$:

$$a_i(t) = \alpha(r, a, \gamma_i(t), \mathbf{P_i}), a_i(t) \in \{0, 1\}, \tag{3.5}$$

triggered by the i–th event impulse with vector input parameters $\mathbf{P_i} = [P_1, ...P_X], \mathbf{P_i} \in \mathbb{R}^X$, which depends on available system resources, and consequently on other actions executions – using a shorter form $a_i(t) = \alpha(r, a, \mathbf{P_i}, t)$. Similarly to resources definitions, some actions are controllable $a_c \subseteq a$. The controller can read only execution time characteristics during the run–time of the actions, including termination, but any other change of the functionality is not allowed. Many action instances of the same action type can execute concurrently, the i–th action type can have many instances executing concurrently, for different values of input parameters.

The *System* model $\mathbf{C}$ is defined as a vector of functions describing the system state space in a discrete time domain $\mathbf{C} : [\mathbf{r}, \mathbf{a}]$. It can be also presented as the following system of difference equations (state space) (Eq. 3.6), containing often highly non–linear functions $r_i(t) \in r, a_i(t) \in a$.

$$\mathbf{C}(t) : \begin{cases} r_1(t) \\ \vdots \\ r_m(t) \\ a_1(t) \\ \vdots \\ a_k(t) \end{cases}, \qquad \begin{array}{l} r(t) = \rho(r(t-1), a(t-1), \mathbf{P_i}, t), \\ a(t) = \alpha(r(t-1), a(t-1), \mathbf{P_i}, t) \end{array} \tag{3.6}$$

The following block diagram (Fig. 3.1) shows dependencies in the system under ASM control between: input event signals $\gamma(t)$, output feedback signals $r(t) \in r, a(t) \in a$, and control signals $r_c(t) \in r, a_c(t) \in a$. It is vital for further descriptions in the thesis to note that building a model of the system containing explicit or aggregated values along

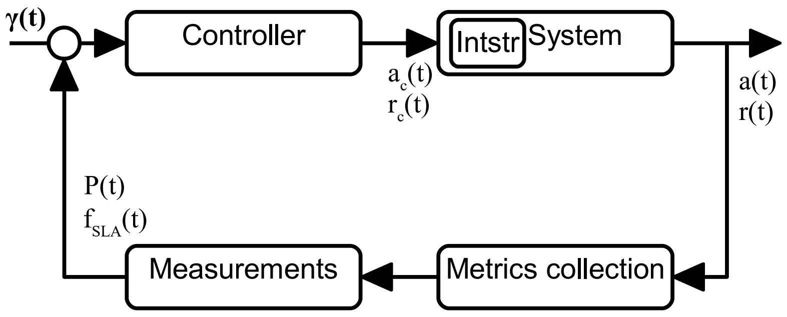


Figure 3.1: The ASM control system block diagram.

the definition of the above equation is extremely difficult. This is discussed further in Chapter 7.

Intuitively, this system of equations (3.6) defines an n–dimensional curve (system states trajectory), in a discrete time domain. The curve projection along individual dimensions can be analysed independently as describing particular elements of the system, which remain in causality relation with each others. This property is used in the similarity matching process and relevant features selection methods presented in Chapters 5 and 6.

3.3 System Performance

Measurements define an ASM control system responses and identify run–time characteristics effectively creating a model of the system performance characteristics. Considering the high–dimensionality of enterprise systems (Grinshpan, 2012) and the large size of the time–series databases, such a model is difficult to be expressed analytically. Moreover, there are many complex execution interdependencies in an enterprise system (Sydor, 2010; Sikora and Magoulas, 2013b) that are effectively recognised while running to operate the requested services.

Resources **r** utilisation is a cumulative result of running system functions, actions **a**, thus:

$$r_k(t) = \sum_i m_{ik}(r_k)a_i(t) + n_k \ , r_k \in \mathbf{r}, a_i \in \mathbf{a} \ , \tag{3.7}$$

where r_k is the k–th resource time series, a_i is the i–th action time series, m_{ik} is an unknown coefficient of a component or service, which causes changes in r_k after a_i calls, and n_k is noise that can be considered as an error. The presence of the error vector n_k can be caused by several factors. Typically it is related to a noise in the system that may be result of unwelcome and often hidden components such as software sleeps, software waits, or waits for unmeasured resources. It can be also related to other operating system or software level framework operations, or simply caused by not monitored activity that utilises system resources. We can assume that the vector n_k consists of non-negative values, as the previously mentioned situations introduce additional execution time in the measured resources utilisation; for instance when resources are used more than the monitored action times suggest.

3.4 System Under Control

Suppose a computing system receives incoming requests for an action a to be processed in order to produce a particular service result by employing resources $r(t)$ according to effectiveness given by an SLA function f_{SLA}. Hence the general operation of the computing system can be defined as a dynamic process of all activities $\mathbf{a}$ and resources $\mathbf{r}$, and is produced in line with some conditions as a result of running the system code and functionality on a given hardware/software configuration.

The controller changes the characteristics of execution time during the run–time of actions. This is realised by using termination actions, but without modifying other functionalities. Many instances of the same action type $\mathbf{a}$ can execute concurrently, i.e. the i–th action type can have many instances executing concurrently for different values of the input parameters $\mathbf{P_i}$.

The output of the system is fed back through a monitoring facility (Fig. 3.1), which employs sensor measurements, and is compared with a reference value that has been estimated from a collection of historic data (measurements/metrics). The ASM controller considers the difference between the reference point and the current output (actions execution times and resources in our case) to change the inputs to the system under control. This type of controller design follows the paradigm of the closed–loop feedback control (Hellerstein et al, 2004), and the controller is suitable for a MIMO enterprise system, defined in a high–dimensionality space of $\mathbb{R}^{n+1} \to \mathbb{R}^{n+1}, n = k + m$ in time domain, where k is the number of measurable system actions and m is the number of system resources (Sikora and Magoulas, 2013b).

The SLA function, f_{SLA}, can be considered as an abstract resource that is not associated with a particular physical component nor is it directly linked with a computational parameter. The function can be derived synthetically by an equation from other system parameters, where resources and actions are taken into account. Therefore, f_{SLA} cannot be controlled directly but plays a key role in later processing for the induction of control rules. As mentioned above, this function can be a composition of many factors, but very often in SaaS/FaaS/PaaS systems it is deducted from the execution time of actions. It is in direct relation to the service provider's productivity as financial performance $P \sim f_{SLA}$, and may be built considering the service provider's revenue but also operational and control costs. The following equation represents the relations:

$$P \sim f_{SLA_a} = f_R(a_e) - f_C(a_{et}) - f_C(a_{term}) - f_C(r) \ , \tag{3.8}$$

where $f_R(a_e)$ denotes revenue, a function of action executions that is the main factor exposed to client as price for a particular service, $f_C(a_{et})$ represents costs of SLA violations, which is a function of action's execution time, $f_C(a_{term})$ represents costs of penalties for terminated actions, and lastly, $f_C(r)$ denotes costs of resources as infrastructure provisioning.

An SLA in most cases is a function of an action execution time, but can be also built as functions of system resources:

$$f_{SLA_i} = f_{SLA_i}(a, r, t), t \in \mathbb{R}, \tag{3.9}$$

where the following condition is met $\exists t_r \in \mathbb{R} : \ f_{SLA} > 0; \exists t_p \in \mathbb{R} : \ f_{SLA} < 0;$ which is later called reward and penalty respectively.

Due to the fact that, in this context, most of the computation costs are specific for a given action, one can derive the cost of resources needed to support specific computation. Thus, $f_C(r)$ considers mainly the infrastructure required to support the service (the current industrial context in this area is examined in Section B.1.6). Effectively infrastructure costs are less dependent on fluctuations in computing, and in the control approach adopted in this work we have assumed that this function is constant. In this work the infrastructure is not subjected to control, therefore any optimisation does not consider this dimension, which can be set as a constant parameter.

It is up to the service provider and client to agree on the level of service that would be delivered per action type. For obvious reasons, the service provider does not have to expose low–level operational details to the client but, nevertheless, has to consider all these issues. Such a system can work under tight supervision, including monitoring and control, of the ASM framework. The control responds to a given run–time situation and exploiting previously–collected data, which define performance characteristics associated with input (e.g., activity frequency, load patters) and output data (e.g., resources utilisation, actions execution times, SLA values), adapts the system to challenging situations with the aim to optimise the economics of the system, i.e. minimise costs, $f_C(a_{et})$, $f_C(a_{term})$, and maximise revenue $f_R(a_e)$.

A system state S at time t is defined as a vector of collected metrics of resources utilisation, $r(t)$, but also system input load to actions $a(t)$, and outputs, such as execution times and SLA values $f_{SLA_a}(t)$. During operation, system states gathered in time form a trajectory in the state space $\mathbf{S}$, where the search for the best available solution takes place:

$$S(t_c) = [\mathbf{a}, \mathbf{r}], \ S(t_c) \in \mathbf{S} \ , \tag{3.10}$$

where $\mathbf{a}, \mathbf{r}$ are vectors of actions and resources collected:

$$\begin{aligned}\mathbf{a} &= [a_0(t_c), f_{SLA_{a0}}(t_c), ..., a_n(t_c), f_{SLA_{an}}(t_c)] \ , \\ \mathbf{r} &= [r_0(t_c), ..., r_m(t_c)]\end{aligned} \tag{3.11}$$

at a specific time instance t_c.

In order to maximise productivity of the system P, as defined in Eq. (3.8), the controller evaluates system states, S, attempting to reconcile the service workload economics defined by incoming activity, $f_R(a_e)$, and utilised resources, $f_C(r)$ on one hand, and the other limiting costs of SLA violations, $f_C(a_{et})$, and actions terminations, $f_C(a_{term})$.

Consequently, the control process aims to generate a sequence of system states that lead to maximising SLA,

$$\max f_{SLA}(t) \in \mathbf{S} \ , \tag{3.12}$$

and so the key measure for the control is the sum of SLA values of all action types, in time, as a cumulative value of service provider productivity that shape the landscape of profitability from service provisioning perspective:

$$T_{SLA}(t) = \sum_i f_{SLA_i}(t) \ , \tag{3.13}$$

where $f_{SLA_i}(t)$ is a time series of all collected values in time t, which incorporates costs and revenue for an action type SLA, f_{SLA_a}.

SLA functions that are driven by action execution time may be exposed to clients and extended by more complicated cost and revenue definitions. The usage of such SLA would create a bilateral contract and may, for instance, enforce the use of fast and stable functions to be executed on code deployed as part of the SaaS/PaaS container. Such a solution may be desirable, especially in serverless computing scenarios, so that the entire operational–service implications are more visible and both parties can benefit from that. This is another reason why it is better to focus the controller directly on maximising f_{SLA_a} as the composition function of the transformation from operational costs, as a function of resources and executing actions, to financial performance.

In certain situations, as mentioned in Section 2.1, mainly in the events of excessive utilisation of resources that are saturated, it is economically viable to terminate incoming calls so that the bottleneck resource can be released, allowing other actions to operate which will bring more profit. This is a type of autonomous control that will be investigated further in the experimental part of the study, presented in Chapters 7 and 8, where challenges relate to the quantity of action types and the different economic contracts defined in their SLAs, any unknown run–time characteristics, and the nature of interference with others actions types accessing the same shared resources.

3.5 The Computing and Business Challenges

As mentioned in previous chapters, the main trend in cloud computing, but also in on–premises corporate data–centres, requires systems to be scalable and easy to be decomposed to smaller independently deployed components following the microservices paradigm. A frequently used practice to provide stable level of service is to add more and more hardware resources to increase computational power of the hosts supporting the working solution.

Nevertheless, in terms of horizontal scaling seen from the engineering practice perspective, there are strong limitations preventing such an approach from being always effective. Not all systems can be easily decomposed nor be refactored to be scalable. Often faced situation is an architecture of the system that prevents its scaling. Legacy systems, heavy reliant on RDBMSs, middleware integration systems delivering routing or switching calls that contain non scalable singletons are just a few examples where such an approach can be challenging.

As far as the vertical scaling is concerned, there are also the obvious limitations of a single host hardware. At the moment of writing the thesis the most powerful single CPU available for general use (commodity segment) seems to be AMD Ryzen Threadripper 3990X with 64 cores (128 threads) tacked by frequency of 2.9 GHz (max. turbo clock speed 4.3 GHz), with L2 cache 32MB and L3 cache 256MB and semiconductor size of 7nm. Limits of fast access memory, but also I/O with disks access and most importantly network bandwidth limitations form far more difficult to overcome bottlenecks.

Furthermore, adding more computational power just to remove the risk of saturating one of crucial resources may not be always necessary. Thus, the research programme leading to formulation of this thesis was built on the assumption that short– or mid– term load fluctuations can be managed and mitigated much cheaper by autonomous control or human–based administration actions to increase profitability, as given in Eq. 3.12, of an enterprise for internal use or cloud computing providers for external. The profitability optimisation considers energy and hardware costs, but also impacts on QoS, as discussed in Chapters 7, 8 and 9.

3.6 The Controller Challenges

It is time to review the controller challenges. To align with the nature of this chapter the discussion below remains at a theoretical level. However in the following chapters the reader will have an opportunity to review further technical sides of the controller challenges while starting the discussion from control objectives, like for example in Section 4.6.

Since the SLA maximisation is the core control objective, as shown in Eq. (3.12), and all SLAs could be possibly wrapped to a single unit, as shown above, one can simplify the formulation of the control problem avoiding the use of multiple objectives. This formulation promotes the concept of monetisation to convert system assets and operation into economic value, taking into account relevant financial statements, and enables adopting a simple method to define explicitly the service required to fulfil a contract. Still, it is important to highlight several difficulties that impede control stability and robustness, such as:

1. Highly multidimensional space having an impact on control decisions and observability of controls actions – feature selection and dimensionality reduction targeting these challenges are discussed in Chapters 5 and 6; for example see Figure 6.4.

2. Modifications of the control targets – dynamic SLAs functions or contracts renegotiation leading to re–definitions of earlier used SLAs.

3. High system inertia – especially present in situations where the actuation is not widely applied in the system under control or the system may not react strongly enough or the responses on control actions are seen with long delay; these issues are considered in Chapters 7 and 8.

4. Poor predictability of load conditions – the load as an input to the system under control may be very chaotic, or difficult to predict, so proactive measures may be impractical. Thus there is naturally a reliance on fast feedback–loop methods, such as monitoring and observing warning thresholds for early detection of SLA violations, as discussed in Chapter 4.

5. Monitoring and control instrumentation intrusiveness – monitoring unavoidably comes with changes to the observed object, as discussed in Section 4.2.4. Instrumentation requires additional computation effort which needs to be considered carefully while switching on the control; otherwise it may lead to dangerous situations and over–control (too frequently executed autonomous control, too expensive evaluations, etc.); these issues are considered in Sections 4.2.5 and 10.3.

6. System resources facing saturation points exhibit high non–linear conditions – uncovering statistical characteristics of such non–linearities for response times is very challenging, particularly when there are sudden changes and low observability due to critically limited resources, which are required to collect probes, as discussed Chapters 5, 6, 7 and 8.

7. Actions parameters, $\mathbf{P_i}$, causing different behaviour and resources usage – impacting not only the direct control, as discussed in Chapters 7 and 8, but also the preprocessing stage where the thesis contributes the algorithms SDCIC and Application Service Management Signal Deconvolution (ASM-SD), as discussed in Chapters 5 and 6.

It is worthy to highlight that some of the challenges mentioned above are also relevant to other domains where control methods are used, such as control systems, computing and

Table 3.1: Comparison of main challenges for control and their relative impact on the difficulty of the problem solution from control system, computing systems and business perspectives.

No.	Challenge	Control system	Computing system[a]	Business
1	Highly multidimensional space	Strong	Medium	Strong
2	Modifications of the control targets	Medium	Weak	Strong
3	High system inertia	Strong	Medium	Medium
4	Poor predictability of load conditions	Medium	Medium	Strong
5	Monitoring and control instrumentation intrusiveness	Medium	Strong	Medium
6	Non–linearity caused by saturated resources	Strong	Medium	Medium
7	Actions parameters	Medium	Medium	Weak

[a]Computing system is understood as a combination of platforms, frameworks, components integration and functionality.

business environments. Table 3.1 summarises the impact of these challenges in the three domains mentioned above.

3.7 Summary and Contribution of the Chapter

This chapter proposed a mathematical formulation of the control problem and defined the relevant variables for designing the ASM controller. The rest of the thesis builds on this approach to develop an appropriate solution that addresses the challenges explained in earlier chapters.

In this context, adaptive control is based on the use of the system resources, energy, APIs or end–user performance considerations, and employs a suitable objective function based on the service contract. The SLA allows to develop flexible functions that enable a control block to adapt an enterprise system running under changing conditions to optimise the performance targets.

This is a multidimensional search space, so to simplify the control design it is practical to introduce a set of target functions that may have a single unit, including a formulation of the financial perspective.

Chapter 4

Monitoring and Control Objectives

"In the fields of observation chance favours only the prepared mind."
—Louis Pasteur Lecture (1854)

"When we had no computers, we had no programming problem either. When we had a few computers, we had a mild programming problem. Confronted with machines a million times as powerful, we are faced with a gigantic programming problem."
—Edsger W. Dijkstra (1986)

This chapter introduces a composite system of metrics gathering and collection. These metrics provide the foundations for deciding how the control should be applied and assessed in order to optimise the system operation in the direction specified by the SLA definitions. First the reader is introduced to observability and controllability properties of ASM systems. Then the chapter explains how these can be exploited in the monitoring and control instrumentation of the proposed framework.

4.1 Observable and Non–observable Dimensions

In most cases, available resources and fundamental system design features, such as scalability, maintainability, availability, predictability, high resilience, well addressed non-functional requirements, use of particular technology and code optimisation aspects dictate the system response levels. Even the best designed systems can face low service quality in situations of high load service operations, or when crucial hardware and software resources are saturated. To reiterate a challenge stated in earlier chapter, almost every parameter tuning change in a system working under conditions of limited resources may cause more harm to other areas of the system than the potential benefits for which the action was intended.

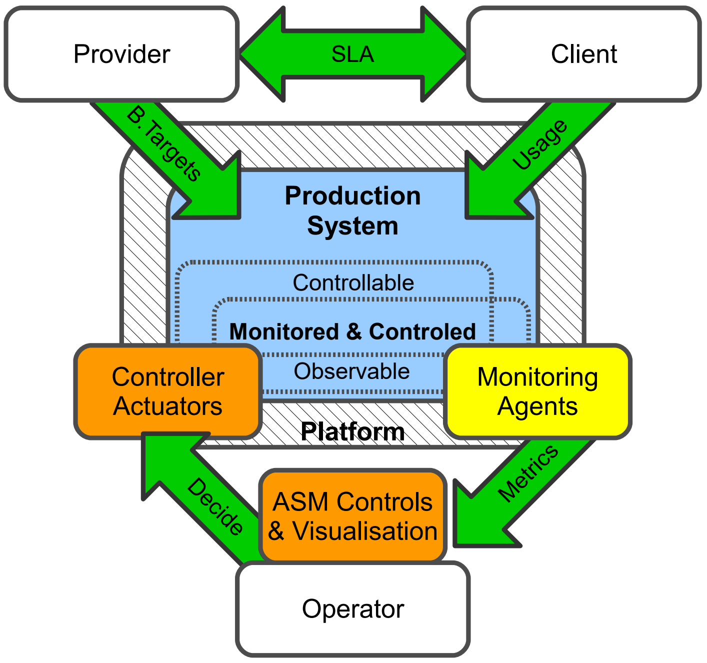


Figure 4.1: Monitoring and control considerations with respect to service provider and client agreements and targets.

An ongoing comparison of production metrics before and after running actions creates an outlook of operators' decisions and of the correctness of the control activities. Operations staff, or autonomous control agents, possessing knowledge about the system usage, can modify a variety of run–time parameters in order to adapt the system to changing situation, e.g. application specific settings, functionalities switches, cache memory thresholds or end–user functional limitations (Haines, 2006; Sydor, 2010; Grinshpan, 2012). A general ASM control cycle has been outlined in Figure 1.1. The monitoring and control considerations of service provider and client have been depicted in Figure 4.1. Service provider business–model targets and contractual SLAs with the client are strongly influencing the adaptive control policies and the operators' decisions.

It is imperative for the quality of the control to analyse the application, system and platform architecture and extract those areas that are observable to APMs, ASMs or custom integrated monitoring, as detailed in Section 4.2, and have a direct measurable effect to service quality and business model targets, as explained in Sections 4.3 and 4.4. The visualisation of the profiling and monitored data in a context of systems structure will be discussed in Section 4.5, while the modification of system elements that can be controllable, adding actuators and setting up control objectives, will be described in Sections 4.6 and 4.7.

The rest of the chapter explains the concept of monitoring in the context of specific metrics collection design as used in the experiments of the adaptive control reported in Chapters 7 and 8.

4.2 Monitoring

Any control action has to be performed on a basis of gathered data. The data can be stored in a *knowledge base*, containing pre–processed metrics describing system run–time situations. The storage mechanism is necessary to secure the datasets for later system debugging, but also to provide transparency for human operators. Persisted metrics are time–sequences of services and resources states, forming a detailed knowledge repository of the system load, requested service actions, responses in the background of the level of available resources, but also adopted control actions. The collected metrics provide a frame of reference for the operation of the system and its potential trajectories in the state space. In this context, the controller operation relates to the exploration of a multidimensional, discontinuous and multi-modal search surface. Figure 4.2 illustrates a general integration of a system with passive, active and resource monitoring.

Adaptive controllers, examined further in Chapters 7 and 8, search configurations of parameters for which the system responds in a better way according to some criteria (e.g. cheaper hosting, faster responses, stabler service, greater total revenue and SLAs footprint). However, in highly utilised environments there can be many cases where any of these criteria may be conflicting with each other. In this context, one has to find a set of criteria that are capable of creating solutions which are closer to the optimal ones considering all defined objectives.

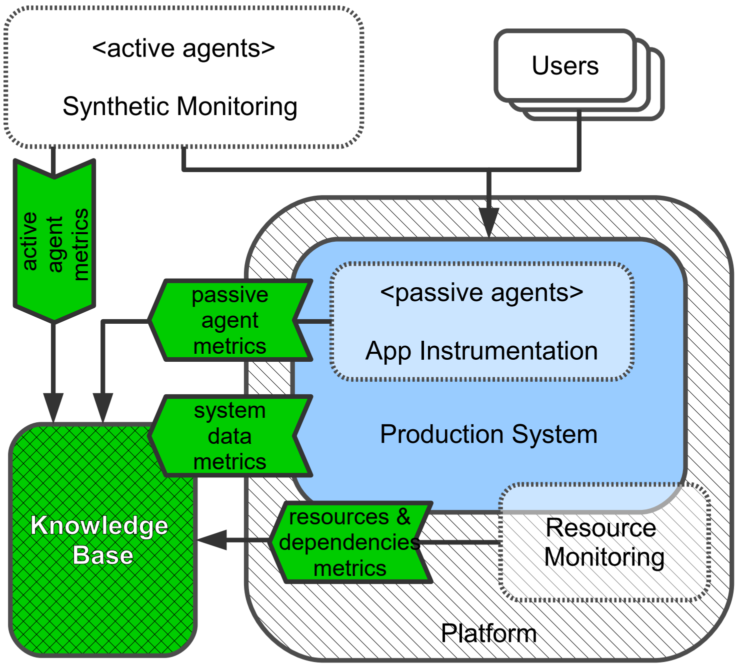


Figure 4.2: Active, Passive and Resources Monitoring.

4.2.1 Passive

Passive monitoring is applied through instrumentation and enables supervision of real–users or API clients interactions with a system. Metrics collected using this approach can be used to determine the actual service–level quality delivered to end–users and to detect errors or potential performance problems in the system.

4.2.2 Active

Active (Synthetic) monitoring actively checks the system states and collects data in scheduled mode, i.e. by scripts that often simulate an end–user activity. Those scripts continuously monitor at specified intervals, for performance and availability reasons, various system metrics. Active agents are also used to gather information about the state of resources in the system. Active monitoring is applied whenever there is a strong need to gather information about the current state of the application, and to detect easily times of lower or higher than normal activity.

In order to provide visibility of applications' health and stability under more complex sequences of actions, an active agent (scripts) can emulate users' behavioural paths or specific APIs transactions. Thus, such an approach to active monitoring is often called synthetic monitoring. It can be an additional proactive detection of system slowness or downtime in the context of the real business usage profile.

4.2.3 Resources Monitoring

Metrics explaining utilisation of resources are typically collected in an active way by agents that are deployed as close to the system as possible. Certain software resources, e.g. Message Queue (MQ) queue length, Content Management System (CMS) data entries, or ESB requests count, Database (DB) data volume size, can be also monitored by passive agents deployed into the software responsible for a service. Change to an important level may fire an event for an APM suite.

Business oriented metrics, synthetic KPIs and SLAs can be monitored in both active and passive ways.

4.2.4 Intrusiveness of the Monitoring Process

Observer effect phenomena are present in physics (e.g. probe effect, uncertainty principle), sociology (Hawthorne effect, subject– and observer– expectancy effect, confirmation bias) and also prevalent in information technology. Following this line of thought, the more an observer desires to know about the ASM environment's dynamics and more probes are passively or actively collecting metrics, the more the system is loaded by this non–functional activity and in effect the measurement uncertainty is raised.

Especially high intrusiveness may be present in (a) passive monitoring due to deep instrumentation and (b) synthetic monitoring involving sequence of interactions. Both can have significantly negative impact on the performance, therefore in addition to other factors their design has to consider working under high systems' utilisation.

The impact of the level of intrusiveness can be reduced by adaptive sampling. This technique allows raising the rate of samples (in a way that is not additionally stressing the system) when more information is needed proactively, or decreasing it when the need for information is not high or the system is reaching a difficult to maintain level of load. Another technique for lowering the intrusiveness of the monitoring is to allow instrumentation to adapt to conditions and switch off (or timely suspend) passive monitoring in areas that do not exhibit very different to the expected level of load, or when the system encounters higher load. Both techniques require a control mechanism that can consist of a simple gauge–threshold or a more complex model that will be able to classify the conditions of the monitoring, which ultimately can change the system under observation and may even introduce another set of instabilities to the main control system leading to over–, under– control or monitoring "heisenbugs".

From technical perspective, the foremost priority in the design of passive monitoring is to find a balance point between the need for collecting metrics (as this can lead to a higher memory footprint in a source system) and sending them (too frequent events can have an impact on performance, I/O and MQs if present) to an isolated from the control system persistence store; in Chapter 7 there is a longer discussion about this issue.

Another matter where extra attention is needed is the intrusiveness of the control instrumentation and control decisions processing. In order to reduce the time delay, the controller decision software need to be deployed as close to the systems as possible and the controller actuating modules shall be tightly integrated with the applications under control. Therefore, in situations where the software uses too much computation power the service may be significantly impacted by the optimisation and the control is not only ineffective but can harm system performance. Chapters 7 and 8 analyse these issues in more detail.

4.2.5 Systems Instrumentation

Instrumentation plays a key role in passive monitoring and control actuation in EIS, as discussed in Section 4.7. It is present in all layers of the system; starting from the OS level deplorable agents (for hardware, network, Berkeley Packet Filter (BPF), Gregg (2019), and OS details visibility), run–time platform hooks (Java with JMX, Python, Amazon Web Services (AWS) PaaS and FaaS CloudWatch, Azure Application Insights, etc.), application level weaved–in probes (UI front–end level instrumentation in JavaScript, web–services for UI and exposed API instrumentation), up to human perception driven feedback mechanisms.

Often enterprise scale systems require auditing features that support various functional and non–functional needs. These code regions can offer a great control value if extended with monitoring instrumentation. Similarly, integration with logging can be a very interesting point for gathering various synthetic metrics. An additional advantage of this metrics collection method is its very limited impact to the existing intrusiveness characteristics of the system.

Description of all the methods lies far outside of the main topics of the thesis, but it is instrumental to mention that both (a) the completeness of the metrics collects from all vital elements and tiers of the system and (b) the fast feedback loop for accurate control applicability are the absolute core elements of an adaptation scheme that provides quality control. More details of the EIS instrumentation for both areas of ASM monitoring and control can be found in Chapters 7, 8 and 9.

4.3 Measures of Service Quality Objectives

Service quality objectives are commonly defined as SLAs, which are contracts between a service provider and a customer (Parekh et al, 2002; Chen et al, 2010). Henceforth the SLA may be composed of one or more specific measurable characteristics of QoS that are combined to produce a target value, ideally scalar. Thus, they can be seen as KPIs of the provided service and be used in the performance service quality level reporting for the administrators and the management (Brooks and Chittenden, 2012), and, naturally, they can provide a base for autonomous control agent actions.

Various cloud computing vendors use SLA credit policies in order to secure the contract with customers. From systems monitoring performance and control perspectives there are two measurement groups that are particularly important. These are about (a) execution time and (b) availability, and are examined below.

4.3.1 Execution Time Measurements Scheme

Passive monitoring instrumentation agents allow to trace the execution times of important for the client and service provider contract software elements, e.g. UI front–end code in SaaS, WSs in PaaS, or API calls in FaaS. Typically, it is easy to persist the scalar values per individual call and apply further data aggregation policies to transform the metrics values to a granularity level that is easy to track for a given software element, an action a, or SLA contract perspective (Kyriazis, 2013; Van Der-Wees et al, 2014). Active monitoring offers the way of collecting those aggregated values.

For the time being, the support for cloud vendors of hard–real–time service delivery is still not adequate enough in neither SaaS, PaaS nor FaaS. However, it is expected that the necessity for these sort of requirements will grow gradually and SLA monitoring should be able to support that need adequately. This aspect is an important consideration for the real–time control constraints investigated in Chapter 8.

Lastly, a crucial consideration in microservices and serverless computing execution models is to tackle time observations related to the agreed maximal time to services SLAs, i.e. the full service readiness after spin–off, or readiness after a long time of inactivity.

4.3.2 Availability and Outages Measurements Scheme

There are three commonly used availability metrics groups, each introducing a slightly different calculation approach to credit an SLA: pro–rata, threshold and percentage. Pro–rata is essentially a proportional method to calculate the time of services unavailability, which is above the guaranteed SLA value, that is then credited as a discount or extra service time to the next accounting period's agreement (although cap may applies). Threshold driven SLA schemes provide a guaranteed availability, but the contract violation is conclusive only after the unavailability time exceeds a given threshold of agreed total time. In other words the service provider has an agreed amount of time to address the problem before the penalty or a service credit can be applied. In contrast, percentage oriented SLA schemes apply credit policy discounts to the next accounting term based on the outage time recorded against the agreed unavailability time as per the SLA.

An additional dimension of an SLA is the platform or provider's system error rate that can be wrapped into a scalar metric collected by an active monitor.

In a context of non–acquisitive control, as discussed in Chapters 7 and 8, negative monetised SLAs, or lower revenue values, are applied in case requests executions have been terminated by the controller.

4.4 Measures of Internal Control Targets

While SLAs are treated as targets that a service provider shall meet as per the contractual bounds with customers, there are many other internal considerations that should be taken into account to measure business–model performance and other underlying resources like energy consumption, equipment utilisation, or amortisation of the business–investment. Several of these aspects will be discussed in detail in Chapter 9.

An example of joint targets are SLAs, which are based on a superposition of execution times, and additional models of bonuses or penalties that help meeting higher client satisfaction goals. Some SLAs may be explicitly defined with extended costs, energy and equipment utilisation considerations to follow an open–computing paradigm and support open standard cloud computing platforms like OpenStack for IaaS (Jackson, 2012) or Cloud Foundry for PaaS (Sellami et al, 2013).

4.5 Profiling and Visualisation

Appropriate visualisation of systems dynamics is a key to understand the subject by operations and development teams engaged in performance analyses (c.f. with Figure 2.4). Well accessible graphical representation of the collected data removes barriers and enhances communication between the teams. As visualisation is out of the scope of the thesis, there is no point to extensively review all available visualisation approaches used in industry. Especially as this is a time of great dynamism and advancement in the ASM and ASM fields. However, it is worth to review briefly the space and declare common nomenclature for the subsequent use.

Active monitoring usually contributes to the collection of scalar values sequences thus it can be directly visualised as time–series charts or in other aggregated way when more dimensions of the data points are taken into account, e.g. heat maps, surface plots, violin plots, colony graphs, scatter–plot matrices, and finally analyses with use of On–Line Analytical Processing (OLAP) cubes and pivot tables.

There can be many rather general ways of presenting collected data, as those just mentioned above, but there are also quite specific visualisation methods in use that aggregate many data and present them together e.g. AppDynamics Application Flow Map showing details of the entire system on a dashboard, or SolarWinds Network Performance Monitor depicting the topology of the network, or Google Page Speed Insights presenting CWV collected aggregates in comparison to specific test performed in synthetic way for a request.

A valuable way of gaining insights for decision making is to analyse visual correlation between systems performance and their usage, i.e. by Google Analytics or Matomo, costs of running the hosting vs revenue for the company financial performance. In this direction, many firms invest in custom models and visualisations dashboards.

Sankey diagram can be a very effective tool illustrating the strength of calls in the services or tiers of application or even to individual classes methods called. This type of diagrams can be also used in causality measures matching between reason and effect on resources and SLAs, as discussed in Chapter 5, where simpler dependency graphs are in use. In order to track the changes in time very effective can be use of alluvial diagrams.

Passive monitoring instrumentation software introduces a great opportunity to collect execution times. The scope of the various execution times between software tiers, sub–services, front–end, back–end, web services, call stacks, or sequences of calls, produces a tremendous field for visual analytics. The most powerful techniques typically involve interactive graphical representation with use of execution traces (Trümper et al, 2013), call stack and call tree graphs, flame graphs (Gregg, 2010; Bezemer et al, 2015), waterfall diagrams (Broda et al, 2019).

4.6 Monitoring and Instrumentation for Control Objectives

This section will provide more technical background to some elements of the problem statement and control approach, outlined in Section 2.5, and to the challenges presented in Section 3.6.

In the thesis, the objectives are stated with use of SLA definitions containing all Capital Expenditure (CapEx) and Operating Expense (OpEx) aspects of running EIS. The typical use of the SLAs would be to minimise the costs of performing the service of computation, but can equally lead to optimisation of key resources footprint, energy usage, or minimising execution time of selected functionalities (actions), which directly and indirectly lead to higher revenue gained.

The control of overloaded systems is a key element from economical, revenue and energy consumption perspectives (Beloglazov and Buyya, 2010b; Beloglazov et al, 2012), when emphasis is put on limiting the costs of redundant computational power being available and provisioned. Reaching a point of resource saturation[1] is effectively leading to performance degradation. On the other hand this is actually the operational area when the energy and amortised hardware is used optimally; there are of course concerns about stability and many other key system operations that have an impact on SLAs, as well as concerns about the level of QoS and functional processing.

The controllers can modify a variety of run–time parameters in order to adapt a system to changing situation i.e. application specific settings, functionalities switches, cache memory thresholds, or end–user functional limitations (Haines, 2006; Sydor, 2010; Grinshpan, 2012). The typical EIS control actuation areas have been depicted on Figure 4.3, where the areas in orange highlight a type of actuation available to the specific tier or system element. More details about the control actuation will be presented the next section.

The resources utilisation and execution times overhead related to the instrumentation introduce yet another complexity to the non–linear control system, where stabilisation next to adaptivity play critical roles. Furthermore, almost every parameter change in a system working under conditions of limited resources, can cause more harm to other areas than potential benefits for which the action has been taken.

As far as instrumentation suitable for non–acquisitive control methods is concerned, there

[1]The point of saturation is when the whole scope of the resource is consumed and the resource begins acting as a bottleneck for the part impacting SLA in question or the system as a whole.

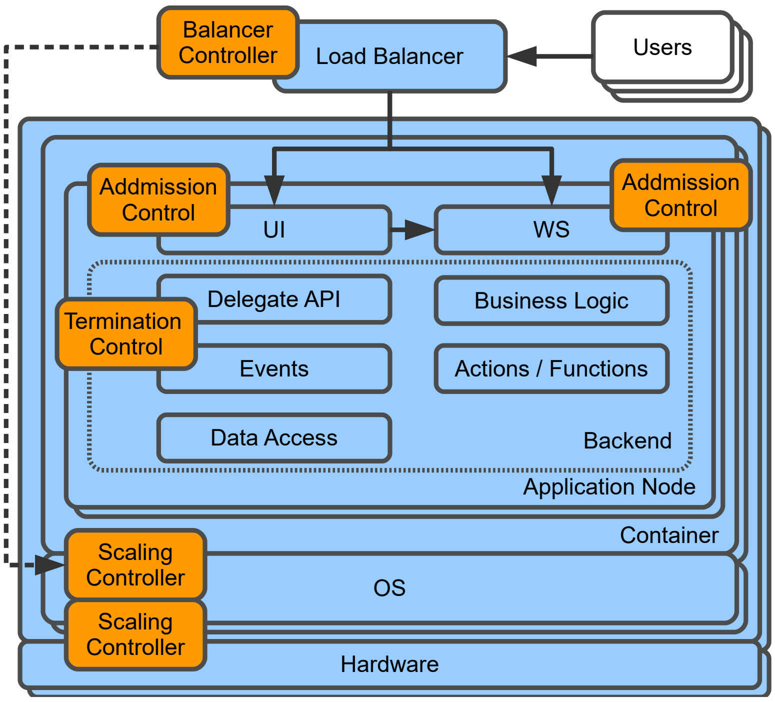


Figure 4.3: Areas of control instrumentation.

are many areas where this type actuators can be applied. A general review of control instrumentation deployable on platform or application level, with use of sequential or event-driven programming models, applicable on–premises of any of the main cloud computing models, such as PaaS, FaaS, and more generally to serverless computing is presented in Section 9.5.5, in the context of potential applications of the approach. The specific control instrumentation approaches applied to the experiments verifying the thesis will be explained in detail in Chapters 7 and 8.

4.7 Control Actuators

In order to enforce control actions the system must be equipped with specific for the type of control applied actuators. Different areas of actuating agents for specific type of control were outlined in Figure 4.3.

It is vital for the controllers decision modules performance to decide on further adaptation actions based on tracking the actuators' work. To this end, each control decision and action ought to be recorded. Furthermore actuators implementation offers a suitable mechanism for passive monitors collecting execution time, call parameters, and deep application metrics that explain details of the run–time situation. Typically, both can be monitored passively by specific controller APIs or instrumentation deployed to decision blocks or to the system under control accordingly.

Further details about the specific implementations of essential for the thesis batch job control and termination actuators can be found in Section 9.5, in the context of potential applications of the approach, after a review of tested non–acquisitive control methods presented in Chapters 7 and 8.

4.7.1 Auto–scaling Actuators

As discussed in Section 2.5.1 horizontal and vertical scalability is a commonly used technique to control the amount of servers, containers or nodes available in the system technical stacks. It is a technique that is present on premises and all cloud computing models. Auto–scaling is an approach allowing to control systems by adaptively allocating and deallocating instances (vertical) or changing their spec (like memory or CPU allotted – horizontal) reflecting the change of the demand on the system resources (see Figure 2.7).

Virtualisation and containerisation massively helped auto–scaling to become more effective and as a result are widely used in industry. Due to far easier creation of identical replicas or virtual machines, containers or nodes, in general auto–scaling in a vertical approach is wider adopted than a horizontal approach.

Lots of providers and technologies have adopted the control approach delineated. There is a plethora of different implementations. Most notable are: open–source platforms supporting containerisation like Kubernetes using orchestration of pods or Docker–swarm, type–1 hypervisor virtual machines VMware ESXi[2], Microsoft Hyper–V[3] or Xen[4], proprietary service–centre platforms like AWS CloudFormation[5] or Azure Resource Manager[6], multi–cloud orchestrators Terraform[7] or Cloudify (TOSCA)[8], both are open-source Infrastructure as Code (IaC) software, and computer server management like Apache Mesos[9] or Red Hat cluster suite, to name a few of the most commonly used ones. From control theory perspective, most of them are simple regulators taking into account defined thresholds of specified dimensions like CPU utilisation, load, or memory allocation.

Although this approach is very effective and is commonly used as a core of service providers business operation it will not be explored further in the thesis due to large adoption and wide literature in the area.

4.7.2 Batch Frameworks and Job Scheduling Instrumentation

The idea of job scheduling and micro–scheduling control has been introduced in Section 2.5.2. Batch jobs actuators are usually built–in the scheduling agents as part of

[2]https://www.vmware.com/pl/products/esxi-and-esx.html
[3]https://docs.microsoft.com/virtualization
[4]https://xenproject.org
[5]https://aws.amazon.com/cloudformation
[6]https://docs.microsoft.com/azure/azure-resource-manager
[7]https://www.terraform.io
[8]https://cloudify.co/tosca
[9]https://mesos.apache.org

the libraries. Most implementations in enterprise systems provide specific functionality and UI to setup, manoeuvre time placement of the jobs, and control call parameters, including job termination prior to execution while some may also allow termination during execution.

If a particular batch framework does not expose appropriate APIs, the batch logic code can be always intersected with the controller framework or perhaps easier weaved–in with the actuators using Aspect–Oriented Programming (AOP).

Properly instrumented code allows ASM suites to be aware of longer execution times from collected metrics and SLAs. Thus an operator can find by doing visual correlation when the usage of resources is typically high and determine a better time for the placement of the job, as depicted on the right part of Figure 2.8. An adaptive controller can use those ASM functionalities to assign an appropriate time for processing a job autonomously, if it is equipped with an algorithm that enables finding relations between cause and effects and searches to find the best matching time and optimise the problem space. This is explained in more details in Chapter 6 that presents an approach to tackle the problem of unmixing activity as input to the system based on observed resources utilisation.

Batch job runs have often desired times of completion that are dictated by business processes. Timely delivery of finalised computation can be defined in SLA together with penalties for any delay. On the contrary, completing an execution too early may just not be right for the business needs. Moreover, jobs can be dependent on each other and scheduled in dependency sequences. A critical path sequence is an additional element that needs to be considered in the search problem as well as derived jobs metadata from the batch framework and the control instrumentation.

4.7.3 Micro–scheduling Instrumentation

The micro–scheduling type of control is concentrating on shifting the core execution forward, as demonstrated on the left part of Figure 2.8. That requires an actuator to be instrumented into the request processing of actions, or deeper, into more core functionality code. Due to the nature of the control there are two groups of actuators (a) in–thread waiting a predefined period of time or waiting for a resume signal, and (b) event–based often implemented with use of MQ also with priorities.

Therefore batch jobs can also be instrumented with additional micro–scheduling actuators allowing fine–tuned execution adaptation, and in conjunction with adaptive job scheduling that primarily focuses on coarse–grained time placement. Still the main use of micro–scheduling is targeted for web requests where even sub second shifts, when critical functionalities or such with hard–real–time SLAs are severely impacted, can cumulatively radically relieve load in line with SLAs for cases that a user expects smooth UX.

4.7.4 Request Termination Instrumentation

In cases when micro–scheduling is not able to offer much reduction in execution load, termination control can still be a valid solution (see Figure 2.9), allowing preventive congestion control to ensure that there are enough resources for authorised, i.e. although leading to higher SLA, penalty calls. Requests termination and admission control have been introduced to the reader in Section 2.5.3. Both can use similar actuation mechanisms weaved–in into the system runtime with the use of AOP or can be integrated using the controller framework APIs. Consequently, this type of control can be used equally in on–premises setups of custom deployed applications, in more complex distributed systems, or in all cloud computing models.

Controller framework APIs can use Object–Oriented Programming (OOP) techniques allowing instrumentation, like the well known from the Gang of Four (GOF) patterns (Gamma, 1995) proxy or decorator pattern. Call termination typically happens prior to a business–logic service, but the framework can be extended allowing termination during the execution. Additional coding (or configuration) is required then to allow the termination to be effective in the depth of the call stack processing.

Software frameworks for various platforms and programming languages, e.g. Java, .Net, Python, Java Script contain direct language support, meta code, or libraries allowing cross–cutting concerns, like the security access control or admission control instrumentation. Widely adopted frameworks such as Spring (cross platform approach), Aspect–J or JBoss–AOP to introduce AOP as a horizontal programming paradigm, where some type of behaviour is applied to several classes that don't share the same vertical, object–oriented inheritance (Kiczales et al, 1997). In Java and C♯, the use of annotations offers a very convenient way for introducing this kind of controllers in on–premises but can also align with typical for APIs, in PaaS, FaaS or even SaaS, standards.

Application servers and web containers control and extend the system security model. Tomcat, Web Logic, Node.js or nginx configuration allows access control with integration of more sophisticated security frameworks. These software areas can be also used as a configurable point–of–control for the instrumentation. As a matter of fact, the same APIs can hold the responsibilities of being delegates to control actions as in the AOP use–cases.

Admission control frameworks share a concept rationale similar to the Call Admission Control (CAC) in the networking domain (Perros and Elsayed, 1996; Yavatkar et al, 2000; Breslau et al, 2000), where a longer connection is checked before is established (e.g. for audio or video flow) to ensure there are enough resources to secure the quality of the communication – preventing congestion issues and stream over–subscription. Container cluster technologies like Kubernetes or Docker–swarm can work with an admission controller that validates and rejects public load balancer services of the cluster. When a deploy, update or other change of the state of the cluster is requested, the modification needs to be validated by the control plane. For example, by default Kubernetes has a number of built–in admission controllers that validate and enforce resource quotas, if necessary, service account automation, and other cluster–critical tasks. The cluster also supports dynamic admission controllers which can be custom implementations of the approach available, as discussed later in Section 9.5.5.

Termination instrumentation has been an important part of the main actuation technique of the ASM control presented in the thesis, as described in Chapters 7 and 8.

The issue of PaaS, FaaS and SaaS implementations on many different tiers of software starting from the DBs and going up to APIs or whole frameworks is further discussed in the context of applications in Chapter 9.

4.8 Summary and Contribution of the Chapter

This chapter explored the areas of monitoring and control of complex applications and systems from a more pragmatic perspective, considering both technical and financial details to inform the work that will be presented in the following chapters. Tackling observability and controllability aspects of ASM systems is of paramount importance, as it has an impact on the effectiveness of the control mechanism in a context of high–dimensionality as represented by actions, resources and architectural aspects of enterprise systems.

Service providers, clients and system operators require tools to observe execution dynamics and steer the system to achieve a stable business model laid on financial and technical foundations. Due to the complexity of the control problem in the ASM field, what should be addressed first in order to optimise the system is to establish the relevance of observed and controlled dimensions. Such reduced dimensionality would allow to concentrate on the most important range of actions. This issue will be explored further in the next two chapters, 5 and 6, describing the relevant methods and the application of dimensions relevance and signals decomposition.

Chapter 5

Dimensions Relevance Selection

"Simplicity is the ultimate sophistication."
— Leonardo Da Vinci (1452 — 1519)

"Felix, qui potuit rerum cognoscere causas"
"Fortunate who was able to know the causes of things"
"Georgics" (29 BC)
— Publius Vergilius Maro, Virgil (70 BC — 19 BC)

This chapter discusses feature selection methods for EISs approaching the problem as metrics time–series analysis. The proposed approach allows searching for similarities in the signals, selecting the most relevant dimensions, and then applying the control in the reduced dimensionality space, which would ultimately scale down the maintenance effort. This approach is further used to assess causal models of the controlled system, significantly simplifying evaluation of defined elements of utilisation dependencies.

5.1 Rationale of Dimensions Relevance Selection

Enterprise systems, employed by large organisations to offer high quantity of computing services, are typically expected to be capable of handling large number of calls and volumes of data supporting the organisation business processes. More and more enterprises deploy EISs with the use of cloud computing with FaaS and SaaS models where the systems functionalities are exposed to external or internal clients, deployed on in–house infrastructure or with use of private, public or hybrid cloud based environments. As discussed in the previous chapter, for better visibility of processes and targeted control, ideally, all metrics of such systems are acquired from all significant elements of existing distributed systems infrastructure and from all tiers of their architectures. Starting from the physical tier, through the virtualisation tier, operating system metrics, services tier (database, application servers, web containers, messaging systems, etc.) and finishing on the application tier metrics (i.e. active users count, front–end execution count/time, application external/internal APIs and web/facade interfaces execution count/time) (Haines, 2006; Sydor, 2010; Grinshpan, 2012).

Diagnostics, monitoring, and auditing components of enterprise class systems may produce thousands of metrics dimensions forming a knowledge base, which is being constantly evaluated in order to specify the control rules. In order to apply efficient, constructive and stable control there must be established knowledge about the system run-time characteristics, as each change can impact other elements. A typical enterprise ASM control system can define hundreds to thousands of distinct actions, tens to hundreds of resources, and tens to hundreds of SLAs used as both key performance indicators and for revenue recognition. Due to many non–linearities autonomous control in such systems has to be applied in a background of well recognised reduced space, as a multidimensional space is difficult to control/regulate. The smaller the number of dimensions, the more rigid the control applied Pérez et al (2004). Thus the right selection of the most relevant dimensions for the autonomous ASM control is crucial. It is imperative to identify which elements (especially controllable actions) have the biggest impact on SLA (indirectly through resources utilisation).

To allow an autonomous controller to operate in a complex EIS system governed by an ASM suite, as it will be explained in detail in Chapters 7 and 8, one has to find which observable and controllable dimensions of the system (see Figure 4.1) give the biggest benefits from run–time optimisation perspective. Since a neural–control system learns by observation of actions, resources utilisation and reactions of the applied control, it would be desirable to perform preliminary evaluation of the whole scope of actions and resources before the controller is enabled. On that basis further control rules are generated and sent to actuators installed in the application runtime environment. Multidimensional similarity matching of metrics signals deserves special attention because it gives an opportunity to view a sequence of changes which were associated with a specific event or group of actions which occurred in the system, so only most relevant dimensions can be considered.

This chapter explains a novel feature selection approach based on metrics time series analysis. The method performs multivariate evaluation calculating the strength of dependency between set dimensions investigating metrics sequences for introduced measures: Similarity, Consequence, Interference, and Clarity (SCIC). The method can tackle time series analysis for dimension reduction as a preliminary and unattended process for the benefit of the autonomous control phase in ASM (Sikora and Magoulas, 2014a).

Section 5.2 describes fundamentals of the problem area that are relevant to the development of the proposed method. Section 5.3 explains the causal elements of the observable system dimensions and places earlier works in the context of the thesis research, linking feature selection and its foundations with previous work on causality (Pearl, 2000) and causal inference (Pearl et al, 2009). Section 5.4 reviews previous work in the related areas. Section 5.5 defines the method proposed, whilst Section 5.6 illustrates aspects of the performance of SCIC approach with real and synthetic data. Section 5.7 highlights some open areas and further considerations. In particular, Section 5.7.2 introduces some of these considerations, focusing on vital elements of causal inference such as intervention in the system and impacts on observable dimensions. Section 5.8 ends the chapter by presenting a summary and contribution.

5.2 Dimensions Relevance: Problem Definition

There are many complex interdependencies in an enterprise system. In order to illustrate the problem we take, as an example, a simple model of operation where (a) incoming requests on actions cause higher resources utilisation, (b) higher utilisation over certain limit causes longer execution times in the system, and effectively higher SLA penalties. Of course following (b) there is an impact on actions execution and therefore on requests count (this has usability implications, i.e. user clicks but the system does not respond in timely fashion, so users tend to click again). Also, actions can be substantially driven by input parameters values depending on functional specifics but we consider these aspects beyond the scope of the thesis.

The above assumption formulates a causal model chain (Pearl, 2000). Thus the typical causal model of enterprise class system can be defined by the following equation:

$$a \rightarrow r \rightarrow f_{SLA} \tag{5.1}$$

Set of actions $a = a(t)$ cause effects on resources $r = r(a, r)$. Actions and resources utilisation cause changes in SLA functions values $f_{SLA} = f_{SLA}(a, r, t)$. Some actions are controllable $a_c \in a$.

The main purpose of this work is to find a dependency measure which can be used to investigate dimensions relevance for better (more focused) control, where relevance criteria are similarity between metrics sequences and strength of utilisation dependence. Dimensionality reduction is typically applied to the field through feature selection (Guyon and Elisseeff, 2003), so only the most depending on controllable actions and therefore relevant for control are selected.

Although the control approach researched does not contain an analytic model of the system, cause–and–effect dependencies between action requests, resources utilisation and SLA are inherent in an EIS. The assumption is that the general causality direction is here quite well set, although it may not be obvious to the relevant actors. Consequently events or related changes in the system can be investigated using similarity of metrics changes in time. Even though SLA definitions are clearly defined and directly related to parameters used, a non–linear system especially under high load may exhibit surprisingly unstable

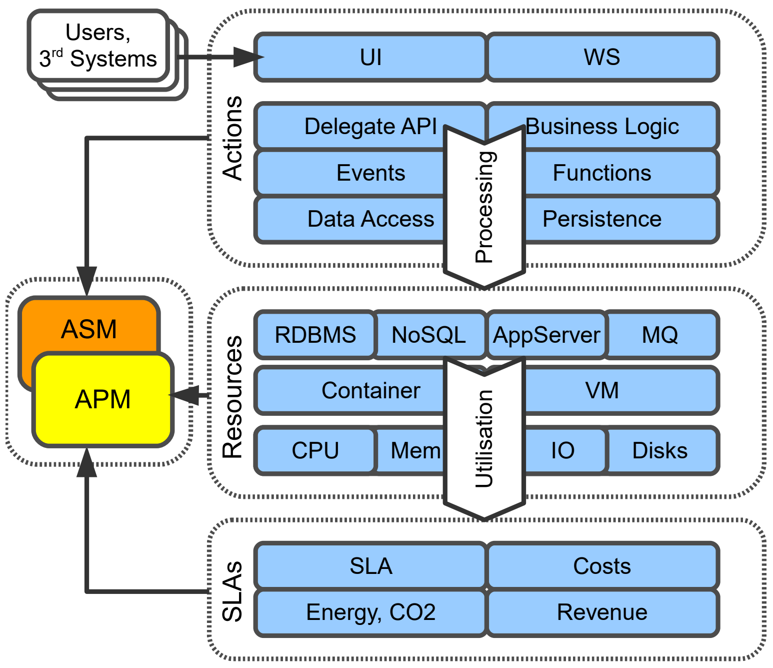


Figure 5.1: Actions, Resources, SLAs.

behaviour. Therefore a method for finding causality strength by matching metrics variations (selecting events times), could help producing more effective control actions; e.g. by setting actuators in EIS's areas where resources and SLA have been found to have the highest causality.

Every event detection approach needs specific treatment, as events' nature differs amongst problems (Ding et al, 2008). Thus it is important to note how we define an event. There are two significant factors in ASM area which we assume in this work: (a) similar signals slopes and shapes (this can be tested additionally with first/second derivative, described in Section 5.5.1), and (b) first appearance of an effect is immediate (no delays).

Four frequently occurring effects/situations which outline time series structure in the ASM field are: (a) Full Match: no delays, this is normal situation when all necessary resources are available, which can be rare in highly utilised environments, (b) Shift: delays occur due to some internal system characteristics (i.e. actions execution time → disk queue), (c) Extension: mainly when system is highly utilised, more and more actions wait for execution (i.e. actions execution time → CPU consumption), (d) Saturation: problem often occurring together with point (c), when a bottleneck is encountered.

It is important to highlight that pure sequence similarity or correlation is not sufficient to measure dimensions dependency. This is an issue that is investigated later in Section 5.5.1. Correlation on the whole dataset does not focus on the sequences of signal changes, so there is a need to consider the time dimension and the interdependencies between dimensions (other events impacts). It is necessary to investigate the changes of signals in time, as explained in Section 5.5.1 and Figure 5.2.

Even though process monitoring, and non–model–based control has been researched thoroughly (Box et al, 2011b), application of dimensionality reduction preceding control in the ASM area have not been widely studied yet.

5.3 Other Considerations

The concept of causality is old, Hume (1758) in his work titled "An Enquiry Concerning Human Understanding" covered the problem with these words: *"...we may define a cause to be an object, followed by another, and where all objects, similar to the first, are followed by objects similar to the second. Or in other words, where, if the first object had not been, the second never had existed..."*. Even though, Pearl and Mackenzie (2018) provide far deeper historical views of the causality and concluding on three different levels in the so called "ladder of causation", the words of Hume cited above are the first historical reference found by the author that are building direct context to a time sequence and the observational events so common to the control problems encountered in the ASM field.

Causal inference is difficult to perform, the computational causality methods are still in their infancy, and there is still a debate amongst scientists about the proper way to determine causality and learn causal structures from data, as existing approaches are rather limited to selected situations (Peters et al, 2017). In order to infer causes and effects Pearl (2000) argues that intervention is a necessary prerequisite for a strong causality confirmation, with use of domain knowledge first.

While this part of the thesis is built on the graphical approach to causality as represented by the work of Pearl (2000), Pearl et al (2009) and Spirtes et al (1999), there is a wide range of methods in the literature that one could choose from relying on signals similarity and ASM domain characteristics (pre–processing of overloaded resources metrics series), as described in the following sections. Some recent application examples of causal inference have been in cloud computing debugging (Geiger et al, 2016) and microservice performance diagnosis (Wu et al, 2021), while an example of the approach used in the thesis can be found in Section 5.7.2.

5.4 Relevant Background

Focusing on ASM dimensionality reduction analysis this work exploits ideas from causal modelling, event detection, time series processing and establishing similarities (by correlation and distances measures), therefore it is necessary to review briefly most notable works from the relevant areas.

Cause–and–effect relationship were researched widely in the past with statistical approaches applied to several areas like econometrics, finance, control theory and many more (Box et al, 2011a). The core finding is that causality is not guaranteed by correlation. In ASM field though we know the causal chain a priori, so investigating changes in time domain can infer events (Holland, 1986).

On the basis of causality models it was proposed a Path Analysis, which helped establishing path coefficients. This was studied by Wright (1934, 1921) who introduced a simple set of path tracing rules, for calculating the correlation between two variables. Pearl (2000) has proposed a theory of causal inference, formulating causal modelling and cause–effect relations, based on directed graph definitions. Granger (1969) proposed a pragmatic test for mainly economics use as a new model of causation applying cross-spectral methods. Extended Granger causality was researched by Chen in the domain of multiple nonlinear time series analysis (Chen et al, 2004). More recently, Eichler (2007) worked on Granger Causality and path diagrams for multivariate time series analysis. Guyon et al (2007) examined areas in which the knowledge of causal relationships benefits feature selection, by explaining relevance in terms of causal mechanisms, distinguishing between features, predicting the consequences of actions performed by external agents.

Event detection and distance measures in time series is the core area of this work. Amongst a variety of works in this field, an interesting distribution based outliers filtering method was studied by Ihler et al (2006) for adaptive event detection. Knowledge Based Event detection in complex time series was researched by Hunter and McIntosh (1999). Correlation ranking for feature reduction were investigated by Geng et al (2007), and Wei and Billings (2007). Multivariate time series in process controllability was researched by Seppala et al (2002) who investigated time series methods for dynamic analysis of multiple controlled variables to enhance feedback controllers efficiency.

For the problem expressed above it is important to review distance measures used in time series processing. Simple general form metric distances, like Minkowski, L1 norm (Manhattan Distance), L2 norm (Euclidean Distance), L∞ norm (Supremum Distance) are natural, simple to use, producing distance which is calculated without a shift (delay). Dynamic Time Warping (DTW), which provides distance between two time series which is less sensitive to lags (effect delays), was researched by Berndt and Clifford (1994) and Keogh (2002). Piecewise Linear Approximation (PLA) for segmentation of plane curves studied by Pavlidis and Horowitz (1974), a modification of DTW as Piecewise Aggregate Approximation (PAA) by Keogh and Pazzani (2000), Adaptive Piecewise Constant Approximation (APCA) introduced by Keogh et al (2001b), Elastic Partial Matching (EPM) (Latecki et al, 2005), are a few examples of widely used similarity measures methods in the time series field. More details and references of comparison of representations and distance measures a reader can find in the work by Ding et al (2008).

Other methods include correlation–based feature selection for machine learning studied by Hall (1999) and dimensionality reduction for fast similarity search in large time series databases researched by Keogh et al (2001a). A review of correlation as distance metric in non-linear time series analysis can be found in works of Kantz and Schreiber (2003).

Events correlation in ASM was researched by Gruschke et al (1998). He described an event management framework added to a system where a component correlating events based on a dependency graph is the core element. The paper does not mention times series analysis nor reduction of the space being controlled. Similarly Keller and Kar (2001) worked on a dependency query facility providing query mechanisms for dependency models definitions as consolidated dependency graph, which can be used as input for event correlators and service management frameworks. Agarwal et al (2006) researched a method using co-occurrence and relevance scores to learn and match fault patterns for automatically generating "change points" defining problem signatures using administrator feedback, but the research did not focus on providing autonomous control. Jiang et al (2011) proposed a rank building mechanism for fault management considering threshold based rules, so the importance of alerts can be better assessed. Kıcıman (2005) in his PhD thesis used extensively correlation techniques in times series analysis, in order to detect usage anomalies, and goodness-of-fit test to confirm deviation between showing component interactions, for more focused application monitoring.

5.5 ASM Control Approach

Time series analysis for dimension reduction as a preliminary processing for the benefit of autonomous control phase in ASM field has been researched earlier by Sikora and Magoulas (2014a), where a novel method was introduced explained below to address the high dimensionality problem.

Before the ASM control system can be started, the metrics time series repository must contain data collected during normal operating system run. In order to secure the knowledge about boundary criteria, it is strongly advised to run as many actions in isolation to each other. This information is most valuable since it describes non-interfered impacts of actions on the system state. So the evaluator is able to detect which elements of the system use which resources, and effectively establish causal dependencies in the system. The steps of the procedure are listed below.

1. Isolated run – in this step the metrics evaluation process collects data about the impact of various actions running in isolation; as discussed further in Section 5.5.1. Planned development team activities along with CD live–cycle, such as CI deploy, User Acceptance Testing (UAT), performance, load and stress testing on the final infrastructure are recommended in this phase.

2. Normal work – the control system collects data about normal operating conditions, so it builds knowledge about casual work signals distributions and interferences between actions.

3. Dimensions selection – this step focuses on statistical evaluation and dimension

selection procedures which are needed to reduce the dimensionality of the control space.

4. Controlled run – the controller adapts the application to the changing conditions; the adaptation is set up to consider only metrics gathered in the last time window of activity, which is reset after changed functionalities during next releases.

Note, this chapter from the following Section onward is starting discussing a method tackling the dimensions selection step (3), whilst the controlled run step (4) was addressed by Sikora and Magoulas (2013b) and further described in Chapters 7 and 8. All the steps (1, 2, 3, 4), have been elaborated further on a macro–level in Chapter 9 explaining applications, use–cases and wider picture of the sequence of joined human and autonomous control in a context of development activities (see Figure 9.2).

5.5.1 Method Description

The proposed approach is based on calculating similarities between measured dimensions and ascertaining strengths of causal relations. To this end, the Clarity measure is introduced. This depends not only on how source and output signals dimensions match each other, but also on how consistent is the similarity relation in time and if the similarity is not interfered by other substantially similar signals in a given time window. This approach allows to establish purity of impact dependencies, focusing on the isolation of patterns of interest (high similarity) from the background of other signals.

Hence, the proposed algorithm establishes the dependency of dimensions based on metrics times series – source s and output o signals are input parameters. Signal pairs are measured using four different factors: Similarity, Consequence, Interference and the final score is Clarity. Thus the method was named SCIC. Computed values describe edges in the assumed causal model graph. When Clarity values are computed, a ranking is built, which directly defines the relevance of dimensions between the tiers of the examined system.

Although the similarity match is not strictly a statistical approach, we can uncover some structure of causality in the signals changes against time (Pearl, 2000). To infer causality let's assume that signals representing activity, resources consumption or services utilisation contain similarly monotonically behaving components.

To improve the efficiency of the main algorithm collected data are pre–processed in two steps. Firstly, data are linearly scaled (normalised) so all amplitudes are standardised in the range $[0, 1]$. This step is needed to ensure that all dimensions are being equally evaluated (curves matching) regardless of the actual values ranges, which can vary widely. Secondly, the data are denoised, assuming that if any of the metrics values in the time series were not collected (null value), i.e. it is unknown or missing, they were replaced by a previous non-null value. During preliminary investigations it was found that fair results are obtained by using the Simple Moving Average (SMA) technique (Pyle, 1999). Similarity check of the denoised signals gives much better results, as more underlying causative features become visible after smoothing out the signals (see Figure 5.2).

The algorithm is searching for the strength of signals causation dependencies:

1. For each dimension of the set of actions a, set of resources r, and set of SLA functions f_{SLA}, calculate Similarity S, Consequence κ, and Interference I measures between pairs of source and output signals. Then build a matrix of the final relation strength – called Clarity C. All measures are calculated in steps using a sliding window of constant n length.

2. Create a ranking based on the descending order of strength of clarity causation. If any of the output signals dimensions correlate with other than this one but with a lower clarity value it will not be considered in the first order.

The Similarity measure S (simx) between two metrics time series, source s and output o signals, is $S_{(s,o)} = \sum_i \mathbf{S}_{(s_i,o_i)}$, where s_i, o_i are the i-th window of source and output vectors, and the i-th vector of similarity series $\mathbf{S}$ is calculated according to the following formula:

$$\mathbf{S}_{(s_i,o_i)} = 1/(\delta(\hat{s_i}, \hat{o_i}) + 1) * \overline{\hat{s_i}} * \overline{\hat{o_i}}, \tag{5.2}$$

where $\hat{s_i} = (s_i - \min s)/(\max s - \min s)$ is a scaled (normalised) vector of signals s and $\overline{\hat{s_i}}$ is the sample arithmetic mean of the vector, the same "hat" and "overline" notations are used for vectors of o signals, whilst $\delta(s_i, o_i)$ is a distance function of source s and output o signals in the i-th sliding window. There were a few distance methods evaluated: starting from the simple euclidean distance, the DTW (Keogh, 2002), and the distance derived from Pearson's, Kendall's, to Spearman's correlation coefficients. Figure 5.3 illustrates the differences between the three distances tested.

In order to improve further the check about the similarity in the source and output signal shapes, another step which is based on root-finding of the first derivative of smoothed signal functions can be optionally added. This minimises a negative effect of high similarity of long high level signals. In such situations distances would be mainly influenced by one of the high signals values (see Eq. 5.2). This step requires the original signals to be smoothed out with use of Cubic Smoothing Spline (CSS) [1] (Reinsch, 1967), $\mu_{s_i} = \mu(s_i)$, then the first derivative μ'_{s_i} root is calculated with use of linear interpolation approach.

Further enhancement of the method is a measure built by calculating the degree of cross–correlation. This optional extension of Similarity confirms the dependence of an output from the input signals evaluated. The Dependence measure is calculated following the

[1]This is to avoid error multiplication problem, since derivative action is sensitive to measurement of noise (Reinsch, 1967; Wang, 1998)

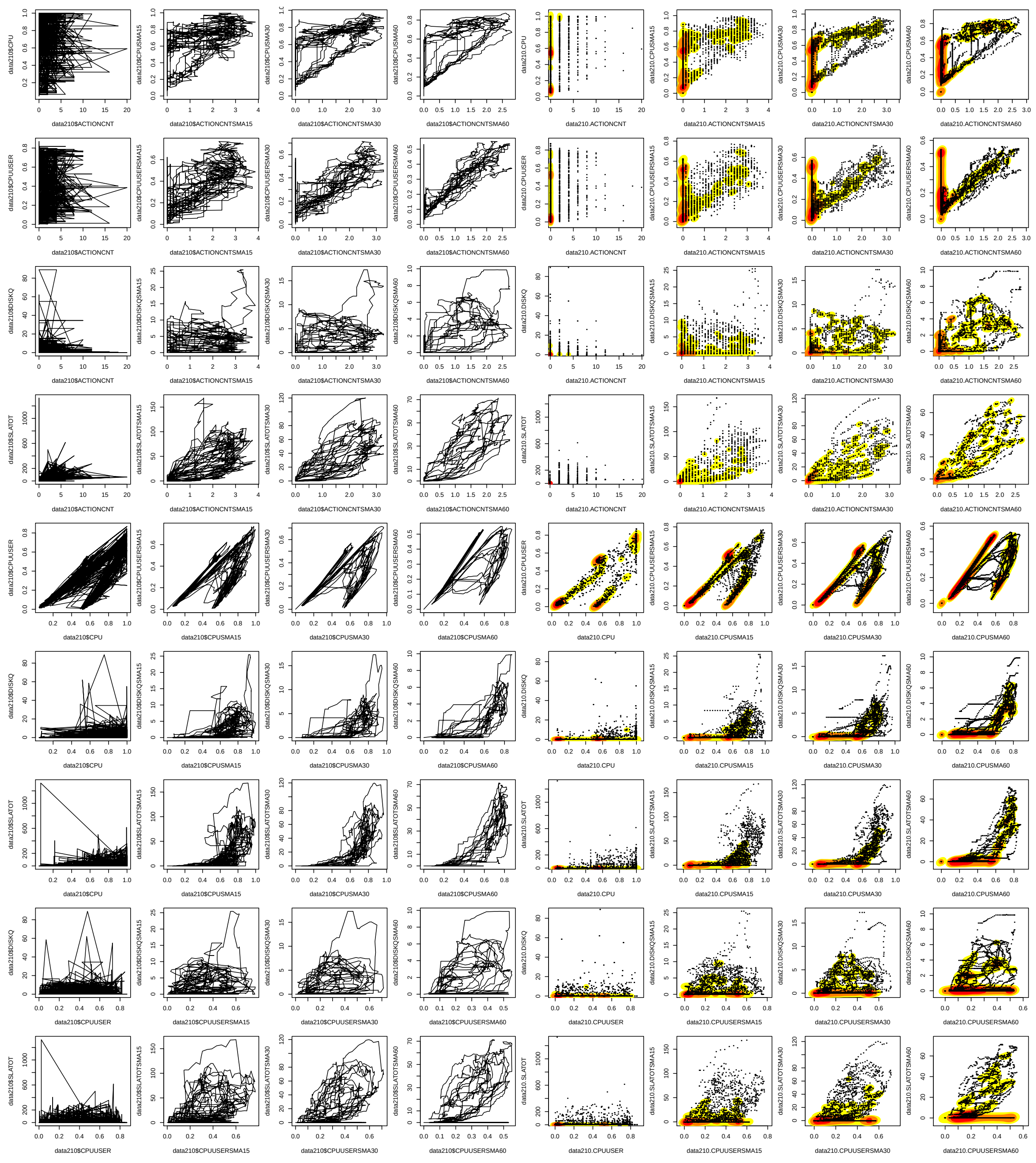


Figure 5.2: Comparison of four different SMA window lengths, 0 (no SMA), 15, 30, and 60 secs, respectively, that are applied to the same time-series of metrics data (window length denoted in the dimensions names suffixes along the horizontal axis). The figure contains two views trajectory-maps as vectors between system state, showing sequence of changes (four columns on left), and measurements distributions, showing statistical properties of measurements apportioning (four columns on right). Each row shows a different bivariate trajectory-map/distributions illustrating the relationship between a source and an output dimension (signal). The wider the moving average window is the less noisy the data get and more clear is the relation between dimension pairs of high causality. By applying the SMA we can uncover some characteristics often hidden in raw data representing system states. The data used in this figure come from the second example scenario, which is presented in Section 5.6.2.

key assumption of causation theory that in a causal system a response never precedes an input (Pearl, 2000). Thus, the measure focuses on checking positive lags of output to input signal. Dependency is a result of cross–correlation (Box et al, 2011a) on input and output signals being evaluated, where the maximal matching lag (the biggest cross–correlation lag in a declared time window) derives the Dependency strength: $D = (l_{max} - t_l)/l_{max}$, where t_l is the maximum product of cross-correlation function estimation $t_l = \max(ccf(s_i, o_i, l_{max})) : t_l \geq 0$ in a set maximum lag window l_{max}. Of course negative lags are not considered since they are found against the assumed causality chain. Shorter lags (delays) are promoted over longer ones, so Dependency is the highest when $t_l = 0$ and minimal, if lag is equal to the set maximal window length $t_l = l_{max}$. Conceptually Dependency D is very related to the Similarity measure, and as a scaling coefficient of the enhanced Similarity $S_{[D]} = S * D$ directly impacts the Clarity measure C explained later (see Eq. 5.5). This is a significant extension of the SCIC method; therefore the SCIC variant that is using the Dependency measure is named SDCIC and will be tested and discussed further in the next section.

The Consequence measure κ (conx) of similarity is a scalar value, indicating how often higher source signal s causes higher output o in the whole time period analysed; the more often similarity $S_{(s,o)}$ (effect) is found the stronger the relation. This measure is calculated to minimise coincidental similarities, and to lower the risk of promoting identified similarities without causal relation. Simply speaking, consecutively found similarities are rewarded, and those which do not reoccur, or occur rarely, effect lower Clarity measure values.

$$\kappa_{(s,o)} = \frac{S_{(s,o)}}{\sum \hat{s} + \sum \hat{o} - S_{(s,o)}} \tag{5.3}$$

The measure of Interference I (intx) has been introduced to limit the significance of high similarity in situations where more than one source signal matches with the same output signal being investigated. In such cases there is generally lower certainty about the causation strength, and therefore weaker dependency relation. We define interference as a sequence of cumulative similarity[2] of all source signals except the currently considered k-th source $s^{(k)}$; the i-th element of interference vector $\mathbf{I}$ is computed according to the equation:

$$\mathbf{I}_{(\mathbf{s_i^{(k)}}, \mathbf{o_i})} = \sum_{j:j \neq k} \mathbf{S}_{(s_i^{(j)}, o_i)} \tag{5.4}$$

The scalar interference value $I_{(s,o)} = \sum_i \mathbf{I}_{(s_i,o_i)}$, where s_i, o_i are the i-th window of source and output vectors.

The Clarity C (clax) is the final measure of dependence of the proposed method. It

[2]All vectors with similarity patterns between source signals are added up creating a cumulative sequence.

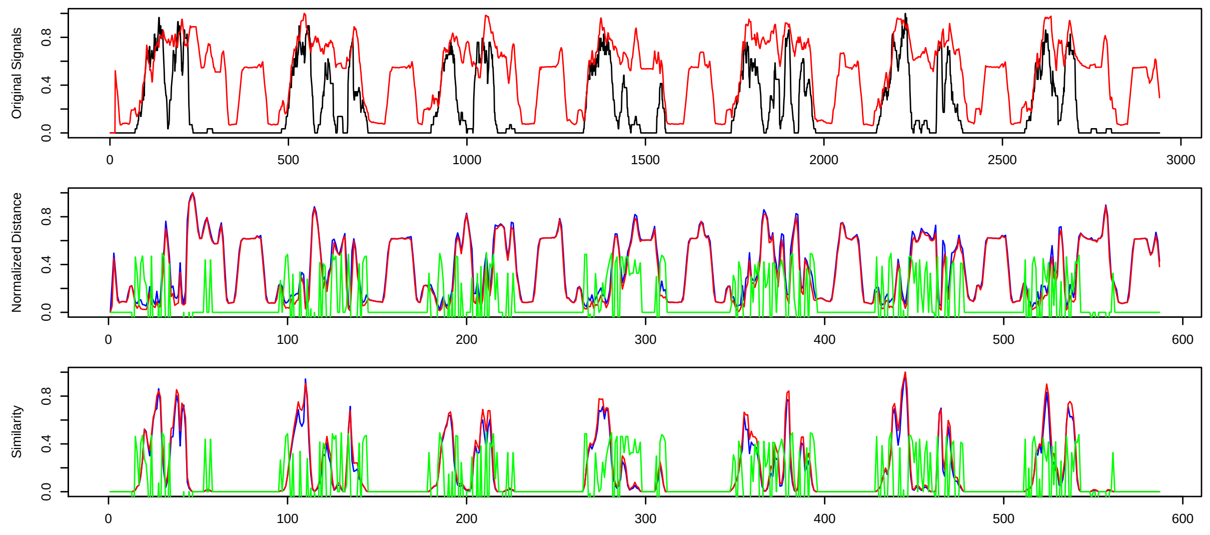


Figure 5.3: Comparison of times series matching process with use of three different distance measures Euclidean (red), DTW (blue), Spearman's correlation (green) in a window of 5 points (seconds). From the top chart with input signals, middle with normalised distances, and bottom with Similarity sequence applied.

represents the strength of impacts dependencies of source to output dimensions. It is based on similarity **S**, its consequence κ and interference **I** of other source signals and is calculated by the following equation, for the i-th element of the vector:

$$\mathbf{C}_{(s_i,o_i)} = \frac{\mathbf{S}_{(s_i,o_i)} * \kappa_{(s,o)} c_\kappa}{\sum \mathbf{I}_{(s_i,o_i)} c_\mathbf{I} + 1} \tag{5.5}$$

The scalar clarity value $C_{(s,o)} = \sum_i \mathbf{C}_{(s_i,o_i)}$.The equation includes also Consequence c_κ and Interference $c_\mathbf{I}$ scaling coefficients, as additional constant parameters configuring strength of measures in Clarity calculation.

5.6 Testbed Design and Experiments

All empirical evaluations were done with the use of R scripting, which can be directly called from the Java hosted ASM controller. Several experiments were performed on datasets of synthetic data (Section 5.6.1), and captured from the testbed system (Section 5.6.2) to test the effectiveness of the approach.

To comparatively evaluate the performance of the distance measures applied in SCIC, we performed a test of different sliding window length n, see Figure 5.4; all charts contain normalised values adjusted by the size of the sliding window. The comparison shows that the ranking method is stable against sliding window length, and fairly similar across presented dimensions method.

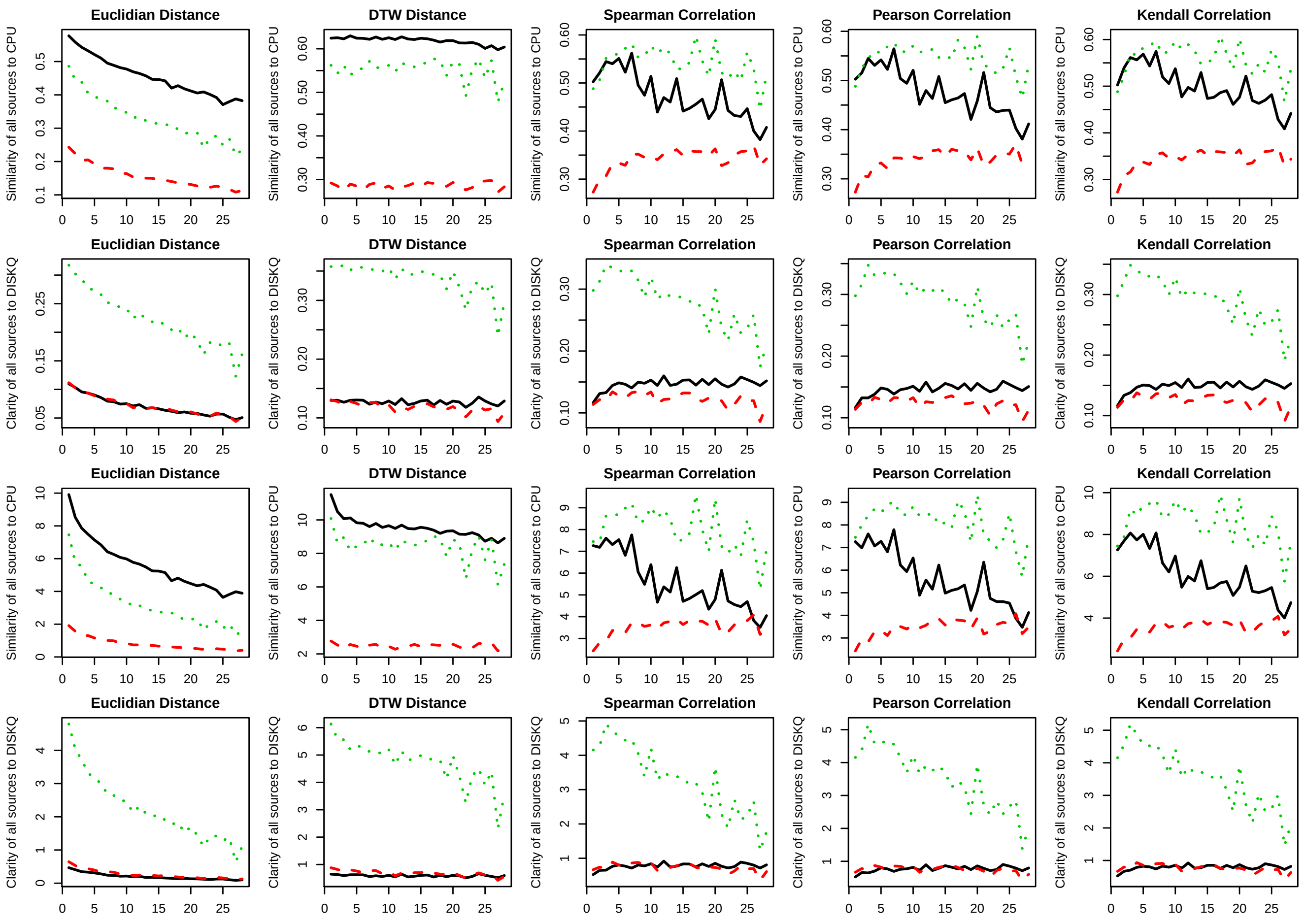


Figure 5.4: Figure presents comparison of Similarity (first two rows) and Clarity (last two rows) in five feature selection tests using different distances measures (in columns). Each of the tests computed the dependency relation of actions to resources of data taken from Scenario 2 (Section 5.6.2), where lines denote different departments' actions: solid–black line: Reservations, dashed–red line: Operations, dotted-green line: Finance. In these tests the SCIC algorithm was executed for different sliding window lengths, ranging from 1 to 30.

5.6.1 Scenario 1

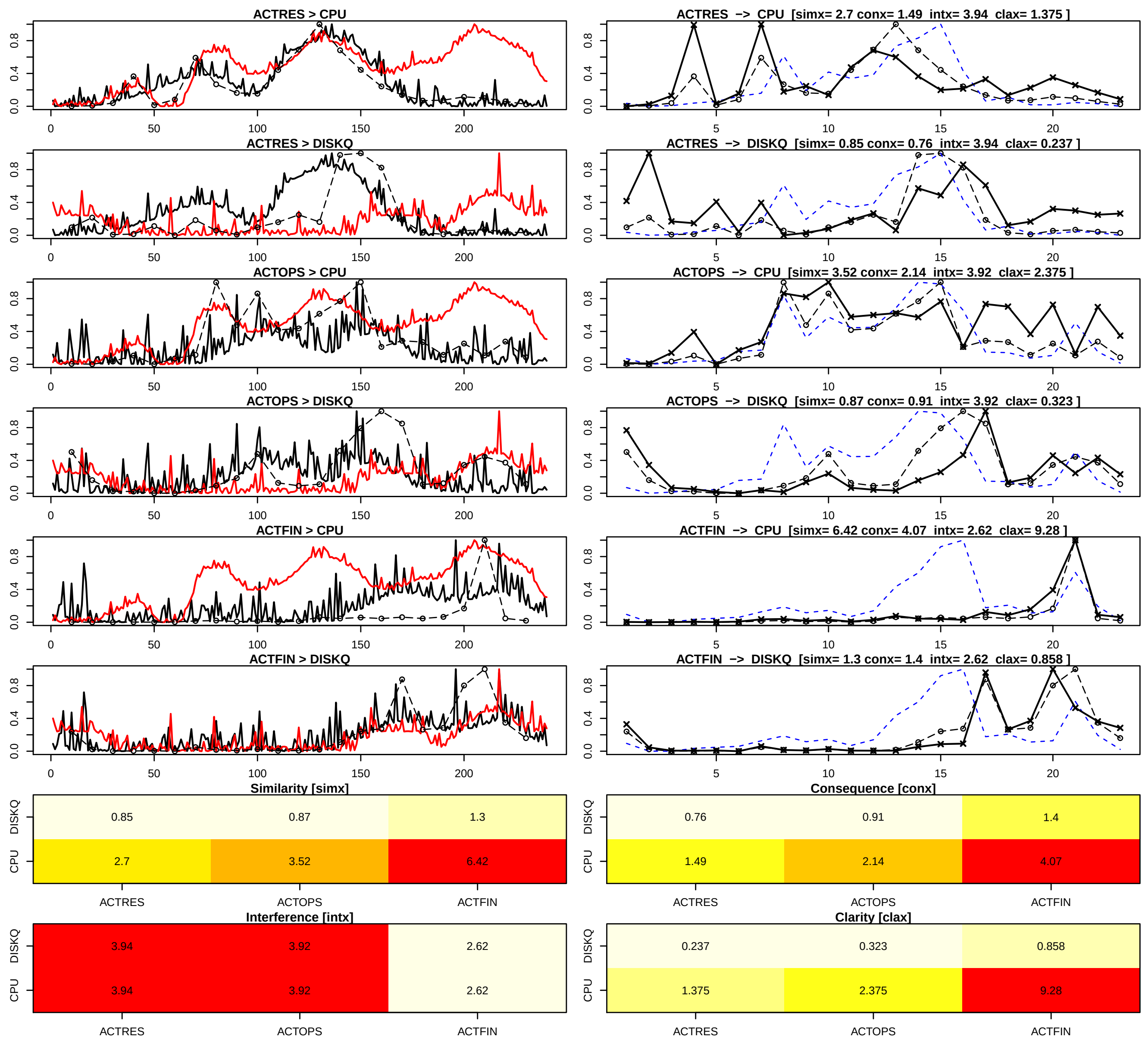
This example contains synthetic data sample of aggregated values collected during 24 hours of normal operation of a hypothetical world-wide used system with 3 departments of the organisation using different system actions. Reservations is based in Europe and America and is characterised by high use, and very frequent calls; Operations, based in India, cause high system use, frequent calls, and long sessions. Financial Reporting, based in America, generates rare actions but they include heavy IO/CPU tasks; an example is shown in Figure 5.5. Due to the different geographic locations, the load is distributed in time.

In this case a summary of metrics about actions execution in time has been considered, so data evaluated are aggregates in a given time bucket. To simplify the presentation only two resources are being analysed CPU and DISKQ. As mentioned earlier, the highest computational load comes around midday from Reservations (ACTRES), and later at the end of the day from the Finance (ACTFIN) department when also the disk is heavily utilised. In this operational context, the actions-resources matching areas are found and the Similarity matrix is shown in Figure 5.5. Although Reservations actions (ACTRES) were substantially interfered by Operations (ACTOPS) activity, the Similarity and Consequence measures were firm enough to produce the highest Clarity level across evaluated pairs. The Clarity matrix shows that disk usage has been revealed to be a relevant dimension for Finance actions mainly. Lastly, the Dependency measure has not been applied in this scenario as no significant effect of delayed system resources signals or SLA response was found.

5.6.2 Scenario 2

In contrast to the earlier test, this dataset was captured during a 48 minutes load test[3] containing 7 subsequent runs of a real testbed application (c.f with Figure 5.8) discussed below). The causal model is defined using a three–tier approach: actions, resources, and SLA. In this case, there are five actions, four resource dimensions and three SLAs. Each of the actions is called with the same pattern but with different load, start of the load or length of the load. The load test was prepared in such a way that only the first action (ACTIONCNTSMA15) causes significant CPU and IO utilisation. The rest of actions consume minimal resources. Nevertheless without knwoeldge of the model of the system, input and output dimensions evaluation must be performed on normalised level as pure actions quantity values do not provide the information required to uncover particular actions' impacts on resources. To hide the real use of the system from action's perspective, only action counts metrics are considered (actions execution time would give much better hint on the actual system usage). Raw metrics data were pre–processed with use of SMA with a 15–secs window. Two of the actions were called exactly with the same pattern length

[3]During the load test, monitoring was capturing a detailed state of the system every second; thus the set consists of 2880 sample points.



Similarity [simx]	ACTRES	ACTOPS	ACTFIN
DISKQ	0.85	0.87	1.3
CPU	2.7	3.52	6.42

Consequence [conx]	ACTRES	ACTOPS	ACTFIN
DISKQ	0.76	0.91	1.4
CPU	1.49	2.14	4.07

Interference [intx]	ACTRES	ACTOPS	ACTFIN
DISKQ	3.94	3.92	2.62
CPU	3.94	3.92	2.62

Clarity [clax]	ACTRES	ACTOPS	ACTFIN
DISKQ	0.237	0.323	0.858
CPU	1.375	2.375	9.28

Figure 5.5: Figure represents matching of Actions to Resources for 24-hours of operation of a hypothetical world-wide-used system. The bottom of the figure presents four matrices with Similarity, Consequence, Interference, and Clarity values for each of signals pairs evaluated. Left column shows input (black line), output (red line) signals, and Similarity sequence (dashed line). Right column contains comparison of sequences of Similarity: dashed line with o–points (where Spearman's correlation coefficient as distance method was used), Interference: dashed blue line, and Clarity: solid–bold line with x–points.

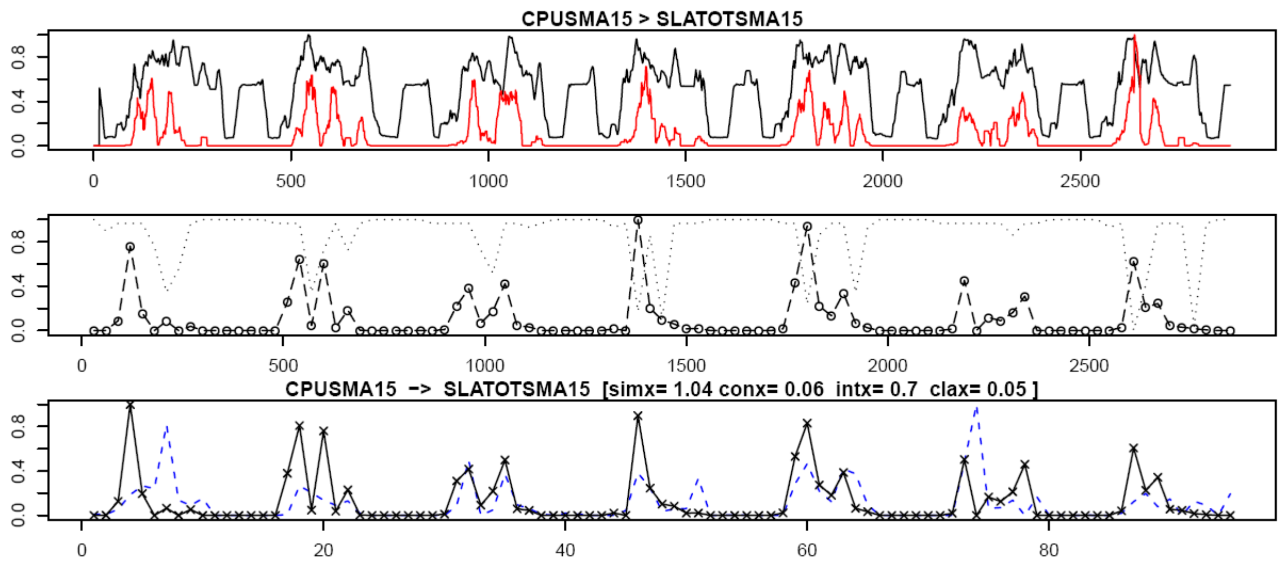


Figure 5.6: Example of evaluated dimensions and sequences of the SCIC measures. From the top, plot of input (CPUSMA15) and output (SLATOTSMA15 in red) signals1ak10n. In the middle, Similarity (dashed line with o–points) and Dependency (dotted loine), and at the bottom plots for Interference (dashed blue line), and Clarity (solid–bold line with x–points.

as the previous action, but with a shift of 100 and 200 seconds respectively, so they are overlapping each other. For two of the other actions, the load pattern was extended two and four times. This type of profile effectively impacted the Interference measure.

Figure 5.6 lends some insight into the details of the Similarity-Dependency and Interference-Clarity pairs. In order to illustrate better the dynamics of the SCIC all measures sequences have been normalised, so values on the charts do not reflect the real values, but this can help readers compare the shapes easier.

Figure 5.7 presents the results of SCIC algorithm execution. Left column shows input (actions) and output (resources) signals. In the bottom of the figure four matrices represent values of main factors calculated for each of signal pairs evaluated. The highest similarity measure was found for ACTIONCNTSMA15 and CPUSMA15, but high values were also for CPUUSERSMA15. Consequence and Interference matrices confirm the strength of these relations. It is worth noting that the Interference matrix contains high values of ACTIONCNT1SMA15 and ACTIONCNT1BSMA15 which most often overlap with other entry signals, so even if similarities for these signals were high, it would be difficult to identify which ones of those had impact on the resources. Interference of ACTIONCNT1CSMA15 is the lowest, but Similarity and Consequence are too low to promote this entry signal as relevant for resources utilisation. Despite the fact that the load test was obfuscated with other non relevant for the resources consumption actions, the method was able to isolate enough information to set the final dependency measure (Clarity) in accordance with the real workload.

Analogical processing was executed for the second tier of the dependency model concerning Resources and SLAs, as shown in Figure 5.8. Notice the very low Clarity of MEMUSEDSMA15

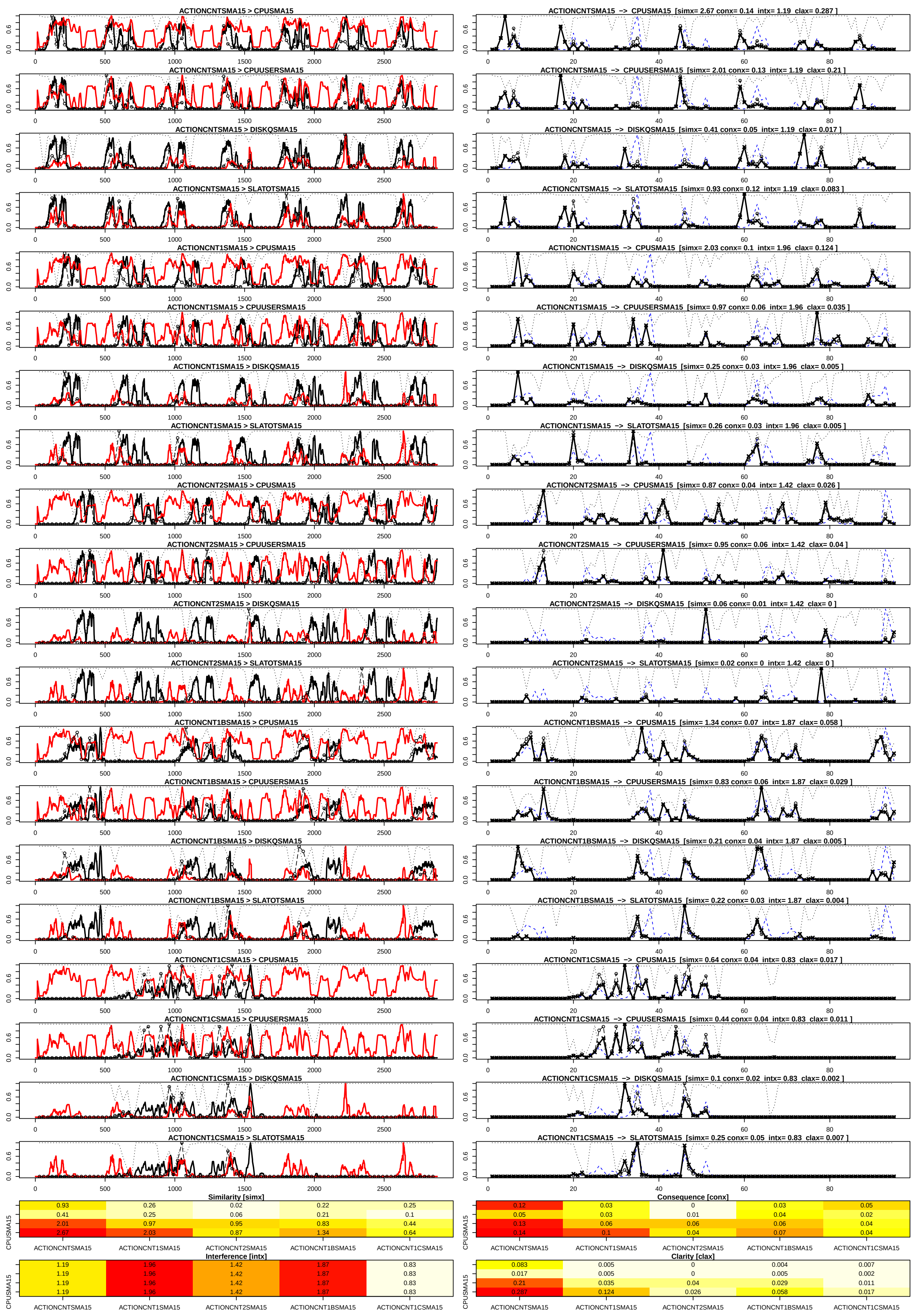


Figure 5.7: SDCIC–based matching of Actions to Resources. At the bottom, the four matrices show Similarity, Consequence, Interference, and Clarity values for each of the signal pairs evaluated. The left column shows input and output signals (black denotes an input while red an output signal). The right column shows a comparison of normalised sequences of all measures, namely Similarity (dashed line with o–points), Dependency (dotted line), Interference (dashed blue line), and Clarity (solid–bold line with x–points).

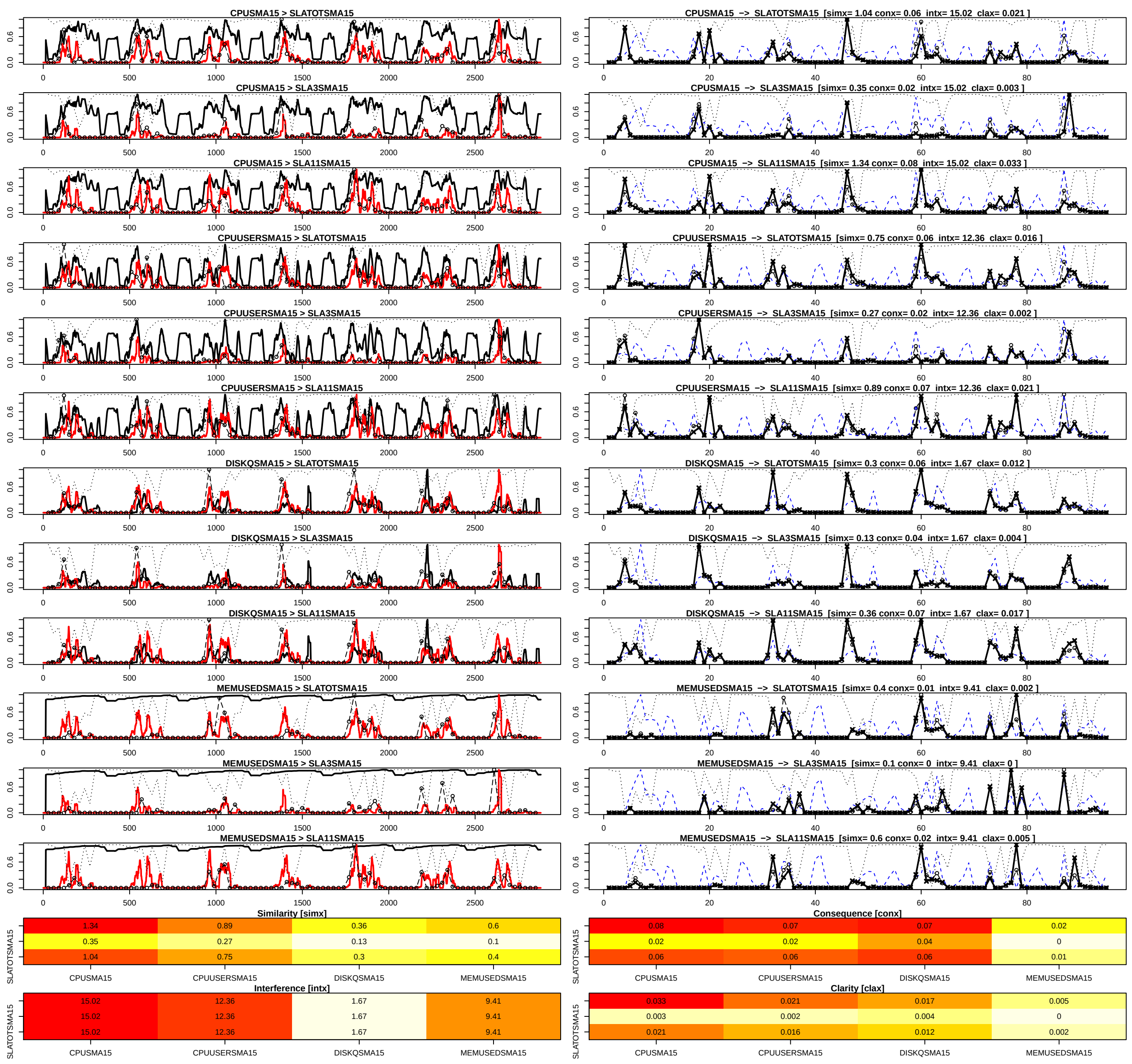


Figure 5.8: SDCIC–based matching of Resources to SLAs signals, with illustration of the process as in Figure 5.7. All measures sequences are denoted as before, including the four matrices at the bottom.

pairs, mainly due to low Consequence. Notable is also the very low level of Interference of DISKQSMA15, which contains fairly rare high signals. Although it did not interfere with others, it had too low similarity with any other SLA dimension to set Clarity to a high level.

On the basis of the SDCIC processes performed against the defined tiers of the causal chain (Actions to Resources, Figure 5.7; and Resources to SLAs, Figure 5.8), a casual dependency graph was generated that is an alternative representation of the Clarity matrices. After selecting the top of the ranked Clarity measures, the graph was reduced containing only the strongest dimensions in the causal chain, see Figure 5.9.

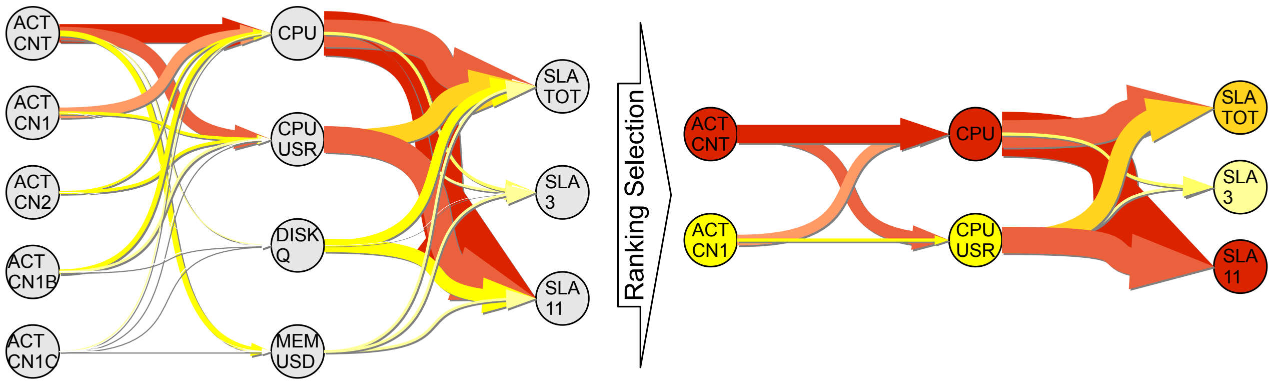


Figure 5.9: Dependency graphs of dimensions used in the three tiers of causal chain. The graph on left shows all dimensions matched (see Figure 5.7 and 5.8) before the future selection step. The graph on right contains only the strongest affected dimensions, according to the SDCIC method, after ranking based selection.

5.7 Limitations and Further Considerations

The proposed approach is sensitive in situations where many source signals are similar to each other (i.e. input dimension values change in time in similar way, resulting in high Similarity measure S between them). Then due to the Inference measure I definition such similarities are strongly minimised in the Clarity C. If the system works only under such conditions then it would be difficult to infer which elements of the causal model are dependent on each other, and the model can be evaluated inefficiently. In such cases requirements may be refined to allow the control system to perform an intervention, with the aim to introduce breaking impulse, changing the high correlation of the similar signals.

Lastly, the method may fail to provide precise results in situations where not enough run–time descriptive metrics are available; actions significantly change the system run–time characteristic after the execution, or data are collected for very short time; so uncertainty is high and estimations are prone to flaws or the causal model is miss–specified.

These are limitations that deserve further consideration and some enhancements are explored in Chapter 6, where methods of signal deconvolution are investigated to improve the dimensions relevance method and complement it. Section 5.7.1 builds a link with the next chapter introducing the approach, whilst Chapters 7 and 8 are dedicated to the control activity, where aspects related to control intervention, as mentioned above, forming a necessary element for wider causality inference (Pearl and Mackenzie, 2018), are explored further.

5.7.1 Uncovering Sources of Causes with Signals Deconvolution

The feature selection technique can be further improved by applying a preprocessing step of ASM signals separation, which is a crucial problem for non–isolated metrics space areas, where many actions use the same resource and by this they impact its utilisation metric. Blind Signal Separation (BSS) and Semi–Blind Signal Separation (SBSS) tackle similar problems in the signal processing domain (Comon and Jutten, 2010). Equation (5.6), which resembles those used in BSS techniques, describes hidden components in a resource signal r_k effected by actions a (actions and resources dimensions were used only as examples):

$$r_k = \sum_i p_i(r_k)a_i + n_k \ , \tag{5.6}$$

where r_k is the k-th resource, a_i is the i-th action p_i is the unknown coefficient of a component causing changes in r_k after a_i calls, and n_k is noise.

However, there are many challenges related to ASM signals decomposition, explored further in the next chapter, that prevent the use of method to blindly unmix ASM metrics data. For example, a core obstacle to employing off–the–shelf BSS methods and software is that the collected metrics sequences can be statistically dependent – thus, many of BSS techniques from signal processing cannot be applied directly.

Another issue in the ASM context relates to the non–addable linearly signals used, i.e in a situation when a given resource is highly or fully utilised, actions using it are “queued” by the system. During such a condition of saturation, the signals are distorted/deformed. In such situations utilisation resources are typically flattened to the maximal value and extended in time while actions execution time signals form much higher peaks. Thus, the parameter p_i in Equation 5.6 may be changing significantly, in a highly non–linear way, impacting most BSS methods that adopt a similar formulation, as their main assumption is not met.

As all signals source data are stored in a database, it is possible to start addressing such problems by building models of statistical properties/measures, so derivation of hidden components can assess the distributions of signals being evaluated. Moreover, assuming that the cause–and–effect delay is known (in simpler cases it is close or equal to zero), particular hidden components can be matched using similarity measures for the unknown coefficient values searched. The ASM Signals Deconvolution (ASM-SD) should be also supported by the assumption that resources r are utilised by actions a, thus cumulative time of action execution is directly related to used resource until a given time.

5.7.2 Causal Inference and Counterfactual Reasoning Methods

As discussed in Section 5.3, causal inference (Pearl, 2010) relies on intervention that is generally speaking a treatment to the observed environment that has been designed to change its characteristics in a foreseen way. This section will review a prototype method that can be used to validate the strength of set causality chain and outline a dimensions' (variables) relevance by ranking the variables defined as those that explain the environment (predictor variables).

Intuitively, whenever observations are far different than the expected source values it is advised to review the effect from a causal perspective. For instance, after a new version release, a new usage behaviour schema change, or after a new marketing campaign, in cases when it is clear when the pre– and post– intervention moment of the new treatment is introduced.

The three–level ladder–of–causation proposed by Pearl and Mackenzie (2018) outlines the model of Causal Inference, where level 1 is "Association" (observing) which invokes purely statistical relationships defined by naked data. At this level the models can learn correlations which is often suitable and sufficient to build powerful predictive models. ASM-SD and SDCIC work at this level. Earlier explained methods and other conventional machine learning approaches, built on correlation analyses and pattern recognition, are insufficient for proper causal analysis. The second level on the ladder "Intervention" (action) comes with the operator *do*, which can be applied together with other techniques in the ASM realm. Figure 5.10 presents an example of integration of methods explained in this chapter and the following one (c.f. with Figure 1.3 depicting the dissertation structure), and Causal Impact analyses after interventions. The highest, third level of the ladder "Counterfactuals" (retrospectives, alternative version of past events) tackles considerations of an alternate version of past events, predicting on what would happen under new, different to the measured already circumstances for the same system. A white model at this level is undoubtedly still in infancy and human factors like experience, intuition in depth understanding of the system structure, interactions topology and the code play chief role in decision making, as discussed in Chapter 9.

In the thesis, there are several possible levels of "intervention" explained in separate chapters in isolation but they brought together here to help see the whole: (a) in SDCIC and ASM-SD, while selecting relevant dimensions, reducing dimensionality of the space and signals decomposition refuters approach can be applied as an intervention to data observed, as elaboated in Chapter 6; (b) in control action rule induction, Section 7.4.4 – separation of data for training and testing the model on NN used for controllers actuating blocks; (b1) in control resolving strategies, Section 7.4.7 – limiting contradictory or inconsistent control decisions in the space of operation of the ASM dimensions by Randomised Controlled Trial (RCT); (b2) in control rules induction of Reinforcement Learning (RL), Section 8.5.2; (c) control as intervention not to data observed but interaction with the real system (not even a model of the system) governed by human operators under PDLC and SDLC.

Figure 5.11 depicts cycles of ASM control and PDLC response to systems structure and functionality. The shorter cycles represent faster control actions and methods requiring less intervation from the human operator. The levels of the cycles indicated are built on the same knowledge base of APM of the system, as mentioned in Section 4.2.

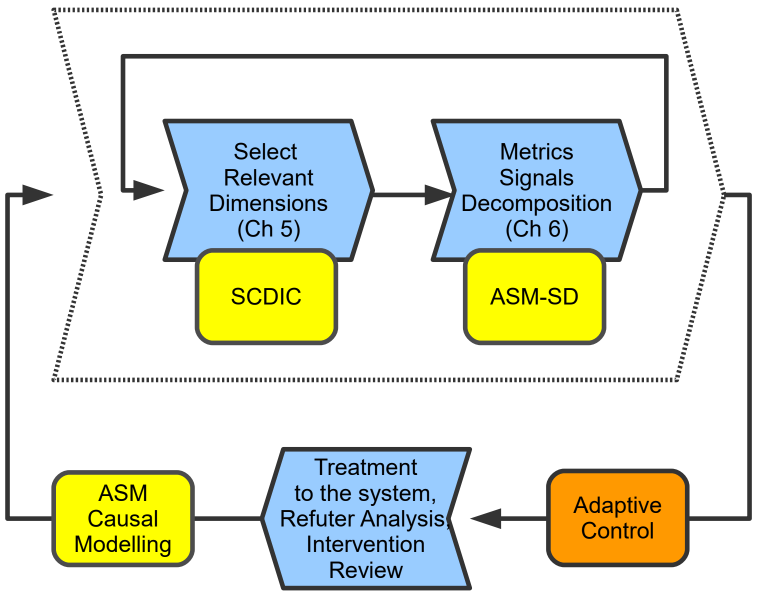

Select Relevant Dimensions (Ch 5)
SCDIC
Metrics Signals Decomposition (Ch 6)
ASM-SD
ASM Causal Modelling
Treatment to the system, Refuter Analysis, Intervention Review
Adaptive Control

Figure 5.10: Integration of SDCIC, ASM-SD with adaptive control and intervention review for ASM causal model building.

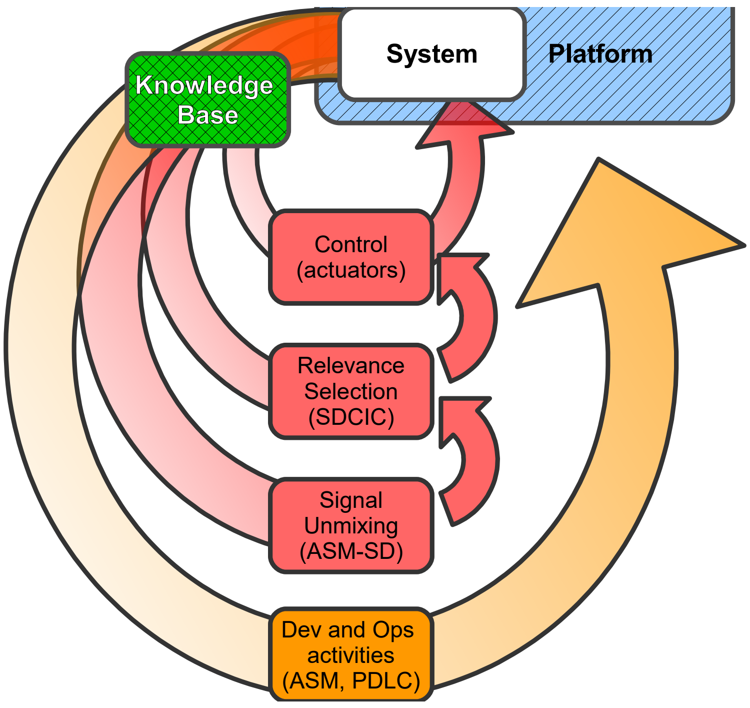


Figure 5.11: Causality Inference, Control Cycle and PDLC

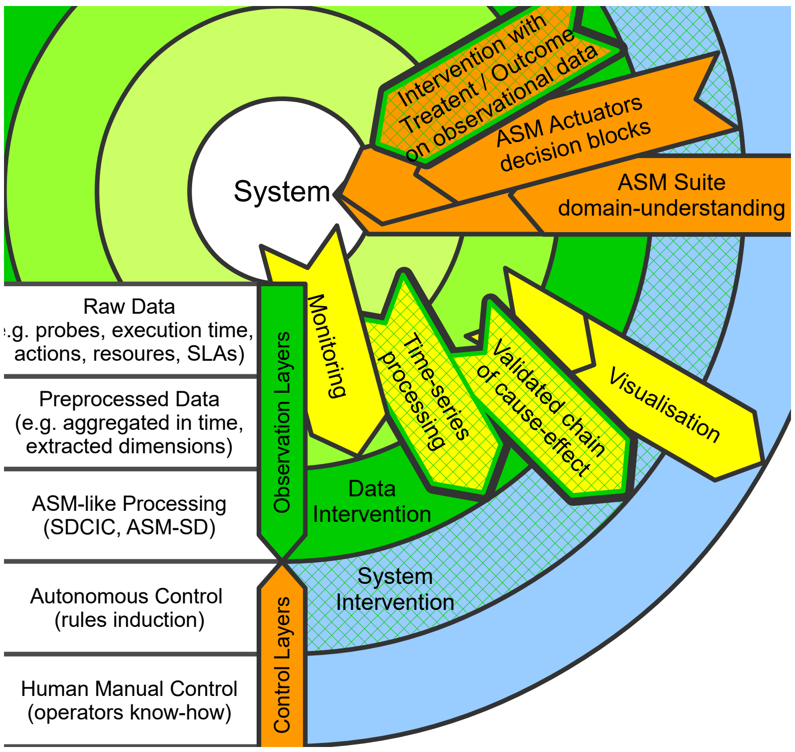

System
Intervention with Treatent / Outcome on observational data
ASM Actuators decision blocks
ASM Suite domain-understanding
Raw Data (e.g. probes, execution time, actions, resoures, SLAs)
Preprocessed Data (e.g. aggregated in time, extracted dimensions)
ASM-like Processing (SDCIC, ASM-SD)
Autonomous Control (rules induction)
Human Manual Control (operators know-how)
Observation Layers
Control Layers
Monitoring
Time-series processing
Validated chain of cause-effect
Visualisation
Data Intervention
System Intervention

Figure 5.12: Layers of information processing and control include observational aspects with SDCIC (yellow chevrons), ASM-SD and control layers (orange chevrons) and different intervention methods (checked in green chevrons).

Figure 5.12 illustrates the integration between the various information processing and control layers. Sliced spheres indicate layers of specific monitoring and control and refer to methods that operate in each layer. In the signpost, yellow denotes observations–related processes, orange indicates that interventions are made by the user or the control system or both, while checked–green highlights the intervention approach used.

In order to demonstrate the practical value of introducing causal modelling, we run an analysis to infer the expected effect of a given intervention (control action, outcome of release, business roll–out, or marketing campaign) had on a response variable (i.e. an SLA of an action) by analysing differences between expected and observed time series data.

The CausalImpact R package performs causal inference through counterfactual predictions using Bayesian structural time–series models [4]. The package implements an approach to estimating the causal effect of a designed intervention on a time series (Brodersen et al, 2015).

To illustrate how the package works in the ASM field data were collected by running simulations of a control system with the same model as in Chapter 8 (see metrics time series in Figures 8.6 and 8.7).

The entry data set for the analysis consists of a response (outcome) variable `y` that is a time series of an action SLA (`A3IOm4.effExecSLA`, effective execution time driven SLA

[4]CausalImpact is available on github https://google.github.io/CausalImpact/ and CRAN https://cran.r-project.org/web/packages/CausalImpact/vignettes/CausalImpact.html

of an action that is IO bound) and a set of predictors variables (covariates) describing the utilisations of disks and processors, as well as execution times and load indicators. The package authors suggest in practice including many more predictor variables and letting the model choose an appropriate subset.

The analysis below concerns causal impact after intervention on an ASM system testbed by switching on the controller in a scenario similar to the one used in Figure 8.6. In this example, the dataset contains 2071 observations. As per the model setup, that will be explained in detail in Chapter 8, the intervention effect is expected to be improving the response variable after time–point 1071, as declared in Listing 5.1. It is worth noting that this method is sensitive to misguidance on the point that the intervention period begins; even one time point too early or too late can significantly negatively influence the causal impact model.

Listing 5.1: Example of CausalImpact.

```
#It consists of a response (outcome) variable y and a set of predictors variables x1..x7.
y  = ex.i4.0.n0.p10.termtrue.evaltrue.ctrlapp3.scen1.A3IOm4.effExecSLA.ap.v30
x1 = ex.i4.0.n0.p10.termtrue.evaltrue.ctrlapp3.scen1.r.diskUtilization.ap.v30
x2 = ex.i4.0.n0.p10.termtrue.evaltrue.ctrlapp3.scen1.r.diskWaitQ.ap.v30
x3 = ex.i4.0.n0.p10.termtrue.evaltrue.ctrlapp3.scen1.r.processorUtilization.ap.v30
x4 = ex.i4.0.n0.p10.termtrue.evaltrue.ctrlapp3.scen1.r.procQ.ap.v30
x5 = ex.i4.0.n0.p10.termtrue.evaltrue.ctrlapp3.scen1.A3IOm4.expExecTime.ap.v30
x6 = ex.i4.0.n0.p10.termtrue.evaltrue.ctrlapp3.scen1.A3IOm4.arrivedTerminated.ap.v30
x7 = ex.i4.0.n0.p10.termtrue.evaltrue.ctrlapp3.scen1.A3IOm4.arrived.ap.v30
#that are all merged in one data matrix:
data <- cbind(y, x1, x2, x3, x4, x5, x6, x7)

#To estimate a causal effect, let's begin by specifying which period in the data should
    be used for training the model (pre-intervention period) and which period for
    computing a counterfactual prediction (post-intervention period).
#Thus time points from 1 to 1070 will be used for training,
pre.period <- c(1, 1070)
#and time points 1071 to 2071 will be used for computing predictions.
post.period <- c(1071, 2071)

#Running an analysis
impact <- CausalImpact(data, pre.period, post.period, alpha=0.05)

plot(impact)
summary(impact, "report")
```

The outcome of the analysis is best explained by the plots in Figure 5.13, and the print-out of the summary in the quote below. CausalImpact analysis returns an impact object visualised by three panels of time series, see Figure 5.13, containing the “original” observed response (see top panel), its “counterfactual” predictions (middle panel), as well as “pointwise” (the bottom panel) and cumulative impact estimates along with posterior credible intervals. Note, the higher values of the bottom panel in area of post–intervention showing “cumulative” values the stronger causal effect has been identified.

> Analysis report - CausalImpact
>
> During the post-intervention period, the response variable had an average value of approx. 2.21. By contrast, in the absence of an intervention, we would have expected an average

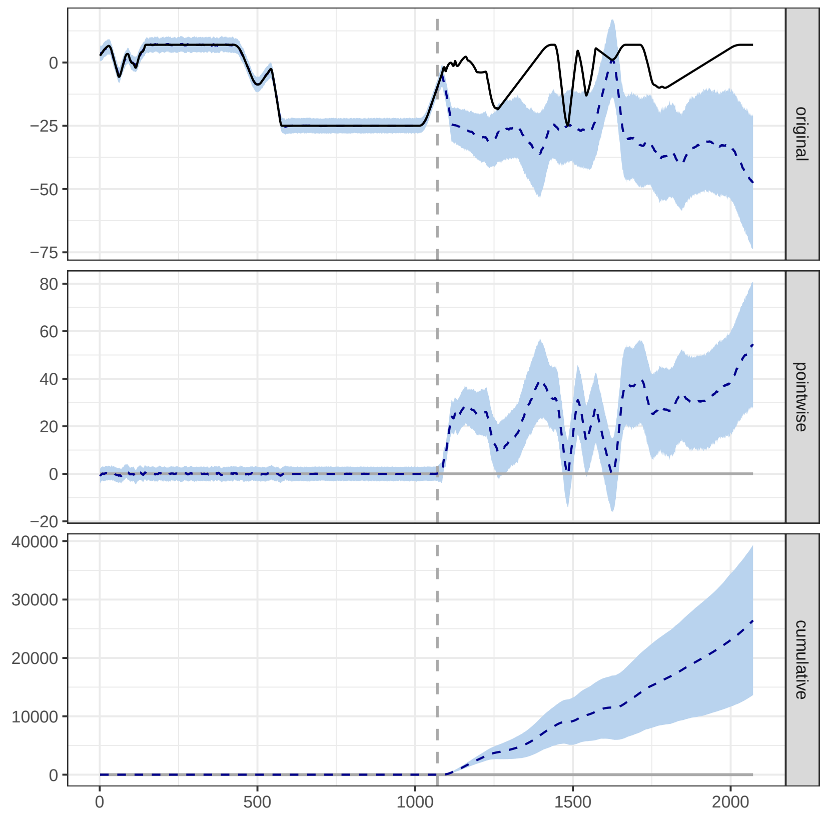


Figure 5.13: Control evaluation using Causal Impact

response of -9.68. The 95% interval of this counterfactual prediction is [-21.74, 2.12]. Subtracting this prediction from the observed response yields an estimate of the causal effect the intervention had on the response variable. This effect is 11.89 with a 95% interval of [0.090, 23.95]. For a discussion of the significance of this effect, see below.

Summing up the individual data points during the post-intervention period (which can only sometimes be meaningfully interpreted), the response variable had an overall value of 2.21K. By contrast, had the intervention not taken place, we would have expected a sum of -9.69K. The 95% interval of this prediction is [-21.76K, 2.12K].

The above results are given in terms of absolute numbers. In relative terms, the response variable showed a decrease of -123%. The 95% interval of this percentage is [-1%, -247%].

This means that the negative effect observed during the intervention period is statistically significant. If the experimenter had expected a positive effect, it is recommended to double-check whether anomalies in the control variables may have caused an overly optimistic expectation of what should have happened in the response variable in the absence of the intervention.

The probability of obtaining this effect by chance is very small (Bayesian one-sided tail-area probability p = 0.023). This means the causal effect can be considered statistically significant.

Figure 5.14 presents ranking of coefficients established by inclusion probability using Causal Impact analysis for a control evaluation (intervention), to understand the predictors variables see Listing 5.1.

According to the calculated ranking the most influential for the causal impact is `x7 = ..A3IOm4.arrived.ap.v30` that is a metric showing quantity of incoming requests of an IO bound action `A3IOm4`, then `x5 = ..A3IOm4.expExecTime.ap.v30` is a synthetically calculated expected execution time of the action in time, next is `x2 = ..r.diskWaitQ.ap.v30`, `x3 = ..r.processorUtilization.ap.v30` and `x4 = ..r.procQ.ap.v30` that are disk wait queue length, CPU and processes queue length respectively.

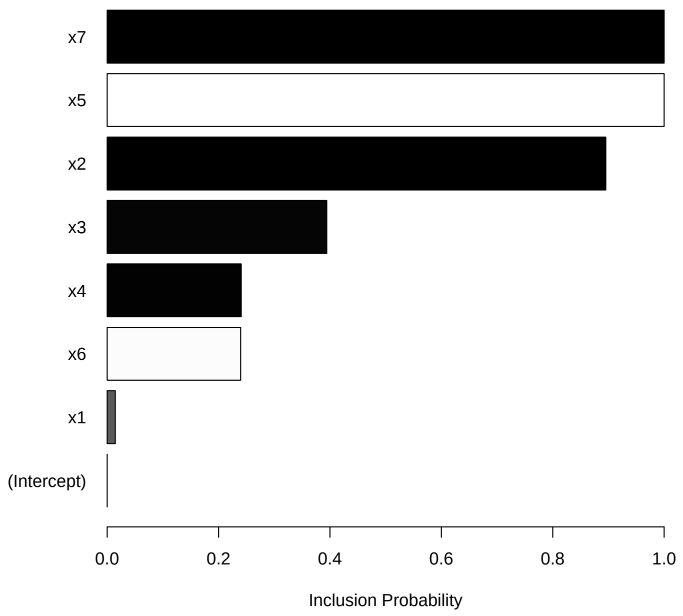


Figure 5.14: Ranking of coefficients by inclusion probability using Causal Impact for a control evaluation (intervention)

5.8 Summary and Contribution of the Chapter

Although ASM control is gaining more and more attention in the enterprise, there is still lack of well researched features selection methods, which could be used to improve autonomous control agents and support ASM operators. To alleviate this situation, the chapter contributed two methods, called SCIC and SDCIC.

Causal dependencies in the ASM control field are known to exists in reference to tier or type of metric (action count, action execution time, resources usage, SLA values). On that basis it is possible to evaluate dependencies between observable dimensions by measuring similarities of input and output signals.

Methods introduced in this chapter build sequences of Similarity, Interference and Clarity values – therefore provide details of interpreted data in time, which can be helpful for understanding the relationships evaluated. In cases when only some actions are controllable $a_c \in a$, it is possible to find out which controllable actions cause most impact on resources and SLAs and include that in the ranking accordingly. This can be an interesting audit property of the method when applied in the engineering problems space and used by operation teams.

Additionally, the introduction of the Dependency measure complements and enhances the Similarity, used in the original SCIC method, by applying further causality observations based on cross–correlation. The use of this measure leads to lower Clarity scores in situations when observed effects were delayed against the set causality chain. The evaluated scenarios confirm the practical potential of the proposed approach. The research about features selection methods suitable for more focused autonomous control in the ASM field is far wider. In particular there are additional methods tackling various aspects relating to signal decomposition, unmixing hidden components, and incorporating knowledge about the system reactions delays, which are presented in the next Chapter of the thesis.

Chapter 6

Signals Decomposition in Application Service Management

"As our circle of knowledge expands,
so does the circumference of darkness surrounding it."
— Albert Einstein (1879 – 1955)

"Understanding is more important than memorisation."
— Richard Feynman (1918 — 1988)

Following up on the previous chapter, Chapter 6 continues the thesis investigation on ASM signals feature extraction. It introduces a methodology that alleviates limitations identified previously by treating the problem as a type of optimisation that can be solved with population–based metaheuristics. This is applied to the problem of activity and resources signals deconvolution as a method of unmixing activity as input to the system from observed resources utilisation. In the following subsections, first a link with previous work in the context of the thesis is established, and then the problem is defined as a type of optimisation. The method is then derived and tested in different scenarios.

6.1 Introduction

The complex nature of EISs requires processes and tools for monitoring and managing application services and their performance levels in order to meet end–users requirements. Resources, system and services activity monitoring tools provide an outlook of the enterprise system operation but gathering measurements and making inferences from the data remain challenging in APM and ASM fields, where usage and performance metrics are acquired from all significant elements of the enterprise systems and from all tiers of its architecture (Haines, 2006; Sydor, 2010; Grinshpan, 2012). Thus, despite the fact that performance profiling and services monitoring are widely used in the enterprise, SaaS, FaaS and wide range of cloud computing businesses, most of the control actions are taken either by humans, or in a semi–automatic way. Typically these are, for example, semi–automated routines, which are implemented using scripts or rules defined in ASM tools, that need to be maintained by administrators. In this context, human decision makers need to react constantly to changing system conditions, or requirements, in order to optimise the performance against a set of objective functions, such as those defined in SLAs. Indeed, human operators or administrators are able to identify which actions are responsible for a particular resource's utilisation by observing actions a and resources r signals, and comparing the shapes and signals characteristics, exploring their understanding of the relation between action type and resources consumption. For example when it is found that a controllable action a_c, or an action dependent on some tuning activities, utilises a resource close to saturation level, an action termination, or an execution redirection, can be applied to limit the excessive computation. Finding and understanding these relations is a key skill–set for any administrator controlling such a process.

However in practice, actions use many resources and the underlying relation is often hidden, even from the expert. Uncovering the nature of the correspondence between systems interactions, utilisation dependencies and performance characteristics requires longer monitoring, while at the same time any metrics time–series gathered should be processed in a high–dimensional problem space in order to take a decision about particular actions. This is related to various kinds of effects present in complex environments, which are often difficult to be anticipated fully before deployment. All these factors make the situation more challenging for human operators and administrators, especially when the problem dimensionality increases and the end–user requirements evolve.

This chapter focuses on a particular stage of the ASM process, modelling of the run–time dependencies between systems (Keller and Kar, 2000), with the aim to support analytics and decision support tools. To this end, it derives a method for decomposing ASM time–series signals, searching for hidden relations between users or systems activity and resources utilisation, that can be used for controlling the impact of unexpected workloads — this is an area significantly underexplored especially in a context of non–acquisitive control methods.

In the next chapter, 7, the reader will have a chance to review the control problem, when no model of the enterprise system is available (Sikora and Magoulas, 2013b). Whilst the previous chapter reported on the use of methods that exploit signals similarity in order to establish causal dependencies (Sikora and Magoulas, 2014a), this approach proved to be sensitive to highly utilised resources; these are normally responsible for serving many actions executions requests.

The method proposed in this chapter, named Application Service Management Signal Deconvolution (ASM-SD), alleviates this situation when decomposing mixed signals, so that more precise signals dependency and analysis of causal chains of events can be provided. The approach is based on decomposing ASM metrics time–series, outlining interdependencies between actions and utilised resources, and can be used to support human operators and administrators, be part of adaptive feedback loop, as outlined in the previous chapter (see Section 5.7), integrated with control evaluation or incorporated decision blocks of autonomous control agents (see Section 8.5.2). In this methodology, ASM signals decomposition is formulated as a search problem that is tackled using evolutionary algorithms.

The chapter is organised as follows. Section 6.2 describes the fundamentals of the problem area. Section 6.3 presents signals decomposition and details of the method proposed for the stated problem. Section 6.4 illustrates aspects of the performance of the method under a few scenarios. Section 6.5 presents outline of signals decomposition optimisation methods. Section 6.6 discusses details related to validation of search methods tested. Section 6.7 outlines other noise models affecting the causality chain. Section 6.8 ends the chapter by outlining future work and discussing a few concluding remarks.

6.2 Signals Decomposition Problem and ASM Dimensions Relevance

Let an ASM system's states defined by activity of services, triggered by users or exposed interfaces, requests, or in other words, actions a and resources r affected by these actions. Passive monitoring of API elements, threads, and sources of calls (e.g. other APIs end–points, user actions or triggered events, authorisation tokens etc.) allows to identify actions usage characteristics. As discussed in Chapter 4 typically resources utilisation is established through active monitoring (Sikora and Magoulas, 2013b). Resources are most often related to elements of the hardware and software infrastructure (e.g. processors, memory, network queue length or discs utilisation) but they can also define more abstract elements requiring synthetic calculations, e.g. SLAs operating as a quality measure of contracted services being provisioned, energy consumed, currently utilised infrastructure prices present in general cloud computing, and other set performance indicators (Neugebauer and McAuley, 2001; Beloglazov and Buyya, 2010b; Beloglazov et al, 2012), including revenue level metrics.

6.2.1 ASM Systems Defined by Metrics Time–series

The measurements define the system responses and identify its run–time characteristics effectively creating a “model” of the system performance characteristics. Considering the high–dimensionality of enterprise systems (Grinshpan, 2012) and the large size of the time–series databases, such a model is difficult to be built. Moreover, there are many complex execution interdependencies in an enterprise system (Sydor, 2010; Sikora and Magoulas, 2013b) that are effectively recognised while running to operate requested services.

Although time–series signals processing for online control that is based on predictions of the dynamics of the application resources has been researched in the past (Gong et al, 2010), how to decompose signals of the metrics that contribute to the resource utilisation for the control of independent actions execution remains under explored in the context of non–acquisitive control; this will be discussed further in Chapters 7 and 8.

Even though concurrently executing actions may have different resources usage characteristics (see more details in Section 6.4), effectively every action execution time is a result of resources r consumption and time spent on each of the elements of the infrastructure. Thus the i–th action execution signal[1] is a function of resources utilised $a_i(t) = \alpha(r, a)$, $a_i(t) \in \{0, 1\}$ that depends on available system resources r and on other incoming or present actions executions a, forming a non–linear system.

Assuming that signals received from resources are convoluted with many action signals[2], then metrics describing actions execution a, can be collated with time spent by the system on using resources, r, monitored (cf. with Equation 6.3 below). In other words we are searching for weights of strength (coefficients) of observed mixtures of actions, executed as resources are utilised, that allow the decomposition or deconvolution of observed signals – that is the key issue in this chapter. In contrast to approaches like standard BSS and SBSS (Bell and Sejnowski, 1995; Cichocki et al, 2002) in the ASM problem we know precisely, most of the time, shapes of the source signals that are formed from actions execution metrics; this is discussed further in Section 6.3.1. When a given resource is highly or fully utilised, actions using it are queued by the system. During saturation the signals are distorted/deformed. In such situations resources signals are flattened and extended in time, while actions execution time signals form much higher peaks. Thus the main assumption of most BSS methods is not met in the ASM context.

Actions a and resources r time–series define the system load–functional performance characteristics, delineating the input and outputs changes, and forming a $\overline{\overline{a}} + \overline{\overline{r}}$–dimensional curve that is a trace of all system states.

In order to address the core problem that is to identify which software elements were responsible for specific resources utilisation, this chapter propose a deconvolution technique to recover the original input signals. The method enables unmixing actions time taken from many monitored areas of a system and establishing coefficients m indicating how executing actions impact the resources usage.

[1]A signal as a vector defines an ongoing flow of information, a measured metrics or any other quantity that changes in time.

[2]Threads sleeps and resources waits form interesting situations. Although both sleeps and waits impact directly action execution times (an action does nothing as defined in the code) and active thread count, their relation to other resources utilisation is very different. Whilst waits for resources are included in the resources utilisation and queue lengths metrics, sleeps have minimal effect on CPU, Disk or other physical resources utilisation.

6.2.2 Industry Practice

An important consideration in operations is to detect, estimate likeliness and prevent a key resources saturation. When it is found that a controllable action a_c, or an action dependent on some tuning activities, utilises a resource in a way that brings it close to reach a saturation level, an appropriate remedial control can be introduced to limit the excessive computation, e.g. action termination, vertical scaling, service migration, rescheduling, or an execution redirection (see Sections 2.6 and 9.4).

A typical practice in the ASM field is to calculate a time–series of action execution times to "action execution load". Such a transformation shows the count of actions running concurrently in time. This preprocessing approach often gives much better correlation, see Figure 6.1.

The core property of the action execution times signal transformation is that the peak values are flattened against the time axis, for so long as the execution time indicates, and the signal better reflects the system load, but still can be much different when a resource is highly utilised for longer time. Also high counts of action calls can obfuscate the transformed signal.

In process monitoring, this is a common way of aggregating changing runtime characteristics for better visibility and lower storage footprint of a number of parts of the system under control.

6.2.3 Beyond the Point of Saturation

A critical operating region of the system, in terms of its performance, is reached when a resource is near saturation (the whole scope of the resource is consumed). When this system state is reached the resource begins acting as a bottleneck for the rest of the system. Often though, this is actually the point when energy and amortised hardware are used optimally. There are of course concerns about stability and many other key system operations that have an impact on SLAs (Beloglazov and Buyya, 2010b; Beloglazov et al, 2012), as well as concerns about the level of QoS and performance related issues, such as those related to queuing and batches processing.

Although it is typically available to use another related measurable derivative resource that present the more about effects of saturation, e.g. CPU and load (tasks queue or thread count), disk usage and disk queue, it is still beneficial to decompose such signals since thousands of different action types can contribute to a single resource usage. There is still a high need to observe and process data of such derivative resources because they better explain the effects of saturation and allow measurements of further impacts while base resources show dynamics before the point of saturation when derivative resources typically show flat, zero level, measurements.

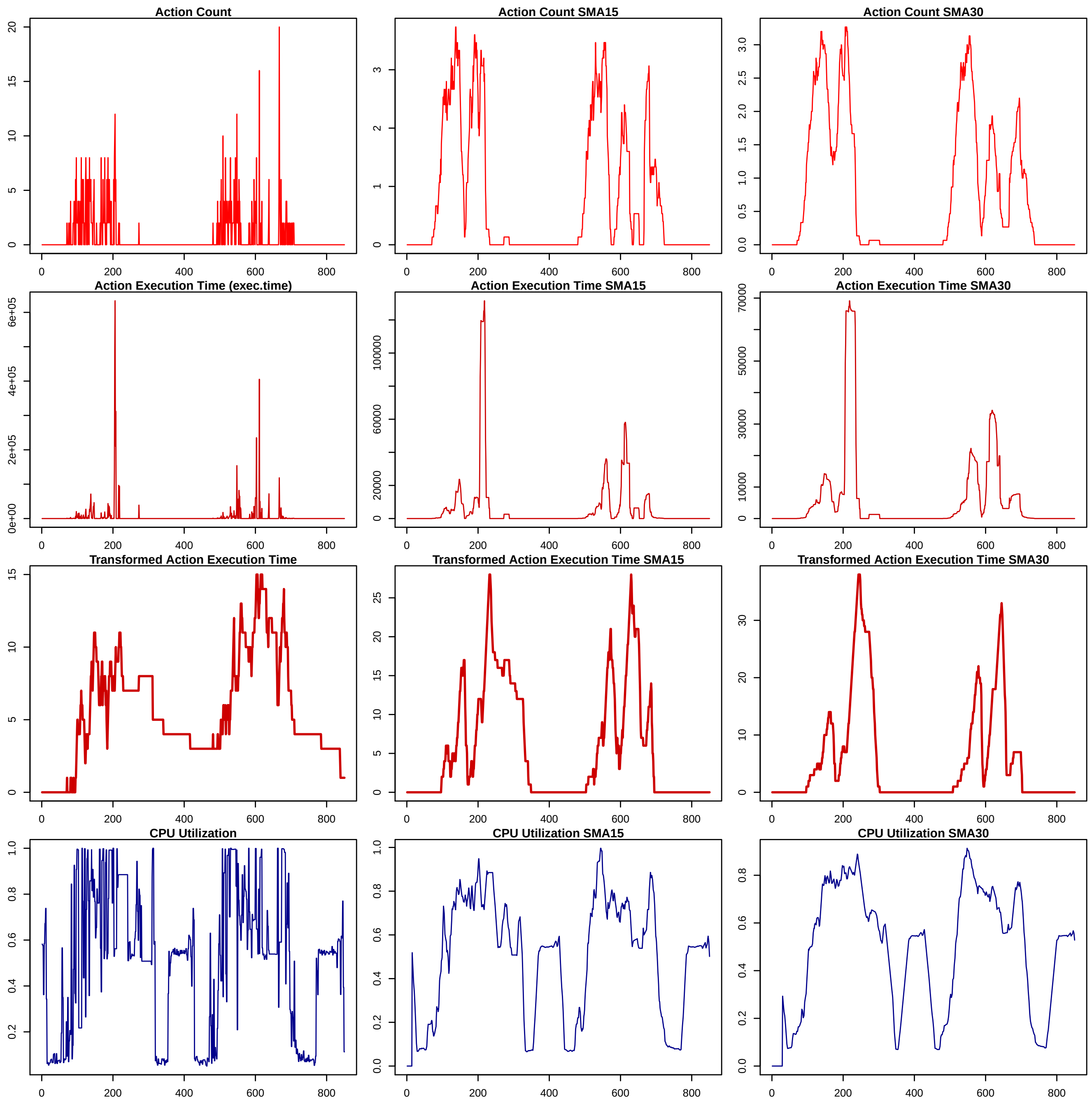


Figure 6.1: Comparison of a sample time–series of CPU–bound action count and execution time in 1 second aggregates. Actions time–series are shown in red whilst resources utilised (CPU) are in blue. The left column shows the time–series of the raw data– no smoothing was used. The middle column contains data processed with SMA that used a 15 secs wide aggregation window. The right column shows an SMA that used a 30 secs wide window. Rows starting from the top show: action counts (light red), action execution time (dark red), "action execution load" as transformed from execution time (dark red), and finally CPU utilisation. It is worth noting the improved similarity between "action execution load" and CPU; this situation is further discussed in Sections 6.3.1 and 6.3. These time-series data have been collected during a real application, not a model–based, experiment.

The approach treats the system as a black– or grey– box searching for relations between actions performance and observed use of resources. Although in most operating systems it is possible to track CPU or IO consumption time per process, or even thread, nevertheless, OS schedulers do not control the operation of applications internals. Hence, building such a model of relations at the application–level can greatly improve application and service level controllers.

This is an area of particular relevance for other communities as well, such as the communities of cloud computing (Emeakaroha et al, 2011; Sironi et al, 2012; Yoo and Kim, 2013; Feng et al, 2012), virtualisation (Weng et al, 2011), and Map Reduce (Ibrahim et al, 2011; Polo et al, 2011), where advanced resource and activity aware adaptive schedulers can be equipped with services activity and resources decomposition methods for better SLA and revenue awareness. In that respect, embedding the proposed ASM-SD method can potentially enhance the performance of adaptive schedulers, especially those working on application level (Bartolini et al, 2012), or of the scheduling policy, extending the predictive modelling of available green energy (Aksanli et al, 2012).

As it will be shown in Section 6.6, experiments provide evidence that the proposed ASM-SD method can be successfully implemented with use of global search algorithmic techniques.

6.3 Signals Decomposition as a Search Problem

The ASM-SD method performs ASM signals decomposition tackling the problem as a form of optimisation. The proposed approach is built on the assumption that the resource utilisation at time point t is a result of actions metrics at this point:

$$r_k(t) = \sum_i m_{ik}(r_k) a_i(t) + n_k \ , \tag{6.1}$$

where r_k is the k-th resource, a_i is the i-th action, m_{ik} is an unknown coefficient of a component, or service, which causes changes in r_k after a_i calls, and n_k is additive noise that can be considered as an error. The presence of the error vector n_k is related with many factors, e.g. noise present in the system that may consist of unwelcome and often hidden components such as software sleeps, calls to other not monitored actions relaying on other resources or 3rd party systems, waits for unmeasured resources, other OS level or software level framework operations, or non–monitorable system activity that utilises system resources. It is normally advised to assume that this vector consists of non–negative values, as the above mentioned situations normally introduce additional execution times against the resources utilised, i.e. resources are used more than monitored action times suggest.

Equations of this form are common in the BSS class of problems (Bell and Sejnowski, 1995; Cichocki et al, 2002) mentioned earlier in Section 6.2.1, and can be reformulated as a system of linear equations of the following form:

$$R = MA + U \tag{6.2}$$

where R, A are vectors of resources and action time–series respectively, $R = [r_1(t), ...]^T$, $A = [a_1(t), ...]^T$, matrix M contains mixing coefficients, and U is the residual vector. That notation would suggest Linear Models Fitting (LMF) to be applied (Chambers, 1992; Wilkinson and Rogers, 1973), (R Core Team, 2014), but the LMF approach tackles the optimisation problem in a symmetric way, where equation may be equally under– or over– determined, and is difficult to adjust it considering all additional calculations that are specific to the ASM area, e.g. boundary limits that are *asymmetric* due to causality direction, and penalties for rare execution or low total execution time, see Section 6.3.1. Thus more details on the optimisation function implementation are needed and, effectively, more flexible optimisation methods have to be tested.

As mentioned above, in the proposed approach the unmixing process is formed as a search problem that minimises a single–objective. Considering that there are many non–linearities, which have an impact on action execution times a and resources utilised r, as outlined in Section 6.2, the optimisation of $f(x) : \mathbb{R}^n \to \mathbb{R}$ can be considered as a general Non–linear programming (NLP) as an optimisation problem (Ruszczyński, 2006), defined as follows:

$$\arg\min \left(r_k(t) - \sum_i \Big(m_{ik}(r_k) a_i(t) \Big) \right) = n_k \ , \tag{6.3}$$

where m_{ik} denotes unknown entries of the mixing matrix M of size $\overline{\overline{r}} \times \overline{\overline{a}}$, for which the equation in brackets attains its minimal value. M is transformed to a solution point s in the search space $S \subset \mathbb{R}_+^{\overline{\overline{a}} \cdot \overline{\overline{r}}}$.

There are, however, several constraints involved: (i) signals are by definition non-negative $a \geq 0, r \geq 0$ and normalised, and (ii) potential negative values of n_k are subject to additional penalty in the optimisation function implementation (discussed further in Section 6.3.1), based on the assumption that actions form the inputs and resources are the system outputs, which are impacted by the results of the actions. Thus, a possible effect cannot be prior to the activity that generated it.

Assuming that there is an adequately rich set of activity measurements A gathered to explain the resources utilised R, n_k as a residual, or fitting error, of unobservable signals is minimal. However, if the solution point s, as a vector of specific values m_{ik} of the mixing matrix M results in a substantially large residual error, then the k-th resource signal is not matching the observed activity, and therefore either : (i) the monitored activity does not adequately explain the resources usage (many important actions have not been monitored), or (ii) the resource usage does not reflect the activity signals gathered (e.g. this is often present in memory consumption cases).

Eq. 6.4, below, is an extension of Eq. 6.3 that factors in the information about actions execution times span and match against a resource, considering the time spent on the execution up to the current time instance t_1 when the rest of the collected samples are aggregated:

$$\arg\min \left(r_k(t) - \sum_i \left(m_{ik}(r_k) \int_0^{t_1} a_i(t)\, \mathrm{d}t \right) \right) = n_k \ . \tag{6.4}$$

The approach of Eq. 6.3 compares similarities of action executions times, but not direct times spent on executing, as in Eq. 6.4. Thus the approach of Eq. 6.3 can be used mainly for very short executions times, or where sampling time or aggregation is wider than most of actions executions, that is quite common in ASM databases. The approach of Eq. 6.4 could be used rather for much longer actions (e.g. batch jobs) that often have significant impact on systems performance and consist of different phases. Very long executing actions could be organised in sequences of monitored sub-actions considered under the assumptions of Eq. 6.3. Moreover, we assume that when action execution times are short, or the span of the time–series aggregate used is wider than most of actions executions, then Eq. 6.4 approximates Eq. 6.3.

In other words the general approach presented above is based on similarity and shapes matching between different domains: activities and resources usage, where units are different. Thus it should rather not consider values comparison directly. Therefore all signals should be normalised before any further processing is done, and the equation $R = MA+U$ is considered to be a good approximation of the system.

Still in situations where activity levels (execution counts or times) are far different between actions in A, the precision can be enhanced when such information is factored in as a weighted vector of action execution times. Actions less often used (thus having lower total execution times) should be considered as having less impact on resources and effectively their error should be augmented as part of another tier of penalty setting.

An iterative optimisation process searches for suitable coefficients of the mixing matrix M. Solving the problem by a greedy approach to calculate the objective function for every designed dimension n of k–th values would be $O(n^k)$, i.e. such a brute–force search may be acceptable for low–dimensional cases only. As mentioned earlier the search space S dimensionality is a product of $\overline{\overline{A}} \cdot \overline{\overline{R}}$, where typically in a real system we can have tens to hundreds of resources and hundreds to thousands of actions.

Each iteration of the method focuses on all resources R. If an action is found during its execution to use time of many different resources, then for a particular action implementation the usage should be consistent over time. Consequently it has been assumed that m_{ik} is constant for the i–th action a and the k–th resource r at iteration t, see Eq. 6.3. Depending on the application context, this assumption may be subject to revision after significant code, parameters or input data changes, or when other factors, like intrusive load or control actions, change the normal system responses behaviour. Although this situation is not considered in this chapter, it is worthy to add a brief comment that a way to potentially overcome this issue is to consider threading actions running with different parameters or input data, or after changes in the release of the software as different – this however needs careful consideration as it may lead the count of dimensions to explode, which in turn could limit the method used.

Lastly, it is worth mentioning the possibility of lowering the count of actions by aggregating actions signals with similar load–functional characteristics that give comparable result on the usage of resources. Then, signal unmixing analysis can be used to confirm any similar relations between actions of different types and resources impact.

6.3.1 The ASM-SD Methodology

The proposed methodology includes the following signal processing steps.

Step 1. Signals Normalisation

Time–series are linearly scaled, so their amplitudes are standardised to the range of $[0, 1]$. This is required because in the practice actions and various resources signals are expressed in different units, and values gathered are difficult to compare directly. Emphasis here is on the signals shapes under consideration rather than the specific values gathered and aims at evaluating all dimensions (e.g. resources, actions) equally.

Step 2. Signals Denoising

System administrators are used to interpret scalar values that represent various system dimensions, like resources utilisation and systems actions (aggregated values), and associate their changes in time. This is a simple but effective way of aggregating data and lowering the scale of resolution of the metrics time–series to be stored. Most of the monitoring tools actively gathering data about systems actions collect counts and execution times continuously through software components (collectors), which are weaved into application runtime (Grinshpan, 2012; Haines, 2006). Such raw metrics are stored in various internal storage for further use by metrics collection tools and need to be aggregated for presentation and further times–series processing purposes. SMA (Pyle, 1999) as a low pass filter has been used[3]. This

[3]As a rule of thumb a fair level of denoising is achieved when SMA aggregates values following the width of the longest action.

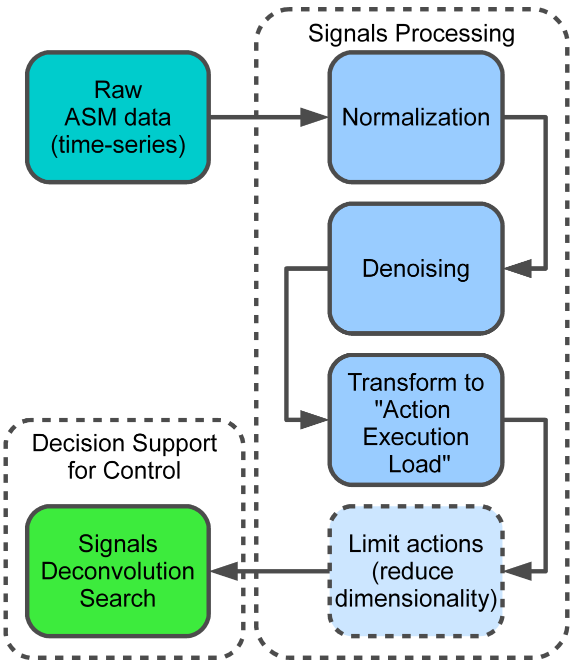

Signals Processing
Raw ASM data (time-series)
Normalization
Denoising
Transform to "Action Execution Load"
Decision Support for Control
Signals Deconvolution Search
Limit actions (reduce dimensionality)

Figure 6.2: ASM system incorporating the ASM-SD processing methodology.

form of signals denoising allows enhancing the level of similarity between the input action and the resources signals which play the role of the system's response, see Figure 6.1.

Step 3. Transformation of the Executions Load

Next, an "action execution load" transformation is executed for each action signal. The essence of the transformation is to change execution time measurements collected at the end of the action call[4] into a time–series that represents the count of actions being executed at a given point in time. Such a transformation smooths signals that may contain many peaks and helps enhancing the level of similarity further- see comparison presented in Figure 6.1.

Figure 6.2 illustrates all stages of the ASM-SD methodology. It includes an additional step of dimensionality reduction which filters out system actions that have lower impact. In ASM industry practice, the number of action signals A greatly exceeds the number of resources R. Thus reducing the actions dimensionality can improve the ASM-SD execution time. Actions' impact can be measured by ranking selected action types– e.g. those which have higher cumulative execution times– or highly called actions[5], or, even in certain cases, by using a weighted product of those two ranking criteria.

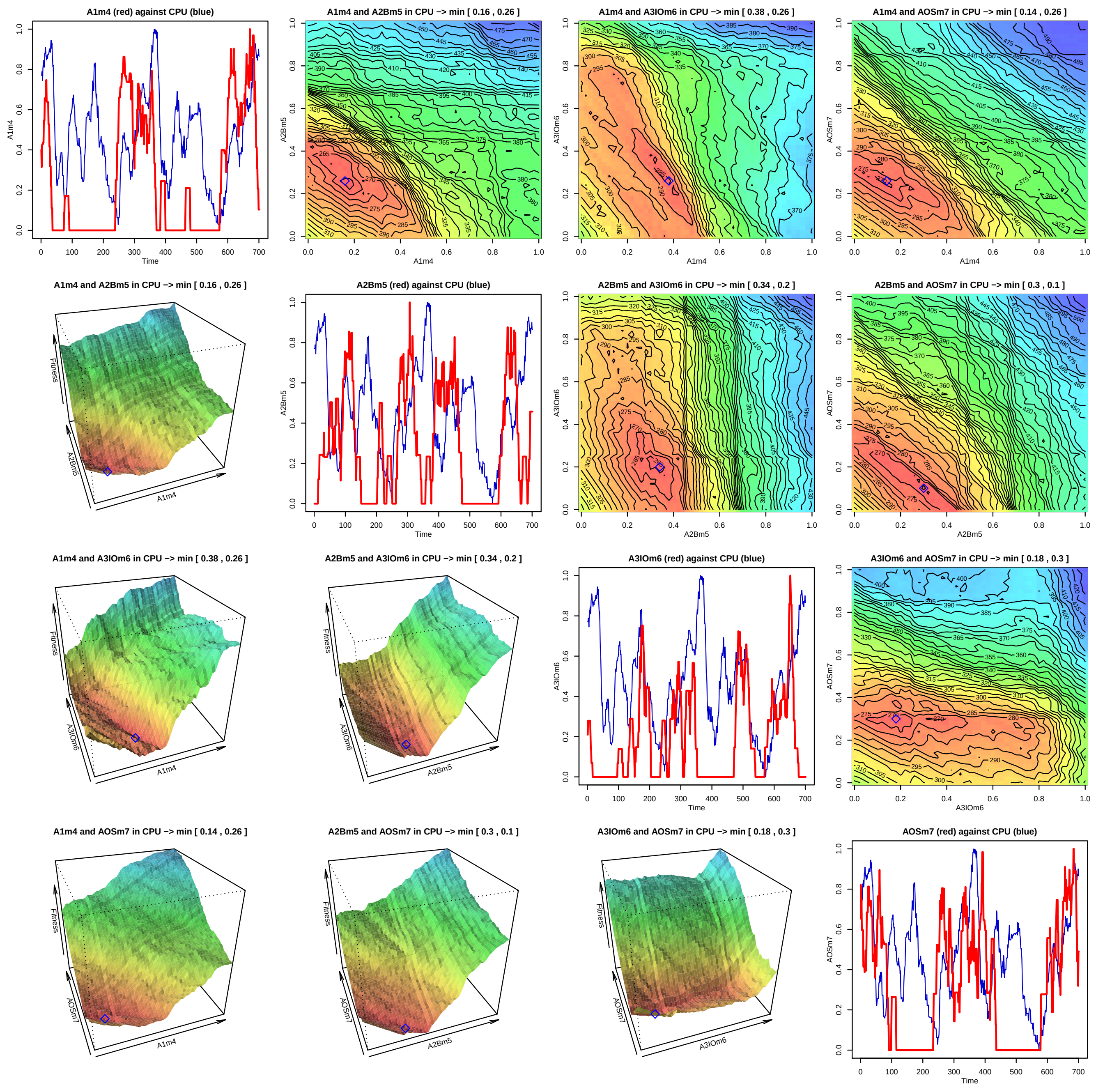


Figure 6.3: The search space of an ASM system having four actions and a single resource being utilised. The figure illustrates the search space with respect to each pair of actions.The original action signals are shown in the plots placed along the main diagonal. The 3D surfaces and the corresponding 2D contour plots depict the fitness function with respect to a pair of isolated action types. The 3D surface plots (at the lower left side of the diagonal) provide a good view of the shape of the fitness function along two dimensions, whilst the corresponding contour plots (at the upper right side of the diagonal) provide a better view of the various local extrema (areas of closed isolines). The lowest point in each of these projections indicates a minimiser and has been marked with a blue diamond.

6.3.2 The ASM Search Space

Seeking appropriate coefficients of the mixing matrix in Eq. 6.3 requires searching a multidimensional space. Below, a simple ASM system containing four actions and a single resource, where the intensity of the actions and usage of the resource are different per action type (see Figure 6.3), is used as an example to illustrate features that are common to ASM systems. The search is conducted in a 4–dimensional normalised hypercube, and denoising of all signals is achieved through SMA filtering that follows a width of 15 secs– see Step 2 in Section 6.3.1.

In Figure 6.3, different slices of the search space are shown by projecting the search surface along two dimensions, which represent pairs of actions against a single utilised resource. This visualisation could help gaining some insight about the characteristics of the optimisation problem. Potential minimisers in these 3D and 2D plots are marked with a blue diamond. Obviously, these are only slices and the full search space explored by the ASM-SD method has higher dimensionality, so the coordinates of the actual problem minimisers differ from the ones shown in these figures. In fact, the ASM-SD search space appears to be multi–modal. This is also illustrated in the 2D contour plots, which show that there are neighbourhoods of local minima (see the plots at the upper right side of the diagonal in Figure 6.3) surrounded by the contour lines. The space is complex with irregular gradient values, from very narrow barrier related slopes (discussed later), to flat regions with many widespread local minima, and valleys– see the 3D plots at the lower left side of the diagonal in Figure 6.3.

Observing the collected data of the experiments reported next, it is clear to notice that the region surrounding a good minimiser is flat and often wide. Moreover, often the neighbourhood of the best solution point found contains many shallow local minima that are not very different in terms of function values to the "global" minimum. Such a plateau areas are caused by many actions of similar resource bound characteristics, having similar impact on the resources footprint, so in simple terms they are hard to be distinguished. They create unfavourable conditions in terms of quality of the ASM-SD method.

As mentioned earlier, in Section 6.3, negative values of residuals of Eq. 6.3 should be avoided, and to this end *bouncing boundaries* are used in the optimisation function implementation. This is done through the introduction of penalties, whose values are passed to the methods as a parameter, that are added to the error for every negative coefficient and residual value (Wright and Nocedal, 1999). Such an approach forms barriers for matching signals whose total impact exceeds resource utilisation. Any candidate solution that violates the problem's assumptions/constraints is removed from the feasible region by assigning a penalty to it.

[4] In engineering practice in most of the cases action executions are assigned at the time the action started.

[5] The execution count can very often cause software or hardware resources utilisation regardless of the execution time.

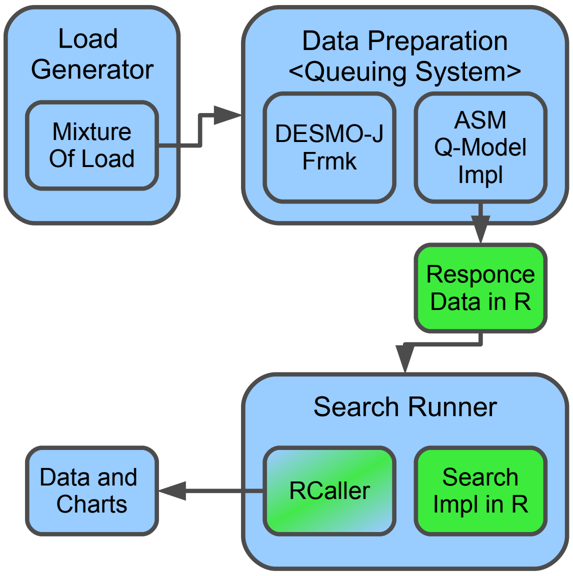


Figure 6.4: The ASM system test framework for Signals Decomposition methods.

6.4 Testbed Design and Experiments

Due to the multi–modal and high–dimensional nature of the ASM signals problem space this chapter is focusing on evolutionary optimisation methods to solve the ASM-SD problem, as this approach appears to be eminently suitable due to the characteristics of the search space, as discussed in Section 6.3.

The testbed framework of Figure 6.4 has been implemented allowing exploration of the potential use of the proposed approach in the industry practice. It provides an ASM environment where iterative experimentation can be conducted, collecting performance data for the various methods tested and generating a rich set of ASM data for testing population–based methods under various loads several times.

Previous research in the ASM field (Sikora and Magoulas, 2013b) showed that the minimal time for a load test – rich enough in terms of different load pattern changes – is around 1 to 2 hours long. To alleviate the problem of conducting long simulation runs queuing model simulations were used. This is a well known simulation modelling approach, as discrete event model simulation in the form of queuing systems has a long history of applications (Lazowska et al, 1984; Banks et al, 2000; Harchol-Balter, 2013).

To this end, a flexible toolset for time event simulation, the Java–based framework for discrete event simulation, Discrete–Event Simulation Modelling in Java (DESMO-J) (Page et al, 2005; Gehlsen and Page, 2001; Lechler et al, 2014), has been used as it utilises plain Java code for the model and the queue networks framework, which allows to integrate it with the rest of the proposed framework[6].

[6]After applying a simulation–based data generator, the speed increased by a factor of approximately 600 to 800 (that practically allows running 24hrs load simulation in about 2 to 3 minutes, depending on count of load sources).

Although the author of the thesis reviewed several existing event–based frameworks for systems modelling and workload analysis, such as: GridSim toolkit (Buyya and Murshed, 2002), GroudSim (Ostermann et al, 2011), Java Modeling Tools (Bertoli et al, 2009, 2006), Palladio (Becker et al, 2009) and CloudSim (Buyya et al, 2009a; Calheiros et al, 2011), he found that for the sake of simplicity, insight and flexibility of direct elements instrumentation for metrics gathering, the framework based on DESMO-J is more appropriate in this case.

The software framework provide APIs for forming custom topologies of queue networks, utilising many different actions, adding custom counters for SLAs and put customised load in a flexible manner. It is a Pure–Old Java Objects (POJO) approach, so it is ready to be equipped with a controller actuators to simulate environments under service control to test operation and performance of the computer system during the design phase, or be applied as a model base for a weaved-in controller to be instrumented directly in the application code to manage and regulate the application behaviour in operation.

Figure 6.4 presents the design of the test–bed that consists of: *Load Generator* that defines execution parameters for the modelled system; Data Preparation block that provides the enterprise system model that generates ASM metrics, which are required to prepare a dataset for the *Search Runner* that uses various evolutionary computing methods.

All empirical evaluations were done with the use of `R` scripting, which can be directly called from a Java–hosted ASM controller. The authors used the approach of RCaller that was proposed and described by Satman (2014). Experiments utilised RCaller v2.1.1 implemented by Satman (2013) as an `R` integration library for Java applications.

6.4.1 Load Generator

The *Load Generator* is a software component that is responsible for creating input data for ASM model deployed in Data Preparation, see Figure 6.4. Identifying bottlenecks, overused or saturated components/resources and entire performance of the modelled system is a key functional prerequisite of the test–bed design, so that those interesting from systems performance perspective operational areas are covered in the ASM-SD experiments. In order to test the insights of system dynamics the same model can be a subject of runs with various load conditions. *Load Generator* provides API that gives necessary flexibility of defining the load mixture by specifying arrival and execution rates and distributions of each of the action types individually. Action type definition contains a vector of load pattern that specifies how the arrival rates are distributed in time of a simulation. Each action must be also equipped with measures showing proportions of available resources utilisation.

Such an approach gives a very concise yet flexible way of modelling a load coming to the system from many sources (users interface, web services and scheduled batch jobs) using the system concurrently. The test–bed can execute simulation from very simple signals mixture to very complex, multi-action workloads.

The example below presents a concrete scenario where actions play the role of input to the queue model. Normally to test the ASM deconvolution process of Eq. 6.3 under a mixture of signals, a set of many concurrently executing actions of many types needs to be run.

The example case bases on a single action type definition that is rather CPU–bound, but also utilises Disk in only 5% of the execution time, with the following execution distribution parameters: exponential distribution with given arrival rate: $Exp(\lambda = 10)$, action execution time defined with use of normal distribution: $\mathcal{N}(\mu = 5, \sigma^2 = 0.1)$, and a variable load defined as a call probability pattern given by a following sequence $p =$ [0, 0.2, 0.2, 0.2, 0.2, 0.2, 0.1, 0.0, 0.0, 0.1, 0.4, 0.4, 0.6, 0.2, 0.1, 0.0, 0.0] repeated 3 times in a time of the single experiment[7]. For better comparison, please see the corresponding Java code snippet below:

```java
new ActionType("A3I02",
  new LoadPattern(      // arrival
    new LoadDistExponential("DistrExp", 10),
      new double[]{     // intensity
        0.0, 0.2, 0.2, 0.2, 0.2, 0.2, 0.1, 0.0, 0.0,
        0.0, 0.1, 0.4, 0.4, 0.6, 0.2, 0.1, 0.0, 0.0},  3),
  new LoadDistNormal(   // execution
          "DistrNorm", 1, 0.1),
  new ResourcesUsage[]{ // resources use
    new ResourcesUsage(ProcessingType.CPU,  0.95),
    new ResourcesUsage(ProcessingType.Disk, 0.05)},
  new SLA());
```

Figure 6.6a shows an example response and unmixed action signals using the ASM-SD method, where one of the actions signals is a a result of the above definition.

6.4.2 Model and ASM Data Preparation

The computing system model implementation (Figure 6.5), applies Processor Time Sharing approach, where each jobs is sliced to atomic subtasks that as per action type characteristics (functionality and load) have different resources distributed (Lazowska et al, 1984; Harchol-Balter, 2013).

As mentioned earlier, DESMO-J (Page et al, 2005) has been found as the best suiting the needs of discrete event simulation framework for this study. A more detailed frameworks comparison has been investigated by Göbel et al (2013).

[7]The load pattern gives a precise way of configuring variability in expected intensity of the frequency of incoming calls for the given action. This parameter is very helpful to define a test highlighting cases such as: actions interference using the same resource, or temporarily higher load to observe effects of spikes in resources consumption, or short reoccurring load changes to analyse a 'delay' factor impacting strength of ASM-SD.

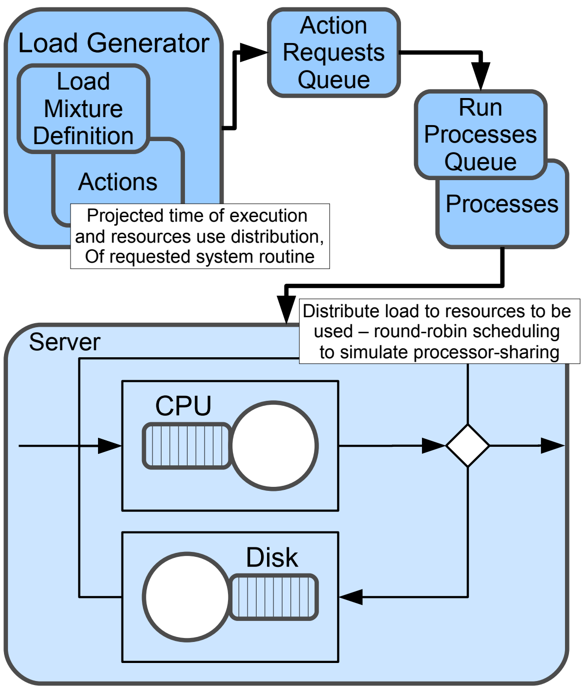

Load Generator
Load Mixture Definition
Actions
Projected time of execution and resources use distribution, Of requested system routine
Action Requests Queue
Run Processes Queue
Processes
Distribute load to resources to be used – round-robin scheduling to simulate processor-sharing
Server
CPU
Disk

Figure 6.5: *Load Generator* and ASM system model for signals generation.

For the benefit of this study the framework provides implementations of CPU and IO load simulations, based on those resources queues. There is no waiting line and all jobs receive an equal proportion of the service capacity, that is essential to model a simple operating system dispatcher, which is a building block of each computing system. The synthetically generated data by the model of such a queuing system design match the metrics collected during real load runs. Rerunning the simulation alllws reproducing hours of load test in seconds.

It is worth mentioning that more advanced computing systems, including distributed components (network delays), multi–core (lower load queues as per simple processor sharing) or with farm of servers covered with load–balancers (each of them are multiprocessor with a disk controller) are feasible to be implemented in the adopted framework. Nevertheless the extensions are not needed to demonstrate the ASM-SD method – as in engineering practice most of the activity and resources metrics can be isolated as per machine or a sub-system, where they can be gathered fairly easy, and consequently the problem is reduced to a single server again. Of course adding resources (network delays, wait to connection pools, etc.) to the picture would multiply the search space to be explored (see Section 6.3).

6.4.3 Search Runner

As mentioned earlier in Section 6.3.2, due to the multiple minima and the high–dimensional nature of the ASM-SD search space, the experimental study focused on comparison of different population–based methods that are considered more capable of exploring globally this kind of search spaces.

Several series of experiments were performed on synthetic datasets to confirm the effectiveness of the approach. Three different population–based method implementation available in `R` are presented in this chapter: (a) PSO (Clerc et al, 2011; Bendtsen, 2012)[8], (b) GAoptim (Tenorio and Tenorio, 2013) [9], and (c) DEoptim (Ardia et al, 2011; Mullen et al, 2011) [10].

The *Search Runner* component provides a software framework abstracting access to different APIs of the search methods `R` implementations. As the `R` optimisation implementations are based on the approach taken by `optim {stats}`, PSO, GAoptim, and DEoptim differ in input parameters, it was necessary to wrap the access with a single access layer giving input parameters abstraction.

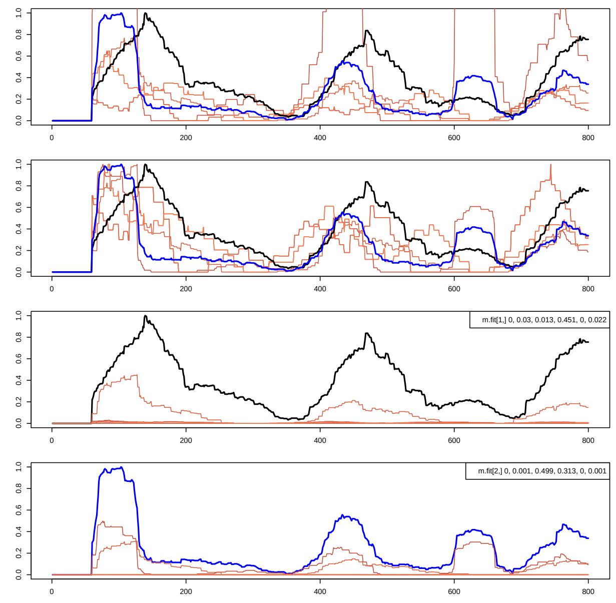
In order to compare the speed of convergence, precision, robustness, and general performance of the different methods we use as our main performance criteria the execution time and the residual error (Eq. 6.3). More detailed description of the evolutionary methods is provided in Section 6.5.

Interested readers can find a performance comparison of non–population–based general–purpose optimisation methods in (Sikora and Magoulas, 2014b), where Simplex (Nelder and Mead, 1965) and Simulated Annealing (SANN) (Bélisle, 1992) were tested. Preliminary experiments in Sikora and Magoulas (2014b) found that population–based methods find good ASM-SD solutions earlier, and are generally resilient to the problem of local minima.

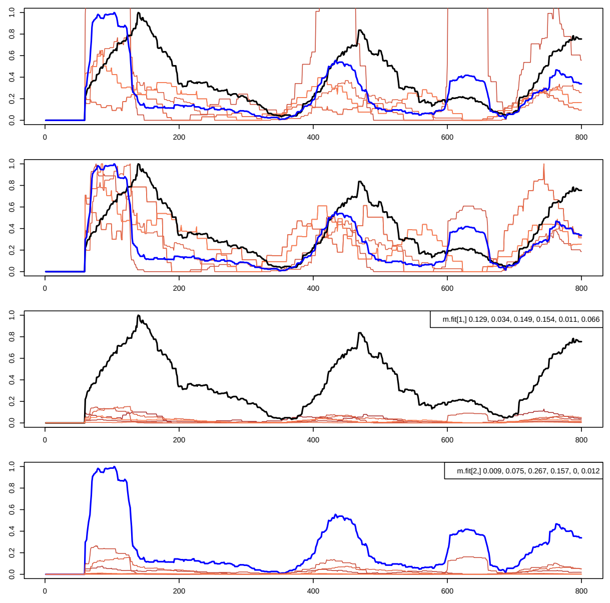
[8]A PSO implementation consistent with the standard PSO 2007/2011 of Maurice Clerc et al., version 1.0.3 (2012-09-02).

[9]Genetic Algorithm optimisation package for real-based and permutation-based problems, version 1.1 (2013-03-24).

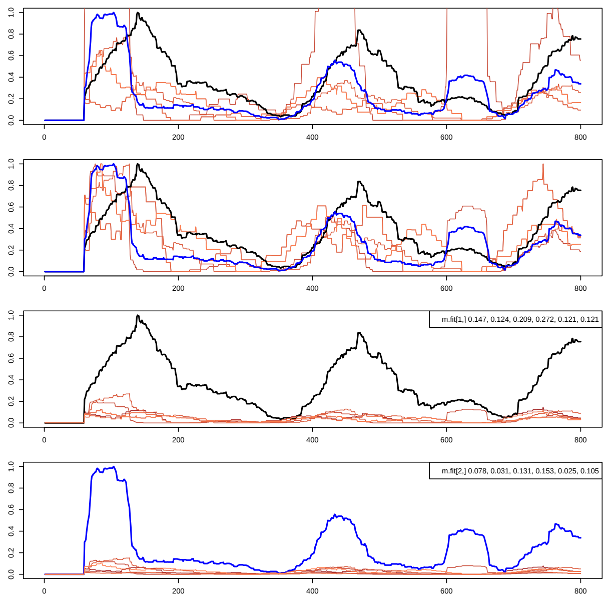
[10]DEoptim package implements the differential evolution algorithm for global optimisation of a real-valued function of a real-valued parameter vector, version 2.2-2 (2014-12-17).



(a) PSO



(b) DE



(c) GA

Figure 6.6: Comparison of deconvolution process of three different optimisation methods. The time series were captured by *Search Runner* debugging during one of the runs, where 6 different concurrently executing actions are present. Most of the actions are mainly CPU–bound but they also use Disk with different proportions of CPU and Disk usage, where one of the actions is rather Disk–bound. Resources are denoted by black and blue thick lines, whilst actions are shown as thin lines using different tones of red. Each of the groups shows from top: original action values signals, all normalised, unmixed actions impacting CPU and Disk respectively.

6.5 Search Methods Performance and Parameters Tuning

This study is not tackling the meta–optimisation or hyperparameter optimisation issue, where another overlaid optimiser for parameter calibration (tuning) is being used (Grefenstette, 1986; Pedersen, 2010a,b). The purpose of this work was to explore the different methods and validate their applicability; thus much work has been done on visualisation comparison of the optimisation characteristics from many angles.

This section provides insights on the examined search methods performance when tested under different experimental conditions and tuning modes in an ASM environment. Series of experiments have been executed checking different load situations (variable arrival rates and difference load patters). The search space is complex with multiple local minima. Thus, we selected meta–heuristic methods such as Particle Swarm Optimisation (PSO) (Kennedy and Eberhart, 1995; Poli et al, 2007), and Genetic Algorithm (GA) (Goldberg et al, 1989; Mitchell, 1998; Michalewicz, 1996) that typically exhibit good performance in such a search conditions.

The experiments focused on `R` implementations only, giving a good comparison ground for selected population–based optimisation. Common for the class of search methods parameters like maximum number of iterations or population size have been extensively tested.

The experimentation with use of earlier mentioned implementations (PSO, GAoptim and DEoptim) have been split to two groups swarm–based and genetic search methods that reuse similar approaches and share common properties that have been presented in Sections 6.5.1 and 6.5.2 respectively.

Due to large number of individual simulation runs conducted applied visualisation of multivariate data–sets have been structured with focus on entire distributions, rather than statistically aggregated values for groups of data, thus optimisation methods comparisons use violin plot (Hintze and Nelson, 1998), that better shows individual experiments than box–and–whiskers plot (McGill et al, 1978). To highlight the directions of the changes Local Polynomial Regression Fitting (loess) trend line has been used (Cleveland, 1979; Cleveland and Devlin, 1988; Cleveland et al, 1992). To generate the figures `R {graphics}` and ggplot2 (Wickham, 2009) plotting systems ware used.

6.5.1 Swarm–based methods

In this section a single metaheuristic of PSO has been presented, that is classified under a broad term of Swarm Intelligence, where a population of simple agents interacting locally forming a collective behaviour, in decentralised fashion, contribute to finding the best

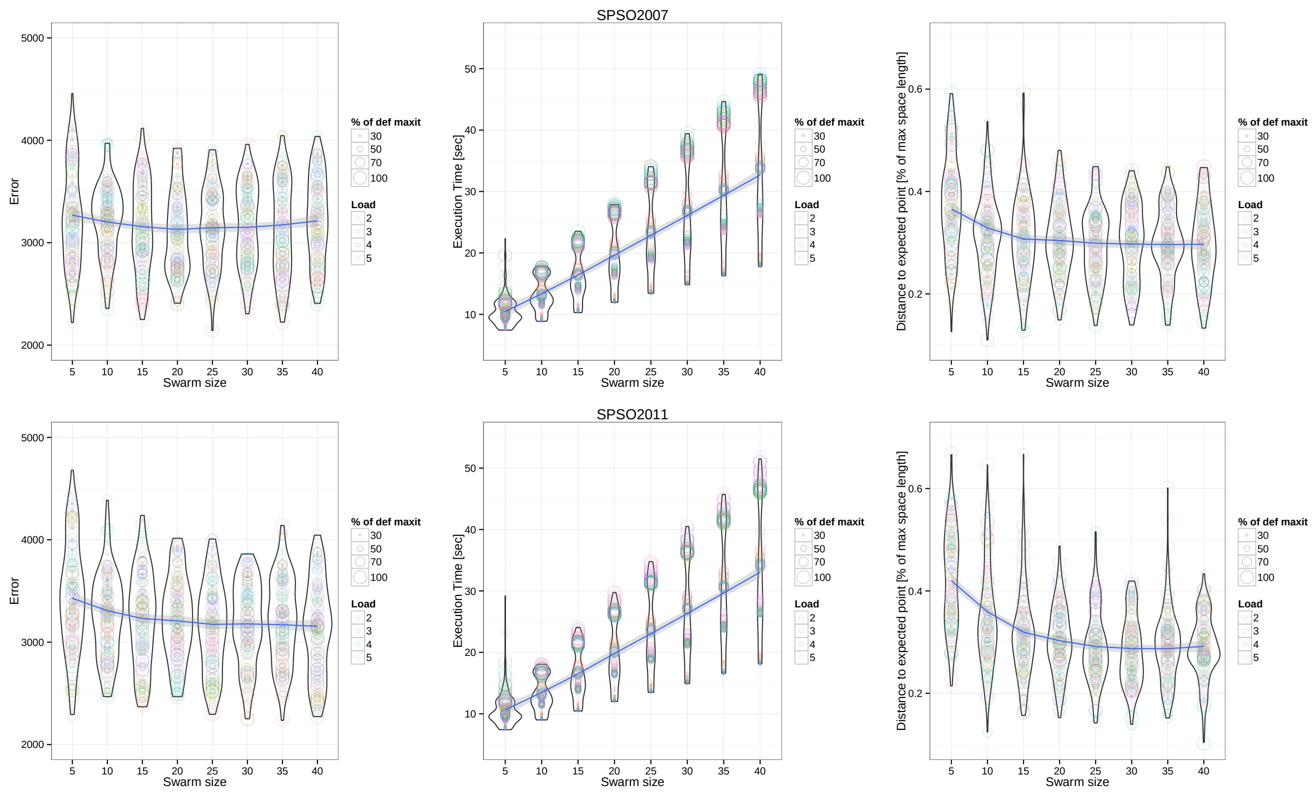


Figure 6.7: Comparison of 2007 and 2011 variants of SPSO.

solution point of the search space (Clerc, 2010).

PSO (Clerc and Kennedy, 2002; Clerc et al, 2011) implementation is consistent with the standard PSO 2007/2011 of Maurice Clerc et al. (Bendtsen, 2012), version 1.0.3 (2012-09-02).

Figure 6.7 shows a comparison of both variants available in the R library implementation tested. Please note that the default swarm size values (population) parameter are quite different between both PSO methods; SPSO2007 defaults to $\lfloor 10 + 2 * \sqrt{\overline{\overline{A}} * \overline{\overline{R}}} \rfloor$, while SPSO2011 use 40. This has been considered in figures showing comparison in function of population size that is always relative (in %) to the default value.

Comparison of SPSO and 2011 has been done for default 1000 and 500 max iterations (maxit) in 8 dimensional space deconvolution problem, see Figure 6.7. Results confirm that performance of and SPSO2007 and 2011 is very similar.

Experiments show that even quite low swarm sizes, e.g. 10–15 swarm size, give good solutions.

6.5.2 Genetic methods

This group of search methods bases on a population of candidate solutions that during an iterative process is evolving toward better solution points of the fitness function (in the search space). Each candidate solution, often called individual, is defined as a set of genotype, which is a basis for selection, crossover and mutation. The population of individuals in each iteration is called a generation (Michalewicz, 1996).

Due to similar parameters and general approach two methods Genetic Algorithm and Differential Evolution have been tested in this section.

Genetic Algorithm (GA)

Genetic Algorithm approach in implementation of `GAReal {GAoptim}`, version 1.1 (published 2013-03-24) (Tenorio and Tenorio, 2013) as an optimisation package for real-based and permutation-based problems has been tested.

Due to complex implementation of custom crossover, selection and mutation operators the default functions have been used. Thus, the `GAReal` function for real-based optimisation has been tested under following conditions. *Selection*: the default option performs a fitness proportionate selection, so that the fittest individuals will have greater chances of being selected. *Crossover*: the blend option was used that performs a linear combination of the individuals chromosomes, and so it introduces new information into the resulting offspring, with the crossover rate equal to 0.9 (comparison in Figure 6.8) as probability of two individuals effectively exchanging genotype. *Mutation*: the implementation uniformly samples given a mutation rate (default 0.01), multiplied by population and present dimensions. Mutation points along the population matrix, where each sampled locus is replaced by a random-uniform number between 0 and 1. *Elite rate*: default value of 0.4 was used as the ratio of the best–fitted individuals amongst the whole population that are automatically selected for the next generation.

GA has been found as a very fast method under high dimensional cases, over 50 dimensions, as shown in Figure 6.14.

Differential Evolution (DE)

Differential Evolution (DE) introduced by Storn and Price (1997) has been applied to many disciplines. ASM-SD experiments used `DEoptim {DEoptim}`, version 2.2-2 (published 2014-12-17) (Ardia et al, 2014) (Ardia et al, 2011) (Mullen et al, 2011).

The choice of DE parameters; population members count NP, crossover probability CR and differential weighting factor F can have a large impact on optimisation performance Storn and Price (1997). Therefore selecting the DE parameters that provide a good performance has been thoroughly researched.

Nevertheless our experiments show that in the ASM-SD problem changes in default parameters have limited value on the method performance and therefore default values are

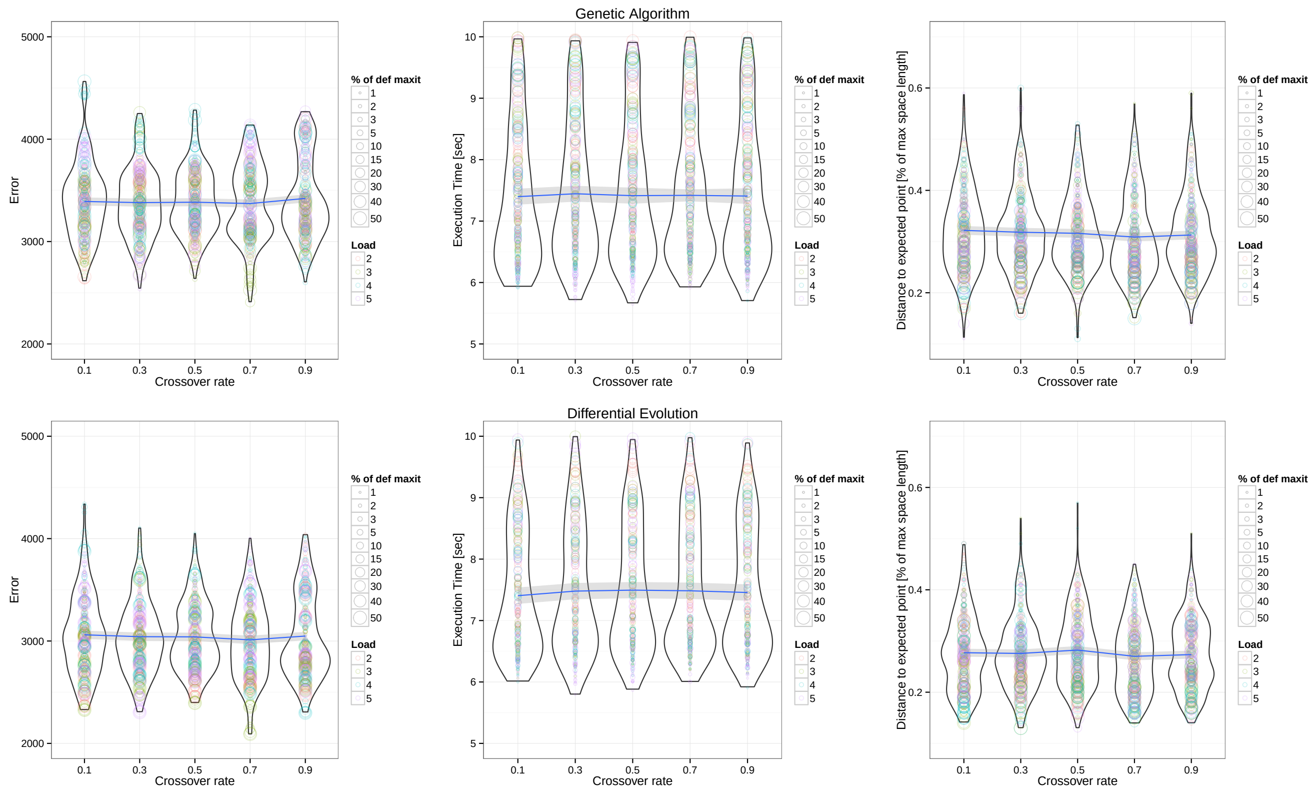


Figure 6.8: Comparison of Crossover parameter of GA and DE using low–dimensional ASM-SD problem set.

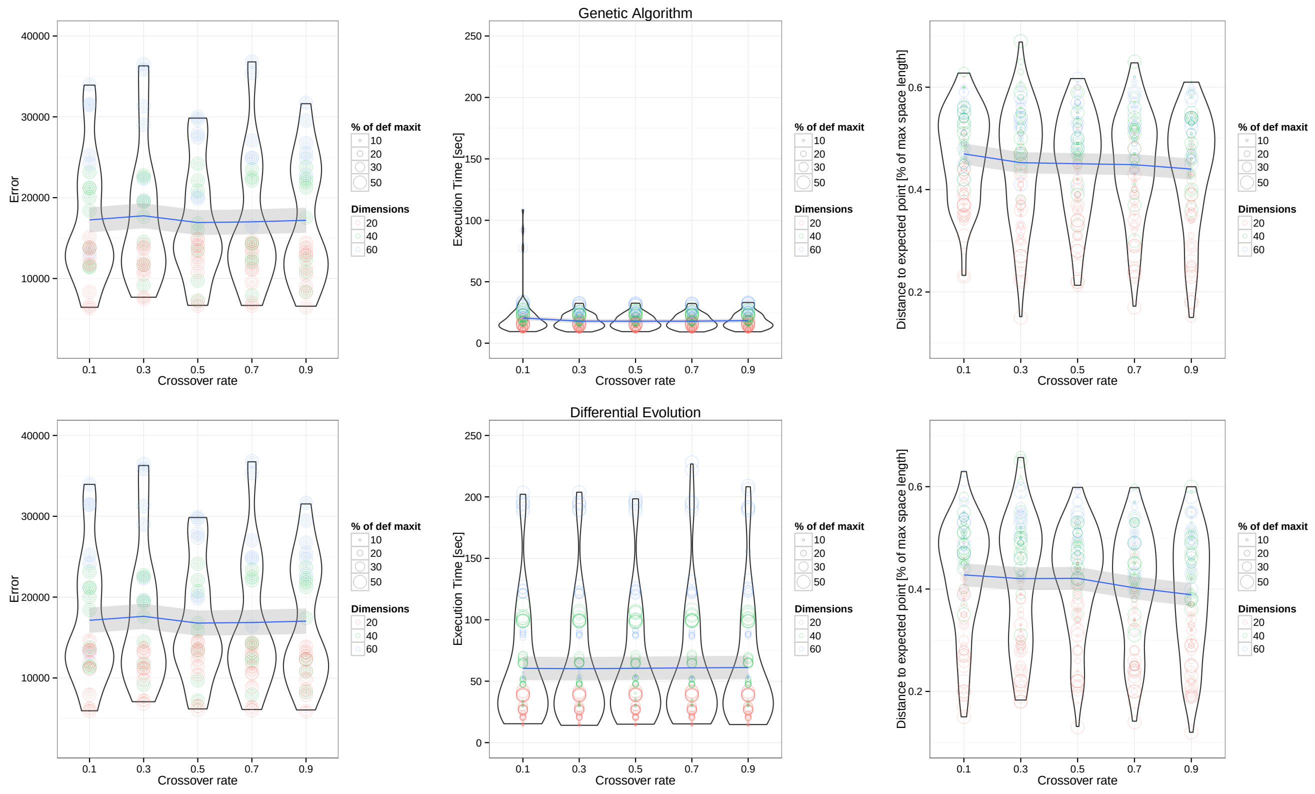


Figure 6.9: Comparison of Crossover parameter of GA and DE using high–dimensional ASM-SD problem set.

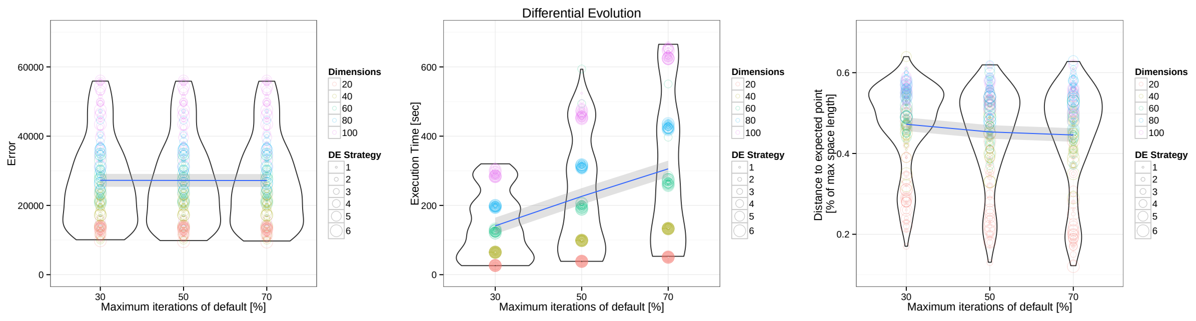


Figure 6.10: Differential Evolution performance as a function of the percent (%) of the default maximum iterations (maxit parameter) needed to converge.

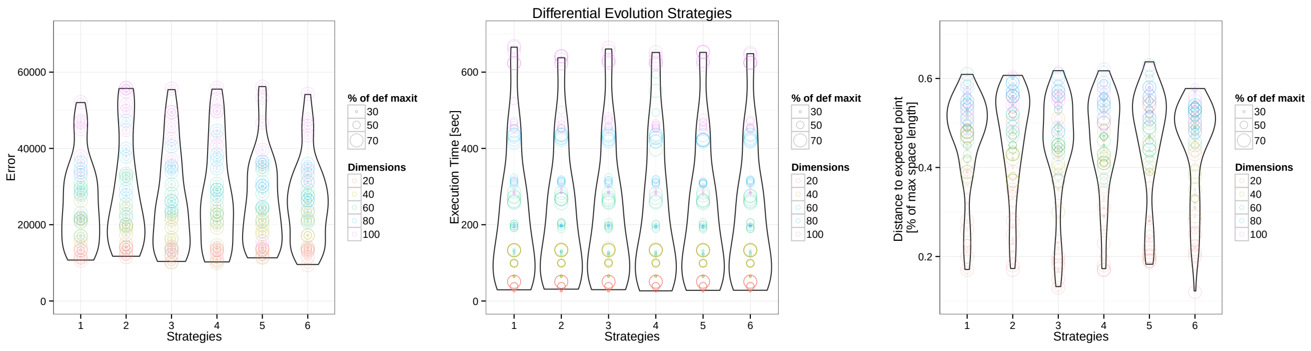


Figure 6.11: Comparison of Differential Evolution strategies, under different dimensions and maximum iterations conditions.

suggested. Comparison of six DE strategies available has been presented in Figure 6.10, crossover probability values review is in Figure 6.8 and population members count is illustrated in Figure 6.16. DEoptim performance as a function of maximum set iterations (maxit) is shown in Figure 6.10, 6.12 and 6.11.

DE finds solution points with much lower error values than GA, with very similar execution times, in low–dimensional cases (see Figure 6.8). However, in high–dimensional spaces, execution times of DE are much higher that the GA's (see Figure 6.9).

This method has been found as with the longest execution times especially in highly–dimensional cases, but with quite low Error values (Figure 6.14). Thus, DE is the most sensible method for low dimensions count (Figure 6.15).

All DE strategies applied to ASM-SD problem give very similar results. All of them give very similar error and execution time characteristics. Figures 6.11–6.12 show a comparison of six DE strategies (the classic DE strategy and five variants) under different load, maximum iteration and dimensions involved conditions.

The best combination for a general ASM-SD use of differential weighting F and crossover CR was found for a pair: F = 0.1 and CR = 0.7; see Figure 6.13.

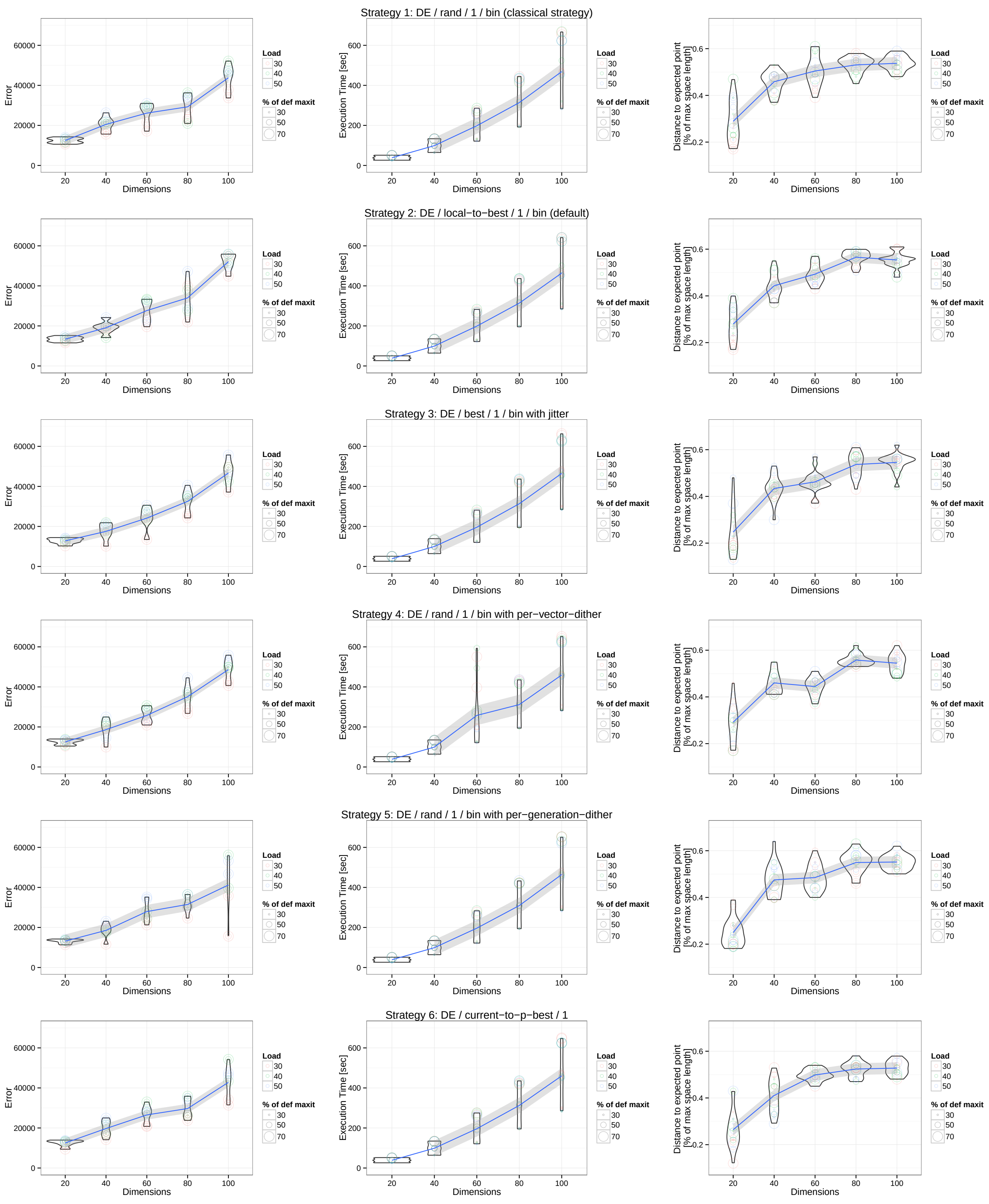


Figure 6.12: Comparative performance results of Differential Evolution strategies as a function of dimensions.

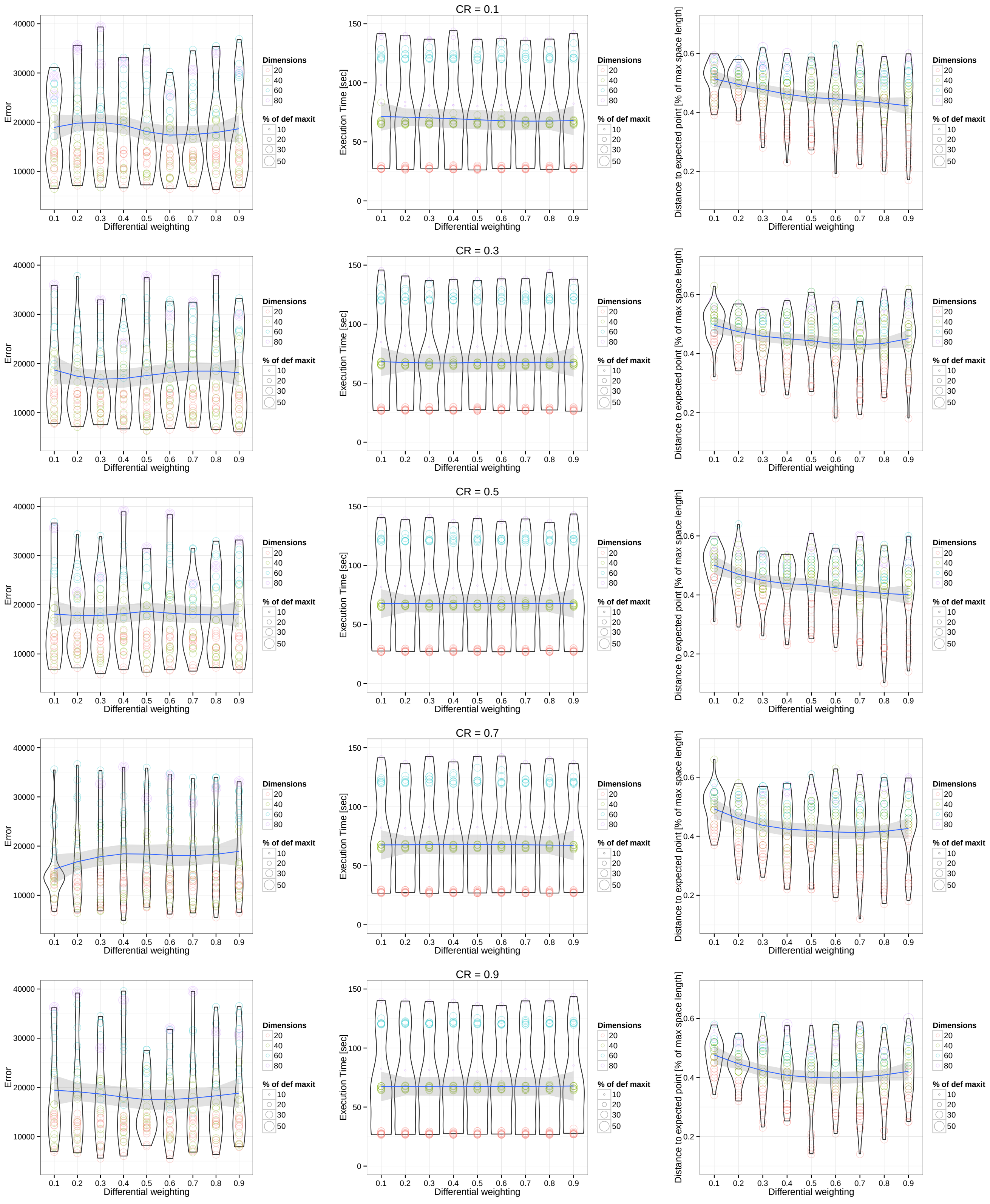


Figure 6.13: Comparison of Differential Evolution (strategy 2 – default), under different crossover probability CR and differential weighting factor F conditions.

6.6 Search Methods Quality Validation

Methods selection focuses on system performance which is established based on the search error – that is the main dimensions of the search as per Eq. 6.3, execution time, and distance of the solution point to the expected result point (precise for simpler, low–dimensional cases, where modelled system is not overloaded) — given as % of maximal space length, that is the search space diagonal of the n–dimensional normalised hypercube, equal to $\sqrt{n}$.

This section presents findings of the above measures for each method with respect to load, dimensions (actions and resources), and maximum iterations (maxit).

6.6.1 Execution Runs

The first set of experiments focused on signals decomposition (ASM-SD) using an ASM system model containing 10 dimensions– 5 action types running with use of 2 resources. To explore the methods in more detail, a series of experiments with the model containing between 10 to 100 dimensions, carried out under variable arrival rates and load patterns, have been used (see results in Figure 6.14, 6.15, 6.16).

In order to validate the methods we must establish another measure than the standard error as the residual of Eq. 6.3. Since *Load Generator* parameters are under strict control, as discussed in Section 6.4.1, it is possible to derive analytically an expected solution point containing values of impact of actions on resources using load model parameters, i.e. arrivals, execution time distribution and load distribution in time.

Such an approach shows a different perspective of the search quality in the ASM context than the error typically used in optimisation.

Common observations from all experiments are: (a) execution time is exponentially dependent on maximum iterations (maxit) (cf. with Figure 6.10); (b) execution time is linearly dependent on populations used (cf. with Figure 6.16). Instead, error characteristics remain flat as a function of either population or max iterations. Thus a concussion would be to limit maximal count of iterations (maxit) and population.

6.6.2 Parallel Population–based Metaheuristics

All experiments have been carried on a single CPU machine (all runs on the same platform, `R` + JVM + OS + hardware) in order to test a serial processing execution times in a rigorous fashion. Parallelised population–based methods, even though provided as an option by some of the tested methods (DE, PSO), have not been included in this study. Computation scalability is a core aspect of the current trends in computing, and so there

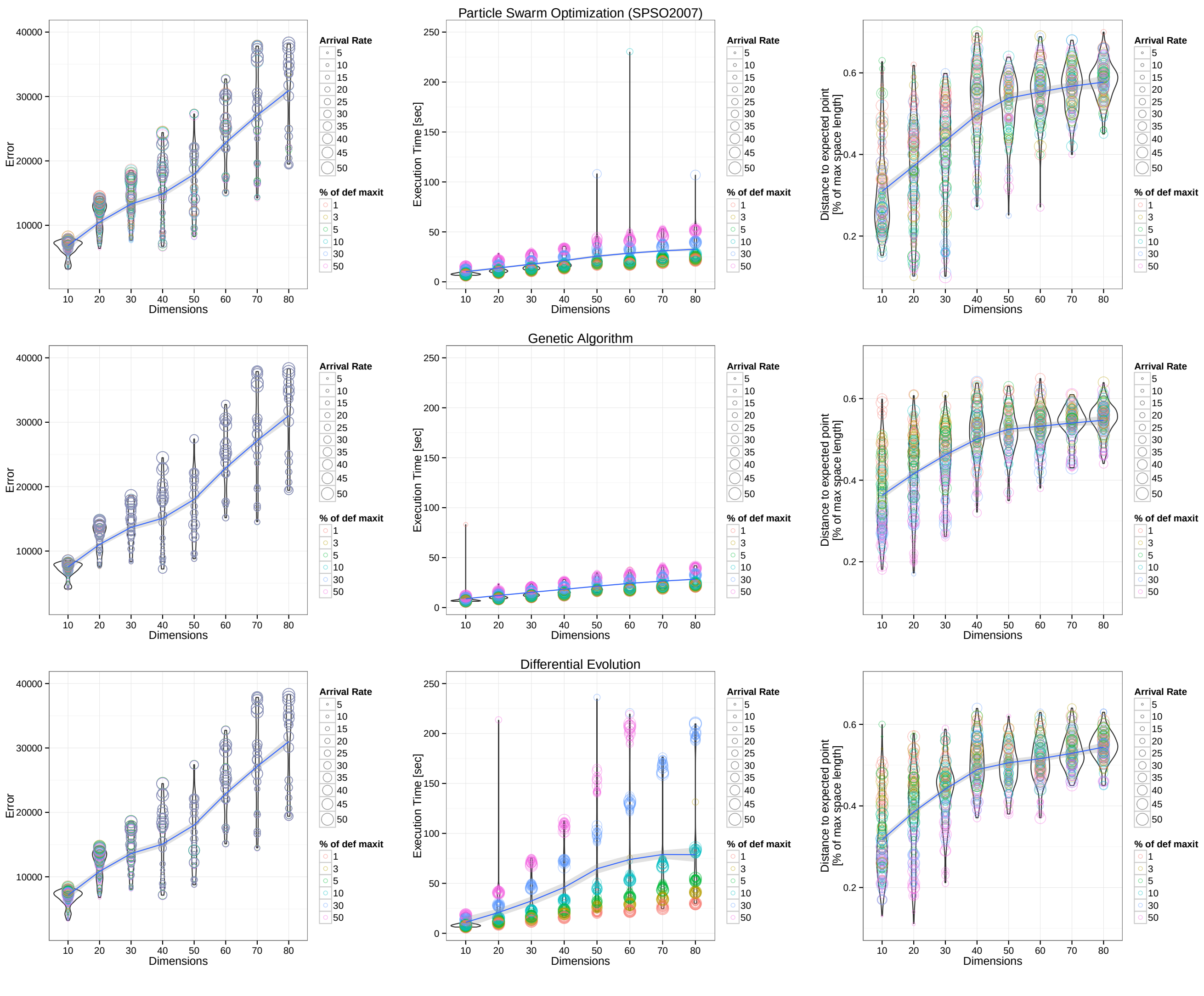


Figure 6.14: Comparison of error, execution time and distance to expected solution point with respect to dimensions count.

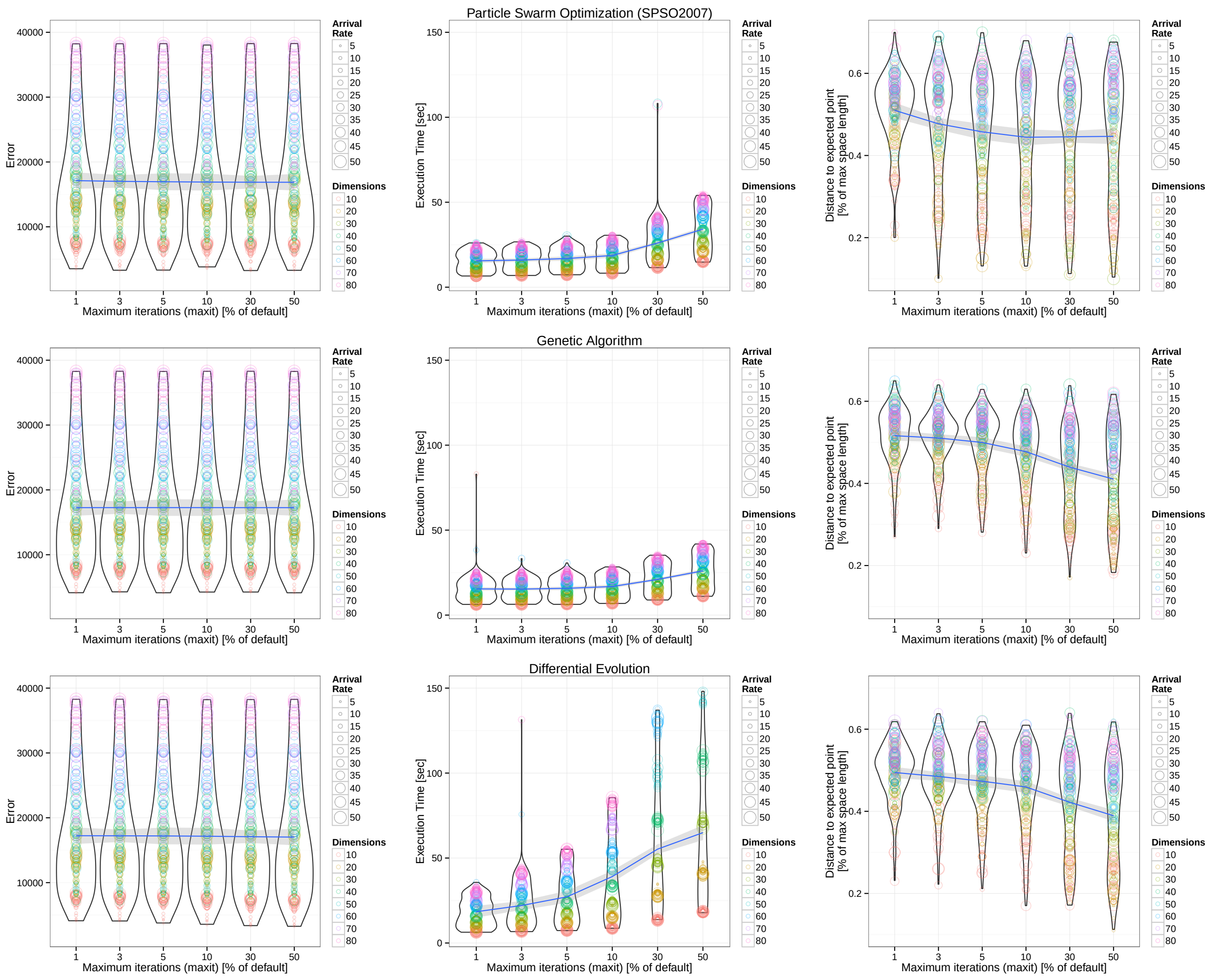


Figure 6.15: Comparison of error, execution time and distance to expected solution point with respect to maximum iterations (maxit, in percentage).

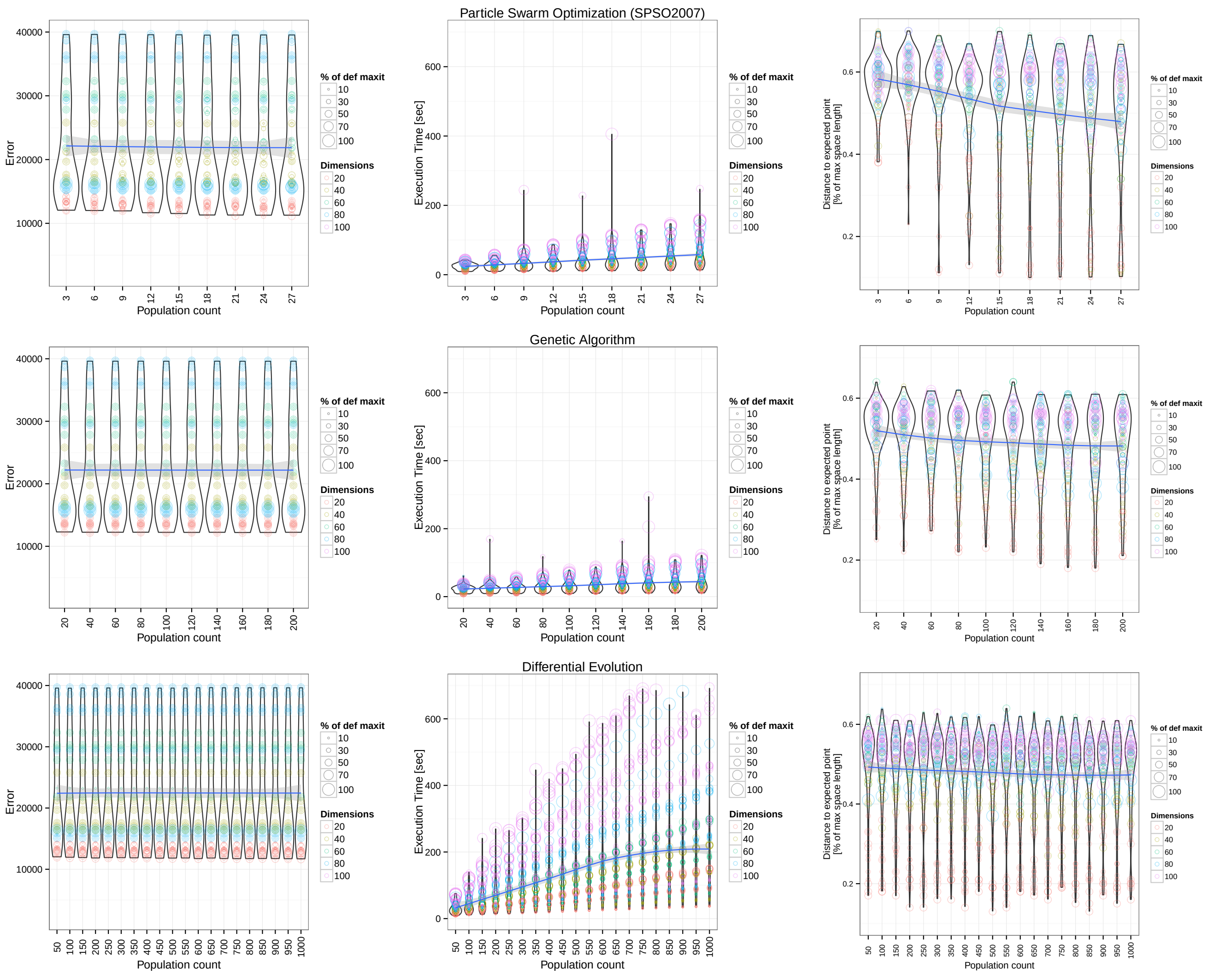


Figure 6.16: Comparison of error, execution time and distance to expected solution point with respect to size of the population.

is much interest in the community about parallel population–based metaheuristics (Nedjah et al, 2006), where concurrent search runs interact to improve the overall solution. Nevertheless, the reported findings offer some insight into the potential of these methods in ASM environments, when parallelised or executed in multiple threads.

6.6.3 Industry Applicability

Experiments show that ASM-SD gives stable results as early as 20–30% default maxit, as illustrated in Figures 6.14 and 6.15, so for engineering applications limiting maximum iterations to 20% could offer a good performance/value trade–off.

Also population sizes can be limited, as demonstrated by the results presented in Figure 6.16, where error characteristics as a function of population size (count) is constant.

Let us consider a real case scenario of a system with a hundred leading action types, and a minute time aggregate collected over a single release cycle usually one month long. Such a release based dataset should be defined by 40320 samples, so the processing time should be around 40–50 minutes, under very limited run–time conditions used in the experiments, that can be easily implemented in a scheduled tasks of an ASM framework.

6.7 Other Noise Models Affecting Causality Chain

As defined earlier in Section 6.3, the BSS problem formulation (Equations 6.1 and 6.2) can tackle other unobservable elements of signals of a system that impact the convoluted signal as noise defined by only considering its additive nature. This approach is a significant simplification that can be seen as considerable impediment in using the method more generally in ASM. Thus, to see the wider picture of possible scenarios it is necessary to highlight other noise models present in causal inference literature.

In order to incorporate an independent noise term in the model to compare the evidences of the two directions.

Below other noise models are listed following the notation of Eq. 6.2, where additive noise of U is simply a residual vector:

- Linear noise: $R = pA + qU$,
- Post–non–linear: $R = G(MA + U)$,
- Functional noise: $R = F(A, E)$.

6.8 Summary and Contribution of the Chapter

The ASM area still lacks research in features selection methodologies, which could be used to improve autonomous control and support ASM operators. This chapter presented an approach that can be applied as a deconvolution technique to help uncovering hidden run–time relations between observed signals in the ASM field.

Metrics signals gathered in normal operation of a system build databases of time–series that contain effectively definitions of hidden causality chains present in the systems. In order to extract the casual relations and establish models of systems interactions, utilisation dependencies and performance characteristics, that a controller or administrator can base on, it is required to match running application responses under given conditions with resources usage.

This chapter described a signal deconvolution method, named ASM-SD, driven by evolutionary search that can be applied in the ASM data analytics and adaptive controllers.

Various optimisation methods have been tested in this signals unmixing problem present in ASM. The evaluated scenarios confirm the practical potential of the proposed approach. The method is introduced thoroughly tested with use of DE, GA and PSO out–of–the–shelf search methods. The high–dimensionality of the search spaces and huge quantities of multivariate datasets require special qualities from the specific search methods applied.

The discussion of features selection for more efficient autonomous ASM control is continued in the next chapters. In particular the investigation of ways to extend the deconvolution approach, enhancing the proposed method to better tackle issues related to specific types of resources and actions signals. That is linked to the work on both autonomous and human-driven control in ASM environments, which is equipped with feature selection methods.

Chapter 7

Adaptive Control

"Truth will sooner come out from error than from confusion."
— Francis Bacon (1561 – 1626)

"Anyone who has never made a mistake has never tried anything new."
— Albert Einstein (1879 – 1955)

This chapter discusses a software controller framework for adaptive control in ASM environments and explores its potential. Control actions are executed in the context of the current system state, which is then again monitored and stored the database called *system state repository*, a knowledge base, extending it and giving views of the correctness or failures of the control actions, which are frequently evaluated. This incremental learning leads to evolving controller behaviour in a particular situation by taking into account consequences of earlier actions in that or similar situations. Experiments have been conducted with use of a real application, that has been instrumented with termination actuators and monitored accordingly (Sikora and Magoulas, 2013b).

7.1 Introduction

High dimensionality and nonlinearities, inherent in enterprise systems, pose several challenges to their modelling and run–time control. Most of the existing research in automating ASM is model–based and is able to accommodate a low number of dimensions, as discussed later in Section 7.2. In practice, enterprise system administrators, cloud based solutions users, FaaS or SaaS supplier staff use manual or semi–automated procedures allowing production level run–time modifications (Buyya et al, 2008), which turn into more flexible adaptive resource scheduling approaches (Zhan et al, 2015; Singh and Chana, 2016).

Introducing a view of expanding the adaptive control in ASM, this chapter proposes, implements and tests an approach that is based on autonomous control in the ASM framework, where selected functionalities pointed to exposed, or internal, interfaces are controllable, in accordance with functions of priorities in times of higher importance or lower system resources– examples of SLAs are presented in Table 7.1. This is a model–free approach that combines NNs with knowledge–based systems, which, to the best of author's knowledge, is innovative in this area (see Figure 1.1). Moreover, the work develops a software framework that is equipped with simple statistical methods for evaluating the system. The framework is able to change internal elements of run–time execution, by taking control actions in background of flexible SLA definitions and resources as part of the current system state.

In the era of "big-data" (Buyya et al, 2008; Brown et al, 2011), a data greedy approach is used, where all observable metrics are collected e.g., actions, resources and control actions, creating a knowledge base that also operates as a repository of system states and reactions to particular control actions. The approach focuses on an application scenario where the controller is only equipped with a termination actuator eliminating expensive to execute actions (not resources tuning control[1]), and can adapt to changing conditions according to modifiable SLA definitions, still without a model of the system. No predictors and no forecasting of service demands and resources utilisation have been used. The general objective is to adaptively control the system to optimise declared SLA functions values.

The chapter is organised as follows. Section 7.2 presents previous work in the ASM domain– the various control schemes proposed so far and their advantages and limitations. Section 7.3 introduces important research concepts and formulates the problem in the tested approach. Section 7.4 introduces and discusses the architecture of the proposed framework used to monitor and control an application. Sections 7.5 and 7.6 presents experimental results. Section 7.7 contains conclusions, discusses applications and outlines future work.

7.2 Previous Work in ASM Control

As it has been already outlined in Sections 2.6 and 2.7 the ASM systems adaptive control gained significant attention in last two decades. Especially in area of resources allocation allowing adaptive vertical and horizontal scaling (Buyya et al, 2009b).

On the contrary, partially due to success of the systems scaling progress, the adaptive application level control has not gained that much interest. The cloud computing community focused instead of flexible functions provisioning under PaaS and FaaS models where serverless computing is a core driving factor (Fox et al, 2017b). Thus the stateless

[1]The system resources can be modified in order to change important run–time characteristics – this scenario is not discussed in this chapter.

functions, mainly short performing cloud calls, enable fine–grained execution control over the performance (Van Eyk et al, 2017). This is reshaping several architectural principles (Shahrad et al, 2019) in order to maximise the setup in terms of costs and performance (Van Eyk et al, 2018).

Although the control of applications in ASM environments still needs substantial research to be conducted, manual ASM administration is widely used, together with the deployment of APM tools. Currently there are many out–of–the–shelf APM and ASM suites (Kowall and Cappelli, 2012), mainly focusing on network and operating system resources monitoring. Nevertheless, the function level performance monitoring and control earn more and more attention.

Despite the recent progress in the use of control theory to design controllers in the ASM field, the use of neural networks and their ability to approximate multidimensional functions defining control actions in system states remains under explored. Bigus over twenty five years ago applied neural networks and techniques from control systems theory to system performance tuning (Bigus, 1994). Similarly to our approach he used several system performance measures, such as devices utilisation, queue lengths, and paging rates. Data were collected to train neural network performance models. For example, NNs were trained on-line to adjust memory allocations in order to meet declared performance objectives. Bigus and Hellerstein extended that approach in (Bigus et al, 2000) and proposed a framework for adjusting resources allocation, where NNs were used as classifiers and predictors, but no extensive knowledge base was incorporated in that approach.

In contrast to the approach discussed above, no system model of any kind is present in the presented approach and a knowledge base is incorporated, as the main system metrics and control actions store. This allows to persist all the metrics and SLA values before any control is taken, and after to reevaluate control actions for changed system characteristics, and retrain actuators with new control rules on–line. Such evaluation of control actions does not require direct settings of control action to the SLAs defined and later delivered. Also in terms of SLAs, the approach provides a flexible way of defining functions, allowing the selection and join of all metrics present in the knowledge base.

In other words, it is a control setup that is isolated from any statistical or causal model or any other form of predefined or online aggregated meta–information describing present and past system states, **S**. Essentially, the simplest form of control deduction still allowing to adaptively govern a real application performance.

7.3 ASM Control Formulation and Key Concepts

This section introduces a few research concepts and defines terminology used later in the chapter. We begin the introduction starting from a control systems formulation of the

ASM problem, going through the considered dimensions and system state definition, and finish with an overview of our approach for rule induction.

7.3.1 Control in the ASM Field

As discussed earlier in Chapter 3, a controller operating in an ASM framework is controlling an enterprise class system which can be modelled as a dynamic system, a *System* model **C** (see Equation 3.6). The model can be expressed as a system of functions: $\mathbf{C} : \mathbb{R}^{n+1} \rightarrow \mathbb{R}^{n+1}$, $n = k + m$ in time domain, where k is the number of actions and m is the number of system resources. In this context, all run-time characteristics of the system are determined by a set of resources and input/output actions, whose current values define the *system state* **S**.

7.3.2 Rules Induction

A feature of our approach is that it considers the enterprise system as a black-box, so it uses no explicit or analytic model of the system; in particular there is no direct knowledge of the code-base structure and the characteristics of the resources. Therefore, the control system has to configure future control actions by observing system states **S** and their changes. It "learns by observation", building a Knowledge Base and solving an optimisation problem.

This type of "learning" is in essence a very natural process as Thorndike and Bruce (1911) discovered while researching animals adaptation facing pleasant and unpleasant situations (Herrnstein, 1970). Thorndike observed animals response causing pleasing effect was reinforced by its consequence, and formulated the law of effect. In other words, actions producing favourable outcomes are more likely to happen again, whilst actions causing discomfort are avoided. This was later extended in the work of Skinner in the area of operand conditioning and defined principles of reinforcement and behaviourism (Skinner, 1938, 1963).

Influenced by the operand conditioning phenomena, research in machine learning introduced variants of Reinforcement Learning (RL) to control by reward and punishment using dynamic programming Sutton (1984); Watkins (1989); Sutton and Barto (1998).

The idea of exploiting background knowledge with positive and negative examples in order to formulate a rule–based hypothesis about the accuracy of an action was researched in the context of inductive reasoning (Muggleton and De Raedt, 1994; Muggleton, 1999), giving grounds of Inductive Logic Programming (ILP).

In this work an approach that builds on the above mentioned schemes is applied, where system states associated with low SLA values are promoted and those which lead to high

SLA penalty values are discouraged in the future actions of the controller. In order to promote this behaviour, an online adaptation procedure through frequent evaluations is performed. A search is executed for each of the process runs to uncover new and evaluate existing associations (indirect mappings) between system states and SLA values. So the problem space of unknown yet possible application system states of observable dimensions of $\mathbf{C} : [\mathbf{r}, \mathbf{a}]$ is being constantly explored (see Section 4.2). Generally, this sort of a trial and error method works quite well, where little a priori knowledge is available.

The control system is able to modify only a subset of all functional actions a_c, those which are expressly instrumented with the controller agents code. A control action in a given system state $c^a(\mathbf{S})$ is a direct result of a rule induced $\mathcal{R}(\mathbf{S})$, approximated with use of Neural Network (NN).

When induced, a control rule expresses a belief (expectation) that the control applied in the background of a system state $\mathbf{S}$, in an area of observable dimensions (a, r), is going to lead to a lower SLA value. When the rule is applied on the system, a new state is observed that extends the system state repository, reflecting the correctness of the control action. More details of the rules induction process are presented in Section 7.4.4.

In this chapter SLAs are treated as penalties of equal importance. Following this assumption a summary of all set SLAs can be computed in order to derive knowledge about the system “health”. The control can be applied on the basis of rules induced under this assumption.

7.4 Proposed Software Framework and Controller Architecture

This section describes the proposed framework and its various elements, such as the monitoring mechanism and the controller architecture, with various components of the environment used for simulations and practical evaluation of the proposed solution.

Figure 7.1 shows the software framework used in the research. All components (Monitoring agents, Collector, Loader, Evaluator, and Controller) are implemented in Java and running in isolated run–times, communicating asynchronously with the use of messaging via a JMS broker (Active MQ). This architecture provides processes separation between metrics data collection, rules generation and control execution. Processes can be deployed and run on many machines, observing system infrastructure as a whole. Communication between components is asynchronous limiting time spent on waiting. The collected metrics are stored in a knowledge base and are persisted in a RDBMS to allow flexible querying (Oracle RDBMS).

Further description of the whole environment will replicate the data life–cycle; starting

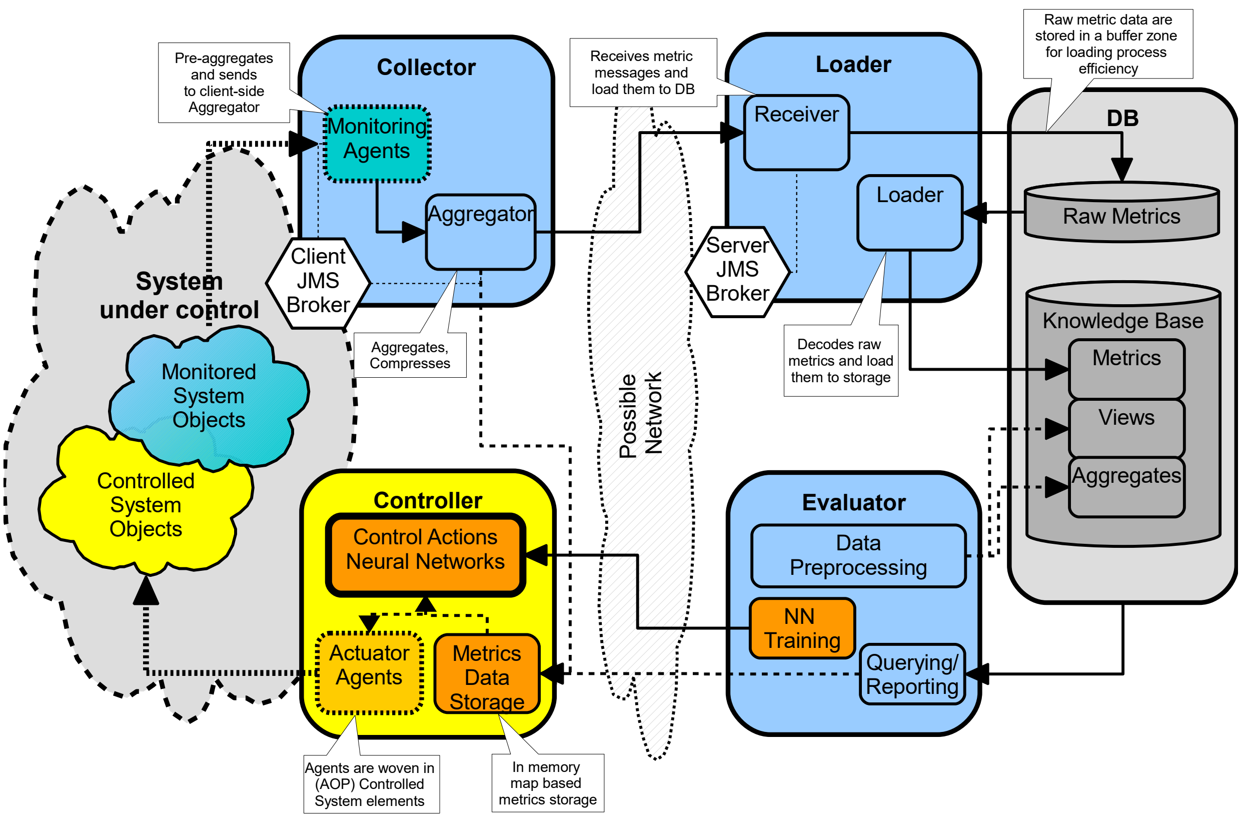

Pre-aggregates and sends to client-side Aggregator
Collector
Monitoring Agents
Aggregator
Receives metric messages and load them to DB
Loader
Receiver
Loader
Raw metric data are stored in a buffer zone for loading process efficiency
DB
Raw Metrics
Knowledge Base
Metrics
Views
Aggregates
System under control
Monitored System Objects
Controlled System Objects
Client JMS Broker
Aggregates, Compresses
Server JMS Broker
Decodes raw metrics and load them to storage
Possible Network
Controller
Control Actions Neural Networks
Actuator Agents
Metrics Data Storage
Evaluator
Data Preprocessing
NN Training
Querying/ Reporting
Agents are woven in (AOP) Controlled System elements
In memory map based metrics storage

Figure 7.1: Deployment diagram: a logical overview of all software components used in the complete data life-cycle for ASM control used in the research.

from the application monitoring and data acquisition, through data transformation, processing and search, finishing on the controller and actuators used.

7.4.1 The Controlled Application and the SLA Engine

The main goal is to control an application, satisfying the technical and users' objectives. Thus the control of the application has to be done in the background of defined knowledge about the system run–time and functional priorities. SLA definitions are used to set up such information in a form that is easily interpretable by both administrators and users, i.e. SLAs are defined as penalty functions in $ units. The system supports flexible SLA definitions; any of the collected metrics as part of system state can be used in SLA definitions.

The concrete SLAs values are computed and stored in the main metrics storage in timely fashion. These synthetic metrics extend the knowledge base. Typically, standard time aggregation is used, so the scalar values are stored per time bucket.

SLA definitions in the current implementation use elements of Structured Query Language (SQL) phrases, as this method was found as a very flexible way of specifying the run–time and the business situations. An example is provided below:

Listing 7.1: Example of penalty drive SLA.

Table 7.1: Examples of SLA definitions defined using SQL phrases (part A)

Attribute Name	SLA Definition Attribute Value
Code / Name	'SLA1: 1$ per every extra second over 2sec execution'
SLA Value Phrase	'case when (sum(metricvalue) - 1000) / 1000 ≤ 60 then (sum(metricvalue) - 1000) / 1000 else 60 end' – summary of execution times but no more than 60$ penalty
Base Resource Filter	'%//HTTP/127.0.0.1%:8081//dddsample/public/trackio [null]'
Metric Value Filter	'metricvalue ≥ 2000' – take into account actions longer than 2 secs only
Group Having Phrase	'1=1' – not used in this SLA
Code / Name	'SLA3: 10$ for every started second of an image processing longer than 10ms'
SLA Value Phrase	'ceil(10 * (count(1) * sum(r.metricvalue) / 1000))' – 10$ of every started second of processing, note: execution time in ms
Base Resource Filter	'ExampleFilter1//HTTP/1.1://127.0.0.1(127.0.0.1):8081//dddsample/images/%' – interprets execution time of images processing on server side, wrapped by a specific filter
Metric Value Filter	'1=1' – there is no filter on metrics values applied
Group Having Phrase	'avg(r.metricvalue) ≥ 10' – but only those time buckets which average metric value is longer than 10ms
Code / Name	'SLA10: 1$ per every terminated action'
SLA Value Phrase	'20 * count(1)' 20$ penalty for each of executed actions
Base Resource Filter	'org.allmon.client.controller.terminator.NeuralRulesJavaCallTerminatorController'
Metric Value Filter	'exceptionbody is not null and sourcename like "%.VoidLoadAdderImpl.generateIOLoad' – checks metrics for which the filtered call was terminated with exception and source call was coming from additional load generator method
Group Having Phrase	'1=1' – not used here

```
p_i_sla_resource_name =>
    'SLA3: 10\$ penalty for every second started of an image
        processing longer by average than 10ms'
p_i_base_resource_like_filter_phrase =>
    'ExampleFilter1//HTTP/1.1://127.0.0.1(127.0.0.1):8081
        //dddsample/images/%.png [null]'
p_i_select_sla_value_phrase =>
    'ceil(10 * (count(1) * sum(r.metricvalue) / 1000))',
p_i_where_metric_phrase =>
    'exceptionbody is not null and sourcename like '%.User.Name''
p_i_having_phrase =>
    'avg(r.metricvalue) > 10' -- post aggregate phrase
```

More examples of SLAs can be found in Table 7.1. The first three SLA definitions have been used in the simulations explained later. The last two examples define business scenarios SLAs, showing a more practical use. The engine is very flexible, so administrators can easily join all metric values collected in the metrics database. Also, service providers would have an option of flexible pricing, which is often limited to flat rates or tariffs based on exposed functionality usage thresholds.

7.4.2 Monitoring Environment

All data for the metrics are acquired from monitored parts of a system by remote agents. The monitoring framework[2] utilised *passive* and *active* monitoring as complementary techniques of performance tracking in distributed systems. *Active (Synthetic) monitoring*

[2]Allmon (Sikora, 2012) is a Java monitoring tool, freely available on Apache License 2.0 (ASF, 2004)

Table 7.2: Examples of SLA definitions defined using SQL phrases (part B)

Attribute Name	SLA Definition Attribute Value
Code / Name	'SLA50: 0.01$ price per every public call shorter than 1 sec'
SLA Value Phrase	'-0.01 * count(1)' – 0.01$ price an action executed
Base Resource Filter	'%//HTTP/%//application/public/%%/%.do%' – interprets all metrics of instrumented public application code
Metric Value Filter	'metricvalue $\leq$ 1000' – actions shorter than one sec only
Group Having Phrase	'1=1' – not used
Code / Name	'SLA51: 50$ extra penalty per every public call longer than 3 secs during peak hours only from 1500 to 1700 hours '
SLA Value Phrase	'50 * count(1)' – 50$ penalty for longer actions, executed from 3 to 6pm (the business peak hours) in working days
Base Resource Filter	'%//HTTP/%//application/public/%' – interprets all metrics of instrumented public application code
Metric Value Filter	'metricvalue $\geq$ 3000' and to_char(timestamp, 'HH24') in ('15', '16', '17') and to_char(timestamp, 'D') $\leq$ 5' – actions longer than 3 secs only
Group Having Phrase	'count(1) $\geq$ 100' – SLA values are calculated only in times when there is more than 100 action calls executed per time bucket

(see Section 4.2.2) collects data in scheduled mode by scripts, which often simulate an end–user activity.

An example of active monitoring setup of OS and JVM via JMX metrics collection is provided below in Listing 7.2.

Listing 7.2: Example of active monitoring setup of OS and JVM (via JMX) metrics collection by AOP configuration.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns xmlns:ws xmlns:wss xmlns:allmon...>
    <bean id="agentContext"
         class="org.allmon.client.agent.AgentContext"/>
    <allmon:active>
        <!-- get all metrics every 10 sec -->
        <allmon:osAgent id="osCheckALL"
      agentContextRef="agentContext"
      cronExpression="*/10 * * * * ?"
      metricType="ALL" />
        <!-- get CPU utilization every 2 sec -->
        <allmon:osAgent id="osCheckCPU"
            agentContextRef="agentContext"
            cronExpression="*/2 * * * * ?"
            metricType="CPU" />
        <!-- get remote jvm:jmx memory metrics -->
        <allmon:jmxServerAgent id="jmxCheck"
            agentContextRef="agentContext"
            cronExpression="*/5 * * * * ?"
            lvmNamesRegexp=".*"
            hostName="localhost" port="9999"
            mbeansAttributesNamesRegexp=
                ".*java.lang:type=Memory.*HeapMemoryUsage/used"
            mbeansObjectName = ""
            mbeansAttributeName = "" />
    </allmon:active>
</beans>
```

Passive monitoring (see Section 4.2.1), enables supervision of actual, real clients and applications' interactions with a system. Metrics collected by this approach can be used to determine the actual service–level quality delivered to end–users and to detect errors or potential performance problems in the system. Passive monitoring agents are deployed into the application with the use of run–time level instrumentation, often by AOP[3], or filters [4] added with use of application server configuration. Agents are woven in the run–time allowing it to access compiled functionalities and exposing internal characteristics, which makes it a very generic audit approach.

Listing 7.3 presents an example of allmon filter–based passive monitoring set on a Java web container, where every servlet is instrumented.

Listing 7.3: Example of web-app XML with filter–based passive monitoring.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee" ..>
    ...
    <!-- monitoring filter definition -->
    <filter>
        <filter-name>PerformanceMonitoringFilter</filter-name>
        <filter-class>
            org.allmon..advices.HttpServletCallFilter
        </filter-class>
        <!-- filter parameters -->
        <init-param>
            <param-name>
                org.allmon..advices.HttpServletCallFilter.filterName
            </param-name>
            <param-value>ExampleFilter1</param-value>
        </init-param>
        <init-param>
            <param-name>
                org.allmon..HttpServletCallFilter.sessionUserAttrKey
            </param-name>
            <param-value>UserProfile</param-value>
        </init-param>
        <init-param>
            <param-name>
                org.allmon..HttpServletCallFilter.captureRequest
            </param-name>
            <param-value>true</param-value>
        </init-param>
    </filter>
    <!-- monitoring filter mapping to url assets on the server -->
    <filter-mapping>
        <filter-name>PerformanceMonitoringFilter</filter-name>
        <url-pattern>*</url-pattern>
    </filter-mapping>
    ...
</web-app>
```

The application can be deployed on many hosts, so metrics have to include host name and instance name of the software area that the metric comes from. Next to the metrics

[3]AOP introduces a few useful concepts improving separability of the monitoring and the controller code from the system functionality. AOP bases on separation of cross-cutting concerns, applying advises, point–cut, and aspects (Kiczales et al, 1997), (Laddad, 2009).

[4]http://httpd.apache.org/docs/trunk/filter.html in Java Servlet specification since version 2.3 (Coward and Yoshida, 2003)

value each object also contains a time stamp, a metric type, the resource name and the source of the collection.

7.4.3 Selecting the Dimensions of the Features Space

A system can be configured to have thousands of observable dimensions – resources r and actions a. In such vast feature space it is imperative to reduce the dimensionality, as discussed in Chapters 5 and 6. This is crucial not only for the evaluation process quality, but mainly due to limitations in sampling the system states after control (evaluations are normally much less frequent than control actions), NNs capacity considerations and generalisation performance.

Various methods for dimension reduction have been proposed so far, with applications to wide range of disciplines (Fodor, 2002). In order to isolate from other general scope methods discussed in earlier chapters, but to support the preprocessing for training phase it was decided to use one of the simplest of feature selection methods (Guyon and Elisseeff, 2003), which essentially keeps subset of most relevant dimensions of the original space. At this stage of the research, for such declared needs of the study, this approach of fighting with the curse of dimensionality appears to be sufficient.

The dimensionality selection based on feature ranking with statistical correlation coefficient (Saeys et al, 2008), using the Java–ML library[5] (Abeel et al, 2009) was applied. Ensemble Feature Ranking with Spearman was found as best working with the ASM data. Only a few dimensions are selected for each single control run according to their fitness in terms of impact on the SLAs values. The control actions are driven by the need to minimise the SLAs values and induced rules are generated on the basis of an evaluation process which considers only the selected dimensions. The evaluation process is described in more details in Section 7.4.4.

Ideally the dimensionality reduction would be an early part of the evaluator process. Nevertheless, the main fundamental problem with applying more extensive reduction, and feature selection in particular, is related to the fact that dimensionality reduction gives as a result an utterly new features space, and has an impact on the system control responses search. When a given selection is chosen (as a small–set combination of all monitored dimensions), the controller will use it for rules generation and effectively for control execution on the application. Assuming that this selection may change in the Evaluator's following runs (in order to improve controller performance), this can potentially change dramatically the controller characteristics. Hence, the used combinations would have to be stored and taken into consideration in the next level of evaluation. This additional processing would significantly complicate SLA values comparison, which is at the core of rules induction, and effectively add the need to consider the combination used as a new

[5]Java-ML is a Java Machine Learning library, available on GPL `http://java-ml.sourceforge.net/content/feature-selection`

dimension in the comparison. Therefore only the feature selection before an actual control phase is in use at this part of the study.

7.4.4 Evaluating Control States

The Evaluator is a software component that selects system states stored in the database that provides a view of past states and control actions. The data selected is processed and used to build training sets for NNs. The controller consists of a pair of NNs with antagonistic rules definitions. Figure 7.2 shows the design of the neural controller decision block. The first "bad" NN is trained to detect states where potentially wrong control would be taken, the second "good" NN is trained to confirm states where good control actions were evaluated.

The proposed rules generation algorithm selects system states where: (a) SLAs take extreme values (maximum and minimum), (b) the total of SLAs when control was applied was lower than without control (details are provided in the following subsection). Those historical system states are merged and create training rule sets for the neural controller. Only selected dimensions (those impacting SLA) are used to train the NNs; same feature dimensions are used for "bad" and "good" networks – aggregated normalised metrics values are passed to training sets. Trained pairs of NNs, as ready–to–use Java objects, are serialised and sent over using topic style JMS published messages to the Controller process.

Because an SLA, as a synthetic metric, is not available in the controller process, the Evaluator has to find matching metrics that are available for the controller and could be used directly in defining the training sets. The analysis starts with time points where SLAs values were the highest, as then the impact is most significant. A search is performed to find which SLAs are impacted and their values could be potentially lowered in the after applying termination control for similar situations.

In the experimental study the evaluation process has been tuned to read time buckets of 15 second aggregates, which were found as most practical. Time–wise aggregation of metrics filters out most of the chaotic short–time changes in the system. Moreover this duration is a good lower limit of observable stable control system responses, and was assumed as a standard/general aggregation. It is worth mentioning that this technique significantly reduces the size of the data set ready for search; when the standard 15 sec time–bucket aggregation is used, there are only 240 system state points in an hour[6].

7.4.5 The Rules Generation Algorithm

As mentioned earlier the Evaluator searches historical systems states. The search is conducted according to following steps:

[6]240 system state points in an hour generate 5.7K in a day, 40K in a week, 2.1M in a year

1. Select extreme SLA states (min-max) from the database. Search high SLA for "good" control states and low SLA for "bad" control states. Limit data samples retrieved, leave only the strongest (more extreme) states for each of the lists– this is further discussed in Section 7.4.6.

2. Select "good" and "bad" control/no-control states. In general, a "good" control decision makes the situation better, i.e. it leads to lower SLA values for the set of state parameters located in the neighbourhood of a particular hyper–cube in the state space. Conversely, a "bad" control decision makes the situation worse, leading to higher SLA values for similar state parameters. Limit data samples retrieved by focusing only on the best/worst decision examples.

3. Merge selected states to create training sets. Add high value SLA states to "good" control state patterns (to encourage taking similar control actions), and low value SLA states to "bad" control states, so control decisions which were considered as harmful are avoided. Add negated high SLA states to "bad" control state patterns and negated low SLA states to "good" control states.

4. Eliminate conflicting states. Remove states with opposite meaning between "good" and "bad" training sets. Resolve conflicts between system states vectors before further training of NNs takes place. This step helps avoiding areas of state instability, where past control decisions were not very stable or consequent to earlier control decision– this is further discussed in Section 7.4.7.

Pseudo code for the proposed evaluation procedure and the rules induction algorithm, which searches across the antagonistic system states to generate training data sets for the NN controller, is presented below:

```
function EvaluationMain                                         ▷ Evaluation process
    for all $a \in a_c$ do
        $\mathcal{R}_G, \mathcal{R}_B \leftarrow RulesGen(a)$;                     ▷ Rule sets
        $nng \leftarrow Train(\mathcal{R}_G)$;                                    ▷ Train NN
        $nnb \leftarrow Train(\mathcal{R}_B)$;
        $Broadcast(nn, a)$;                                     ▷ Send NN to actuators
```

```
function RulesGen($a$)
Select from past system states "good" and "bad" control actions to define control rules.
                                                ▷ Search states of "low" and "high" SLA
    $\mathbf{L} \leftarrow EvalSLA(\text{"}low\text{"})$;
    $\mathbf{H} \leftarrow EvalSLA(\text{"}high\text{"})$;
                                              ▷ Search "good" and "bad" control states
    $\mathbf{G_c}, \mathbf{G_{nc}} \leftarrow EvalControl(\text{"}good\text{"}, a)$;
    $\mathbf{B_c}, \mathbf{B_{nc}} \leftarrow EvalControl(\text{"}bad\text{"}, a)$;
                                       ▷ Merge selected states to create training sets
    $\mathcal{R}_G \leftarrow \mathbf{H} \bigcup \mathbf{G_c} \bigcup \mathbf{G_{nc}} \bigcup \neg\mathbf{L}$; [7]
```

[7]Merge sets of states of high SLA values and "good" control to promote taking control actions rules.

```
    $\mathcal{R}_B \leftarrow \mathbf{L} \bigcup \mathbf{B_c} \bigcup \mathbf{B_{nc}} \bigcup \neg\mathbf{H}$; [8]
                                                        ▷ Remove not-consequent rules
    $ElimConf(\mathcal{R}_G, \mathcal{R}_B)$
    return $\mathcal{R}_G, \mathcal{R}_B$
function TRAIN($R$)                                     ▷ Train the control block NN
    $nn \leftarrow TrainMLP(\text{``sigmoid''}, R)$
    return $nn$;
function BROADCAST($nng, nnb, a$)
    $o \leftarrow Serialize(nng, nnb)$
    $Send(o, a)$ [9]
```

```
function EVALSLA($type, a$)
SLAs are considered as penalty functions, so higher values can be associated with bad control actions.
    if $type$ = “$high$” then [10]
        for $i = 1 \rightarrow m$ do
            $\mathbf{H}(i) \leftarrow \max(SLA^a \in \mathbf{C} \setminus \mathbf{H})$
        return H
    if $type$ = “$low$” then [11]
        for $i = 1 \rightarrow m$ do
            $\mathbf{L}(i) \leftarrow \min(SLA^a \in \mathbf{C} \setminus \mathbf{L})$
        return L
function EVALCONTROL($type, a$)
Evaluate good/bad control actions (for both no-control and control states).
                                                        ▷ Select the best control [12]
    if $type$ = “$good$” then
        for $i = 1 \rightarrow c$ do
            $\mathbf{G_c}(i) \leftarrow \mathbf{S} : \min(\sum SLA^a_c(\mathbf{S}) - \sum SLA^a(\mathbf{S}))$
            $\mathbf{G_{nc}}(i) \leftarrow \mathbf{S} : \min(\sum SLA^a_{nc}(\mathbf{S}) - \sum SLA^a(\mathbf{S}))$
        return $\mathbf{G_c}$, $\mathbf{G_{nc}}$
                                                        ▷ Select the worst control [13]
    if $type$ = “$bad$” then
        for $i = 1 \rightarrow c$ do
            $\mathbf{B_c}(i) \leftarrow \mathbf{S} : \max(\sum SLA^a_c(\mathbf{S}) - \sum SLA^a(\mathbf{S}))$
            $\mathbf{B_{nc}}(i) \leftarrow \mathbf{S} : \max(\sum SLA^a_{nc}(\mathbf{S}) - \sum SLA^a(\mathbf{S}))$
        return $\mathbf{B_c}$, $\mathbf{B_{nc}}$
function ELIMCONF($G, B$)
Eliminate conflicting states - remove similar states.
    for all $\mathbf{S}_G \in \mathbf{G}$ do
        for all $\mathbf{S}_B \in \mathbf{B}$ do
```

[8]Merge sets of states of low SLA values and “bad” control to prevent harmful control decisions.

[9]Trained networks with approximated rules definitions are sent to controllers actuators, so they are directly available in the controlled application runtime.

[10]Select high SLAs for “good” control states, where penalties for the control were still lower than SLAs - in both cases where control was applied, and control was not applied at all.

[11]Select low SLAs for “bad” control states, where penalties for the control were higher than SLAs (too strong control) - in both cases, where control was applied, and control was not applied at all (so keep it like that, do not apply any control for such states).

[12]A “good” control decision is the one that makes the situation better, so select system states **S** where total of SLAs is lower for similar state parameters.

[13]A “bad” control decision is the one that makes the situation worse, so select states where SLAs are higher for similar state parameters.

```
if $\mathbf{S}_G \sim \mathbf{S}_B$ then RemoveState [14]
```

7.4.6 Extrema Search and Previous Control Actions

High SLA values based on actions execution time can be a result of saturation of the utilised resources. Moreover, some of the SLAs values could be more expensive in real terms than the violations of SLAs due to terminated actions. The precise situation depends, of course, not only on the SLAs definitions but also on the enterprise system's load and the amount of potential violations. In most practical cases, however, the terminating actuator can have an impact on SLAs that are based on an action execution time. The termination actuator can release load and effectively lower the SLA values. This approach promotes termination of actions running for a system state identified as related with the highest SLA footprint, and condemns termination decisions when the control caused more harm than benefit in terms of selected SLA levels.

Previous control actions are compared according to summary of SLAs in a given system state $\mathbf{S}$. As noted above only the best and the worst control actions are considered. The size of the set c is tuneable and typically depends linearly on the number of dimensions selected, and the total number of the system states in the repository.

It is worth mentioning here an interesting effect. There is possibility to change the SLA definitions while the control system operates. Although there is no technical problem to recalculate new SLA values, as past system states are available, it is not possible to evaluate all reactions to control decisions, as such modification could fundamentally change the dynamics of the control system. A similar problem was described earlier in Section 7.4.3, where impacts of frequent dimensions selection and effective system space search were noted.

7.4.7 Conflict Resolution Strategies

Strategies for conflict resolution were introduced in order to deal with a problem of inducing potentially not consequent rules. Due to high–dimensionality and responses instabilities on control action, good and bad rule sets $\mathcal{R}_G, \mathcal{R}_B$ may contain conflicting directives for similar system state $\mathbf{S} : \mathbf{S}_G \sim \mathbf{S}_B$.

Similarity criteria used here are based on Chebyshev distance in the normalised metrics data space, so a hyper–cube shape neighbourhood is considered. Thus, two states are similar $\mathbf{S}_G \sim \mathbf{S}_B$ when the following similarity condition holds: $\forall_{(d_G \in \mathbf{S}_G, d_B \in \mathbf{S}_B)} |d_G - d_B| < \xi$.

In other words, the Chebyshev distance $D_{\mathrm{Ch}}(p,q) := \max_i(|p_i - q_i|)$ between system states

[14] In the simplest form both system states are removed. There have been more variants of resolving conflicts researched as well.

is lower than the neighbourhood distance $D_{\mathrm{Ch}}(\mathbf{S}_G, \mathbf{S}_B) < \xi$. The neighbourhood distance ξ is tunable, but normally is set to 5% of a dimension range.

Initially, all similar system states from both "good" and "bad" rules vectors were removed in order to avoid any conflicts– this strategy is named Variant A. This was found too strict, in the sense that too many data points were removed when adopting this strategy. Training sets for NNs appear often degenerated and poor–in–value during simulations when too many contradicting rules are generated in the same states neighbourhood. In the experiments, this scenario mainly occurred at the start of the controller operation, or when the run–time situation changed drastically.

Therefore slightly more advanced algorithms for preventing rule conflicts are suggested. In order to implement these strategies the system state vector has to be extended with additional measures, calculated from data points within the system state neighbourhood, such as the total of SLA values, the count of similar system states (cardinality), the average of SLA values. That leads to the following set of strategies:

- Variant A – "clarity only" – is a very strict strategy that eliminates all system states if they lay in the same neighbourhood.
- Variant B – "habits first" – this strategy adopts a simple approach checking the cardinality of similar rules induced in the same system state neighbourhood. This variant emphasises frequently occurring events, named "habits", and only eliminates less frequent states, i.e. those of lower cardinality which possess less occurring instances. Thus, overall fewer states are eliminated compared to Variant A.
- Variant C – "avoid traumas" – this strategy eliminates system states that lay in the same neighbourhood keeping the ones that possess higher average of SLA penalties. So the level of SLA violations influences conflict resolution as this strategy emphasises high penalty events, named "trauma", which may be rare but introduce significant losses from SLA perspective.

A comparison of the controller behaviour using above strategies is included in the experimental part of the work in Section 7.6.3.

7.4.8 Training Set Generation and Rules Induction Implementation

To approximate the discovered control rules by training NNs, a simple hierarchical approach has been used. Figure 7.2 provides an overview of the neural networks-based controller, as discussed earlier. Each of the controlled actions (java class.method, servlet, etc.) has its own dedicated pair of NNs instances which are trained to produce specific for this action control decisions depending on system states conditions and past experiences.

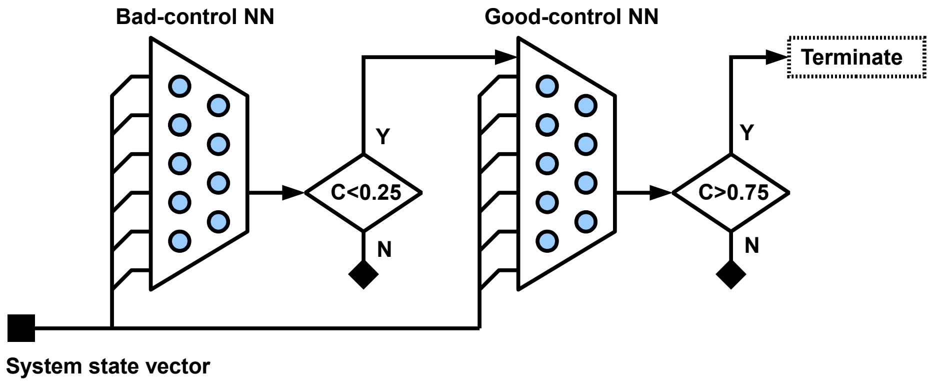


Figure 7.2: Neural decision block structure. Neural networks-based decision module consisting of two NNs for each of the actions that should be controlled.

The NNs are trained with “bad" and “good" control action sets selected from the past. The controller first considers decisions of the “bad" control network and if the current state does not match any patterns learned by the NNs, then it begins checking the “good" control actions.

NN are trained with normalised data, within min/max boundaries; system state values which are exceeding the declared scope are not taken into account by the controller, so no control decision is taken. Of course such a situation is monitored and when occurs it extends the boundaries in the next evaluation iteration.

Neuroph[15] library with Multi Layer Perceptron (MLP) implementation and sigmoid activation function was applied. Experimentally, it was found that for the proposed training set structure, NNs with two hidden layers of 4 neurons trained using MomentumBackpropagation algorithm, worked well in most situations. Activation thresholds for termination actions in the control decision module were 0.25 and 0.75 for “bad" and “good" control actions respectively, as shown in Figure 7.2.

An example of rules induced, as NN training sets and actual control actions have been presented in Figure 7.3. Two main aspects of the rules induction and NN training sets for the control are highlighted: (a) target NN in the decision block – marked with shape, and (b) termination action – marked with colour. Circles show actions of the “good" neural network, while squares denote contributions of the “bad" part of the decision module. Target neural networks are trained with termination and no–termination actions states. Thus, light red shapes indicate supported termination control, whilst black promote no control.

Figure 7.3 depicts also a comparison of total of SLA values (top) and cardinality (bottom) of each of the system states identified in the rule set. Shapes size indicate the value of SLAs or cardinality in a given system state. Crosses indicate real system states where

[15]Neuroph 2.4 is an open source Java neural network framework `http://neuroph.sourceforge.net/`. It contains reference to another NN library, called Encog `http://code.google.com/p/encog-java/`. Both of them are published under Apache 2.0 license ASF (2004).

NN–based decision module was checked in order to decide on a particular control action to be taken.

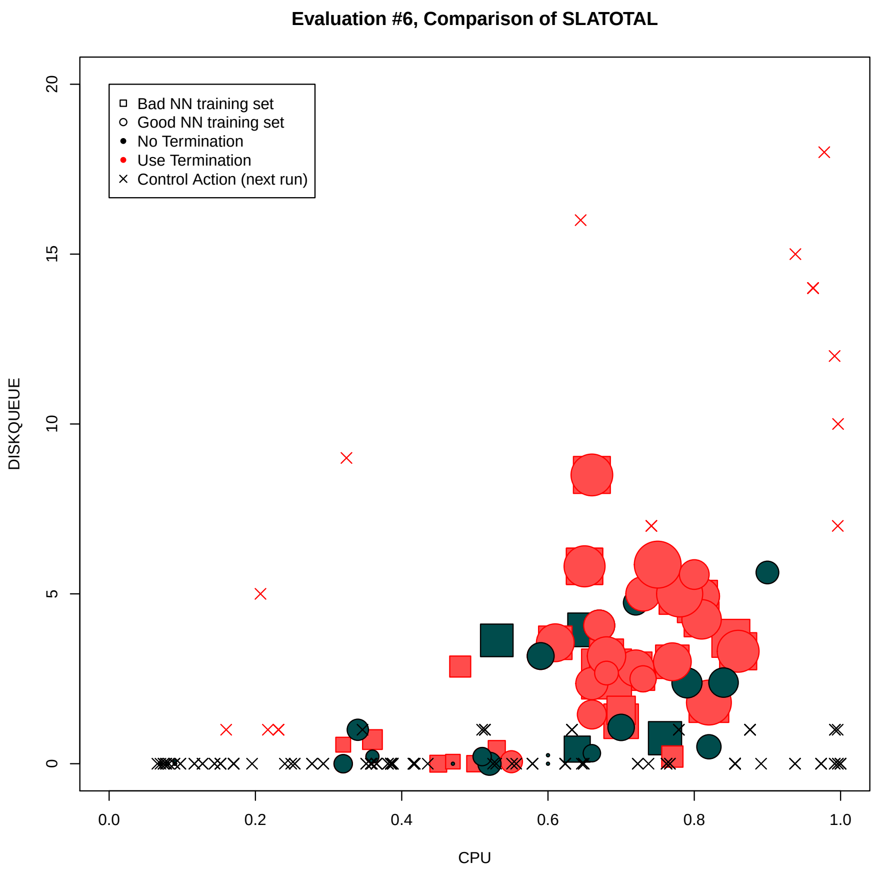
7.4.9 Controller and Actuator Agents

Actuator agents are weaved into an application (analogous to passive monitoring). The controller receives serialised trained NN instances from the Evaluator process that are stored in memory for easy and fast access by actuator APIs before each of the controlled actions calls. To minimise the time spent on accessing the structure a `Map`, as a data container, has been used.

One of the key conclusions drawn in preliminary simulations was that the fast access to metrics (in order to retrieve the current system state) with minimal delays is essential for the control system quality. The controller must take control actions based on the exact state that the system is currently operating; otherwise the reactions are not adequate or precise enough.

Access time to the metrics map is constant during normal conditions. By average an actuator instance needs 0.5ms to gather all values of the system state vector. In order to limit memory allocated it has been assumed that only the last 5 minutes of any of the metrics are stored. Thus only the latest system activity metrics data are available for an actuator. All other older metrics objects are flushed from the container by an independent to the controller thread running.

The termination actuator allows the controller to stop executing an action before its core functionality is called. The controller has access to all resources and actions metrics acquired in the plant (this is up to the monitoring configuration). To check the current state and perform the control operation only very limited resources are needed. The controller, as a part of the model application run–time, uses a small associative table (`TreeMap`), which stores all rules as pairs of trained neural networks.



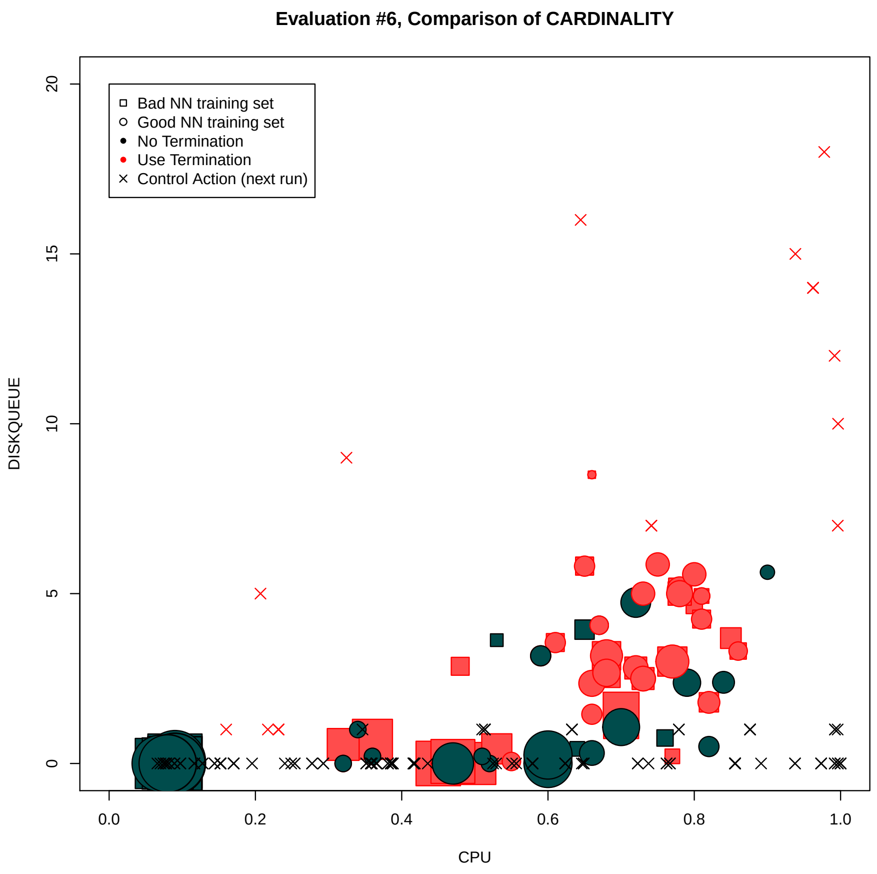


Figure 7.3: The figure shows training set of a single evaluation and later control actions in the background of the observed system states.

7.5 Experimental Results

This section discusses the testbed structure and the results of simulations runs providing example of simple control scenarios.

7.5.1 Testbed Structure

The testbed contains the model application[16], the load generator, and the monitoring and controller components described in the previous section. Details of the implementation are provided below.

Load generator: The implementation calls a Grinder[17] load script (calling web application) using an array with load distribution values and effectively transforms this into run–time characteristics. During the simulation the same bimodal load pattern was used, executed consecutively many times, in two subsequent phases with different load–levels: the first phase had a load of 10 running threads while the second had a 20–threads load. The aim of this testing to explore how the controller changes termination action characteristics adapting to different system conditions.

Monitored resources and actions: During this exercise only Operating System resources and system responses times were monitored. Therefore only OS and system actions metrics were included in the system state vectors. Actions were monitored by http filter agents (no direct java code instrumentation). The load generator calls only one jsp page */public/trackio* (+static assets pointed to it). The page has been extended with a block with high IO intensive code (what is consequently causing significant CPU utilization). In a situation where all resources are available, the code to execute takes from 100ms to 1100ms with uniform distribution, but this rises significantly when the resources used are entering into a saturation state; this is illustrated in Figure 7.4.

Controlled actions: The model application action */public/trackio* contains an object where a termination control actuator was directly woven in. A termination exception is thrown in situations when the actuator finds that the current system state corresponds to a termination action determined by the approximated control actions function defined

[16]The dddsample application (Evans, 2012) was used as a model enterprise application with a few modifications mainly concerning the load characteristics. It is a sample application which was developed for demonstration of Domain-Driven Design concepts introduced by Evans (2004), and is freely available under the MIT License `http://www.opensource.org/licenses/MIT`

[17]Grinder (Aston and Fitzgerald, 2005) is a general purpose Java load testing framework specialised in using many load injector machines for flexible load tests (jython scripts) in distributed environments/systems – it is freely available under the BSD license, `http://www.opensource.org/licenses/bsd-license.php`

in the neural decision block (see Figure 7.2). This mechanism was explained earlier in sections 7.4.8 and 7.4.9.

Evaluator: System states are generated for selected dimensions only. The evaluation described above is executed based on three SLAs definitions set: (a) `SLA3` – 10$ penalty per every second of an image processing task, which takes longer than 10ms on average in a given time bucket; (b) `SLA1` – 1$ penalty per every extra second over 1sec execution of 'trackio' action, but not more than 60$ penalty; (c) `SLA10TERM` – 20$ penalty per every terminated action. Evaluations were executed every 5mins[18]. In the simulation, a one–hour sliding window was used to access system states repository data. These SLAs definitions were used in all simulations presented next in the chapter. The definitions used are nontrivial in order to demonstrate the adaptive control behaviour in the background of nonlinear system performance requirements (where nonlinear factors are present in dependencies between the system execution and characteristics of services performance perceived by clients).

7.6 Simulations and Results

This section describes the experiments and discusses results confirming the adaptive nature of the controller using data from three scenarios. In these simulations the focused was put on the runs of medium length, i.e. up to 4hours for a single simulation run.

7.6.1 Single load testing

This scenario investigates the controlled system dynamics using a single load test simulation with two subsequent load levels.

System states were generated for the three main dimensions being evaluated (`CPUUSER`, `DISKQUEUE`, and `MEMORY`). Figure 7.4 shows a comparison between SLAs values (circles), action execution times and main resources (CPU utilisation and disk queue length) as a function of time (in 5 minutes time buckets – aggregates). All metrics were recorded during a 3-hour simulation run under two load levels. The main objective was to control the system to optimise SLAs values; thus the controller tries to keep the `TOTALSLA` penalties on minimal level, balancing with termination penalties (`SLA10TERM`) long running actions (`SLA1`) and potentially massive static assets execution penalties in cases of high resources consumption (`SLA3`).

[18]During the simulations the Evaluator needed from 20 to 90sec to process above described algorithm. When the Evaluator was running CPU was significantly utilised, which was impacting the application running under load and effectively was changing whole system characteristics. The system had to adapt to such conditions.

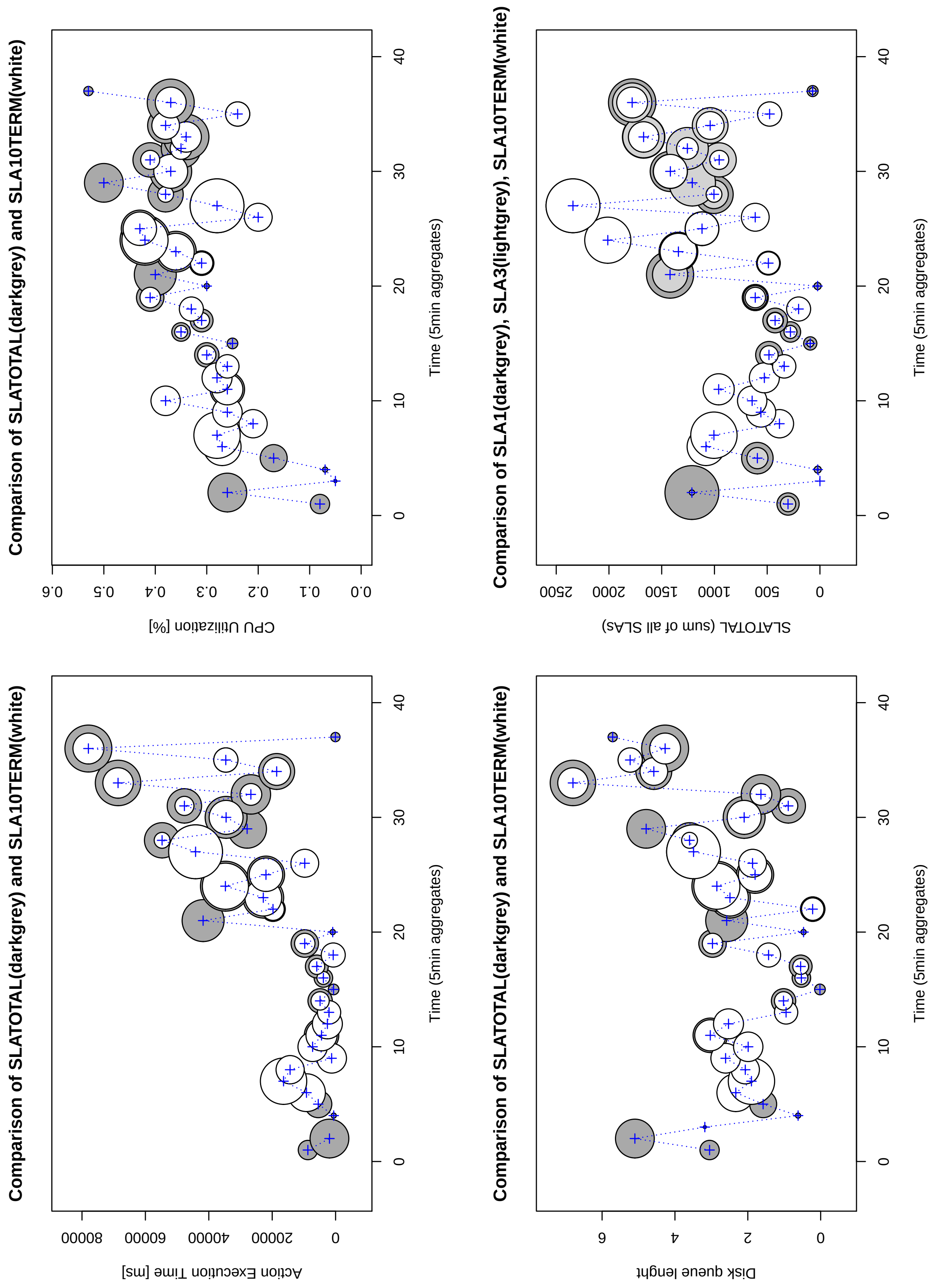


Figure 7.4: Termination action and total SLA values changes comparison during real–ASM system control test run. Data collected during 3 hours simulation under two load levels. Note that there is significant increase on the action execution time, after the higher load is added to the system (at around 20 time bucket), but still SLAs are maintained on reasonably stable level.

The result not only provides a reasonable constant level of total SLAs but also maintains low level of resources utilisation (the controller did not allow saturation of resources). Simulations showed that the neural controller approach can adapt and optimise the operation of the system in certain conditions based on objective functions defined by SLAs.

The two load patterns used during the run are best visible on the top left chart on Figure 7.4, when action execution time rises significantly after adding twice the load to the system. At the beginning of the run no control was applied (`SLA1OTERM` was low), because the evaluator had not established any "good" and "bad" states for potential termination control yet. Just after the 5th time bucket the controller decides on the first termination actions (white circles), lowering `DISKQUEUE`, `SLA1` and `SLA3`. It is worth noting the gradual decrease of cumulative SLA value during the first phase. The trend was broken after 20th bucket when more load was added to the system.

Surprisingly the first time buckets of the second phase show no termination action (no white circles) – that was because the new conditions were so different that the controller could not match the new system states with those represented in the trained NNs. A new evaluation process was needed to train the NNs and propagate the decision blocks to the application in order to give the guidance in the new situation. Just after the controller begins terminating actions, this causes massive growth of total SLA (mainly penalties for termination), still maintaining reasonable low resources consumption. Around the 30th time bucket the controller (considering recent past states) changes the strict mode, so significantly fewer terminations are applied. Consequently, more actions are called and all resources are more utilised. At the end of the simulation the controller contains state definitions, which allow the system to operate on a level of total 2000$ penalty per time bucket, with quite high execution times but reasonable resources utilised.

7.6.2 Load testing with different SLA penalties

This scenario demonstrates the adaptive nature of the controller running in the background of different SLA definitions. In order to test adaptability for different SLAs, we chose to change only a termination penalty and to keep the rest of SLA unmodified (`SLA1` and `SLA3` mentioned above) effectively shaping the system performance needs. Variant B of conflicts resolution was used, so in conflicting areas states with lower cardinality are removed.

To simplify the discussion and the presentation of the data collected, the Evaluator engine was setup to manage only two resources dimensions through the actuator agents. `DISKQUEUE` and `CPU` were selected as best meeting the SLA definitions dynamics. Thus, the Evaluator process and the agents consider only those two resources values while applying the control.

Four load tests are executed per experiment. Each load test takes around 2 hours, and after each run a full system restart takes place. Firstly, three identical load test were executed for different termination penalty SLAs. For each run the SLA TERM was modified starting from 5, 10 to 20$ per termination. After that another load test was run (no control – NT) – this time the Evaluator process was switched off, so the actuator agents didn't receive a trained NN control module with rules. Effectively no control can be applied, so every action requested is executed.

During each of the load runs an Evaluator was triggered every 8 minutes, so there was minimum of 14 rules inductions processes. During the first 7 evaluations, the system was tested with 10 and later with 20–threads load. The system specification was same as the one used in the first scenario, so the same saturation effect took place – execution times of actions after saturating a crucial for the functionality performance are growing significantly.

Figure 7.5 shows data of three snapshots of rule sets generated for evaluations number 6, 9 and 13. This is analogous to the chart presented in Section 7.4.4. Evaluation number 6 was the last one before the load was changed. Note that on the snapshot number 9 a few more rules promoting termination appeared in the area of higher `DISKQUEUE` and `CPU`, which are quite important from SLA value perspective (`SLATOTAL` – the top row), but still not often recorded (low cardinality, bottom row). In the evaluation number 13 we can see a much more mature behaviour, where rules are moved towards higher values of the observed dimensions. So the controllers would act according to the changed situation in the system.

Figure 7.6 presents a comparison of testbed dynamics during four load runs, "05", "10", "20" and "NT", from four different perspectives. The top two charts contain changes of two main dimensions values in time of the simulations, using two minutes time buckets. The bottom left chart presents the system states with both dimensions illustrating the relationship between them. The bottom right shows density of `DISKQUEUE` per each run. The curves are plotted against the dimension value using Locally Estimated Scatterplot Smoothing (LOESS).

During 10–threads load the control system executes on a fairly stable level – both dimensions are stable for the control runs. The situation changes at around the 30th time bucket when 20–threads load is put to the system. The utilisation raise for a few minutes, but than the controller learns the new dimension values and adapts to the new conditions lowering the utilisation, especially `DISKQUEUE` which is primarily used by the tested action functionality.

It is worth mentioning that the controller setup with the "cheapest" SLA termination penalty (run "05") tends to use more termination actions, but the overall cost for the controlled system should remain at a lower level than the SLA costs of running an overloaded system. Apparently, when the penalty function is low the evaluation process is limited mainly to the size of the rule set, not to the termination penalties – this generates

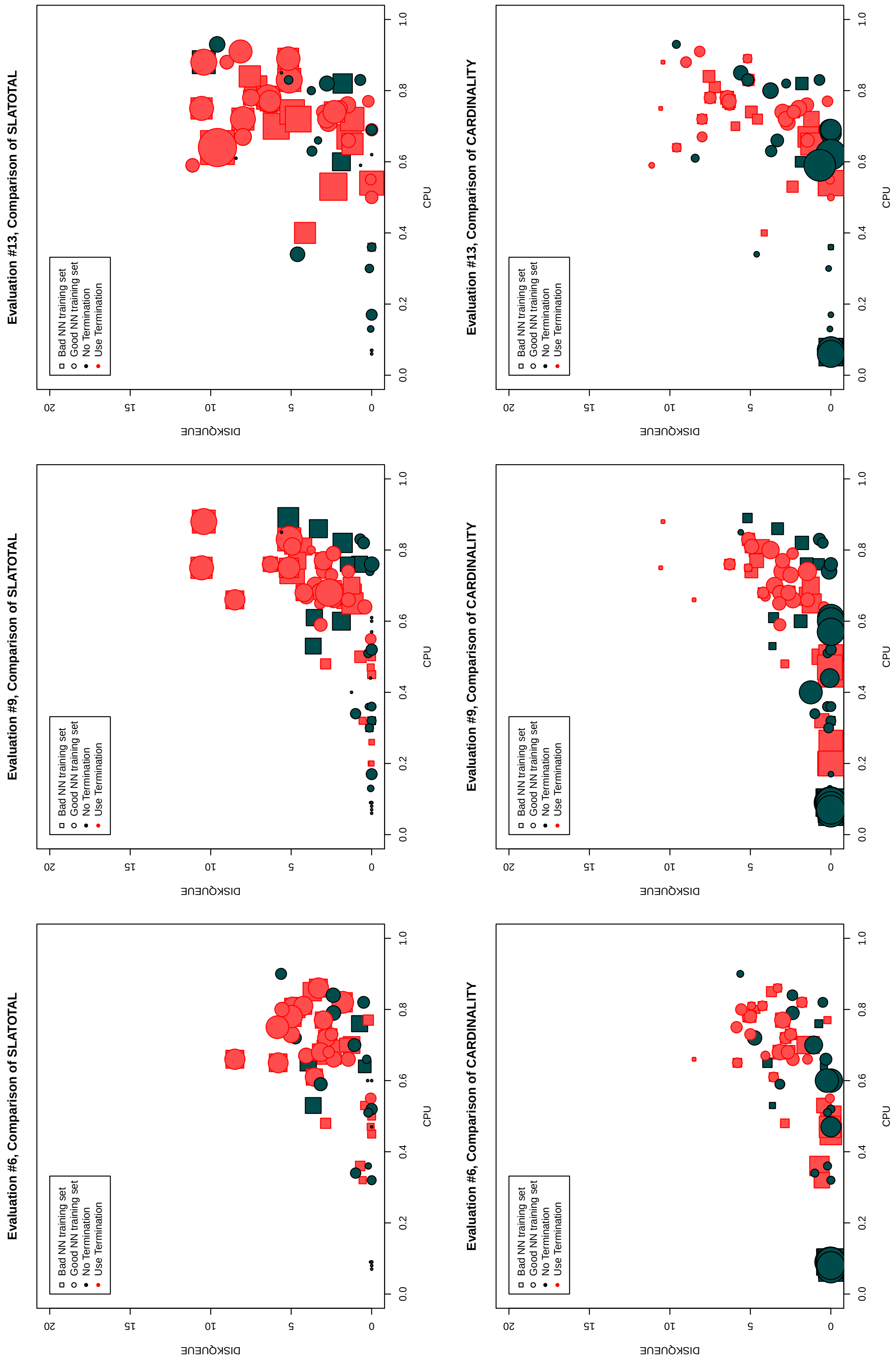


Figure 7.5: Training sets comparison generated by three different subsequent evaluation processes. Data collected during a 2-hour simulation run under two load levels. Note the adaptive behaviour between evaluations after changing the load level (just before Evaluation #9). Shapes denote target NN in the decision block, and colours termination action. Circles show actions of the "good" neural network, while squares the actions of the "bad" part of the decision module. Light red shapes indicate supported termination control, whilst black promote no control. Further details can be found in section 7.4.8.

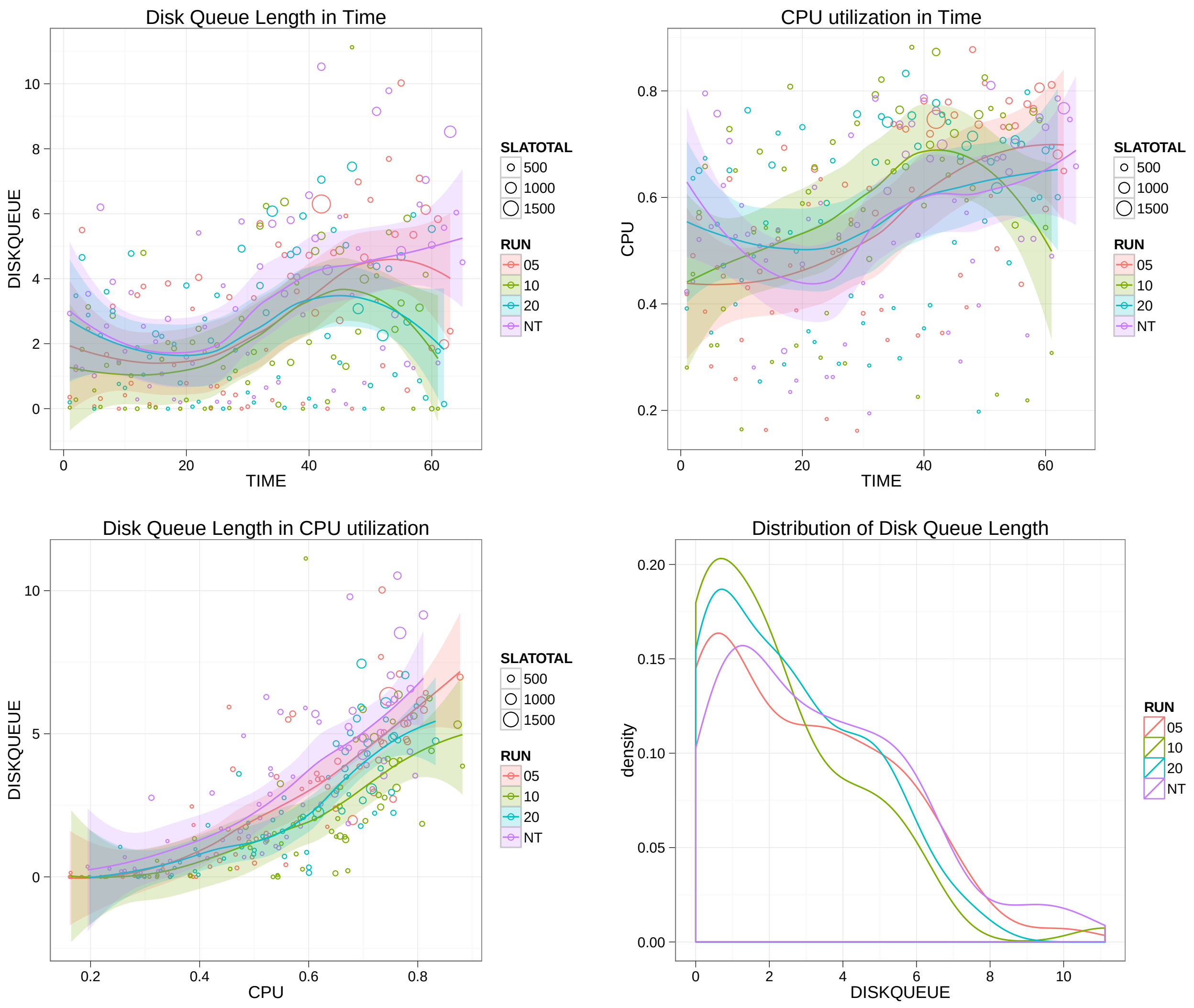


Figure 7.6: Comparison of the main dimensions and total SLA values in four test runs for three different termination penalties SLAs and a case of load with no–control. All data were collected during a 2-hour simulation run under the same load patterns with two subsequent load levels. The plots show LOESS smoothed (Cleveland and Devlin, 1988; Ripley, 2012) trends of adaptive change of selected resources utilisation (LOESS level of confidence interval is set to 0.95 by default).

imprecise control actions.

The reader may also notice that the controller reacts to changes faster in the “10” and “20” runs. During the load test with the non-controlled application (“NT”) the trends are constantly growing from the point of resources saturation, which is shortly after the 20–threads load begins.

7.6.3 Load testing with different conflict resolution strategies

This testing scenario concerns a set of load runs that demonstrate the controller behaviour using three different strategies of resolving conflicts. These are Variants A-C which were explained earlier in Section 7.4.7. The load patterns, SLA definitions (penalty for termination is 10$), and the rest of the testbed parameters remain the same as in previous tests. Note that a slightly different dimensions set has been used – `CPUUSER` instead of `CPU`.

Variant A in the tested scenarios produces higher values for the SLAs and the adaptation response is slower than with other variants (promoting states that are considered as part of "habits" or "trauma"). This is caused mainly by the fact that training sets are much shorter, and simply not rich enough in terms of control information. As a result the controller does not have many control actions approximated by the neural decision block, then there is lower control activity and effectively there is less to learn from the system responses after a control application. Situations where too many contradicting rules appear in the same neighbourhood of states are mainly occurring at the start of a simulation run (i.e. a type of cold start problem), or when the run–time process characteristics change drastically.

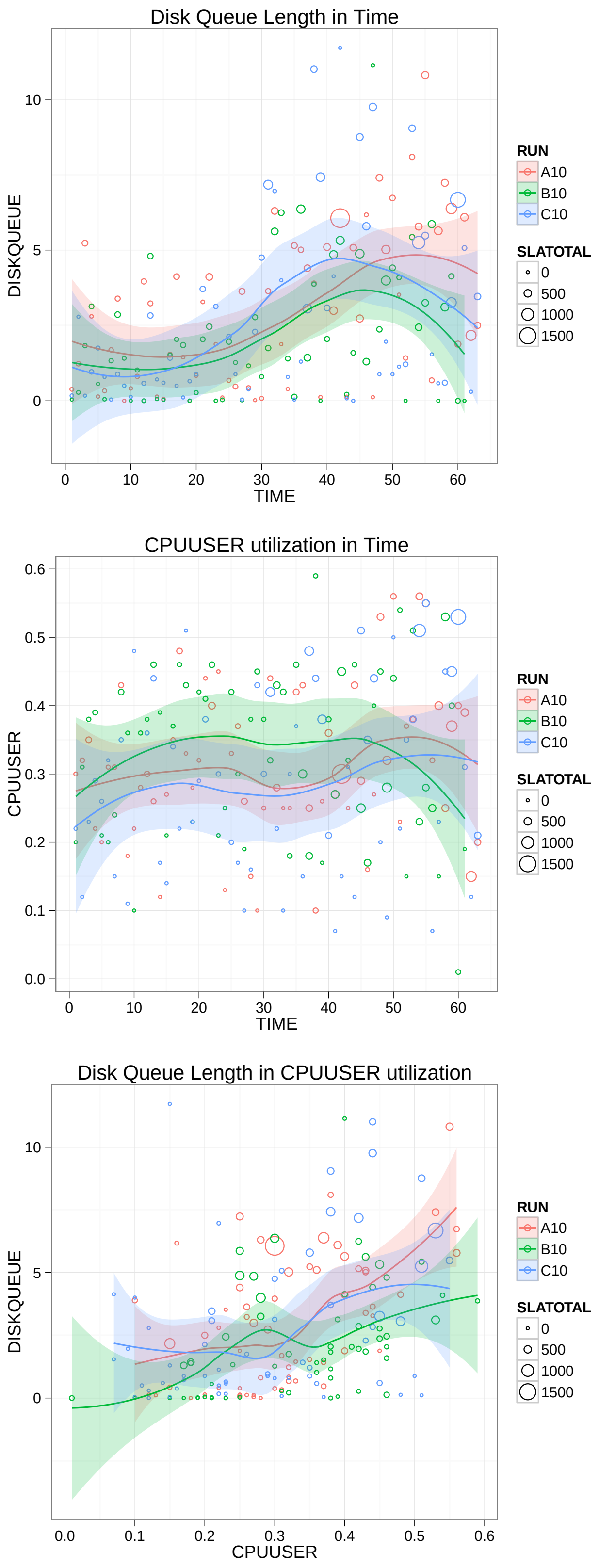


Figure 7.7: Comparison of test runs with different variants of conflicts resolution strategy. Data collected during a 2-hour simulation run under two load levels. LOESS smoothed trends of adaptive change of selected resources have been plotted against the data points.

7.7 Summary and Contribution of the Chapter

This chapter discussed a model free approach to an ASM control system allowing flexible generic SLA definition for scalable distributed environments. It exploited the use of a knowledge base containing many gigabytes of system run–time metrics. Empirical results showed that neural networks–based controllers equipped with a simple states evaluator are able to adapt to changing run–time conditions based on SLA definitions.

Huge amounts of data cause challenges for search in the systems states knowledge base; we will study more efficient mining methods for patterns search and dimensions selection (identifying most influencing dimensions for highest SLAs). Multivariate metrics search and analysis can be the basis for further prediction and classification research in the field. Furthermore, the proposed evaluating algorithm search of a minimal total SLA as a single aggregate objective function (without investigating landscapes of each of functions in the given system space) is in fact the simplest of Multi Objective Optimisation (MOO) techniques (Carlos et al, 2002). Therefore applying more sophisticated techniques of this type could bring interesting practical outcomes.

Chapter 8

Advanced Methods for Adaptive Control in ASM Environment

"Disce quasi semper victurus, vive quasi cras moriturus."
"Learn as if you were to live forever, live as if you were to die tomorrow."
— Isidore of Seville (560 — 636)

"Nothing in life is to be feared, it is only to be understood.
Now is the time to understand more, so that we may fear less."
— Maria Curie–Skłodowska (1867 — 1934)

This chapter presents other adaptive control features that have been tested with the use of a more advanced, modelled system, applying grey–models and reduced dimensionality spaces. It is a continuation of previous chapter in a context of real–time systems, where high penalties present introduce additional constraints for adaptive controller operations. In accordance to the declared scope and focus of the thesis the action termination and scheduling as non–acquisitive control methods are investigated.

8.1 Introduction

As discussed earlier in Chapter 2 both cloud computing providers and cloud services internally used by enterprise data centres, still, allocate much more computational power, machines and energy than it is required to provide the contracted level of service and ensure customer satisfaction. This is a result of holding extra servers in reserve, ready to be brought into operation in case of high demand, and is a consequence of the effect of high switching on/off inertia of the reserve systems (Mao and Humphrey, 2012). Theoretically, this strategy would appear as inefficient but in practice it is considered as an effective approach to secure some strongly desired non-functional requirements, such as resiliency, scalability and reliability (Zhang and Lin, 2010). The real–time requirements only cause higher resources reserves impacting the financial and energy footprint of the solutions.

The cloud computing paradigm tackles this problem distributing computational power more efficiently through the allocation and reuse of resources, hardware and services to many clients. Cloud auto–scaling based on resources availability has been researched thoroughly, including considerations about deadlines and budget constraints, both from practical and theoretical perspectives (Mao et al, 2010; Dutreilh et al, 2010; Zhang et al, 2010; Gong et al, 2010).

In an attempt to provide solutions that would optimise energy usage and available equipment utilisation under dynamic workload changes, and at the same time they would offer the flexibility of a computation framework, industry has developed a range of cloud computing approaches ranging from IaaS, SaaS, PaaS, to serverless and FaaS computing (Eivy and Weinman, 2017; Van Eyk et al, 2018). Other relevant areas that have provided additional benefits are advances in flexible containerisation, micro–services, and distributed computing (García-Valls et al, 2014).

All these cloud computing and distributed computing developments aim at the efficiency of power and infrastructure use, and ultimately seek to fully utilise the entire computational power available. However, the closer we are to that goal, the probability of facing risks related to fulfilled capacity and overused infrastructure rises. This can lead to operating scenarios that generate much harder problems for task scheduling, more difficult control situations, much longer processing times, and in effect can have an impact on SLA. Any violated SLAs may cause high fines to be paid as a result of contractual penalties. Whatever has not been optimised in a SLA negotiation phase on a market level (Sim, 2013) will have to be dealt with in later operation. ASM tackles control aspects of the operating environment by attempting to balance cost and service levels objectives. This chapter considers that these objectives can be reduced to monetisation, where all resources, usage profiles (services calls and load profile) and SLAs can be aggregated down to cost and revenue calculations under consideration in SLA definitions.

Real–time conditions for SLA aware control are even more difficult to be met. System reactions must be instantaneous, too fast for the scaling paradigm, which cannot react so promptly, and service providers are required instead to leave computational power in standby reserve. Still SLA definitions in such conditions offer additional incentive in the form of financial gain, which is a key direction for non–acquisitive control methods, like action termination and scheduling.

Currently, the cloud computing industry does not offer effective hard real–time guarantees as the emphasis is on resources provision, e.g. existing hypervisors provide no guarantee on latency, and there is lack of SLAs that consider latency. However, these types of requirements are going to be more and more present in the near future, as the demands of latency sensitive applications increase, e.g. cloud gaming, communication, streaming, making simple cloud outsourced resources allocation, with calculated headroom, not adequate (Liu et al, 2010; Duy et al, 2010; Zhu et al, 2014). This may be an interesting trend in FaaS and serverless computing area in general, which still lack widely accepted and integrated execution time driven penalties included in the SLA contracts.

Cloud computing enables flexible, dynamic outsourcing while improving cost efficiency. These operational and economic benefits, however, are today not available to the prominent class of safety–critical and mission–critical real–time applications due to the lack of timeliness supports by current cloud infrastructures. Unlike traditional real–time scheduling environments, providing real–time guarantees in cloud environments faces several novel challenges due to the unpredictable system performance as a result of virtualisation, failures, heterogeneity, and multi-tenancy.

This part of the thesis extends the current cloud ready control framework with real–time resource management techniques to enable real–time guarantees on the cloud. The chapter (i) reviews a formal framework for modelling and performance evaluation of real–time applications in cloud environments, (ii) develops approaches for scheduling batch jobs of real–time cloud applications, (iii) designs techniques that are auxiliary measures to address issues of scaled systems introduced by virtualisation and machine failures; and (iv) evaluates these techniques in the context of practical real–time applications.

In this context, the chapter investigates how embedding neural–based admission control into the application run–time, terminating actions before execution on a server but also during their processing in the application, can help managing different types of real–world load patterns and at the same time can meet requirements of profit/penalty oriented SLAs. Moreover, the chapter tackles batch jobs control that can extend termination in more predictive operating areas of control for planning longer execution procedures that are orchestrated within the system and are not directly triggered by ad–hoc users, 3rd APIs, or other clients requests. The planning is improved by dimensions relevance, SDCIC, and signals decomposition, ASM-SD, methods introduced in Chapters 5 and 6 respectively.

The rest of the chapter is organised as follows. Section 8.2 reviews literature that relates to this research. Section 8.3 introduces the formulation of the problem used in this part of the thesis. Section 8.4 introduces the reader to aspects of the testbed design, and describes architectural aspects and components. Section 8.5.1 presents the proposed control framework and strategy. Section 8.6 presents test cases and experiments conducted to validate the proposed adaptive control strategy. Section 8.7 highlights a few challenges and limitations faced with application of termination control to enterprise and cloud computing. Section 8.8 introduces a non–acquisitive control method of adaptive batch job scheduling that augments and complements the capabilities of action termination. Section 8.9 reviews the experiments conducted on the test bed to explore search methods for batch jobs scheduling. Section 10.3.2 discusses evaluation extended by model–driven prediction enhancements to the approaches discussed in the chapter. Section 8.11 ends the chapter by providing concluding remarks and insights about contribution but also aspects that deserve further investigation.

8.2 Related Work

Scheduling mechanisms for tasks and requests, resource allocation, load–balancing, auto–scaling and admission control on the cloud have been the topics of considerable research attention in recent years. Most of the research done so far has tackled the control problem using a combination of components working in an inter–dependent manner, often operating in a sequence where admission control is done first and then scheduling and resource allocation are utilised. For example, such an approach can be found in the work of Urgaonkar and Shenoy (2004) that aims to manage CPU and network bandwidth in shared clusters. Also, Yu et al (2008) used admission control in the context of control of virtual network embedding, while Sharifian et al (2008) applied admission control for a cluster of web servers together with load balancing, and Almeida et al (2010) proposed joint admission control and resource allocation for virtualised servers. Lastly, Ferrer et al (2012) introduced a framework, called OPTIMIS, for cloud service provisioning where admission control is a key component.

More recently, there has been a lot of interest in cost–aware control. For example, Malawski et al (2015) proposed methods for cost– and deadline–constrained resources provisioning in IaaS using a priority–based scheduling algorithm. Also, Yuan et al (2016) built workload scheduling integrated with admission control for distributed cloud data centres, whilst Bi et al (2017) test application–aware dynamic resource provisioning in cloud data centres. Whilst Ranaldo and Zimeo (2016) applied more proactive measures and propose capacity–driven utility model for SLA bilateral negotiation to optimise the utility for cloud service providers, costs and penalties prices, Messina et al (2014, 2016) discuss an agent based negotiation protocol for cloud SLA.

There have also been approaches that exploit learning algorithms working in supervised or unsupervised model. For example, Muppala et al (2014) applied a model independent reinforcement learning with cascade neural networks technique for load proportional auto–configuration of virtual machines and session based admission control. Other researchers have proposed to steer admission control with neural controller and support vector machines (Mohana and Thangaraj, 2013), and use scheduling algorithms based on meta–heuristic optimisation Hoang et al (2016). Database access with profit–oriented control has been researched by Xiong et al (2011), while requests preemption in admission control context was studied by (Salehi et al, 2012). Singh and Chana (2015) introduced policy–based QoS–aware resource scheduling framework that can be applied to cloud computing. A hierarchical framework of cloud resource allocation with power consideration using deep reinforcement learning has been studied by Liu et al (2017). Alfakih et al (2020) studied task offloading and resource allocation for mobile edge computing by

SARSA deep reinforcement learning.

Another part of research has concentrated on requests termination or actions cancellation. Although work in this area has been limited, Cherkasova and Phaal (2002) proposed an admission control mechanism based on server CPU utilisation using four different strategies, and Leontiou et al (2010) used Kalman filtering and ARMAX models to predict changes in incoming load in order to support stable adaptive admission control of distributed cloud services. Zheng and Sakellariou (2013) built admission control with work–flow heuristic to evaluate schedule plans that meet budget-deadline constrains, while He et al (2014) employ network calculus to perform admission control on the cloud tackling large number of parallel service requests.

Work in this area has been summarised in a number of survey papers, which cover general cloud control aspects (Buyya et al, 2009b), present a network embedding perspective (Fischer et al, 2013), focus on auto–scaling techniques for elastic applications (Lorido-Botran et al, 2014), or review machine learning approaches for energy–efficient resource management in cloud computing environments (Demirci, 2015). Although, as mentioned above, there have been several control approaches and experimental platforms, which have been deployed in both model–based and real–world systems with the use of synthetic and benchmark workloads, there is still no clear established standard to enable an easy performance comparison.

Despite the massive progress in pre–call request termination as the main admission control strategy, the author is not aware of any documented research that focuses on action termination, tackling pre– and during– request execution inclusively. In this vein, it is imperative to explore the potential of these control approaches in isolation without utilising scalability and resource allocation methods, which could introduce obfuscation to cumulative results.

In this context, the chapter contributes a framework that allows embedding the neural admission control into the application run–time. It builds on a previously introduced learning controller framework for adaptive control in ASM environments (Sikora and Magoulas, 2013b), see Chapter 7. This framework is extended; it is equipped with re–training capabilities and it is incorporated into a model–based testbed, where the collection of results is more efficient allowing the execution of long test runs involving real–systems experimentation. The emphasis is placed on more complex, business–oriented scenarios, where SLAs are defined as functions of not only call counts, as in our previous work, but also execution time, and are converted directly into revenue, as per the defined monetisation model of SaaS, PaaS or FaaS systems (see further discussion in Chapter 9).

The enhanced framework enables to terminate actions before execution on a server but also during the processing of the requests in the application. It also supports building synthetic workload profiles that are able to reproduce behaviours of real–world load patterns, where priority is set as part of profit/penalty–oriented SLAs with termination supporting real–time systems and harsh time–wise conditions. The new control approach is validated through synthetic and real–world scenarios, built from several system and workload profiles, which have been used by other researchers in the literature. The proposed action termination solution is further evaluated using four different control types, aiming at assessing both performance and cost–effectiveness.

It is an often expressed opinion that cloud computing, in any form IaaS, PaaS, FaaS or SaaS, offers huge advantage to batch processing in a form of high scalability for close to linear "cost associativity". Thus in effect one could assume that a job that take a thousand hours can be distributed to a thousand of machines and run in 1 hour window of parallel processing (Fox et al, 2009; Armbrust et al, 2010; Avram, 2014). In fact, the cloud model offers the great advantage of elastic computing by allowing clients to reduce the amount of computing power that would be poorly utilised on–premises. Of course all that is true but in an optimistic scenario, which is a class of edge cases, but it does not really represent many other scenarios where the long intensive computation cannot be parallelised. Furthermore, in certain scenarios even in cloud computing and under conditions of significant trust in effective parallelisation, there are challenges on scaled computation due to limits of (a) software specifics, (b) services, and (c) resources allocation as highlighted earlier in Chapter 2. For example, processing data dependencies can be computationally heavy and may be a huge challenge to be processed in parallel. Also, the use of relatively old software and classical database systems, which do not always scale up well, are considered a substantial force slowing down efforts of parallelisation.

The limitations of scaling together with costs optimisation considerations give a strong incentive to develop non–acquisitive control methods like adaptive termination and scheduling.

The next section introduces the main concepts behind the approach and presents a theoretical formulation of the problem.

8.3 SLA–Driven Computing Services Management for Financial Performance Gain

In the problem formulation, introduced in Chapter 3, the financial performance is tightly associated with the ability to offer the best available service under a contractual SLA. Financial performance P is defined as a composition of revenue R and costs C, defined as an SLA, f_{SLA}; let's recall here Equation 3.1, $P = \sum(R - C) \sim f_{SLA};\ R, C \in \mathbb{R}_{+} \cup \{0\}$. Hence, the core objective is to optimise the service of the system, i.e. maximise the effectiveness in the background of load, resources usage, and performance characteristics of the service by reducing the costs defined in the SLA. All those aspects are time–variant; thus, adaptivity is a key requirement in such a control system. The controller should be capable of readjusting the characteristics of actions execution time at run–time using only termination actions and without changing other functionalities.

The scope of possible actions as undertaking revenue and costs related decisions is in line with what the parties have agreed to do when the conditions are met. This allows to transform the SLA, defined as a set of service–level objectives represented as *if...then* structures, into a function of actions execution time and incorporate it into a more complex function, where costs and revenues of the same argument are considered (see Equation 3.1 and Equation 3.8 in Chapter 3).

8.3.1 Defining the Control System Targets

Suppose an enterprise system receives incoming requests to process action a in order to generate a particular service outcome. This is achieved by employing resources $r(t)$, according to an effectiveness criterion defined in an SLA function f_{SLA}. Hence, the general performance of the system can be considered as a dynamic process of all activities $\mathbf{a}$ and resources $\mathbf{r}$, and is produced under some conditions, as a result of running the system code and providing functionality on a given hardware–software–architecture configuration.

In order to maximise the productivity of the system, P (Eq. 3.8), the controller evaluates system states, $\mathbf{S}$, attempting to reconcile service workload economics, defined by incoming activity, $f_R(a_e)$, with utilised resources, $f_C(r)$, on one hand, and other limiting costs of SLA violations, $f_C(a_{et})$, and actions terminations, $f_C(a_{term})$, on the other hand. Consequently, the control process aims to generate a sequence of system states that would lead to $\max f_{SLA}(t) \in \mathbf{S}$. Thus, a key performance metric of the control system is the sum of SLA values, in time, for all action types, which can be considered as a cumulative indicator of productivity of the service provider that shapes the landscape of profitability from service provisioning perspective:

$$\min T_{SLA}(t) = \min \sum_i f_{SLA_i}(t) \ , \tag{8.1}$$

where $f_{SLA_i}(t)$ is a sequence of collected values in time t (i.e. a time–series) that incorporates costs and revenue for an action type SLA, f_{SLA_a}.

8.4 Testbed Design

At this stage of the research, the model–driven experimentation is used as the main approach to analyse the framework and its components, and evaluate its performance. This is to gain more flexibility of mass testing and in order to speed up the experiments of various control scenarios and combinations of actions control. The testbed constructed as part of the research allows profitability analysis of SaaS, FaaS or PaaS for software and enterprise cloud providers, respectively, interpreting flexible SLAs measures, actions count and SLA functions evaluations in the background of complex system dynamics, which are represented by actions, a, performance, and resources, r, consumption.

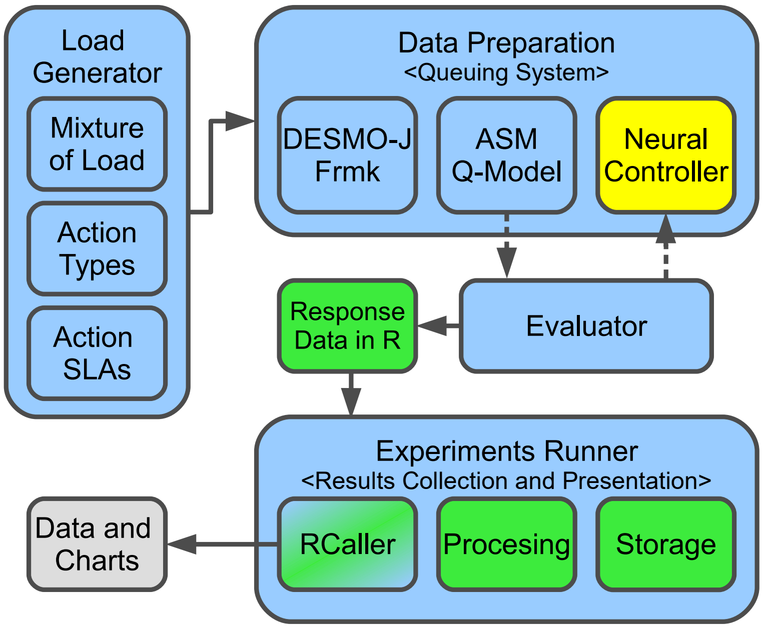


Figure 8.1: Testbed design contains model components, experiment preparation framework, evaluator with direct connection to controller, and data collection for visualisation and debugging.

Experiments for various scenarios are conducted using the testbed. The scenarios target functional and serverless computing, where an action (or a function) is an atomic portion of requested computing service that a cloud service provider manages. The cost/revenue model incorporates costs incurred for starting, or stopping, virtual machines with workers to serve requests and execute an action by measuring resources required to satisfy the call. The testbed framework contains an Experiments Runner that is responsible for presentation but also collection of data gathered in earlier experiments; the testbed design is illustrated in Figure 8.1. More details about the various independent components[1] and their function are provided in Sections 8.5.2, 8.5.3 and 8.5.4.

8.4.1 Discrete Event Simulation and Software Framework

In order to model computer systems a discrete–event simulation approach has been applied that is built upon the Java library DESMO-J[2] (Page et al, 2005; Göbel et al, 2013; Lechler et al, 2014). The library has been extended with a framework wrapping the APIs, and implementing specific features of such computing systems, replicating queuing models for server, process, disk controller, action requests, and action types. As mentioned in

[1]The proposed framework allows experimentation and testing of the model, which is equipped with a neural controller directly applied to SaaS Cloud Service provisioning, in the light of revenue and cost of service usage.

[2]DESMO-J which is freely available on Apache Licence version 2.0 (ASF, 2004)

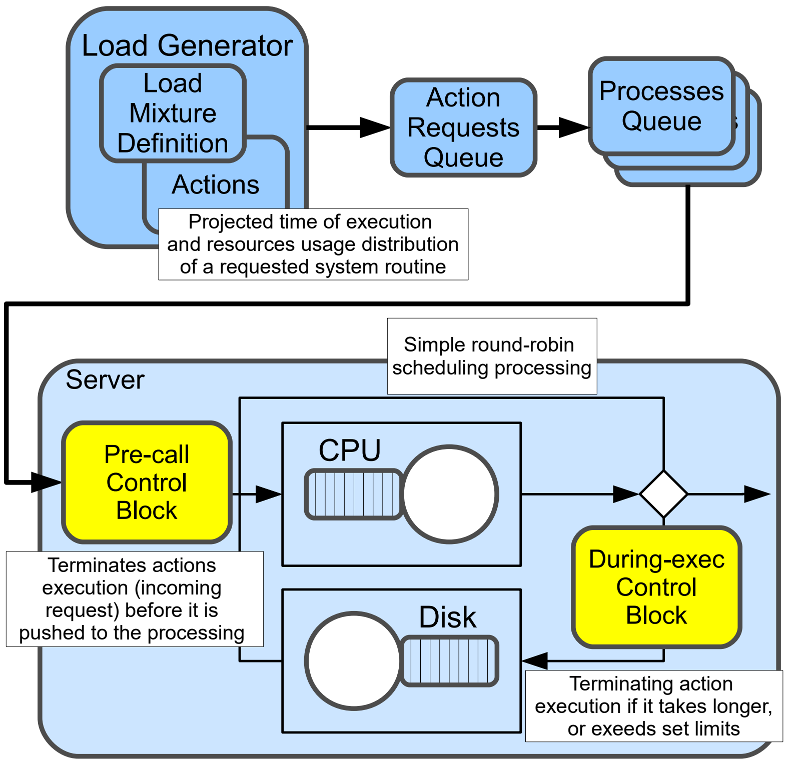

Load Generator
Load Mixture Definition
Actions
Projected time of execution and resources usage distribution of a requested system routine
Action Requests Queue
Processes Queue
Simple round-robin scheduling processing
Server
Pre-call Control Block
CPU
Disk
During-exec Control Block
Terminates actions execution (incoming request) before it is pushed to the processing
Terminating action execution if it takes longer, or exeeds set limits

Figure 8.2: Architecture of the ASM control system applied to a computer system model. Two different control blocks are deployed– one is responsible for pre–call and the other for during–call termination.

Chapter 6 Sikora and Magoulas (2021) have found that lower–level discrete event modelling of the machines and scheduling approaches with DESMO-J gives better control and extensibility than other comprehensive cloud modelling frameworks like CloudSim[3] (Calheiros et al, 2011). Moreover, the focus of this research is more on application dynamics, and therefore, there is a stronger requirement for detailed definition of action types that reflect execution of the application functionalities, convoluted load profiles, and SLA definitions. All that has to give stable and rigorous ground for adaptive control weaved into the model. Thus, DESMO-J as a Discrete–Event Simulation (DES) library, which offers a high level of flexibility and insight into the low–level operations, appears more suitable for this task.

The testbed framework contains a model of an application that is a set of action types defined to express the resources usage that are going to be replicated accordingly in the discrete event simulation run–time. As explained earlier in Section 4.7 the control system actuators are artefacts that will need to reside in the application run–time. Since the simulation framework classes execute on the same Java platform the actuators objects can be directly weaved–into the model of the application.

The framework uses the Not–weighted Round Robin Scheduling (Stallings, 2014). Each

[3]CloudSim is able to simulate an entire cloud computational centre as well as federation of centres, offering a complete cloud model with provision of hosts, network topology, virtual–machines, and resources utilisation of CPU, memory, disks and bandwidth.

process is given a fixed time to execute, called a quantum. Once a process is executed for a specified time period, it is preempted and another process executes for a given time period (Jensen et al, 1985). Each action request is transformed into a process that is decomposed into smaller chunks, which are served by resource controllers, according to the distribution set of the action type definition. Context switching is used to save states of preempted processes [4].

In this part of the research, two types of resources have been considered into the model: processors and disk controllers, see Figure 8.2. This reflects the real systems context and allows to get insight into the execution of complex action workloads. In such environments, other resources, such as network, multiple servers or memory, as well as software components, such as message queues or databases, may be interesting for implementing more sophisticated models of modern enterprise distributed systems but are not considered as essential for investigating termination control in server–based or Serverless computing environments, which is the focus of this chapter. This issue is discussed further in Section 8.5.7.

8.4.2 Load Generator and SLA Contracts

The load for each of the action types can be generated according to a statistical profile considering: (a) the probability distribution of arrival requests; (b) the load pattern evolution in time for the particular experiment with repetition (allowing to loop a pattern so that it repeats itself); (c) the execution time in relation to resources usage distribution. Load profiles will be explored further below, whilst some examples are provided in Section 8.6.1.

An important consideration under load conditions is maintaining the stability of the service, as this has implications on the performance of the systems and significant impact on usability. Thus, in practice, providers secure additional computational resources than effectively needed in order to ensure stability. Naturally, this strategy affects the costs of the provided service. Therefore, we expect that the use of reward/penalty–driven SLAs will become more widely adopted. Such SLA contracts specify precisely up to what level of execution time, the provided service is acceptable to the client, and, at the same time, profitable to the service provider. Of course execution time is a function of the required resources and of the algorithm that is being executed.

The easiest way to reduce resources consumption is to optimise the application code adapting it to the specific conditions, or to the requirements of the platform on which it is running. However, since the application, software, or routine, is complex, or may

[4]This is a fairly simple yet efficient scheduling algorithm. Round Robin compared to other standard approaches, like Shortest Job Next, Priority Based Scheduling, Shortest Remaining Time, or Multiple-Level Queues Scheduling, performs particularly well in conditions of overloaded systems, when jobs characteristics are unknown (Arpaci-Dusseau and Arpaci-Dusseau, 2015)– an area of special consideration for this research.

have been deployed by the customer (as per SaaS, FaaS on service provider system), this may not be possible. The introduction of SLA contracts gives to clients a clearer understanding as to what the highest–acceptable level of execution time is, and to service providers guidance on what resources have to be allocated in a given point in time.

8.5 Actions Termination Control for Real–time SLAs

In certain situations, mainly when there is excessive utilisation of saturated resources, it is economically viable to terminate incoming calls[5] so that the bottleneck resource can be released. This will allow other actions to execute which can potentially bring more profit. This is a type of control that will be investigated further in this part of the experimental study. In this context, there are several challenges involved, such as the number of action types and the different economic contracts defined in their SLAs, the impact of unknown run–time characteristics, and the nature of interference caused by others action types accessing the same shared resources.

Real–time processing requirements are translated into SLA of soft– or hard– constraints depending on the definition per action type. Where the penalty price for too long executing actions is much higher for hard real–time than soft.

One can argue that real–time systems should not come with a price for violation. Of course this is a subject to a "real–time constraint". In this part of the research, focusing on the cloud–driven (especially serverless computing oriented FaaS) and distributed systems where network latency problem introduce substantial instability that needs to be factored in, the execution time is measured from an event of receiving an action to the system response.

In practice, real–time processing fails if execution is not completed within a specified time frame, or by a specified deadline, and this depends on the action type. Deadlines must always be met, regardless of system load or functionality. Attempting to optimise resources allocation in a system experiencing higher load is a typical control scenario (Buyya et al, 2011; Al-Dahoud et al, 2016; Hwang et al, 2016; Tran et al, 2017).

Optimising resources consumption in this environment requires balancing the goals of service providers and clients, e.g. the service provider may want to allocate many different types of activities, or even many tenants on the same resources, whilst the client may not be able to deliver a solid implementation. Although all functionalities following this regime must finish on time, some may end before the deadline producing a named error, which, then, should be interpreted by the client. This is softening the definition of *real–time* but it may resolve the problem of optimising resources consumption, and offer more flexibility to both parties. This is a desirable feature and it can be implemented in the framework

[5]Similar to calls termination is a concept of throttling, or the introduction of time-outs, that would cancel the execution of a request to prevent overloading.

at the API level, especially in serverless ecosystems where audit and control are possible.

Since typically real–time services must guarantee response within specified time limits, the service provider can avoid contract violations by terminating the execution before a hard deadline, just before processing the requested function but also during execution. This scenario is the main focus of this chapter, where the assumption is that already allocated resources are not scaling up or down due to control decisions but can be released by terminating incoming actions requests. This is a technique widely used in practice by human administrators. Nevertheless, it has not attracted considerable attention in cloud computing, cloud–based enterprise systems or operating systems implementations yet. Sill, it is foreseen the method will be introduced to FaaS and SaaS soon, see Chapter 9.

Although both resources allocation and actions termination scenarios can be used in conjunction, this chapter concentrates first on the termination control. Following a scaling–up strategy, e.g. adding extra resources, requires higher investment and introduces inertia, which may limit flexibility to deal with high instantaneous (spike) usage, and ultimately increase SLA costs. In contrast, termination control applied either before or during execution offers an attractive alternative, as it will be demonstrated in our experiments presented below.

8.5.1 Action Termination Control Framework

The control framework implementation sets the base for an autonomous agent that monitors the complex enterprise environment through sensors and acts upon the current run–time situation using actuators. Through appropriate control actions, it directs the system activity towards goal states that fulfil SLA requirements. Figure 8.3 shows the agent-based control blocks, which are able to control processes on application level, deployed into a cloud environment.

The adoption of an agent–based implementation allows the controller to collect instances of earlier control decisions and use a learning mechanism in order to refine the acquired knowledge and achieve the desired goals. Integrated into the framework are a component, named Evaluator, containing history of system states **S**, and a Control Block API, supporting actuator features, to adapt the current run–time dynamics in a way that makes the service more profitable for the service provider.

Similarly to the design of an action actuator used in (Sikora and Magoulas, 2013b) and explained in Chapter 7. The control block incorporates neural agents that are able to work independently and execute concurrently, utilising a model built based on data generated by the system under control.

This type of control introduces benefits especially in soft real–time systems, where the expectation is that the system responds within a given period of time, and this is explicitly

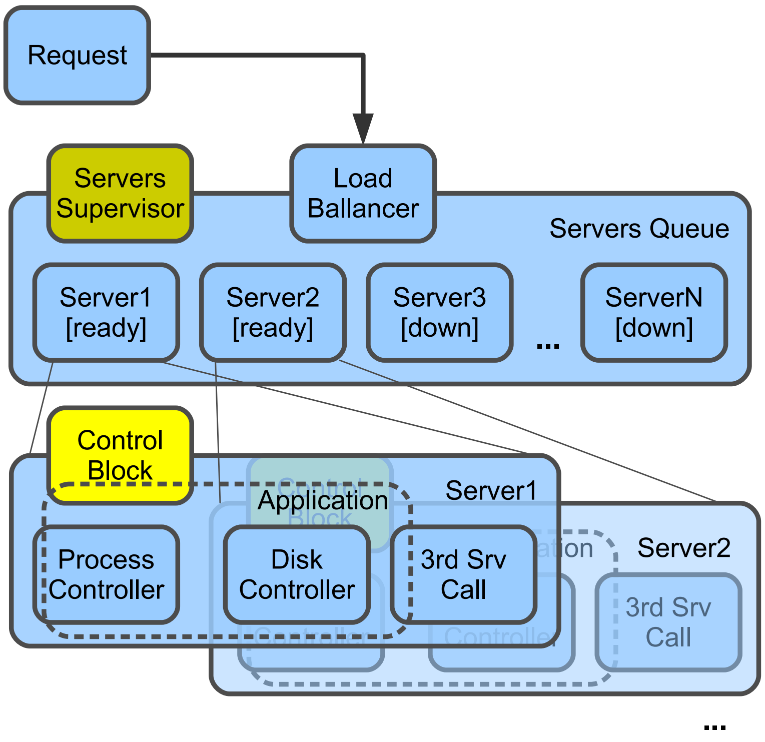


Figure 8.3: Deployment architecture of control blocks into application running on a cloud–based system.

defined as a service–level objective in the SLA definition. Examples of such functions, which better illustrate the concept, are provided in Section 8.6.3 below.

Special consideration is needed for termination errors, which may have appeared suddenly due to a termination actuation, so as to avoid instabilities to the rest of the client's integrated systems. The control framework enables termination errors to be handled on the client side at a price of maintaining a tighter conversation – more frequent responses on requests – with the client systems, and in this way supports higher availability across a variety of applications. This strategy may be very effective especially in micro–services architectures, where many weakly–coupled components are present. In such systems, instabilities caused by network issues, or over–utilised services, must be considered by design in the handling framework. Thus, the cost of extending error management with termination mechanisms is fairly low. Furthermore, such an approach enforces integration of time–constraints, since an error message caused by a termination event is in essence part of the contract that must be followed between interconnected services. This is consistent with event–driven distributed services deployed on serverless computing (Baldini et al, 2017) workers, or micro–services (Dmitry and Manfred, 2014), that are getting more and more attention amongst programmers (García-Valls et al, 2014).

Termination control, whether before or during execution, provides better potential for serverless computing to support real–time systems constraints. For example, it can provide an additional level of flexibility in the case of soft real–time systems, as mentioned above, where the code is deployed to PaaS services, or in situations where processes have types of actions with long execution, e.g. asynchronous messaging, batch systems, etc, that are of lower importance. This is further discussed below in Section 8.6.5.

8.5.2 Evaluation Control Decisions and Rules Induction

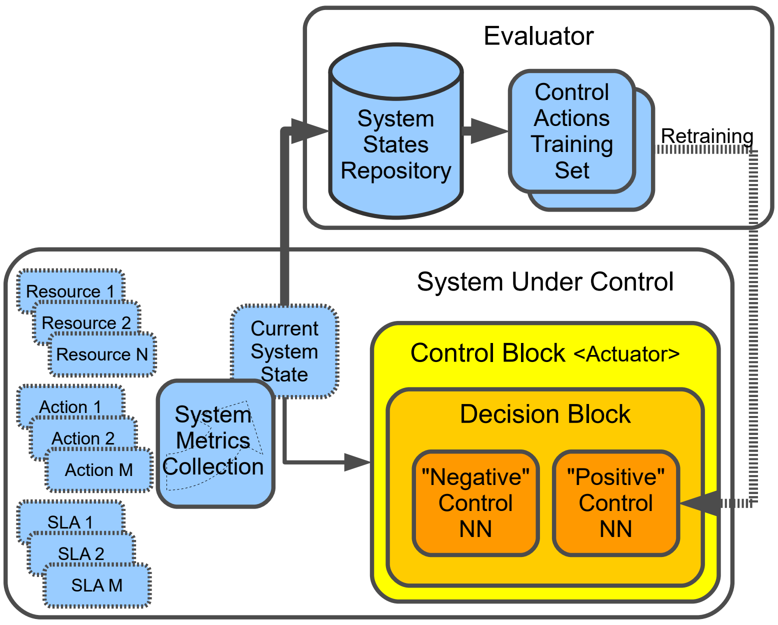


Figure 8.4: Architecture of the neural networks–based decision block. It combines two multi–layer perceptron networks and is weaved into the system code, identifying run–time situations in system's states and executing appropriate control actions.

A key component of the control framework is the so–called Evaluator. Its role is to interpret run–time situations, stored in the Systems States Repository and collected through monitored activity, at the input, read gathered data about resources consumption, and generate appropriate output states in order to create an appropriate trajectory in the system states space, **S**. This type of outputs form an actuating signal that can be used in order to execute or terminate a service.

Following the fail first, learn, adapt, and succeed feedback–loop approach[6], which is an essence of agent–environment type of interaction with delayed reward, like in RL (Sutton, 1992; Sutton and Barto, 1998), analogically as explained in Chapter 7, the extended Evaluator selects systems states that formulate the training set of the Decision Block. This process selects states in operating regions where minimum and maximum SLA function boundaries are evaluated and generates the training set. Then a control action is executed and its impact is evaluated once more before the next training round. The process is iterative, so that the latest control actions can be evaluated together with results of previous actuation events. Figure 8.4 illustrates the software components that support the data flow and the control feedback–loop, whilst Figure 8.6, described in the next section, provides an example of actions sequence, illustrating iterative evaluation of actions with subsequent control phases for one of our experiments.

[6]Similar approach to fail first, learn, adapt, and succeed is also used in Test–Driven Development.

Table 8.1: Rule-based Strategies

Rule	Description
Rule 1	Search for “low” –enough and “high” –enough SLA states based on total of all SLAs, T_{SLA}, see Equation 8.1, and label control states that are below a predefined threshold of low–mark–percentage or above a high–mark–percentage (by default 5% and 95%), as “positive” or “negative”, respectively. This is similar to the strategy used in (Sikora and Magoulas, 2013b), which presented a different control framework that is used as a baseline.
Rule 2	Evaluate “positive” and “negative” outcome of previous control actions based on a comparison with gathered actuation decisions. Select all system states where executed termination actions had led to an SLA decrease, as measured within a given time window. In all these cases, the system’s reaction on the actuation had a positive outcome so all those states are labelled as “positive”. These marked systems states are passed for training the decision block of the controller, unless there is a similar system state that generated the opposite effect on the SLA values. This is an important element of decision making for conflict resolution, which is an internal part of the evaluation process. In the opposite situation, when termination control brought an increase of SLA value, and this is confirmed in other observed system states under control, these states are labelled as “negative” because the effect is counter–productive and the controller should avoid actuation in those system states.
Rule 3	Introduce stronger exploration strategy with epsilon–greedy policy (Sutton and Barto, 1998; Scheffler and Young, 2002) randomly selecting states, where the control actions will be applied and evaluated in the next evaluation round. System states selection is arbitrary, i.e. regardless of SLA values or reactions to control actions. By default 10% of all the collected system state points, amongst those not already selected, are added to both “positive” and “negative” sets.
Rule 4	This rule considers specifics of actions types scenarios, allowing to run independent evaluations and generate training sets per action type. Such an approach creates a multi–agent system (Busoniu et al, 2008), where control blocks operate independently. Each evaluation cycle is guided by reward of SLA values linked to particular action types rather than the overall total of all SLAs– this will be discussed in detail in Section 8.5.5.

Previous control states are evaluated according to a rule–based strategy, as listed in Table 8.1. These rules can be seen as different strategies for selecting suitable data to train the Neural Networks–based Decision Block. The rules can form groups that define different control schemes or policies, which helps dealing with the exploration–exploitation trade–off. Hence, in our implementation the Evaluator can derive training sets effectively supporting four control schemes– these include rules combinations as shown in Table 8.2. Further decision making or states conflict resolution strategy is not included in the evaluation phase but is covered by the neural controller and the training process.

Both “positive” and “negative” operating conditions are important and should be “learned" by the controller. Considering “positive” control states are the main area of concern of the Evaluator. “Negative” control states operate as an additional protective measure, making sure that no control, or termination control objectives, are applied to states that were selected by mistake, producing negative outcomes. This could happen for example in a very rapidly changing environment, where observed “disturbances” may have an impact on the stability of the control. Consequently, sets of “positive” and “negative” states create operating regions in the Neural Networks-based Decision Block of the control system, as explained in Section 8.5.3.

Figure 8.4 depicts a clear distinction of the processes: the evaluation (at the top), which represents more strategic objectives, and the tactical decision making components (at the bottom) which are part of effective actuators called Decision Blocks, explained in further

Table 8.2: Control Schemes

Type	**Termination On/Off**	**Rule**	**Description**
0	Off	No Rule	No automatic control is applied. This type is used in simulations as a baseline for comparison purposes.
1	On	Rule 1	Search states of “low” and “high” SLA value within a given band, e.g. 5%, and allocate states for “negative” and “positive” control regions to be used for training the neural networks-based control block.
2	On	Rule 1+2	Select “low” and “high” SLA states. Evaluate marked “positive” and “negative” control states based on comparison with earlier control actions, so that recent control decisions are validated.
3	On	Rule 1+2+3	Select states of SLA extrema, evaluate earlier control decisions, and add random control points. Introduce “curiosity” effect by randomly selecting states with uniform distribution where control actions will be enforced in the next runs.
4	On	Rule 1+2+4	Select states of SLA extrema and evaluate earlier control independently per action type, considering dedicated SLA values, and execute termination targeted per each action type separately.

sections.

8.5.3 Neural Controller

The controller is a software component deployed into the system. It contains a decision block that is responsible for holding the generalised knowledge about earlier system states and a model of control actions/actuating signals. It is equipped with actuating logic that terminates the execution when the current system state maps to particular operating regions in the model and enforces the effective control over the incoming system requests. It is applied to the system code API (pointed to selected methods, batches, user interfaces, etc.), either weaved in with the use of AOP, or coded explicitly as per the control framework exposed to the application run–time. The controller decision block is equipped with two NNs, MLP trained with backpropagation with momentum, one working as an “expert" on “positive” and one operating on “negative” control states, as evaluated during earlier system runs.

The controller processes the current vector of the system state S and makes decisions about termination. For each received request, the system state is evaluated by the “negative” control neural network first, in order to confirm that the potential control action is not going to harm system's behaviour. Then, if found promising, the second step is to validate this state against the mapping learned by the “positive” control neural network to make sure that it lies within control regions that introduce positive response increasing the SLA. Figure 8.2 illustrates the deployment of the controller on the system infrastructure. Figure 8.4 presents in more detail the architecture of the neural-networks based controller.

Every iteration of evaluation creates a new updated model instance, exploiting information from recently executed controller termination actions through retraining. This approach allows to apply more precise control in the next cycle, and to adapt the controller decisions to changing run–time or load characteristics of the system; this is further discussed in Section 8.6.

8.5.4 Action Termination Actuator

Popular approaches to implement action termination are: (a) the use of AOP– in the simplest case when the neural controller considering the current system state takes a decision to terminate a run–time process, an exception is thrown from the object that has been woven–in the code under the instrumentation. This may be a very good solution for all internal APIs, User Interface facades, and batch job actions; (b) the use of framework API integration by adding APIs in a serverless architecture (Kiran et al, 2015), and event–driven function–oriented service provision (Baldini et al, 2017).

Although in practice the controller and the termination actuator would use some part of CPU, for simplicity, the testbed model does not consider this part in the resources utilisation. The typical execution time for decision–making by the neural controller is $7-9ms$, and then less than $1ms$ is needed for action termination (actuating the decision). Figure 8.2 illustrates the deployment and the queuing considerations of the modelled, or real, resources.

8.5.5 Concurrent Multi Actions Control

In the previous discussion, the main measure driving the focus of the controller on "positive" or "negative" operating regions in the system's state space was the total of SLA values, T_{SLA}. That cumulative measure allows to simplify the formulation of the problem by considering it in terms of the cost and financial performance of the service provision, without the need to implement more complex MOO techniques (Zhu et al, 2016).

Intuitively, controlling each action type independently may produce better results, mainly because different actions can have far different load, performance, resources usage and most importantly SLA characteristics. Furthermore, the evaluation of earlier control actions, per action type, allows to consider specific SLA values and execute targeted termination for each action type separately. For instance, action type A can execute longer using a significant portion of resources, whilst action type B is rather shorter, called occasionally, but contracted with high SLA penalties for longer execution times. In this scenario it is desirable to terminate action A at times when there is a higher demand for many type B actions. Of course certain functionalities are not consuming much processor time nor disk but they may still have long action execution time due to their code structure. In such cases, although running multiple actions on the environment may not have direct impact on key resources consumption, some of these actions may still influence the execution of other actions. The best way to establish the effective run–time characteristics and interdependencies of the various action types is to allow them to execute, collect data representing the situation under a given load, and try to apply control so as to assess whether the anticipated operational changes are providing the expected benefits.

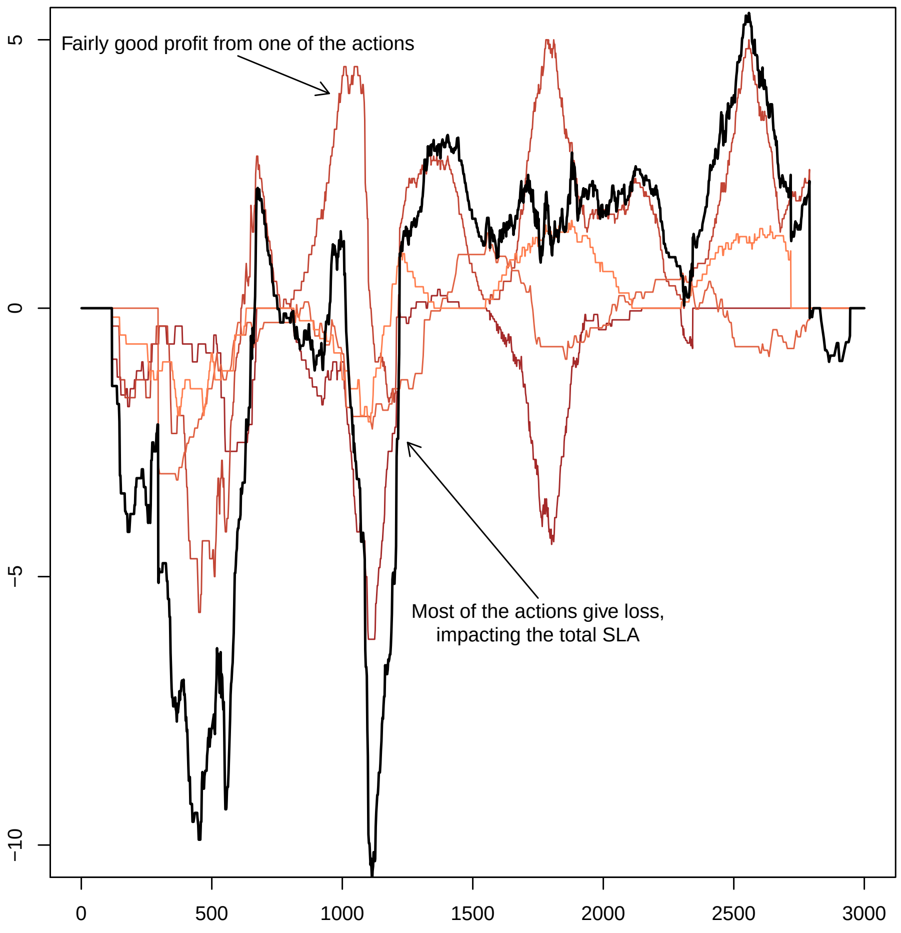


Figure 8.5: Sequence of SLA values for many action types working under different load depicted using a range of red and brown shades. Total SLA, line in black, is the main cumulative measure driving the control schemes described in Table 8.2, apart from type 4.

Figure 8.5 illustrates an example of load activity related to functionalities of five different actions, shown using a range of red and brown shades. It includes a case where the SLA of one of the actions and the total SLA value move to opposite directions. This is representative of situations where part of the system may be perfectly productive, and, thus, no termination control is required, although the SLAs values for most of the other actions drive the controller to take termination actions. A control scheme that has been trained using a Type–4 policy (cf. Table 8.2) would focus on the termination of the actions that cause negative SLA values, leaving the most productive action (in red) running.

It is important to note that the Evaluator must review the low and high SLA values independently for each of the actions, see time–series in the bottom row of Figure 8.6. The direct measure of the total of all SLAs, T_{SLA}, is used for simpler control but still the importance of influence, or strength, of termination control per type of action needs to be proportionate to the total SLA measured in a given system state. This allows to weaken termination control on types of actions that do not substantially impact the total SLA[7], and vice versa, i.e. apply termination control to actions whose SLA values are matching the total SLA profile [8]. To resolve a potential conflict between the total of all SLAs, T_{SLA}, and the SLA of a particular action, the algorithm considers states with “low” and “high” SLA values for all selected types of actions scoring them based on total SLA. Subsequently, a search is applied across all action types using a specified threshold– by default this is 30% of the worst action points. As a result, control is more likely to be formulated for an action type, whose SLA values are the lowest in the system’s state subspace that has been defined by the selected “low” states.

There are two potential implementations for the controller’s decision block: (a) training a single neural network that can be used for all action types but when in operation, the network decodes a control decision depending on the given execution, or (b) training separate, dedicated neural networks to form the decision block, see Figure 8.4.

In this chapter, the second approach is used; hence, independent NNs are trained based on the same training data source. Such an approach offers more flexibility compared to implementation (a) mentioned above; for example, it allows to create an array of neural networks dedicated to different action types, if necessary. In addition, the research shows that this controller design approach usually produces better control system performance, although there are associated costs for designing different neural networks-based decision blocks, running the training procedure and an independent evaluation process. The performance of such a control approach is promising especially in cases when there is a significant difference in the run–time characteristics of the various types of actions. It is worth noticing that for scenarios where far different action types are used, i.e. batches and user interface functions, Type–1, 2 and 3 control schemes (cf. Table 8.2) can be used separately, so that there are isolated instances of the same framework used. Both implementation strategies can be easily applied to SaaS and PaaS models, but also to internal on–premises enterprise systems with ASM controllers.

8.5.6 Terminations During Action Execution

The previous sections considered the termination of an action when the requests are being received by the server, i.e. before any specific computation for a particular action type has been done, see Section 8.5.4. In this context, the controller takes action termination decisions based on expectation, as derived through training from response and usage patterns of previously observed execution instances. This aggressive approach is effective as long as the action execution times are predictable, and enforces the control

[7]Weak–influencer actions types– there is no point in terminating actions whose SLA values do not cause big losses, i.e. where SLA values are not very low compared to others.

[8]For example, terminating actions with worst SLA values whilst saving well performing actions.

system to maximise the saving of resources used by the action. Although this can be especially attractive in soft real–time systems where the code is deployed to PaaS, FaaS and SaaS services, in scenarios tested where the execution times are quite constant under non–overloaded system conditions, little overall benefit was observed, see Fig 8.11. However, the more unpredictable the execution time is, the more this method is expected to outperform simple termination control working before action execution only.

It is worth mentioning that the longer an action runs, the more resources are consumed, so terminating the action at the end provides lower value since computational power has been consumed for processing and execution already. Thus, often, it is much more beneficial to decide not to produce a response in a given time line than to produce it with delay. This is especially true for functions that are directly exposed to users or human operators, and may cause stress to the user, such as when users make successive clicks on the same button. For example, if a service is slower than usual, users tend to confirm the requested action by clicking a button again and again. In most cases, the first request has been processed correctly but there are just not enough resources to execute it, or the performance might have been altered. Consequently, right after the second request comes to the server, there is no point in executing the first one[9].

PaaS/SaaS/FaaS systems may be equipped with a control API specifically tackling both action termination schemes following the application code structure, where a specific control block type is weaved into different types of actions for best performance. For example, short and predictable calls, like user–interface or web–services, controlled by pre–execution control, and longer less predicable ones, like batch processes or asynchronous messaging, controlled during action execution by terminator actuators.

In engineering practice, instrumentation required by this approach can be difficult to implement or may be using significant resources due to the repeatable additional checks that should be performed during action execution. In many cases, this technique can be simply too intrusive to be effective. However, it can be appropriate for batch based systems or for selected longer actions. Further extensions could allow the controller to detect execution times or distribution, predictably, and choose which control scheme is best for a given action type under specific run–time conditions.

8.5.7 Simplifications and Limitations

Enterprise systems are complex in nature so in order to simplify the experimental data collection and analysis two types of resources are considered in this chapter, namely CPU and Disk. Of course, other types such as network, memory, virtualisation layer aspects, or

[9]A potential solution to this problem in practice is to block the function from being clicked again, or simply generate the fastest action possible. Ideally, the entire end–user–performance execution time should be less than 1 second. That is considered to be the limit for the user's flow of thought to stay uninterrupted, while 10 seconds is the limit for keeping user's attention focused on a human–machine dialogue (Jakob, 1993; Miller, 1968).

calls to other services (introducing idle time and latencies) could be added in a similar way to the already implemented resources. Moreover the computational effort related to the collection of the metrics, storage, evaluation and neural–control block training were not included in the tested model. There are two reasons for that: (a) the monitoring facilities and the Evaluator can be isolated from the system under control, having minimal effect on the system performance; (b) investigating engineering details of the neural network training was not part of the research[10].

Although extending the framework to deal with many servers would give simulation results closer to real–world data centres, it is not considered essential for action termination control, which involves single server run–time, node instance or container and interconnections with other nodes do not impact the core observations. Connectivity to other servers, applications or systems, with waiting effects and networks utilisation considerations were also left for future work in order to keep the range of the experiments more focused on the potential of action termination. Therefore in this study a single CPU and Disk queue per server has been chosen in order to simplify the set up, and aspects, such as multi–threading, virtualisation overhead, multi–node coordination that could introduce further complexity and obfuscate the observations were not considered. In the future, the model could be easily extended with more CPUs. However, it is expected that any change of characteristics would be close to linear due to the used scheduling algorithms.

As discussed in Section 8.3, the execution time is very important for profitability of PaaS/SaaS/FaaS running in tight SLA contracts. However, there are also factors such as cost of equipment and energy usage that can be factored into the aggregate cost values, which are added to the SLA functions associated with action types. The chapter does not focus on those dimensions although the framework can incorporate those values as constants in the SLA function definitions.

Several simplifications and limitations of the termination methods have been observed and discussed in forthcoming after experiments review in Section 8.7.

8.6 Actions Termination Experiments

A set of experiments is presented in this section to evaluate the proposed control framework under various conditions and demonstrate its adaptiveness and effectiveness to generate control actions that optimise financial performance in ASM environments. Each of the experiments is executed in the context of a list of action types with specific load patterns, distributions of resources usage and SLA function definitions. All these elements create a complex environment even in a fairly simple server–disk control scenario.

Section 8.6.1 elaborates more on the mixture of components used to define the load

[10]It takes around 1–5 seconds to train a neural–control block per action type in a 2000–time steps experiment on a single i7-6500 CPU@2.59GHz with use of around 11000 system metrics collected.

profiles and the load parametrisation. The discussion of the experiments conducted starts from simpler cases and then progresses to more complex and elaborate scenarios. The first experiment in Section 8.6.2 demonstrates operation in a simple scenario, which is first executed without control and then with control applied on the system in order to compare changes in resources usage and SLA function optimisation between the two configurations. Then, in Sections 8.6.3 and 8.6.4 the simulations exhibit more adaptivity and illustrate financial performance details providing a comparative evaluation for different types of control schemes (cf. Table 8.2) under various load profiles.

8.6.1 Load Parametrisation

Different types of actions are realised by application functionalities that utilise CPU or Disk. The load is specified by three factors: (a) load intensity pattern, multiplied by (b) the quantity of incoming requests to perform the action, generated by a probability density function, and (c) an execution time distribution.

Listing 8.1: Example of load parameterisation, action type: Java code snippet of action A3IO2, its load profile, and load distributions.

```java
ActionType actionA3IO2 =
  new ActionType("A3IO2",
    new LoadPattern(
      // arrival density distribution
      new LoadDistExponential("DExp", 10),
        // load pattern
        new double[]{  // intensity
            0.0, 0.2, 0.2, 0.2, 0.2, 0.2,
            0.1, 0.0, 0.0, 0.1, 0.4, 0.4,
            0.6, 0.2, 0.1, 0.0, 0.0},
            3),        // LM repetition
    // execution time distribution
    new LoadDistNormal("DNorm", 1, 0.1),
    // resources utilization distribution
    new ResourcesUsage[]{
      new ResourcesUsage(
          ProcessingType.CPU, 0.1),
      new ResourcesUsage(
          ProcessingType.Disk, 0.9) },
    new SLA());
```

For example, action type A3IO2 definition, in Listing 8.1 and Table 8.3, provides the relevant statistics for a functionality that is rather Disk–bound (90%), utilising CPU only 10% of the execution time, and has the following execution distribution parameters: exponential distribution with arrival rate $Exp(\lambda = 10)$; action execution time defined by a normal distribution $\mathcal{N}(\mu = 5, \sigma^2 = 0.1)$; a variable load intensity set as a request probability pattern repeated 3 times within the time frame of a single experiment[11].

[11]The load profile offers a precise way of configuring variability in the expected intensity of the frequency of incoming requests for a particular action. It helps defining different test scenarios, highlighting cases such as actions interference when using the same resource, higher load applied temporarily to observe the effects of spikes in resources consumption, or short reoccurring load changes to analyse the impact of a "delay" factor on the strength of ASM signal deconvolution (Sikora and Magoulas, 2014b,

Examples of SLA functions are provided in Table 8.4. The SLA function is formulated according to an agreement made between parties; for example, it can be a function of service quality provided that depends on execution times, for instance: "I as a service provider expect payment (positive: revenue/profit) for short executions, and pay back penalties for the longer executions and potentially for termination actions".

Additional examples of action types that are used in the experiments are shown in Table 8.3, while Table 8.5 describes experimental scenarios and defines SLA functions. There is a mixture of synthetic– and real– workloads. The later represent loads captured on the 1989 World Cup website and on Wikipedia in October 2017, and have been used in the past by adaptive admission control researchers Xiong et al (2011) and Ferrer et al (2012).

8.6.2 Experiment 1: Comparing operation with and without the Neural Control Block

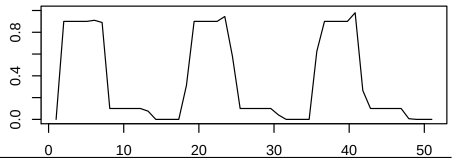
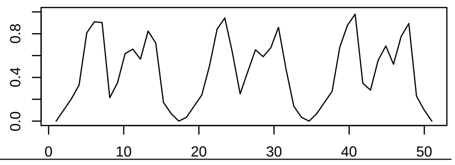
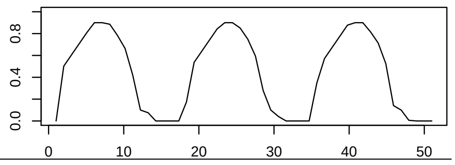
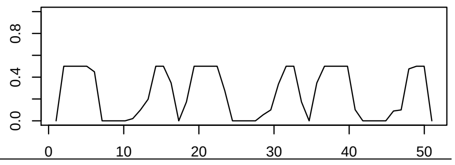
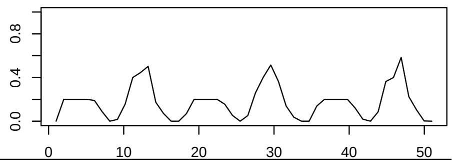
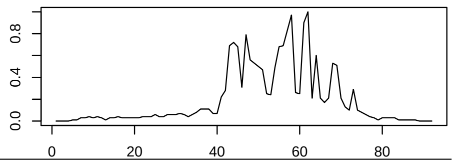
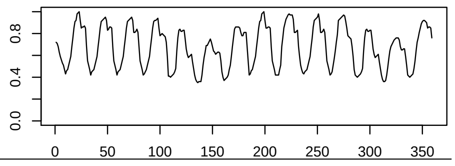
This experiment aims to give an overview of the operation of the controller and demonstrate its adaptivity and effectiveness, starting from the system operating without control and then, as the load profile and SLA function remain the same, the controller is activated. The control system performance is evaluated and gradually optimised through controller retraining.

Figure 8.6 presents time sequences for input activity, impact on resources and financial performance, in terms of SLA values for an action type. The same load profile is applied and the system operates without control (on the left) and with termination control (on the right). Actions executions are presented in shades of red, whilst resources utilisations are in blue and black. The bottom row presents the sequence of the total SLA values for the experiment. The total SLA includes the costs of overrunning actions/SLA violations, termination penalties, but also the revenue/rewards for actions serviced in a normative time. Thus, the higher these SLA values are, the more profitable the scenario is. Termination control is executed after the first evaluation that was called at the 500th time slot. Later, the controller cancels the selected action requests to free up resources for another action and in effect the total SLA is improved. Count of terminations is shown in the sequence on the 6th row.The response diagrams on the 4th and 5th row illustrate utilisation of disk and process queue (CPU usage by actions computations). On the right hand side plots, the resources are clearly less utilised due to the lower number of serviced actions, as shown by the sequences on the 2nd and the 3rd row.

The mean arrival time and quantity of different action types used in this experiment expose the system to load, which is around 4 times bigger than the system is able to handle; that is around 4 times above the saturation threshold. Although the system load is generally more CPU–bound, disk is significantly overused as well. Thus the controller attempts

2015).

Table 8.3: Action Types used in the experiments, calls quantity, expected execution time and resources usage proportion, frequency and load intensity.

Action Type	**Quantity of incoming calls**[a]	**Execution time distribution**[b]	**Resources Usage CPU, Disk(IO)**	**Load Intensity Pattern**[c]
A1	$\mathcal{N}(\mu = l, \sigma^2 = 0.1)$	$\mathcal{N}(\mu = 4.2, \sigma^2 = 0.1)$	CPU=100%, Disk=0%	
	This action type imitates a business driven function that utilises only CPU. Load pattern of request calls and execution time are following a normal distribution with average interval l and μ equal to 4.2.			
A2B	$\mathcal{N}(\mu = l, \sigma^2 = 0.1)$	$\mathcal{N}(\mu = 2.0, \sigma^2 = 0.1)$	CPU=80%, Disk=20%	
	This action type models another business related function, where request calls frequency is constant and execution time follows a normal distribution with average set to 2.0. This action uses 20% of Disk and 80% of CPU to compute the results.			
AOS	$\mathcal{N}(\mu = 10, \sigma^2 = 0.1)$	$\mathcal{N}(\mu = 0.5, \sigma^2 = 0.1)$	CPU=90%, Disk=10%	
	This action type was introduced to mimic load coming from operating system activities. Actions are very short and it is rather CPU driven.			
A3IO	$\exp(\lambda = 5)$	$\mathcal{N}(\mu = 3.0, \sigma^2 = 0.1)$	CPU=1%, Disk=99%	
	This action type computes responses by mainly using disk resources– only 1% of execution time is used by CPU. The frequency of calls is defined by an exponential distribution, so it is less predictable than other actions shown above.			
A3IO2	$\exp(\lambda = 10)$	$\mathcal{N}(\mu = 1.0, \sigma^2 = 0.1)$	CPU=10%, Disk=90%	
	Similar to A3IO but calls are 3x shorter, two times less frequent, and is using less disk time. The code snippet of Table 8.1 provides implementation details.			
WC1, ..., WC12	$\mathcal{N}(\mu = l, \sigma^2 = 0.1)$	$\mathcal{N}(\mu = 4.2, \sigma^2 = 0.1)$	CPU=90%, Disk=10%	
	This set of action types imitates a business driven function, which utilises mainly CPU. Request calls and execution time are the same as in a sequence of A1, A2B, AOS.			
W.En, W.De, W.Ja, W.Es	$\mathcal{N}(\mu = l, \sigma^2 = 0.1)$	$\mathcal{N}(\mu = 4.2, \sigma^2 = 0.1)$	CPU=90%, Disk=10%	
	This action type imitates a business driven function, which utilises mainly CPU. Request calls and execution time are same as in A1.			

[a]Quantity of incoming calls defines load produced by requests following a probability distribution: normal, $\mathcal{N}(\mu, \sigma^2)$, or exponential $\exp(\lambda)$.

[b]Execution time of a single action call defined by normal distribution.

[c]Load intensity patterns depicted in the table have been applied for Load Multiplier (LM) equal to 3.

to reduce the load by terminating some of the actions. Naturally, both the termination penalty and the penalty for longer execution times are considered (see the example of SLA function definition in Table 8.4). We can see that the controller drives the system into "profitable" states just after the first training cycle, where SLA function values get significantly higher around the 750th time slot. The training phase is repeated every 500 time steps, and it should be noted that not all actions are terminated, whilst resources consumption is reduced soon after the first training cycle. The training set used each time consists of "positive" and "negative" states identified by the Evaluator, as described in

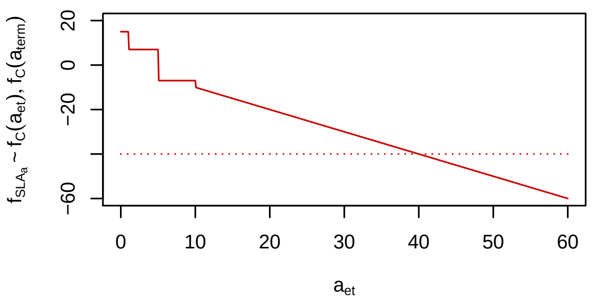
Table 8.4: Example of execution time SLA cost functions; code and visualisation

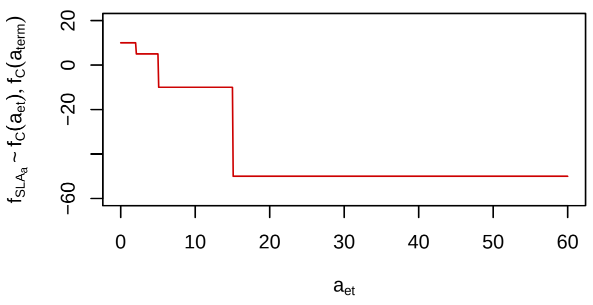
Listing 8.2: Example of SLA definition with termination penalty.

```
// such SLA costs are common
// for Soft Real-Time Systems
ExecutionTimeCostsSLA sla =
  new ExecutionTimeWithTermCostsSLA(){
    public double price() {
      // linear (not constant) penalty
      if  (getServiceTime() >  10)
        return -1*getServiceTime();
      else if (getServiceTime() >  5)
        return -7;
      // sub-second call, good price
      else if (getServiceTime() <= 1)
        return 15;
      // normal positive price
      else if (getServiceTime() <= 5)
        return 7;
      else return 1;
    }
    public double terminationPenalty() {
      return -40;
    }
  };
```

Listing 8.3: Example of SLA definition without termination penalty.

```
//
//
ExecutionTimeCostsSLA sla2 =
  new ExecutionTimeCostsSLA() {
    public double price() {
      // high penalty
      if (getServiceTime() >  10)
        return -50;
      else if (getServiceTime() >  5)
        return -10;
      // sub-second call
      else if (getServiceTime() <= 1)
        return 10;
      // normal positive prize
      else if (getServiceTime() <= 5)
        return 5;
      else return 1;
    }
    // no termination penalty
  };
```





Section 8.5.2. An example is shown in the scatter plot matrix of Figure 8.7, where positive and negative states are denoted by Control Mark “Right" and “Wrong", respectively. This figure shows all the key dimensions of the system states in pairs illustrating their relations.

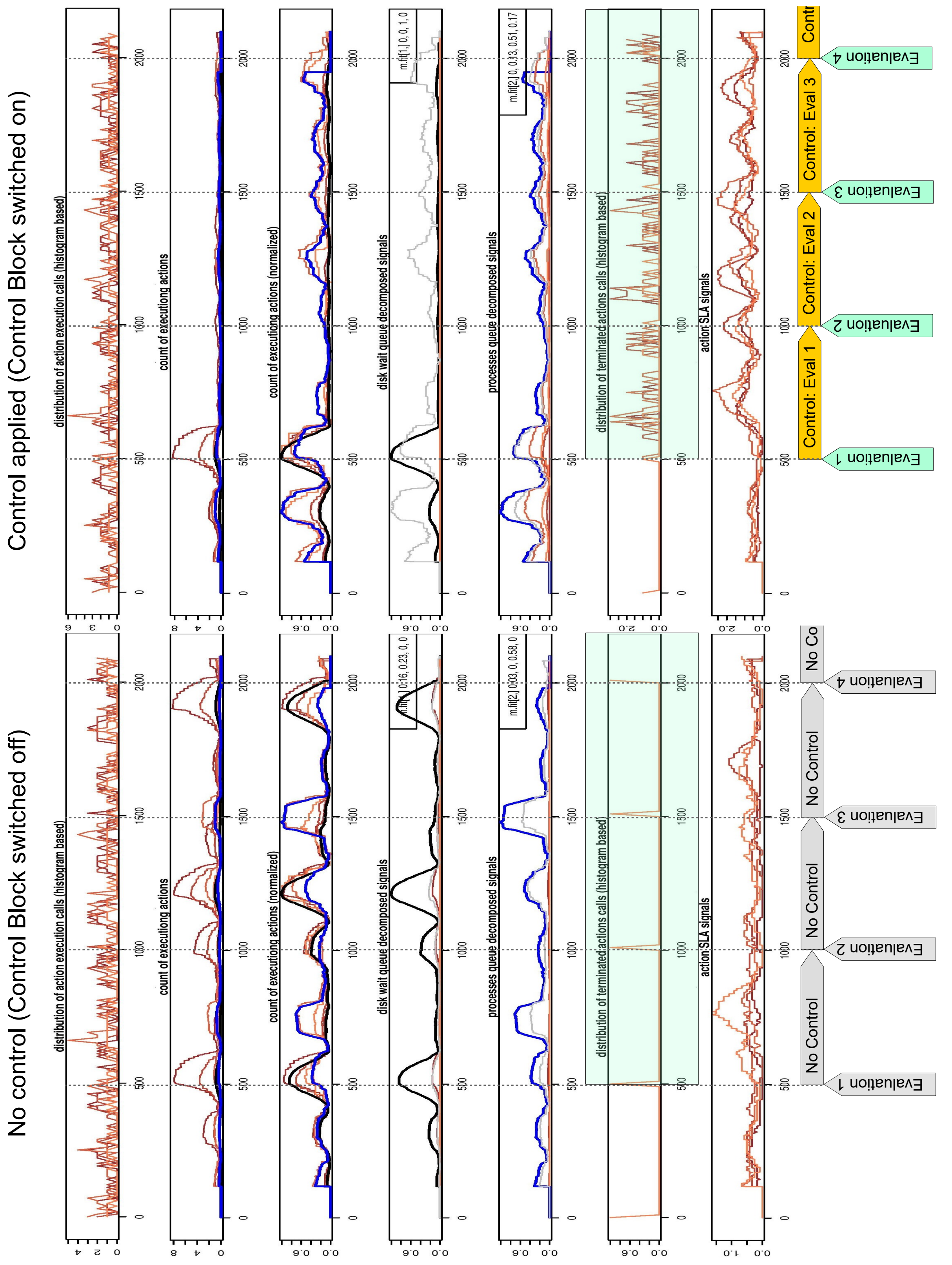


Figure 8.6: Metrics time sequence and system responses for operation without controller, on the left side, and with neural control block that uses Type–2 control scheme (cf. Table 8.2), on the right side. In both cases, the experiment was executed under the load pattern and SLA function of scenario 4 (cf. Table 8.5).

All system states presented have been processed by the Evaluator and some of them were labelled with "positive" and "negative" control marks. Note how the spikes in Disk Queue impact total SLA, lowering the values due to high execution times (see red points). This is simply due to the fact that all actions were executing simultaneously saturating the CPU. After the 500th time slot, when the controller is engaged for the first time, resources queues are reduced substantially. High utilisation of the resources is still allowed, offering high computational power for service provisioning, but overloading the system and very long executions are avoided. The number of states with high SLA values, violet points, after 500 time steps in the scatter plot of the pair CPU utilisation/Time (column 1, row 3 of the matrix) indicate revenue generation and more effective utilisation of resources, and thus a much more profitable operation. The same effect can be seen in Figure 8.6, where the time sequence in row 3, on the right side, shows a huge spike of concurrently executing actions between the 480th and the 600th time slot. Although the same load was applied repeatedly, i.e. 4 times, during the run, such a big accumulation of waiting actions was not repeated, which is attributed to the application of termination control. In general, after the controller takes over, financial performance, represented by the values of the total SLA, gets better, as exhibited in Figure 8.7 (see plot in the first column, last row of the scatter plot matrix). Due to the changing nature of the load profile, saturation issues occurring quickly and system inertia, resources utilisation is not highly correlated with SLA value changes, but directly relates to the queue length that far better corresponds with high SLA function values observed.

Dimensions representing resources queues appear to be sources of rich information for the system entering saturation. Thus, control actuators monitoring resources usage and queues are able to react to the changing conditions and execute effective control actions.

8.6.3 Experiment 2: Adaptivity to Load Profile and Financial Performance

The testbed allows to perform many experiments in accumulated collection of runs exploring various problem dimensions such as: comparing types of control schemes (cf. Table 8.2), verifying system performance under changing load patterns and different values for the mean time of arrival, repetitions with different random generator seeds, and, lastly, testing various model parameters using a set of scenarios with different action types under specific load patterns and SLA definitions; see Tables 8.3 and 8.5 respectively.

This section presents experiments to test the above conditions and discuss their results. Eight different system scenarios are discussed below, as summarised in Table 8.5. The scenarios differ in the way their SLA contracts are defined. The scenarios corpus starts from the most aggressive SLAs– mainly driven by operations monetisation perspective– where the control system has the most challenging situation to manage from both financial and load perspectives. The first three scenarios are penalty driven, and as a result the

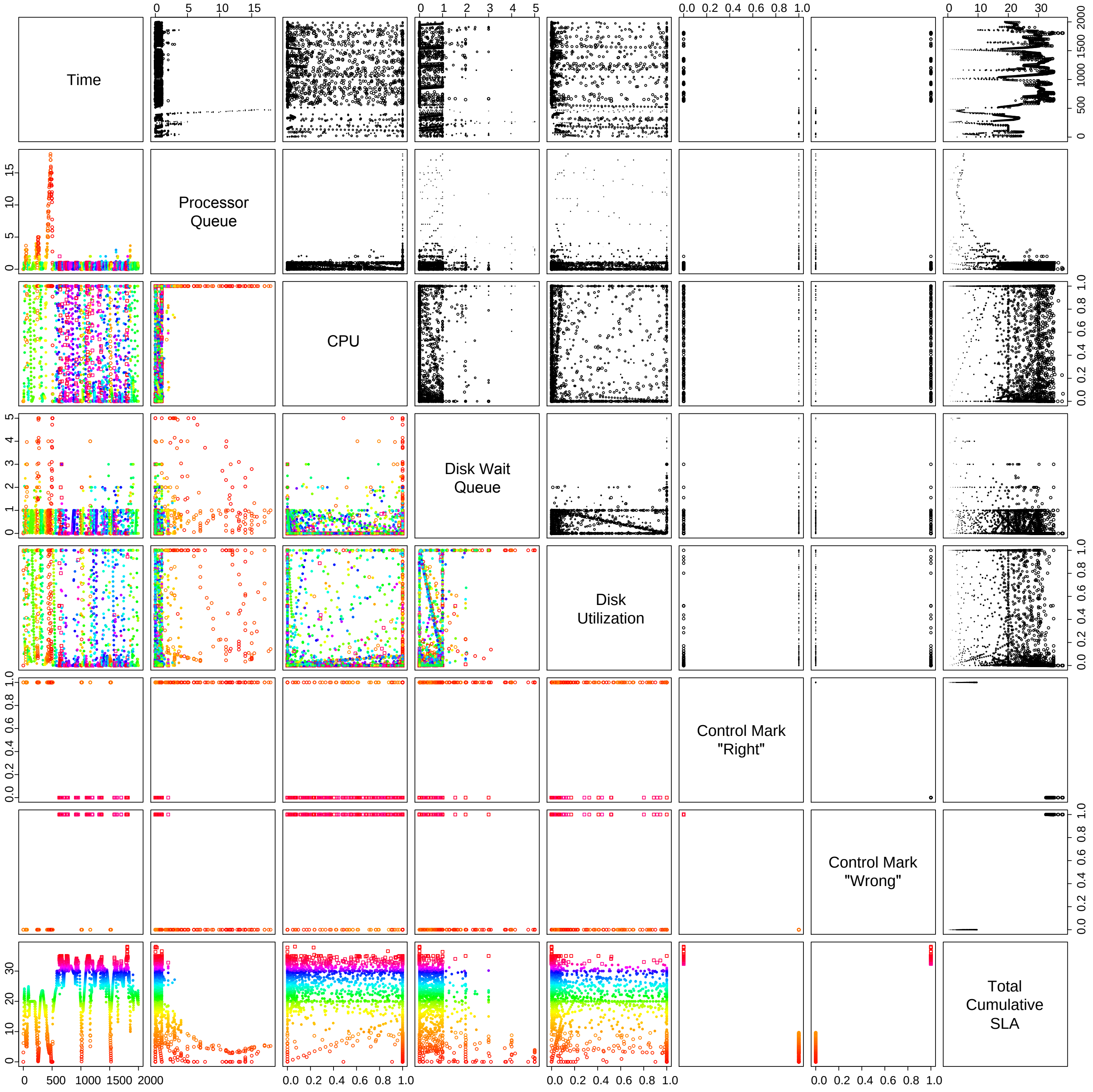
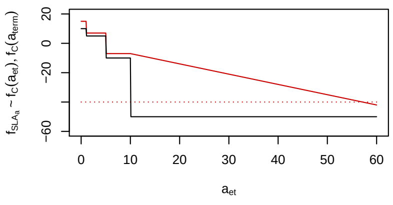
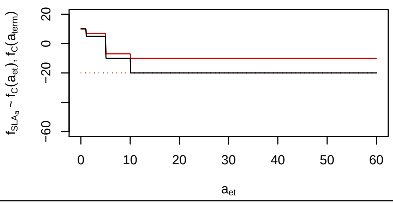
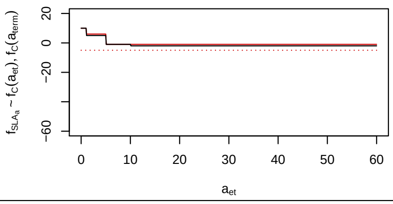
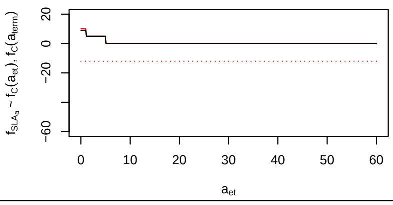
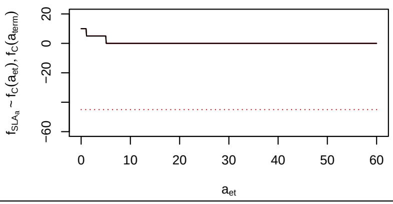
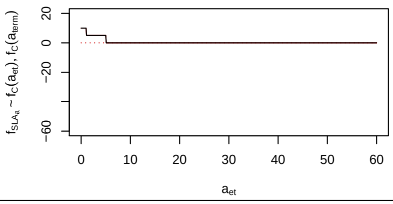
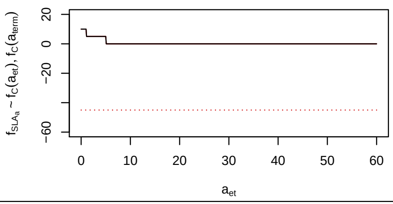
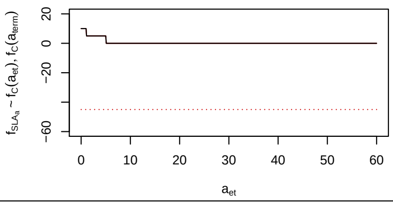


Figure 8.7: Scatter plot matrix of the most important dimensions of the system states collected during the last evaluation iteration in Experiment–1.
The neural control block uses a Type–2 control scheme (cf. Table 8.2), whilst the load pattern and SLA function follow scenario 4 (cf. Table 8.5). The dimensions named in the diagonal, starting from top–left, are: Time, Processor Queue, Processor Utilisation (CPU), Disk Queue, Disk Utilisation, "Positive" and "Negative" Control States (denoted by Control Mark "Right" and "Wrong", respectively), Total Cumulative SLA. The system states have been processed by the Evaluator (cf. Section 8.5.2) and have been labelled with "positive" and "negative" control marks used later for neural networks training. In the plots below the diagonal, a rainbow colour scale is used, starting from red for low values and ending with pink-violet for high values. The size of a point in the scatter plots above the diagonal represent its SLA function value, so that system states with lower SLA value are smaller and less visible.

Table 8.5: Examples of Load Scenarios and SLA function definitions

Scenario	**SLA Function**	**Description, Actions Types (SLAs), Load Pattern, LM**[a]
Scenario 0: This is a very challenging control scenario that employs a very aggressive penalty scheme for an overloaded system.		5 action types, namely A1, A2B, AOS, A3IO, A3IO2 with very aggressive SLAs, where the system is profitable only for calls shorter than 5 time stamps. For execution times longer than 10 stamps, the penalty is linearly increasing, whilst the termination penalty remains very high, equal to -40 for each termination decision. The code for this SLA is shown in Table 8.4. Effectively, termination control makes sense only for very long execution times. Actions load pattern is described in Table 8.3 and load is applied for LM=4.
Scenario 1: A slightly less aggressive penalty scheme for an overloaded system.		4 action types, namely A1, A2B, A3IO, AOS. Execution time less than or equal to 5 time steps generates a +7 award, but longer executions produce a penalty of -7, while the ones that are longer than 10 time stamps produce a penalty of -10. The termination penalty is -20 (two times smaller than in Scenario 0). Actions load pattern is described in Table 8.3 and LM=4.
Scenario 2: A scenario with low penalties for long execution times and termination penalty, which is cheaper than long execution time penalty.		4 action types, A1, A2B, AOS, A3IO, are used. There are low penalties for long executions: only -1 penalty for execution time longer than 5 time steps, also reasonably low penalty, -10, for each action termination, while for AOS and A1, termination penalty is only -5. Actions load as described in Table 8.3 and LM=4.
Scenario 3: This scenario has no penalties for long execution calls, so that the control can focus on revenue optimization only.		4 action types are used: A1, A2B, AOS, A3IO. There are no penalties for long executions but also zero award. There is a substantial penalty, -12, for termination of A1, A3 and AOS, but also awards for shorter calls; that is +10 for calls shorter than 1 time slot and +5 for shorter that 5. Action load, see Tab. 8.3, LM=4.
Scenario 4: This scenario demonstrates the effect of termination control on overloaded system without penalties.		4 action types are executed: A1, A2B, AOS, A3IO. No penalties for longer executions are applied but there is substantial penalty for termination. Shorter calls are rewarded, enabling active control. This scenario uses a load pattern based on Wolf's Sunspot Numbers Hathaway et al (2002). Actions load is standard, except for AOS that follows a pattern of annual sunspots[b], LM=3.
Scenario 5: This is a scenario that demonstrates the effect of termination control on overloaded system without any penalties.		4 action types are executed: A1, A2B, AOS, A3IO. There is no penalty for termination nor for longer execution times (SLA function has no negative values). Still, shorter calls are rewarded which drives the evaluations and the controller optimization direction. Details of the actions load pattern are in Table 8.3; load is applied for LM=2.
Scenario 6: The "World Cup 1998" scenario, described by Arlitt and Jin (2000), shows a three month period of quite bursty traffic and aperiodic workload.		12 actions of type A1, A2B, AOS, A3IO are executed, representing 1% of the most popular pages of the website that received 75% of all requests (Arlitt and Jin, 1998). There are no penalties for longer executions and termination, but still higher reward for shorter calls. The load pattern distinguishes this scenario from the rest, as shown in Table 8.3, and it is applied for LM=1.
Scenario 7: This scenario uses Wikipedia workload in October 2007 (Urdaneta et al, 2009) demonstrating operation under cyclic traffic.		There are 4 action types, namely W.En, W.De, W.Ja, and W.Es, responsible for 63.18% of the total load to Wikipedia in October 2017. There are no penalties for longer executions and termination, but still reward for shorter calls is higher. The load pattern is distinctive, shown in Table 8.3. Although LM=1, the pattern is periodic.

[a]LM = Load Pattern Multiplier is a parameter that multiplies the sequence of a base entry load profile, so that repetitive sequences of system states can be tested. An LM equal to 4 correlates very well with the frequency of the Evaluator cycles, see Section 8.5.2, which executes every 500 time steps. In all experiments, a total of 2100 time steps is used, see Section 8.6.2.

[b]"Annual sunspot relative number 1936-1972" and "Wolf's Sunspot Numbers 1936-1972", source `https://datamarket.com/data/set/22nu/annual-sunspot-relative-number-1936-1972#!ds=22nu&display=line`

service provider will incur penalties for action executions times that are longer than agreed. In the last three scenarios no penalties are included but the reward/revenue for longer calls is zero; thus, in the case of a slow environment the service provider will be operating the requests for computation service with minimal revenue generated, which will incur internal costs for the resources utilised.

In the first round, we compare the system operating without control against the first two of the termination control schemes presented in Table 8.2. Tests are conducted under eight different load scenarios and SLA function definitions, as presented in Table 8.5. This test examines termination control before execution only, i.e., just before the request for the functionality to be executed is processed. The results of the experiment are exhibited in Figure 8.8. The plots illustrates the impact of the control scheme on financial performance, represented by the Total Cumulative value of SLA units measured for each of the monitored system states T_{SLA}. As the time between calls increases, represented by the mean arrival time l that is defined according to a probability distribution for each scenario, the lower the frequency of calls gets, and, effectively, the load that the system needs to consume. Table 8.6 shows a comparison of financial performance increase $P \sim f_{SLA}$ in the test runs, per scenario and load l, using cumulative figures of T_{SLA} collected during these simulations.

Depending on the functionalities produced by the different actions in the scenario, the control system reacts differently to the load pattern it is exposed to taking into account the SLA function's definition. In effect, the controller adapts to the particular load conditions. In all scenarios, the controller improves the revenue generated by the provided service when the system is under higher load, as denoted by the lower mean arrival time l values. Furthermore, Type–2 control scheme performs better than Type–1 in most of the cases. In scenarios 2, 4 and 5 the control system was able to improve the situation regardless of the load that the system was exposed to. Note that even for action types, whose SLA function definition does not include penalties set explicitly (i.e., no negative SLA values for longer executions) but there is a higher reward for shorter execution times (see for example scenarios 3, 4, and 5), the controller drives the system to states that generate revenues. Also, the controller tends to terminate the longer actions, although these may be cheap in terms of associated penalty. This is because it considers that a cheaper action that is consuming resources is not economically viable to operate, even if termination incurs additional cost. This is a key finding for the monetisation of computing services such as PaaS/FaaS/SaaS equipped with this type of control. The controller shapes the run–time situation in order to achieve better conditions for running many types of actions under the same shared pools of resources. It favours cheaper executing actions that generate revenue over more resource expensive actions and their revenue, taking into consideration the potential costs of termination. Intuitively, such control could be most efficient when system's utilisation is high. In that case, releasing resources required to compute competing for access to resources actions benefits high–revenue actions. Therefore, it is not surprising that the best performance of the termination control in the experiments was in the heavily loaded system cases. The highest efficiency was noted under scenario 4 and 5, for arrival time lower than 4.0. In scenario 5, Type–1 and Type–2 control schemes were able to improve the system's operation value by 30–52% and 13–33%, respectively. This is clearly due to the less restrictive SLA and the lack of penalty for termination. Still in scenario 4, where additional cost for termination is considered, the improvement is significant– in the range of 15–35% and 8–25% for Type–1 and Type–2 control, respectively. Note that in scenario 4 there is still good improvement after applying control in the lower load as well. This is possible due to the relaxed nature of SLA, and the lack of penalties for termination. Scenario 5 has quite good overall improvement that is gradually reduced– the higher the load (l value gets low), the worse the total profit is. Interestingly, the operation performance is much greater than the intuitively expected increase of the revenue generation due to higher load (more actions to account) but impeded by the overused system resources, see the distribution shown on the violin charts. Very cheap termination in scenario 3 shows fairly good improvement, 5–12%, in cases of higher load, for arrival time lower than 4. The lower the load is, the fewer the chances to reduce the usage of resources by termination actions, so the operational improvement is gradually lowering but it is still better than no–control. This behaviour is repeated even when lower load is applied, e.g. mean arrival time gets equal to 6, where the system was profitable for all system states monitored.

Performance in scenarios 0, 1, 2, where SLAs contain penalties for longer termination, is lower than in scenarios 3–5, but still there are improvements made especially for higher loads. The financial performance in these cases is mainly impacted by the fact that SLAs contain substantial penalties for longer execution times that widen the band of profitability distribution and reduce the scale of the potential improvements that can be made on the system. Of course high prices for termination additionally impact the chances to improve the operation. The most difficult scenarios, 0 and 1, show practically optimisation of the loss of a service provider that is exposed to extremely challenging contracted SLA conditions. The control system was able to improve the situation by 5% only under very high load, where mean time of arrival is 3.0.

It is important to note that in the case of difficult to control scenarios where SLAs are aggressive in terms of penalties, like in scenario 0 and 1, the lower load may carry risks such as introducing degradation of the operational performance and even greater financial losses.

8.6.4 Experiment 3: Resources Utilisation and Financial Performance

The focus of this experiment is to evaluate how the controller operates under different load conditions when the executing actions require resources to be consumed, and the impact of the control scheme on resources utilisation and cost optimisation and revenue

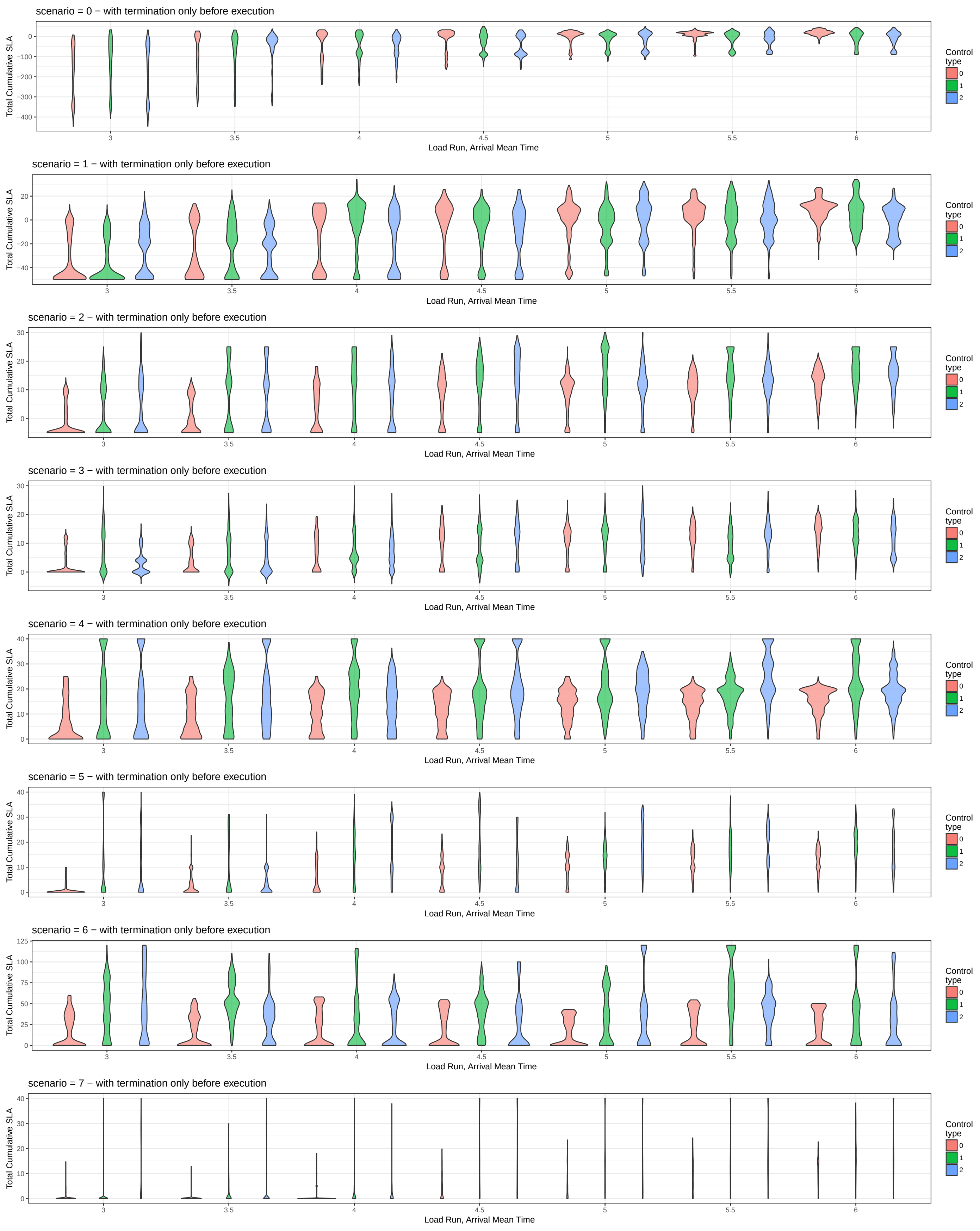


Figure 8.8: Comparative results for operation without control and with two of the termination control schemes, namely Type–1 and Type–2 (cf. Table 8.2), which apply termination control only before action execution. The diagram presents the distribution of system states as a function of load run. The focus is on financial performance, represented by the Total Cumulative value of SLA units measured for each of the monitored system states. SLA function definitions consider costs/permissions and revenue generated. The higher the values reached in the vertical axis, the more profitable a control scheme is for the scenario.

Table 8.6: Financial performance increase $P \sim f_{SLA}$ between control (Types 1 and 2) and no–control (Type–0), in independent test runs per scenario and load, in terms of T_{SLA} values measured during simulations execution (cf. Figure 8.8)

		ΔT_{SLA} [%]							
Control Type	Scenario	3.0	3.5	4.0	4.5	5.0	5.5	6.0	Total
1 to 0	0	5.63	-18.87	-25.52	-5.14	-6.18	-22.04	-43.66	-16.54
	1	0.60	2.72	3.64	-23.69	-16.89	-16.16	-29.22	-11.28
	2	9.07	9.09	28.70	21.85	14.77	15.02	7.71	15.17
	3	12.30	5.62	0.55	-8.14	-3.31	3.18	-16.21	-0.85
	4	34.64	30.50	15.36	14.46	22.57	20.41	14.69	21.80
	5	47.03	52.26	8.09	9.67	3.64	11.35	10.50	20.36
	6	96.41	75.14	33.95	25.70	54.86	64.44	62.26	58.97
	7	146.41	58.33	38.88	34.74	55.07	56.78	59.86	64.30
2 to 0	0	-1.89	-14.48	-31.35	-2.67	5.39	-19.58	-25.46	-12.86
	1	0.52	13.87	6.84	-22.51	13.31	-21.78	-36.02	-6.54
	2	5.84	66.99	22.75	36.18	18.82	3.42	7.34	23.05
	3	12.89	3.32	-4.26	1.61	-13.41	-4.79	-13.57	-2.60
	4	8.54	24.72	13.78	10.61	23.31	12.99	11.21	15.02
	5	16.81	33.40	13.63	16.82	6.64	14.03	9.44	15.82
	6	98.12	22.87	27.51	21.45	57.38	45.18	54.27	46.68
	7	146.41	61.61	34.23	31.94	52.71	55.78	58.36	63.01

generation.

It is clear that action execution times will heavily depend on the utilisation of resources; thus, high SLA values are likely to be found in the situations where the system is reaching a saturation state. From this point on, the system will be queuing the requests and SLA values will grow significantly. Figure 8.9 show this effect as observed in the experiments, while Figure 8.10 illustrates the states' SLA values. Each row in these figures presents several independent runs in one scenario, illustrating the distribution of system states for process queue, disk wait queue and total cumulative SLA for each one the three control schemes (Types 0, 1, and 2) tested under various load patterns.

Similarly to the previous tests, to simplify the results presentation[12], this one also examines termination before execution only.

One can see that in scenario 0 the CPU is massively over–utilised, e.g. there is a large process queue length with up to 80 tasks awaiting. For other scenarios, namely 1, 2, 3 and 4, where SLA functions get gradually less restrictive (e.g. especially when penalties for actions termination are lower), the controller is able to gradually free up more and more resources in order to give an improved SLA response. The only exception to this “rule" is the last scenario, number 5, where the load–pattern multiplier–LM (cf. Table 8.5) is too low to allow the controller to experience more repetitive sequences of systems states, and thus to reinforce the notion of “good”/“bad” control states. Effectively, there are fewer high quality control actions executed, and only the lack of penalties compensates the cumulative SLA value. Another effect observed in cases of lower LM is the much higher queues due to the longer duration of high load.

[12]Figure 8.9 and 8.10 shows distributions of systems states collected in 336 test runs (168 experiments executed twice) that generated 2GB of trace data.

Overall, for every scenario, the controller using either Type–1 or Type–2 control scheme manages to reduce resources consumption regardless of the load used in such a way that states with better financial performance are reached.

8.6.5 Experiment 4: Timing of Termination and Impact on Financial Performance

The next experiment focuses on aspects related to the timing of the termination action: (a) just after the request has been received but before it has been processed, or (b) during execution when the action is being processed.

In Figure 8.11, violin plots are used to illustrate the distribution of Total SLA in system states measured during the experiments for the two approaches mentioned above. The left column shows control approach (a) whilst the right one shows approach (b). Four control schemes, Types 1, 2, 3 and 4, and no–control (Type–0), (cf. Table 8.2), have been tested with the two approaches. The plots focus on one load condition with mean arrival time equal to 4 time buckets, while the expected mean execution time of actions, l (cf. Table 8.3), was set to 4.2. When the load pattern reached around 95%, the system was already above limit and was going into saturation. Table 8.7 contains aggregated general performance improvement values per type of control scheme collected in simulations which involve a wide range of load conditions across all scenarios.

Termination during execution, namely control approach (b), offers substantial flexibility allowing the controller to improve the system's revenue/costs performance significantly. Although, in some cases, the added value seems low, as shown in Figure 8.11 and Table 8.7, cumulatively approach (b) is an attractive option whenever an action takes longer and/or is affecting resources consumption, deadline violation costs are substantial and/or termination penalty is not high.

Of course, performance is impacted by the fact that allowing the execution of an action causes resources consumption, and thus, it introduces costs. So after termination, the benefits of releasing these resources are lower than applying approach (a) directly. Nevertheless, the potential penalty for the termination remains the same, even though in certain cases the controller may find that it is still worth applying termination immediately after the start of an action.

In most cases, Type–4 control is the best. Only in scenario 5, simpler control schemes (Type–1, 2, or 3) are better, as there are no penalties. Although the model does not take into account the computation loss related to control instrumentation processing, the termination during execution approach (right column) is only slightly better. The difference is clearly visible only in scenarios 1, 3, 4. Termination approach (b), applying control during action execution, was best performing for Type–1 control scheme where an average improvement of 19% on average was noted across all scenarios. For Type–3

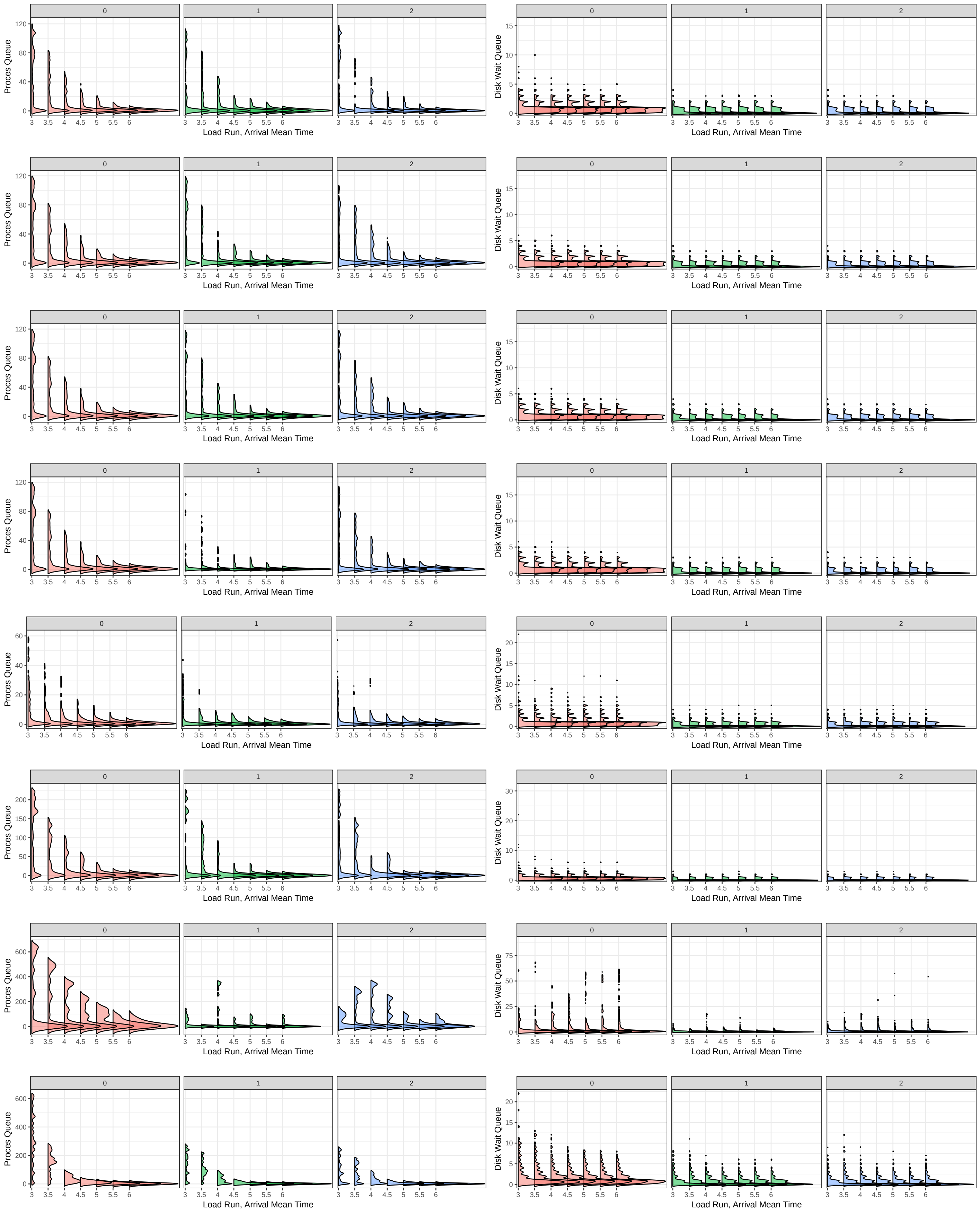


Figure 8.9: Experimental results for Type–0 (no–control) and Types –1 and –2 control schemes (cf. Table 8.2) for each of the eight scenarios (cf. Table 8.5). The diagram shows the distribution of system states as a function of load run. Each experiment contains around 500K states sampled during 168 tests: 3 control types, 8 scenarios, 7 load runs, across 2500 time steps. Every experiment was run twice for incremented pseudo-random number generator seed. The focus is on the effect of the control on resources utilisation for all 3 control types, namely Process Queue (left columns) and Disk Queues (right columns), as a function of load run.

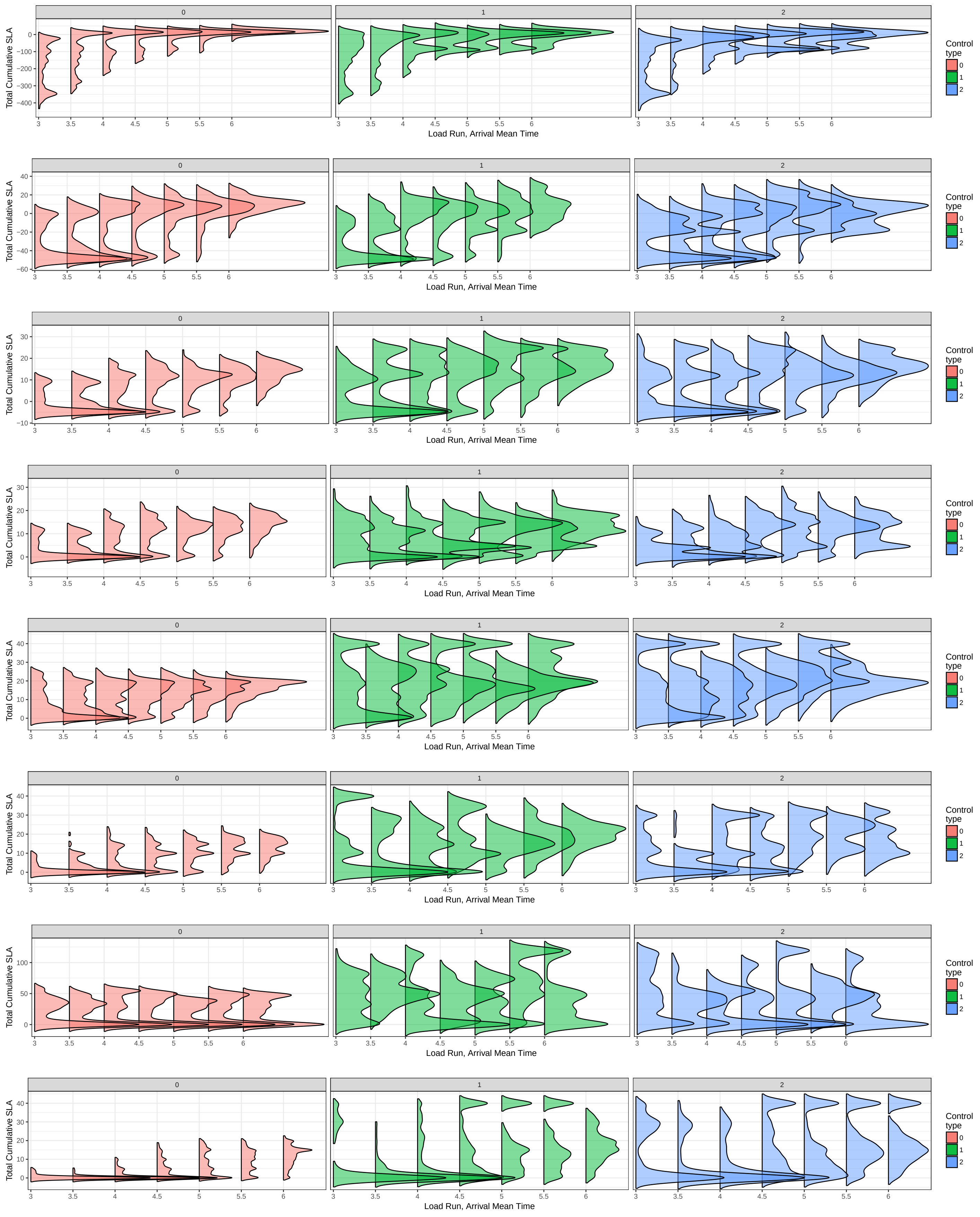


Figure 8.10: Effect of the control on financial performance, represented by the Total Cumulative SLA values of the system states with respect to load run. Results are for the experiments of Figure 8.9, i.e. around 500K states sampled during 168 tests: 3 control types, 8 scenarios, 7 load runs, across 2500 time steps.

Table 8.7: Financial performance increase $P \sim f_{SLA}$ between control (Types 1, 2, 3 and 4) and no–control (Type–0), in independent test runs across all scenarios (cf. Table 8.5), in terms of mean T_{SLA} values of the simulations (cf. Figure 8.11)

Control Type	Only before (a)	During (b)	Difference [%]
1 to 0	16.23	19.84	22.28
2 to 0	18.34	19.18	4.59
3 to 0	19.67	19.99	1.65
4 to 0	32.16	35.38	10.01

control the improvement was 14%, while for Type–4 the amelioration was only 4%, even thought Type–4 gives the best results over all.

Above experiments have been described in Sikora and Magoulas (2021).

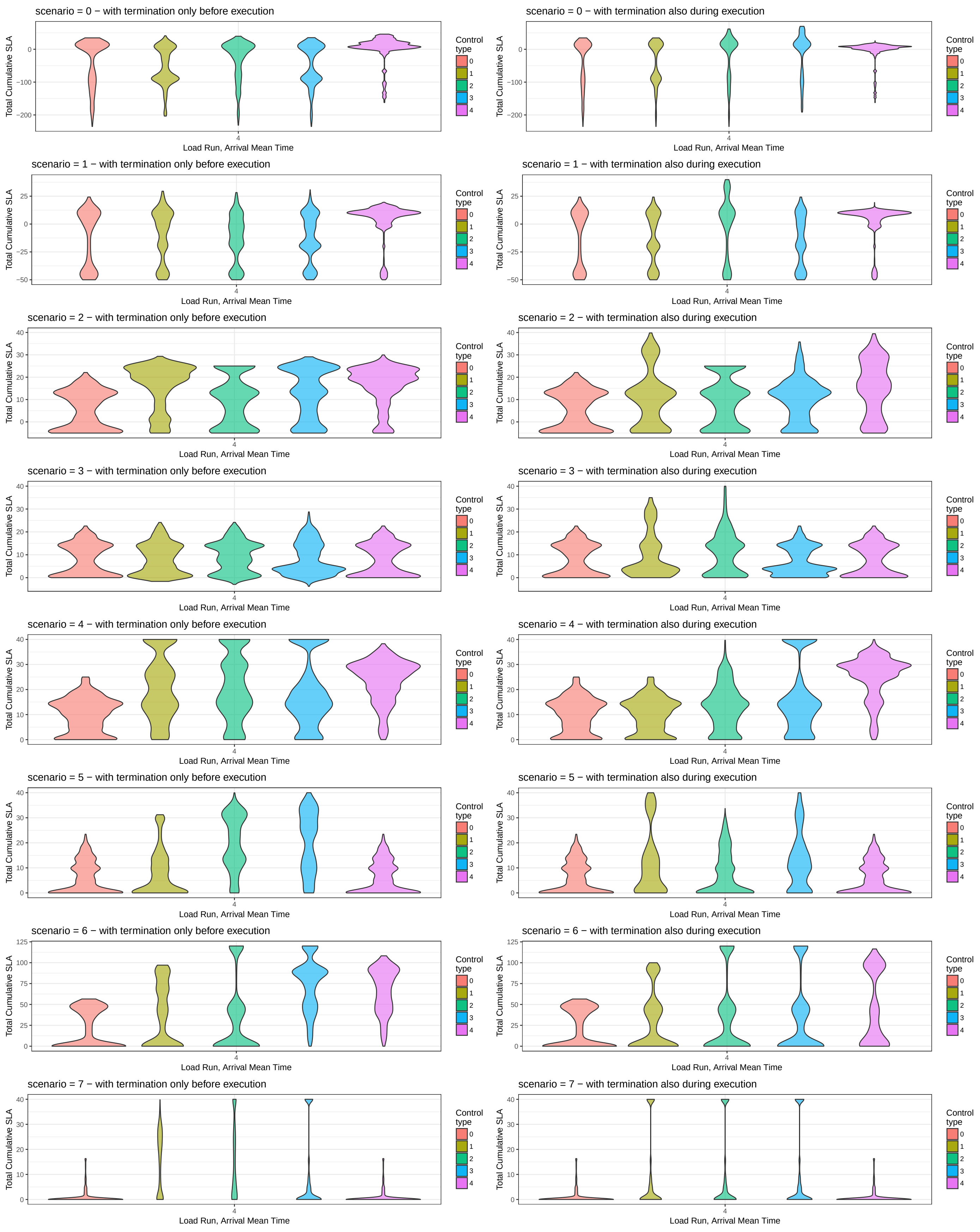


Figure 8.11: Results of experiments testing four control schemes (Types 1, 2, 3, and 4) and no–control (Type–0), (cf. Table 8.2), in all load and SLA definition scenarios (cf. Table 8.5) for a given mean arrival time equal to 4. The violin plots show distributions of Total Cumulative SLA values measured at each one of the monitored system states. The column on the left side shows values for states where termination was performed only before action execution, whilst the column on the right side refers to states where termination was done also during the action execution.

8.7 Terminating Actions in Complex Scenarios

In this section, we review challenges identified during the research and experimentation phase with the use of a termination actuator.

8.7.1 Actions Types of Different Performance Footprint Characteristics

The problem of highly convoluted scenarios is common in larger server setups, not necessarily in big data centres with distributed deployment strategies. Thus it is often encountered in larger enterprise systems rather than cloud providers. The issue has been discussed in Chapter 5 and 6. As the focus in this chapter is on control actuation, the experiments do not tackle such scenarios. In order to tackle the problem there are two broad approaches suggested: (a) to find actions, resources and SLA relevance and prioritise focusing on the most significant in the observable setup (see Chapter 5) limiting the scale of the control problem, (b) to decompose signals and introduce sequence of cycles around the Evaluator feedback (see Chapter 6) to extract essential for given actions load resource reliance and identify systems states sequences that lead to bottlenecks producing high SLA values.

8.7.2 Activity Input Signals for the Neural Control Block

The tested control block as the main actuating unit does not take into account current activity, other actions and effectively load function inputs, it focuses the final actuation decision making by resources consumption measurements. This is an assumed simplification extending the list of the setup limitations in Section 8.5.7, which reduces not only the complexity of the decision block design and control judgement execution performance, but also significantly decreases the number of dimensions that are interpreted by the decision block (not the Evaluator), which in effect simplifies the search and control space complexity.

It is worthy to remind that the collected systems states, S, defining the current load, usage and system response to SLAs, are processed during the evaluation process and are used to train the control block NNs, so SLAs as outputs are present in transformed form to (middle–level) inputs, such as resources usage (utilisation or queue length). However, direct input of activity, such as each action types executions count and any currently (or in recent past) observed execution times, is not available to the decision block of termination actuators. Consequently, the rules induction, the training vectors and the neural–block model could be extended by considering activity inputs in order to form potentially more precise local to action run–time control decisions. The general approach described in the study is fully compatible with this extension.

8.7.3 Early–system–life Limitations

There is also an important internal limitation of the evaluation mechanism that should be considered at the beginning of the controller operation, when no much knowledge is still available, or in cases when many contradicting states are disregarded (filtered out), and the training sets are shorter than in later stages, which can impact the NN control block training and the initial precision and quality of the actuation decisions. This can be solved by using synthetic Gaussian mixture added over the rules points, creating an expansion of a given training set. Such a procedure to combat over–fitting and improve the generalisation ability was studied by Karystinos and Pados (2000).

8.7.4 Long Executing Actions

The longer the action the heavier the cumulative execution footprint on the resources (i.e. CPU, disks, network, memory), and in effect on other defined SLA, but not necessarily on those actions SLAs. The control system needs to consider that termination will not only impact the potential penalty of terminated actions but the operator will still need to expect the routine to be requested for execution again, which needs to be included in the decision process. Wider time aggregates of collected metrics can help to demonstrate such situations, and steer the adequate control rules induction.

In general, this is an operating area where termination during execution should give better results (see Section 8.5.6), e.g. short and predictable calls like user–interface or web–services controlled with pre–execution control, and for longer less predicable like batch processes, or asynchronous messaging with during–action–execution terminator actuators.

The longer the action the more difficult it gets to map the execution characteristic to simple DES model. On the other hand any larger action is built with subroutines whose calls can be wrapped with proxies of control actuators. Still such scenarios are cumbersome and require better knowledge of the system run–time and finer grained control blocks configuration.

Since modern UI–UX requires fast, sub–second responses, a typical industry practice for longer than 5–10 seconds executions is to design an interaction of user–system that is taking into account the delay adequately, i.e. incorporating MQ with UI showing progress, or batch processing, effectively introducing space for other control methods.

Long executing actions like manually ad–hoc requested batch jobs have been tackled in the following section.

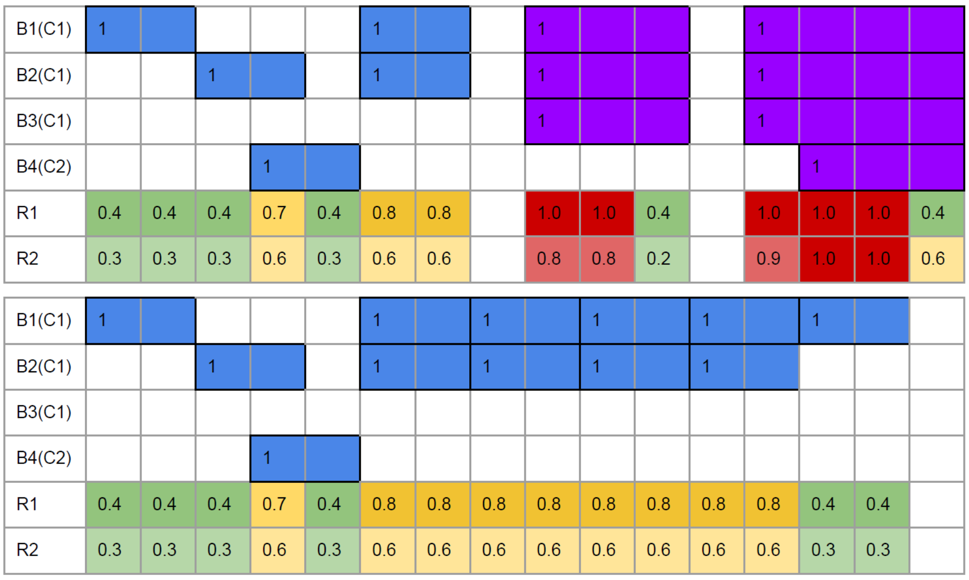
8.8 Actions and Batch Jobs Reschedule

It is time to review another method of non–acquisitive control that is working under the regime of real–time NFRs. Cloud computing research has investigated the potential of tasks scheduling optimisation (Soltani et al, 2017; Arunarani et al, 2019), including the use of RL (Peng et al, 2015).

The thesis tackles the area for completeness, discussing the whole field of methods for optimising the resources utilisation, and showing the potential of their coexistence in ASM setups.

As highlighted earlier in Sections 2.5.2 and 4.7, when explaining possible ways of actuating the work of an enterprise system, incoming actions execution can be delayed or moved earlier than originally planned for scheduled jobs in order to place the processing at a time when critical resources are expected to become available in such capacity that will allow sufficient support of the service per defined SLA.

The methods examined so far in Section 8.5.1 and in the previous chapter allow termination but may do not well support critical and often long, thus expensive, batch jobs. In a simple scenario, the importance of the execution, although long, may be far exceeding the costs of longer execution. In a more complex scenario, in presence of a harsh time for the placement of the processing, an operator may be faced with additional costs for penalties for longer processing. Therefore, termination cannot be applied.



B1(C1)	1	🟦				1	🟦		1	🟪	🟪		1	🟪	🟪	🟪
B2(C1)			1	🟦		1	🟦		1	🟪	🟪		1	🟪	🟪	🟪
B3(C1)									1	🟪	🟪		1	🟪	🟪	🟪
B4(C2)				1	🟦									1	🟪	🟪
R1	0.4	0.4	0.4	0.7	0.4	0.8	0.8		1.0	1.0	0.4		1.0	1.0	1.0	0.4
R2	0.3	0.3	0.3	0.6	0.3	0.6	0.6		0.8	0.8	0.2		0.9	1.0	1.0	0.6

B1(C1)	1	🟦				1	🟦	1	🟦	1	🟦	1	🟦	1	🟦	
B2(C1)			1	🟦		1	🟦	1	🟦	1	🟦	1	🟦			
B3(C1)																
B4(C2)				1	🟦											
R1	0.4	0.4	0.4	0.7	0.4	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.4	0.4	
R2	0.3	0.3	0.3	0.6	0.3	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.3	0.3	

Figure 8.12: Batch jobs in time with impact on resources utilisation and execution times. The two tables show an example of the same computation scheduled for 12 batch job executions in four batch threads, B1,..,B4, for two different classes, C1, C2: before (table on top) and after (table on bottom) the rescheduling optimisation exercise. It is clearly visible how the better placement of jobs helps in resource utilisation.

Batch jobs are typically executed at a scheduled time but general use and corporate frameworks allow to execute a batch job on an ad–hoc (manual) basis when needed.

8.8.1 Rescheduling Control Instrumentation

Shallow instrumentation can be provided by batch frameworks like the "Quartz" job scheduling library or Java Spring `@Scheduled` annotation, the "schedule" package in Python, or the "clockwork" Ruby module, in the same way as in termination weaved–in objects approach (see Sections 8.5.1 and 8.5.4). A deeper code instrumentation, wrapping the sub–function, can be delaying further execution of a longer routine and may be also a help in situations where the system is exposed to excessive utilisation due to an unexpected load coming for very important actions. This utilisation can lead to saturation of resources currently used by a running batch job that is normally expected to finish soon. Thus, it is economically viable to delay the long job processing rather than terminate the incoming calls, so that the bottleneck resource can be temporarily cured, or released long enough to allow a scaling controller (Sections 2.5.1 and 4.7), to allocate more resources from the cloud pool.

For these and more complex situations the scheduling controller can adapt autonomously distributing the load in time, see Figure 8.12. Critical in this scheme is the knowledge about (a) impact on the resources that come from the execution functionality– Chapter 6 tackles that need, and (b) economical and time constraints of the job processing that are typically given in batch job execution parameters and SLAs– see Section 8.3.1.

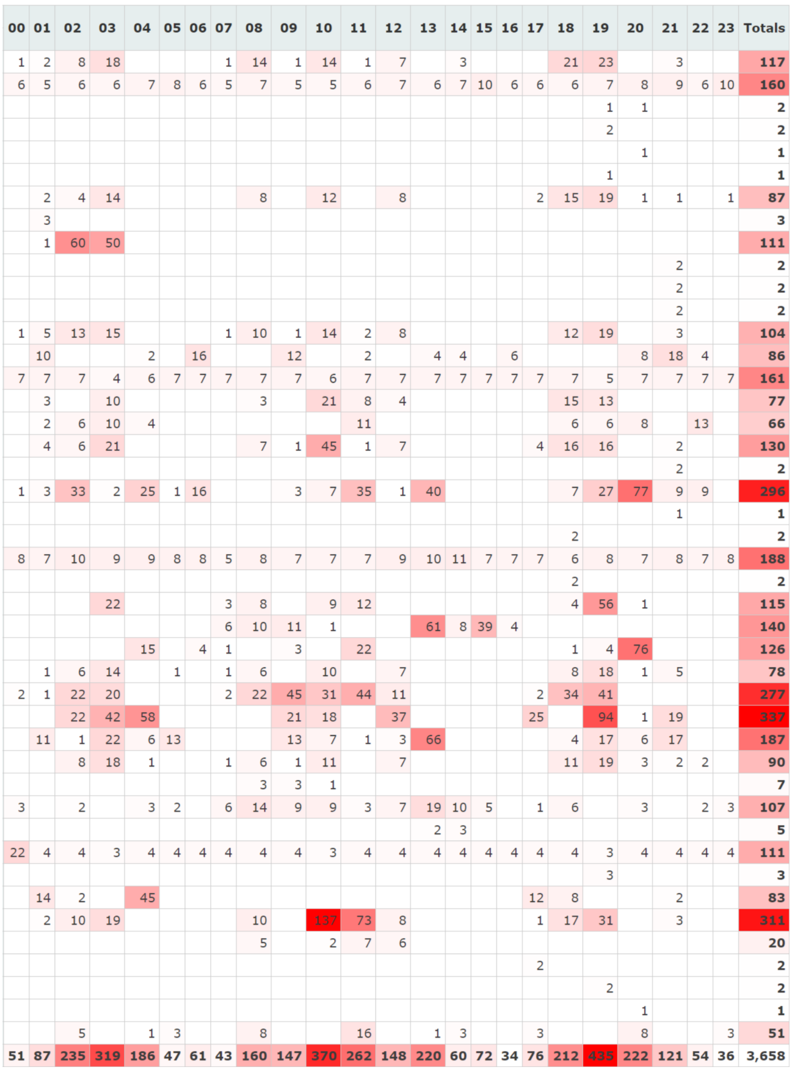
A simple but typical to ASM field scheduling problem is illustrated in Figure 8.12. The general goal is to spread the load of requested computation in such a way that resources are utilised only to a secure limit providing stability and batches are distributed in time to not impact delivery times.

Operators, equipped with administration consoles tuning batch jobs plan, can change the run–time utilisation and optimise the system considerably. Still in order to use human element their attention could be moved from time–wise scheduling to boundaries definitions.

8.8.2 Examples of Rescheduling Use–cases

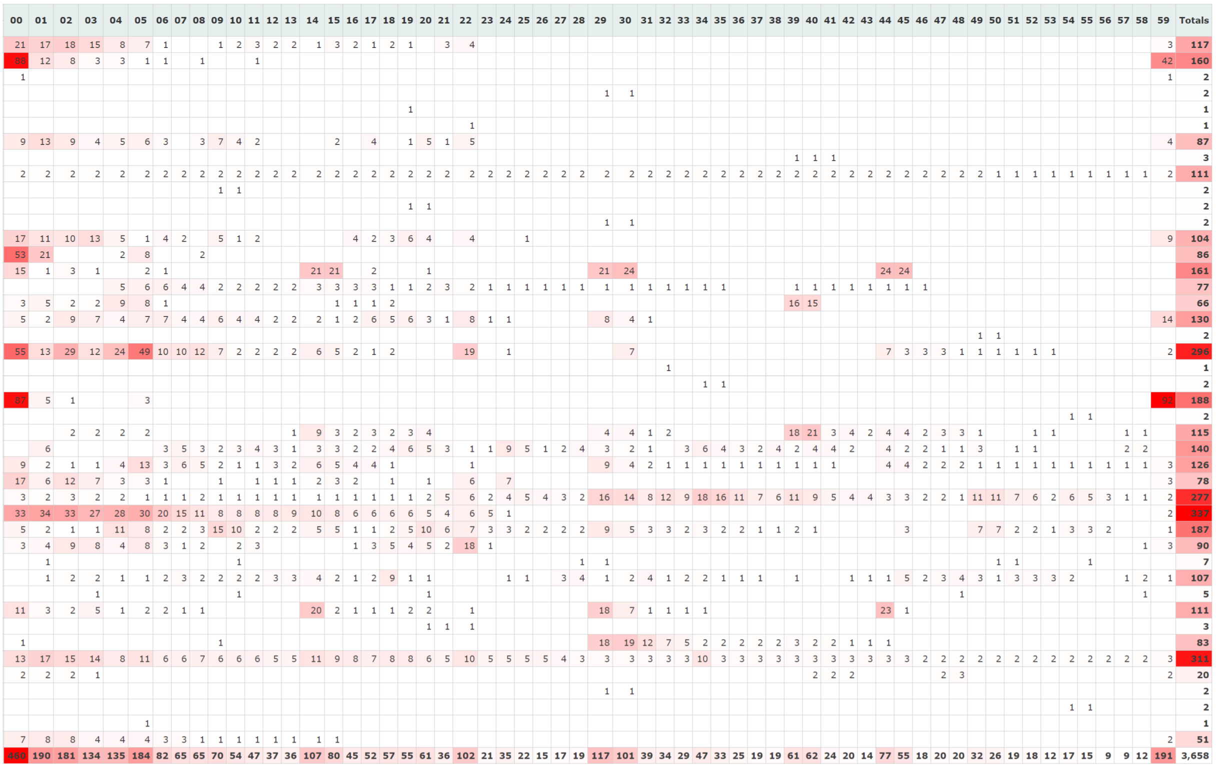
There are two main visual representations of the jobs (previous, current, and expected runs): (a) per batch job type, jobs count and execution time aggregates in time (easy to be shown by pivot table), as shown in Figures 8.13 and 8.14, and (b) in–time Gantt charts explaining more details of event placement, as shown in Figures 8.12 and 8.16.

The general target of the scheduling is to deliver the expected computation on time. Since computations in a busy enterprise environment require resources, the goal is in fact to place executions in time buckets when resources are expected to be still available and reduce the processing demand in times when the key resources are under risks of saturation, e.g. when CPU usage, thread count or queues length demonstrate bottlenecks, and SLAs are



00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	Totals
1	2	8	18				1	14	1	14	1	7		3				21	23		3			117
6	5	6	6	7	8	6	5	7	5	5	6	7	6	7	10	6	6	6	7	8	9	6	10	160
																			1	1				2
																			2					2
																				1				1
																			1					1
	2	4	14					8		12		8					2	15	19	1	1		1	87
	3																						3
	1	60	50																					111
																					2			2
																					2			2
																					2			2
1	5	13	15				1	10	1	14	2	8						12	19		3			104
	10			2		16			12		2		4	4		6				8	18	4		86
7	7	7	4	6	7	7	7	7	7	6	7	7	7	7	7	7	7	7	5	7	7	7	7	161
	3		10					3		21	8	4						15	13					77
	2	6	10	4							11							6	6	8		13		66
	4	6	21					7	1	45	1	7					4	16	16		2			130
																					2			2
1	3	33	2	25	1	16			3	7	35	1	40					7	27	77	9	9		296
																					1			1
																		2					2
8	7	10	9	9	8	8	5	8	7	7	7	9	10	11	7	7	7	6	8	7	8	7	8	188
																		2					2
			22				3	8		9	12							4	56	1				115
							6	10	11	1			61	8	39	4							140
				15		4	1		3		22							1	4	76				126
	1	6	14		1		1	6		10		7						8	18	1	5			78
2	1	22	20				2	22	45	31	44	11					2	34	41					277
		22	42	58					21	18		37					25		94	1	19			337
	11	1	22	6	13				13	7	1	3	66					4	17	6	17			187
		8	18	1			1	6	1	11		7						11	19	3	2	2		90
								3	3	1														7
3		2		3	2		6	14	9	9	3	7	19	10	5		1	6		3		2	3	107
													2	3										5
22	4	4	3	4	4	4	4	4	4	3	4	4	4	4	4	4	4	4	3	4	4	4	4	111
																			3					3
	14	2		45												12	8			2			83
	2	10	19					10		137	73	8					1	17	31		3			311
								5		2	7	6											20
																	2							2
																			2					2
																				1				1
		5		1	3			8			16		1	3			3			8			3	51
51	87	235	319	186	47	61	43	160	147	370	262	148	220	60	72	34	76	212	435	222	121	54	36	3,658

Figure 8.13: Comparison of batch jobs in time of a day, grouped by hour.



00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	Totals
21	17	18	15	8	7	1			1	2	3	2	2	1	3	2	1	2	1		3	4																																					3	117
88	12	8	3	3	1	1		1			1																																																42	160
1																																																											1	2
																													1	1																													2
																			1																																									1
																						1																																					1
9	13	9	4	5	6	3		3	7	4	2				2		4		1	5	1	5																																					4	87
																																							1	1	1																			3
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	2	111
									1	1																																																	2
																			1	1																																							2
																													1	1																													2
17	11	10	13	5	1	4	2		5	1	2					4	2	3	6	4		4			1																																	9	104
53	21			2	8			2																																																				86
15	1	3	1		2	1								21	21		2			1									21	24														24	24															161
				5	6	6	4	4	2	2	2	2	2	3	3	3	3	1	1	2	3	2	1	1	1	1	1	1	1	1	1	1	1	1	1				1	1	1	1	1	1	1	1														77
3	5	2	2	9	8	1									1	1	1	2																					16	15																				66
5	2	9	7	4	7	7	4	4	6	4	4	2	2	2	1	2	6	5	6	3	1	8	1	1					8	4	1																												14	130
																																																	1	1										2
55	13	29	12	24	49	10	10	12	7	2	2	2	2	6	5	2	1	2				19		1						7														7	3	3	3	1	1	1	1	1	1						2	296
																																1																											1
																																		1	1																									2
87	5	1			3																																																						92	188
																																																						1	1					2
		2	2	2	2								1	9	3	2	3	2	3	4									4	4	1	2							18	21	3	4	2	4	4	2	3	3	1			1	1				1	1		115
	6					3	5	3	2	3	4	3	1	3	3	2	2	4	6	5	3	1	1	9	5	1	2	4	3	2	1		3	6	4	3	2	4	2	4	4	2		4	2	2	1	1	3		1	1					2	2		140
9	2	1	1	4	13	3	6	5	2	1	1	3	2	6	5	4	4	1				1							9	4	2	1	1	1	1	1	1	1	1	1	1			4	4	2	2	2	1	1	1	1	1	1	1	1	1	1	1	3	126
17	6	12	7	3	3	1			1		1	1	1	2	3	2		1		1		6		7																																			3	78
3	2	3	2	2	1	1	1	2	1	1	1	1	1	1	1	1	1	1	1	2	5	6	2	4	5	4	3	2	16	14	8	12	9	18	16	11	7	6	11	9	5	4	4	3	3	2	2	1	11	11	7	6	2	6	5	3	1	1	2	277
33	34	33	27	28	30	20	15	11	8	8	8	8	9	10	8	6	6	6	6	5	4	6	5	1																																			2	337
5	2	1	1	11	8	2	2	3	15	10	2	2	2	5	5	1	1	2	5	10	6	7	3	3	2	2	2	2	9	5	3	3	2	3	2	2	1	1	2	1					3				7	7	2	2	1	3	3	2			1	187
3	4	9	8	4	8	3	1	2		2	3					1	3	5	4	5	2	18	1																																			1	3	90
	1									1																	1	1																					1	1				1					7
	1	2	2	1	1	2	3	2	2	2	2	3	3	4	2	1	2	9	1	1				1	1		3	4	1	2	4	1	2	2	1	1	1		1			1	1	1	5	2	3	4	3	1	3	3	3	2			1	2	1	107
			1						1									1																											1										1		5
11	3	2	5	1	2	2	1	1						20	2	1	1	1	2	2		1							18	7	1	1	1	1										23	1															111
																				1	1	1																																					3
1									1																					18	19	12	7	5	2	2	2	2	2	3	2	2	1	1	1																83
13	17	15	14	8	11	6	6	7	6	6	6	5	5	11	9	8	7	8	8	6	5	10	5	5	5	5	4	3	3	3	3	3	3	10	3	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	3	311
2	2	2	1																																				2	2	2					2	3											2	20
																													1	1																													2
																																																						1	1					2
					1																																																						1
7	8	8	4	4	4	3	3	1	1	1	1	1	1	1	1																																												2	51
460	190	181	134	135	184	82	65	65	70	54	47	37	36	107	80	45	52	57	55	61	36	102	21	35	22	15	17	19	117	101	39	34	29	47	33	25	19	19	61	62	24	20	14	77	55	18	20	20	32	26	19	18	12	17	15	9	9	12	191	3,658

Figure 8.14: Comparison of batch jobs in time of an hour, grouped by per minute.

severely impacted.

Thus following the example of Figure 8.12, the goal is to shorten execution times on the top right side of the table. Shorter executions of action batch jobs B2, B3 and B4 are in time buckets where resources R1 and R1 are still available (top–left), far longer when the system is overloaded (top–right). When the jobs are placed in such time buckets where resources are available the execution times are far shorter (bottom–right).

8.8.3 Adaptive Rescheduling in ASM

Scheduling of batch jobs as an optimisation problem belongs to the family of packing problems, and is a variant of Job Shop Scheduling, as an extended by time dependency Knapsack Packing problem, that is NP–complete (Chandra et al, 1976; Khuri et al, 1994; Taillard, 1993). Job Shop Scheduling problem without dependencies is Open–shop Scheduling problem (Gonzalez and Sahni, 1976; Williamson et al, 1997), whilst Flow Shop Scheduling is a special case of Job Shop Scheduling where the emphasis is put on the order of all operations to be done in all jobs (Gupta and Stafford Jr, 2006; Liu et al, 2007). ASM scheduling resembles the Knapsack problem in the weights of items, which are priorities or SLA values, and the capacity, which is execution time and footprint on resources used. However in the ASM field the set of constraints is far extended, i.e. it is required to schedule all of the batches planned in a given time window. In order to avoid risks of over–utilised resources like resource contention or bottlenecks, and in effect shorten the execution time and lower SLA penalties the algorithm should seek a new schedule of jobs that are spread in time.

Exact algorithms or traditional heuristics have been studied in the literature(Martello, 1990; Pisinger, 1995); however as for most NP–complete problems, approximation algorithms are commonly used (Khuri et al, 1994).

Adaptive rescheduling methods in batch mode running in grid computing initially or in the cloud computing later have been largely explored for many environments and diverse types of ASM systems and applications by Gao et al (2005), Abawajy (2009), Xhafa et al (2007), Jian and Wang (2014), Prabhakaran et al (2015), (Li et al, 2010), Cheng et al (2017), Chadha et al (2020), (Awad et al, 2015), (Moon et al, 2017), (Jang et al, 2012), (Lin et al, 2014), (Kaur and Verma, 2012), (Jena, 2017), (Arunarani et al, 2019), (Yu and Chen, 2008).

In Section 8.9 of this chapter several metaheuristic method are tested– they all have a good research track for job shop and scheduling optimisation applications: e.g. Simulated Annealing (SANN) by Osman and Potts (1989); PSO by Jerald et al (2005), Abraham et al (2006), Wu (2014a), Ramezani et al (2014), and Jordehi and Jasni (2015); GA by Ruiz et al (2006) Omara and Arafa (2009) and Park et al (2003).

8.8.4 Rescheduling as an Optimisation Problem

Batch jobs scheduling for malleable routines and shorter actions micro–scheduling optimisation is basically a search problem, where the main objectives are:

- (final rank) minimise SLA, f_{SLA} — improve financial performance P (see Section 8.3);
- (effective rank) minimise total batches execution time — lower impact on other parts of the system and reduce total UI/API actions execution time;
- (working rank) minimise resources consumption — using decomposed signals data from earlier executions, see Chapter 6,

under various constraints such as, for example:

- fastest end of processing time;
- resources usage: e.g. maximum SMA of DB pools utilisation, maximal CPU utilisation;
- reduce (minimise) the amount of schedule changes.

A similar problem was already explained in Chapter 6 while discussing the ASM-SD method.

Let $b(t) \in b$ denote a batch job run that is effectively one of the systems actions, $b \in a$, of normally far longer execution time than other actions $b(t) \gg a(t)$, and where the system starting the routine has planned $b(t_0, t)$, called by a scheduler in time t_0 rather than another entity like users for UI or 3rd party system for API, making the sources of load much less predictable.

Financial performance increase $P \sim f_{SLA}$ as a composition of revenue R and costs C (see Equation 3.1 and Section 8.3), effectively relies on allowing stable high activity and low resources usage.

Each of the batch job runs use resources $r(t) \in r$. After SDCIC selection and ASM-SD deconvolution phases each of most important batch jobs b_i can have m_{ik} entries in the convolution (mixing) matrix M assigned to resources r_k (cf. Equation 6.3).

Again, there are many non–linearities, which have an impact on batch jobs execution times and resources utilised (see Section 6.2), thus it will be treated as a general NLP optimisation. Therefore, formally:

$$\arg\min_{b_{t_0} \in T_s} \left(\sum_k \left(\widetilde{r_{a_k}} + \sum_i \left(m_{ik}(r_k) b_i(t_0, t) \right) \right) \right), \tag{8.2}$$

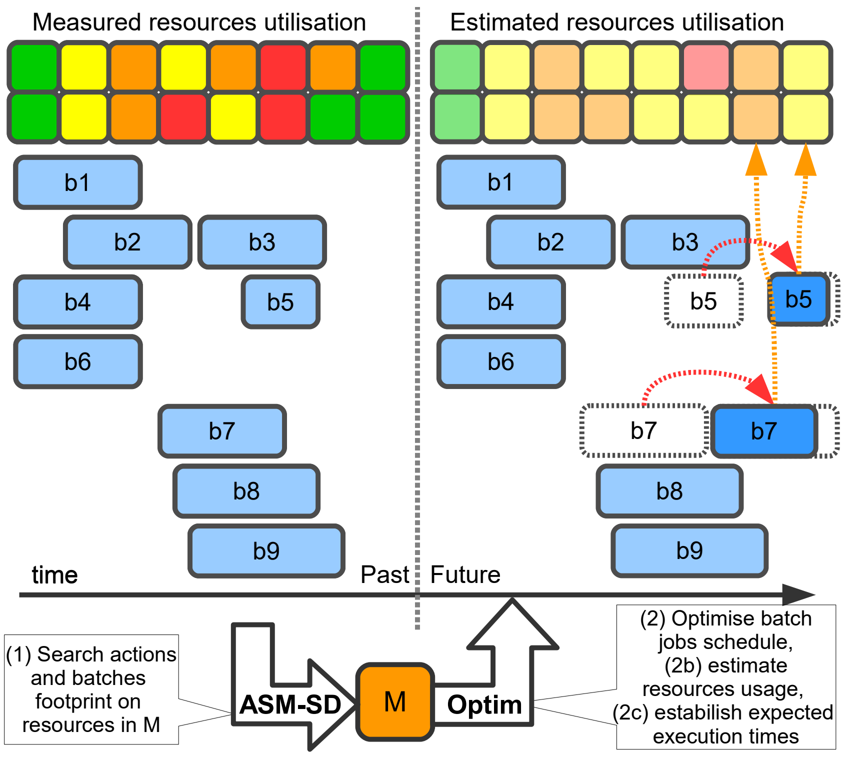

Measured resources utilisation
Estimated resources utilisation
b1
b2
b3
b4
b5
b6
b7
b8
b9
time
Past
Future
(1) Search actions and batches footprint on resources in M
ASM-SD
M
Optim
(2) Optimise batch jobs schedule, (2b) estimate resources usage, (2c) estabilish expected execution times

Figure 8.15: Sequence of reschedule processing, ASM-SD search and batch scheduling optimisation.

where t_0 is unknown (searched) time in future and indicates start of a job, so b_{t_0} is a vector of new batch schedule that is part of new feasible settings of all batch jobs $T_s \in C$ for which the equation in brackets (the objective function) attains its minimal value, under C as a global set of job constraints. m_{ik} denotes coefficients of service load (found in the ASM-SD process) forming a mixing matrix M of size $\overline{\overline{r}} \times \overline{\overline{a \cup b}}$, which causes changes in r_k after a_i and b_i calls. Batch jobs and resources are denoted by b_i as the i-th batch job, r_k as the k-th resource and $\widetilde{r_{a_k}} = \sum_i \Big(m_{ik}(r_k)\widetilde{a}_i(t) \Big)$ is a footprint on resources usage by expected in prognosis $\widetilde{a}$ actions, where $\widetilde{a}_i(t)$ is the anticipated t-th action.

In this context the objective function values explain the system load under expected actions a and scheduled batch jobs b in a scope of the considered time t and the solution is a schedule of batches b_{t_0}.

Notice that resources $r_k \in r$ are weighed with the same priority when the mixing matrix M coefficients are taken into account in the search objective function. It is possible, however, to introduce weights and expose them to a calibration process if some of the resources are more valuable for the stability or any other NFR to reflect protection over given resources.

8.8.5 Objective Function Processing

Figure 8.15 depicts the wider process of optimisation. Let's review in more details the sequence of processing phases and their nature, which is analogous to process of rules evaluation explained in Section 8.5.2:

1. Preliminary search of batches load characteristic:

   As defined in Eq. 8.2 the objective function requires mixing matrix M as an entry parameter. The metrics are being found in the ASM-SD process (see Chapter 6). This step introduces information on effective footprint of given action/batch on resources. Not only batch jobs but action executions are also taken into account in that processing. This is essential to consider other activity in the system to establish wider matrix M, and to factor in the additional expected resource intensive actions activity.

2. Batch jobs optimisation process:

   (a) Search run:

   The problem space is being searched with use of the objective function, Eq. 8.2. This phrase is checking the parameters for function calculation and the function itself is extended by boundaries and constraint conditions, as shown in Section 8.8.6.

   Notice that many EIS allow application super users to trigger batches manually regardless of scheduled times. Those manually requested batches naturally extend the knowledge of the systems States S and contribute to better control in the future after load characteristics are derived during ASM-SD processing. Thus, such manually requested (not scheduled) batch jobs, often even of the same class types as those that are scheduled, should be treated as other actions a.

   (b) Estimating new batch jobs schedule and their effect on resources utilisation:

   This step would effectively change the search from static to dynamic system. To simplify the further processing and to avoid recursion, only one cycle is performed to (i) calculate new batch execution based on .... and (ii) effect on resources utilisation.

   (c) Projecting new execution times of batches:

   This step is important especially under real–time conditions. Since batch jobs times are functions of resources allocation, their execution time change significantly when required resources are expected to be over–utilised – the earlier collected data are in use and the sooner the objective function takes that into account. Therefore, after using the found coefficients of the mixing metrics M to calculate the expected footprint on resources, the objective function can adapt the execution times. Of course, the updated execution times will have an impact on resources used, but a single cycle is in use to avoid expensive recursion. In experiments it was found that a single adaptation cycle could give satisfying approximation.

(d) Projecting new batches SLAs:

In a context of the control framework of this work the priorities are set by monetised SLAs. Since both SLA definitions are known and planned batch jobs have estimated the execution times (a), and resources utilisations (b) have been projected, it is possible to calculate synthetically the SLAs.

8.8.6 Scheduling Constraints

The optimisation must be performed under several constraints (boundaries) representing technical, regulators and business requirements, which demand listing and explanation:

1. Time–windows boundaries in the search space:

   In order to reduce the search space dealing with practical maintenance aspects the schedule perspective is limited to time–windows, e.g. next hour, 8–hours shift, a day, or a week. Depending on algorithm used the optimisation can be controlled by search arguments parameters or barrier function given by additional penalisation coefficient for which the optimisation function value is sharply rising when arguments are above the expected limits (Di Pillo and Grippo, 1989; Smith and Coit, 1997).

   The time horizon is split to time buckets that reduce the search problem and convert it to discrete space. So the new start of a batch job time t_0 can be limited to a time bucket precision. Typically the buckets are big enough to reduce the space and adequately small to reflect the nature of the batch jobs execution times. In a typical case for heavy batch jobs hours in a day or a week, see Figure 8.16, and for shorter batches minutes in 8–hours or a day should be sufficient, see Figures 8.17 and 8.18.

2. Execution time boundaries:

   There are two categories of this boundary type: (a) certain batches have deadlines and must finish before the time, and (b) the time horizon of the scheduling plan is limited and the search criteria should steer the batch jobs to not execute after the limit[13].

   In this case a dynamic penalty solution of barrier function applied in a similar to the aforementioned use–case gives very good results. Whenever the expected batch job execution under tested t_0 is going to finish breaking the constraint of the permitted time the penalty is being calculated to the optimisation function.

3. Lowering quantity of schedule changes that introduce limited value:

   An operation team may face huge challenge when the new schedule is completely different than previously set and overall expected value after calculation is still quite comparable with improvement gain when shifting one or limited portion of the jobs

[13]This criteria also reduce a type of optimisation errors due to pushing all batches out of the window of the scheduled plan and by this minimising the function to lowest possible value.

subset. From the business and human–operator perspective it is better to shift one batch that hundreds for the similar value; simply it is just easier for humans to monitor the action and evaluate the next round.

In order to prevent the optimisation to give as a solution, one could argue a chaotic set of schedules (likely to be find when the search space is flat) with new batch job times b_{t_0}, the objective function can be extended by an extra coefficient that would inflate the value slightly when the t_0 is further from the original position.

4. Avoidance of high SLAs of batch jobs.

   High SLA values in history data found for system states $\mathbf{S}$ are under consideration in the search. High SLAs differences and disproportions to earlier observed values may be symptoms of large utilisations. That should be avoided even if total SLA under whole time bucket and search time window is lower and optimisation on high executions times works as expected. In most cases a new execution time of a batch job can be derived from calculated resource footprint by found in ASM-SD mixing matrix M.

   Additional boundaries should prevent such situations.

5. Evaluation of "positive" and "negative" outcome of previous control actions:

   Since the control process of batches has more proactive planning activity characteristic rather than very reactive control style as in action termination, see Sections 7.4.4 and 8.5.2, the earlier control evaluation process and rules induction are integrated–into the search process.

   This part could be equally analysed above when details of the optimisation function processing were tackled. Nevertheless, it has been explained as a constraint since the evaluation part of the new schedule search process relies on further penalisation as avoidance of "negative" control decisions implications. Those "positive" will not be additionally reinforced.

   Therefore evaluating "positive" and "negative" outcomes of previous control actions, as in Rule 2 explained in Section 8.5.2, is now part of the ASM-SD process and fitness function implementation with constraints[14]. Rule 3, the epsilons–greedy strategy is applicable to scenarios where search space around minima is flat – in such case stochastic optimisation algorithms like metaheuristic search, e.g. SANN, PSO or GA naturally support that fit.

6. Intervention on the batches selection:

   Based on performed SDCIC and ASM-SD methods the human operator or part of an automated decision maker can allow only certain batches to be scheduled. This is a similar scenario to outlined above since one could argue this part is the core scheduling fitness function. Analogously an administrator could select a shorter window of time for the schedule optimisation, i.e. per a given geographical region

[14] In fact this reflected implementation of the Control Scheme of Type 2, see Table 8.2.

business time, e.g. during the night before the business day in Europe making sure all batches end when the business start there.

7. Grouping the batches:

   The batches could be grouped in categories of different execution profiles, where batch classes of similar resources footprint and execution times are replaced with a new type of a batch for the global batch system optimisation. Such a step will greatly reduce the search space and smaller the mixing matrix M. Another dimension worthy consideration is a entry–results–parameters chain dependency between batches. Especially in situations when the expected execution time of the whole sequence of batches is an important factor, such linked batch jobs may create a new batch type.

8. Cap on selected resources utilisation:

   Even after considering footprint on resources of expected from the previous cycle actions, an operator group would like to set up a margin of headroom for critical resources like CPU, IO, DB pools, or memory so they cannot be higher than a certain configurable level.

   Linked with scenario (4) this boundary is especially important under real–time conditions declared under SLAs of other action classes. The boundary can be present in specific time–windows or operating regions of system states $\mathbf{S}$ when there is projected risk of higher load coming from actions.

9. Regulating expected tardiness versus promptness:

   In certain situation it may be more in favour to delay batch processing than shift them backwards. The operator team may request in calibration phase this condition. This is especially important in batches that are scheduled in sequences since there is a data level dependency (input of some rely on results of earlier batch jobs) or there is a hierarchy of batch jobs.

   In such situations the objective function can be extended by another linearly or exponentially building value of penalty when the t_0 increases.

   This can be an interface for yet another meta–regulator used by human–operators who can chose expected tardiness versus promptness of a given or group of batch jobs in the new schedule.

8.8.7 Objective Function and Constraints Implementation

The search space of Eq. 8.2 is complex with multi local minima. Thus, selected meta-heuristic methods that typically exhibit good performance in such conditions are tested (Martello, 1990; Pisinger, 1995).

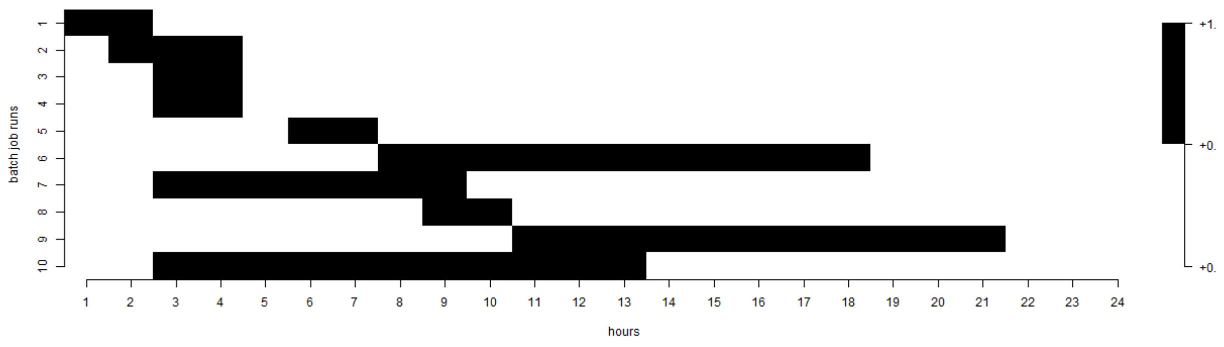


Figure 8.16: Visualisation of batch jobs execution time for the simple scenario, where the time bucket is an hour and the optimisation time window is 24 hours.

Exact algorithms or traditional heuristics that also give satisfactory results even in high–dimensional problem will not be tested.

In order to determine the next round of control strategy by finding better schedule of batch jobs following search approaches have been explored: SANN (Kirkpatrick et al, 1983), PSO (Kennedy and Eberhart, 1995; Poli et al, 2007), and GA (Goldberg et al, 1989; Mitchell, 1998; Michalewicz, 1996). In all of those stochastic optimisation methods solutions evolve following a compatible style of objective function, so the core functionality does not have to change significantly. All of the approaches are well explored and have generally available, stable implementations in many modern platforms.

Intuitively the data representation of the problem is a two–dimensional table with batch jobs and time buckets as presented on aforementioned Figures 8.13 and 8.14. However, in order to ease the search of new start times b_{t_0} it is easier to convert batch jobs data into form of Gantt–like charts presented on Figures 8.12 and 8.16, where there is no risk of a collision with another job[15]. Figures 8.17 and 8.18 demonstrate ordered schedule by start of the batch job in order to help the reader to see the overlapping time regions and those times where there is no actively operating batches.

The testbed is implemented in Java, thus the easiest to integrate it was to use Java implementations of above search algorithms. Following have been used: for SANN: jannealer[16] (Mégnin, 2003), for PSO: JSwarm-PSO[17] (Cingolani, 2006), for GA: Jenetics[18] (Wilhelmstötter, 2012).

[15]Still, an attention needs to be paid to problem of dependencies between batches highlighted above.
[16]jannealer: Java Simulated Annealing Package, version 1.1, updated 2003-04-02
[17]JSwarm-PSO, version 2.08, updated 2009-08-18
[18]Jenetics: Java Genetic Algorithm Library, version 4.3.0, updated 2018-10-28

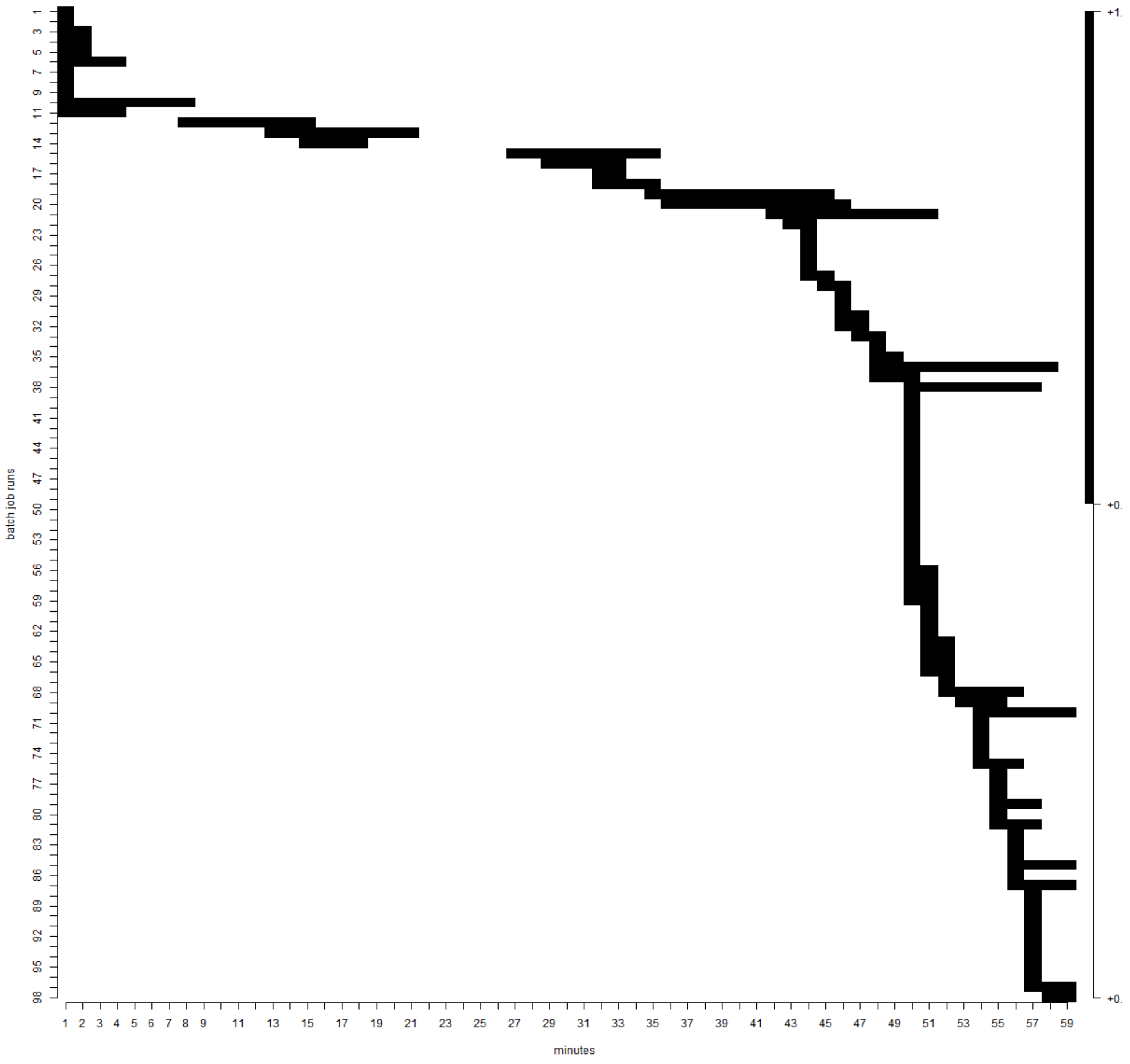


Figure 8.17: Visualisation of batch jobs execution time for the medium complexity scenario (before optimisation), where the time bucket is a minute and the optimisation time window is one hour.

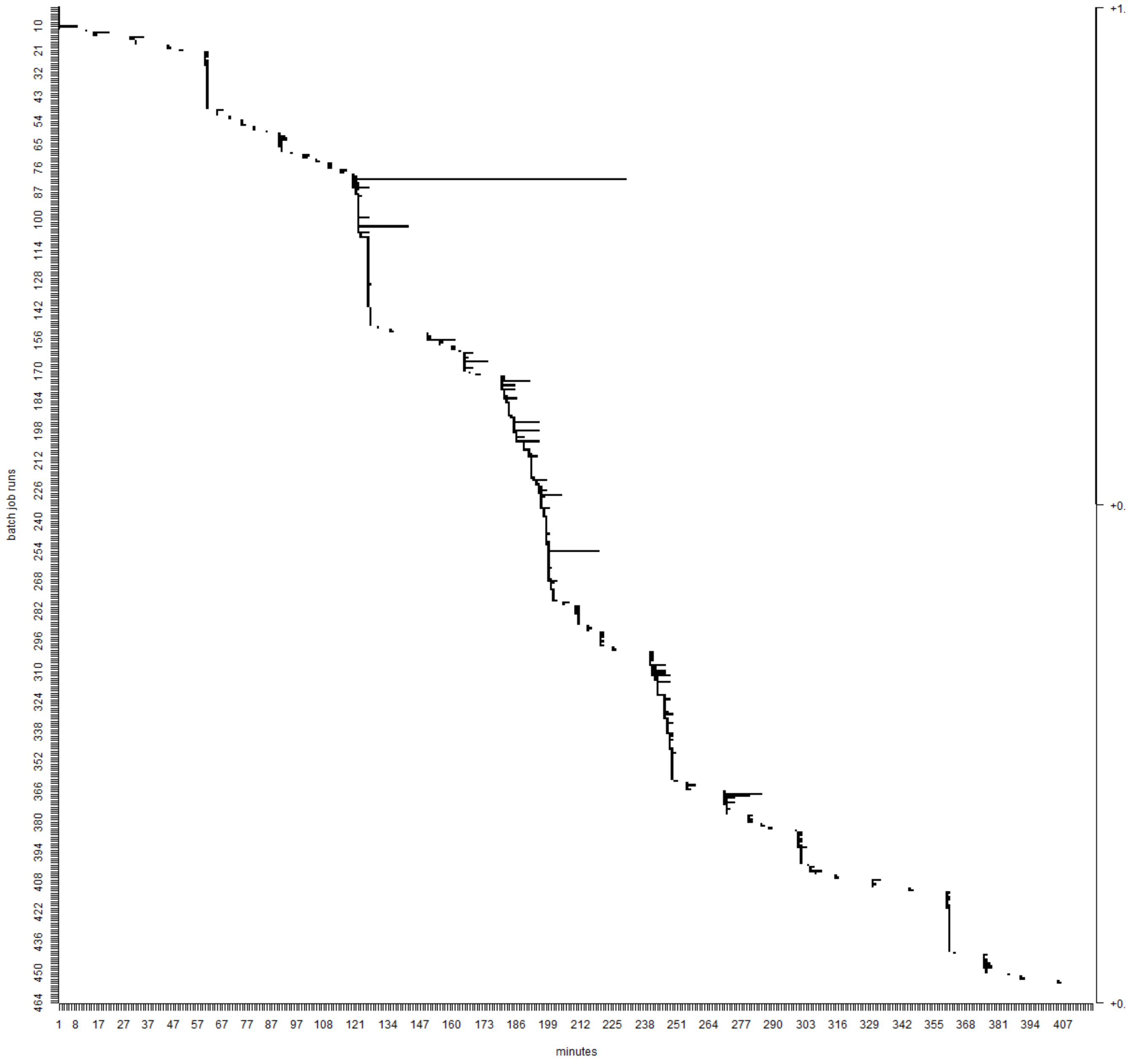


Figure 8.18: Visualisation of batch jobs execution time for the complex scenario (result after the optimisation), with 465 batch jobs and where the time bucket is a minute and the optimisation time window is 7 hours (time before a start of the business day)

8.8.8 Actions Micro–scheduling

Action with termination during execution is in fact a form of micro–scheduling, see Section 8.7, where the neural block model contains direction on control, and no extensive search process is needed. Also the required instrumentation to perform micro–scheduling actuation is the same. Thus, in a pure form, micro–scheduling is difficult to be applied and perhaps not much beneficial for most of the modern UI/API driven actions that as per design execute under a sub second. Search is an expensive process, so there are two options to alleviate that challenge: (a) reoccurring online–optimisation for selected action types setting a tight end of job execution actioned by a termination actuation, (b) the entire class of actions could have a cyclically re–calculated delay, this could be a good solution especially for functions running under FaaS model. Furthermore, ESB or asynchronous brokers normally use queuing messages and could employ delays to reduce key resources utilisations.

8.9 Scheduling Process Experiments

Regardless of a scale of details used in the optimisation function delineated in Section 8.8.5 the metaheuristic methods used ought not to change the general nature and complexity of the search. The simplifications due to elimination constraints calculation or limitations in core optimisation function should have constant effect on search execution time.

In order to investigate the runtime characteristics of different search methods there are three scenarios generated based on ASM model data, that explain different system load situations for the search problem in measured SLA conditions, named: simple, medium and complex. Figure 8.16 presents the simple scenario with a schedule of 10 very long running batch jobs (could be groups or longer sequences of jobs), where the time bucket granularity is an hour and the time window of search is 24 hours long. Medium complexity scenario was shown in Figure 8.17 containing 100 batch jobs which run up to a few minutes, so the precision of time bucket length was set to a minute and planning horizon an hour. Whilst the most complex scenario containing 465 jobs planned of total expected time equal to 978 minutes for 7 hours planning window with a minute time bucket precision was presented in Figure 8.18.

As mentioned earlier in Sections 8.8.3 and 8.8.7 the approach to the scheduling problem was tested with use of three general scope metaheuristics methods: SANN, PSO, and GA; Figures 8.19, 8.20, and 8.21 show the details of dynamics of the search trajectories for all applicable scenarios respectively and different base populations. Navy blue cross points indicate solution values in function of subsequent generations, red points mark the worst and green the best solutions found so far.

Table 8.8 contains totals of performance observations of the scenarios experiments. The

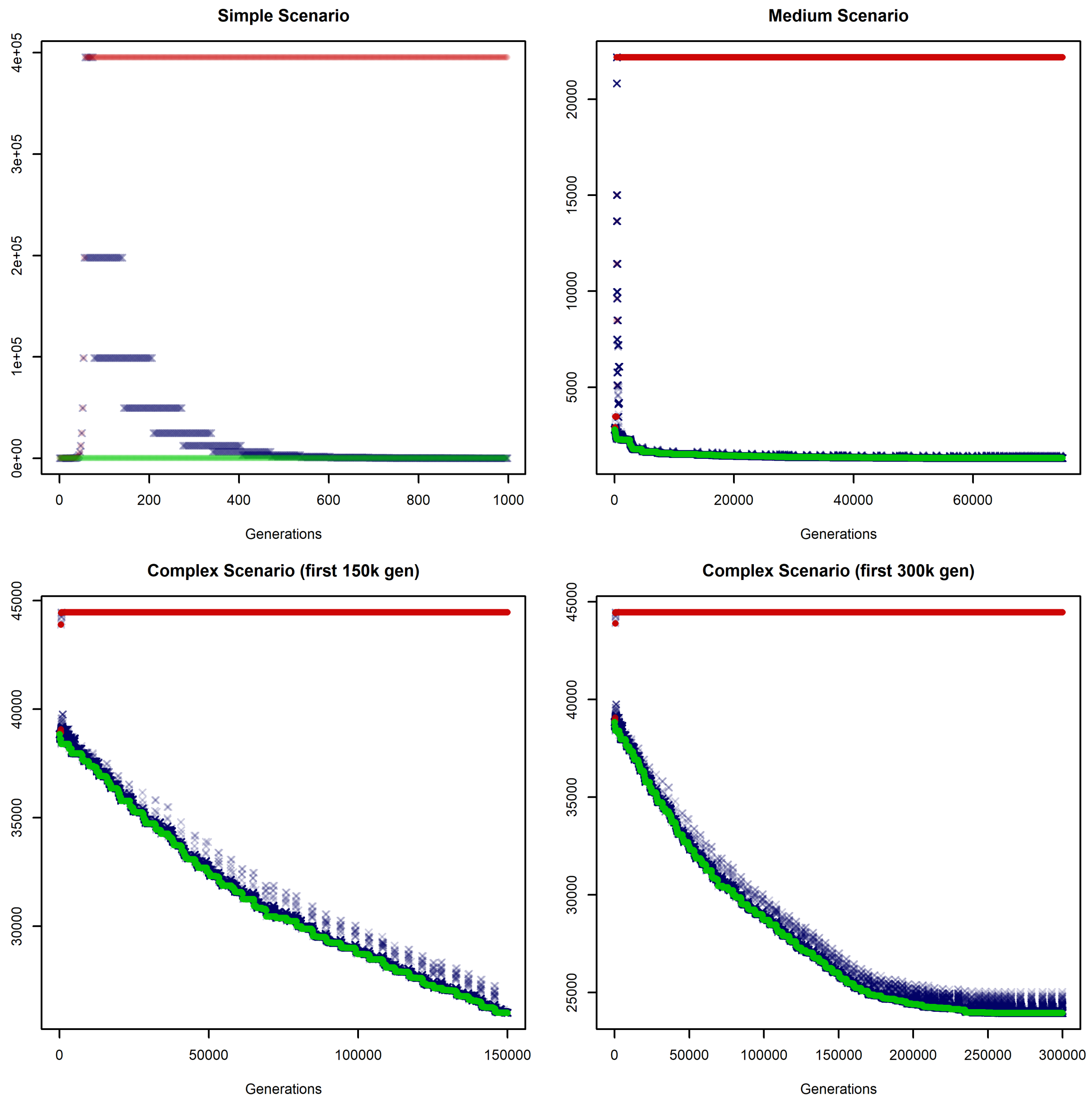


Figure 8.19: Comparison of batch jobs optimisation for SANN across all 3 scenarios.

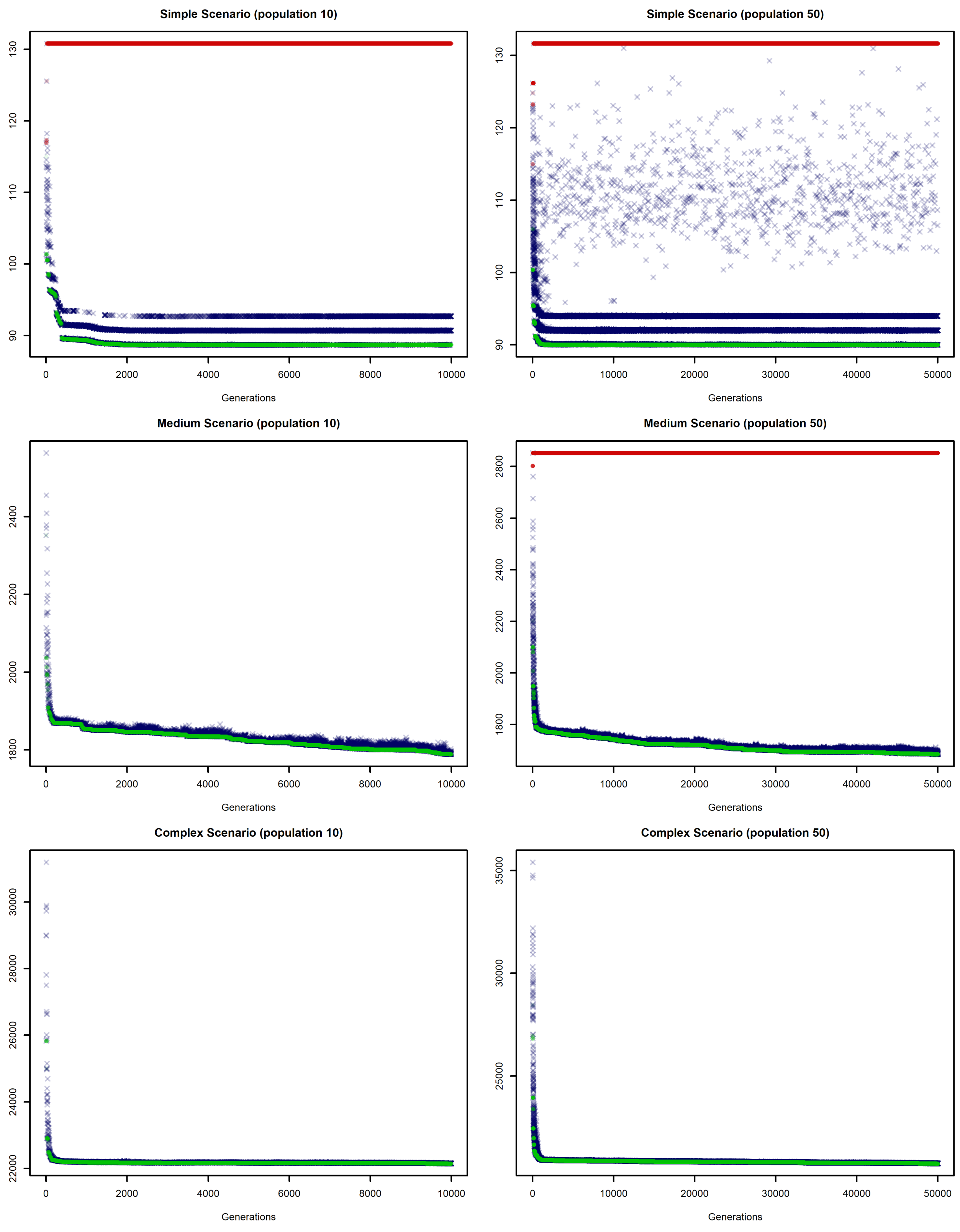


Figure 8.20: Comparison of batch jobs optimisation for PSO across all 3 scenarios.

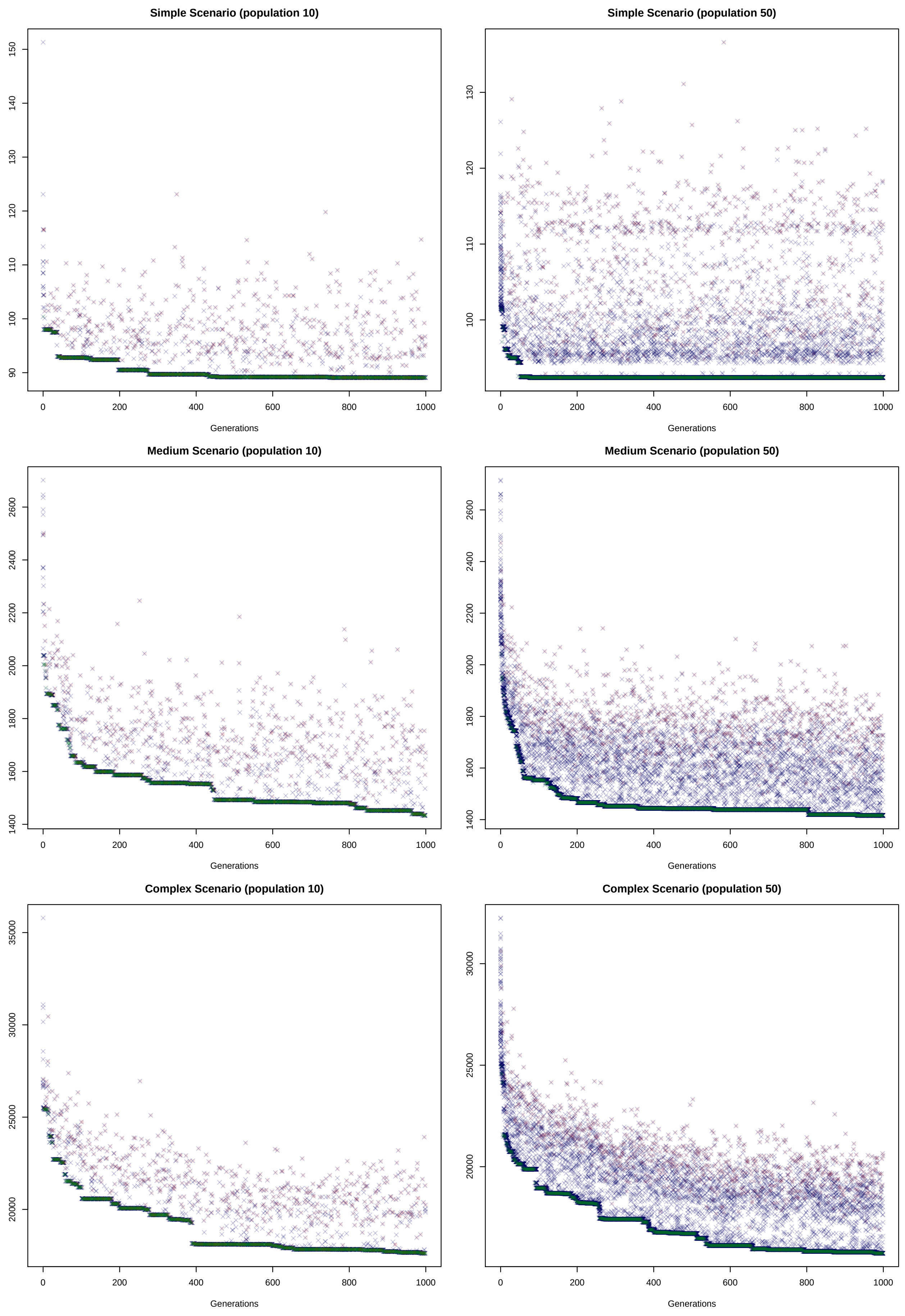


Figure 8.21: Comparison of batch jobs optimisation for GA across all 3 scenarios.

Table 8.8: Results of batch jobs rescheduling process execution times for simple, medium and complex scenarios.

Problem, Space Size	**Evaluations (Population x Iter. Limit[a])**	**SANN (jannealer)**			**PSO (jswarm-pso)**			**GA (jenetics)**		
		ET[b]	BS[c]	(I%[d])	ET	BS	(I%)	ET	BS	(I%)
Simple [e]	50k (50x1k)	173ms	88	(76%)	131ms	93	(81%)	587ms	89	(77%)
	100k (100x1k)	in 6248 iterations			225ms	89	(77%)	770ms	88	(76%)
Medium [f]	10k (10x1k)	11sec	1303	(46%)	840ms	1680	(60%)	899ms	1507	(53%)
	50k (50x1k)	in 224k iterations			2967ms	1611	(57%)	1307ms	1407	(50%)
	100k (100x1k)				9356ms	1571	(56%)	2524ms	1380	(49%)
Complex [g]	10k (10x1k)	3700sec	23917	(62%)	11sec	18718	(48%)	6sec	19249	(49%)
	50k (50x1k)	in 11.5M iterations			37sec	18638	(47%)	18sec	15220	(39%)
	100k (100x1k)				53sec	18632	(47%)	31sec	14488	(37%)

[a]for PSO and GA iteration limit is 1000 (1k) for all runs
[b]Execution time
[c]Best solutions value
[d]Improvement as a percentage of original solution value
[e]Simple scenario, $(24h)^{6b}$, no change solution value is 115, Figure 8.16
[f]Medium scenario, $(60min)^{100b}$, no change solution value is 2802, Figure 8.17
[g]Complex scenario, $(7h \times 60min)^{465b} = (420min)^{465b}$, Figure 8.18

comparison shows, in columns respectively: search space size, algorithms run parameters (number of evaluations as stopping criteria, population size, iteration limit), execution times, best solution values and percentage of optimised solution (the lower the better) for each of the methods. All values presented are averages of 10 subsequent search runs.

As we can see SANN seems to be outperforming other methods in simple scenario able to find better solutions faster. Unfortunately it is quickly loosing ability to perform well under more complicated scenarios. Both PSO and GA operate very well in medium and complex scenarios. GA is finding better solutions and faster which may be a result of more recent implementation able to utilise all cores in full capacity.

8.10 Evaluation with Model–driven Prediction

The experiments scaffolding constructed allows on profitability analysis of system deployed in SaaS, FaaS or PaaS realm. It permits researchers, software developers, enterprises and cloud service providers, respectively, interpreting flexible SLAs measures, actions count and SLA values in a context of load to the systems and its global state. DES model contains built–in assumptions of vital elements of general queuing characteristic that is essential and core phenomena of computing systems. As explained earlier in Section 8.4 the testbed design follows the model-–driven experimentation approach, where the DESMO-J model implementation of the computing system can explain the details of CPU, disks and memory interactions on load and type of action. Based on queues parameters the model can differently reflect specifics of action types and their run–time interference in the same EIS ecosystem, see Section 8.4.1.

Since the model of the system can give answers to queries of the resources footprint characteristic and actions execution time, it can be used to prepare the extended training set of future control actions for more hypothetical, possible, and with some certainty expected system states $\widetilde{\mathbf{S}}$. Therefore this property can be used as a predictive measure that can be used by autonomous control system in a form of a grey–box system. Model–based performance prediction using DESMO-J has been studied by Becker et al (2007, 2009) and by Reussner et al (2011, 2016) in order to simulate performance to support design and specify component–based QoS–driven software architectures in a parametric way.

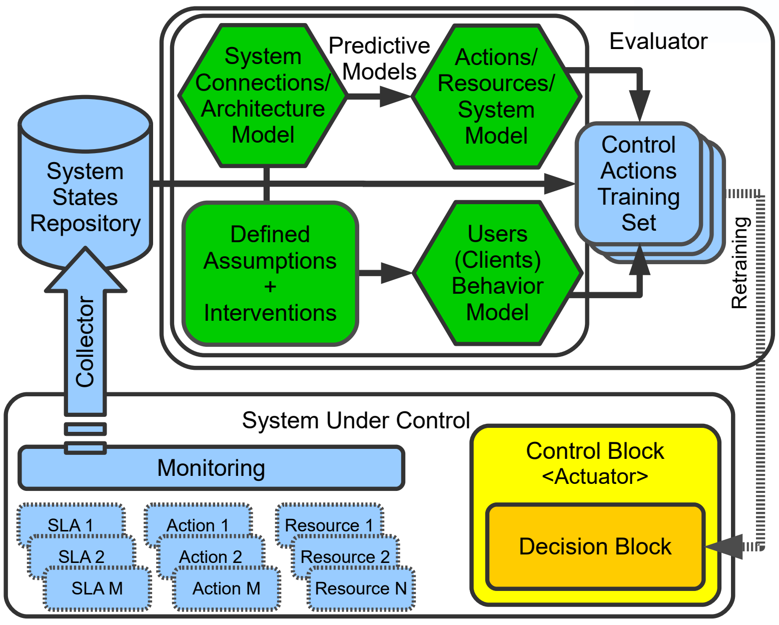


Figure 8.22: Architecture of the controller with predictive Evaluator (top) and neural decision block (bottom). The evaluation processing combines response of many models that support the Evaluator in simulations of anticipated situations and other not yet measured system states that give more precise training set for neural decision blocks deployed to the system or weaved into the code of application.

Figure 8.22 illustrates an example of a controller architecture allowing the evaluation process to rely on many complementary models like neural, statistical or queue models that support the Evaluator component, see Section 8.5.2, in simulations giving base to predictions of anticipated situations and other not yet measured system states, $\mathbf{S}$, that give more precise training set for neural decision blocks, see Section 8.5.3, deployed to the system or weaved into the code of application to deal with action termination, see Section 8.5.4. Those models can depend on collected systems states data[19], with methods and capacity discussed so far in the thesis, however, can also incorporate such elements as: (a) description of the systems explaining structure of the architecture, systems connections, or even code–base, that would offer stronger ground for causal inference of control actions enforced by (b) assumed by operators characteristics and tested interventions. The models containing joint run–time characteristics, $\mathbf{C}$, of actions, a, resources, r and response to measured SLA, f_{SLA}, can assist in more comprehensive evaluation process and better

[19]The model parameters explaining of how resources are used can be established in load test or based on selected form data in isolated runs, and then in normal work can be adjusted by admins or retrained automatically.

training sets for actuators. While the Evaluator (at the top) can work simultaneously to the EIS and ASM control system operation, the decision block (at the bottom) needs to constantly stay active in alert to deal with ongoing monitoring and control actuation. Thus, the responsiveness and performance of the selection of system states, **S**, observations consolidation, categorisation, judgement of earlier and future control actions is not that critical as efficiency of decision block where actuations must be done in a sequence of the request processing. Furthermore, the evaluation can be executed more sporadically, as per a given schedule (as in the above sections), or per a specific event like quite new situation of far different system usage or lower control quality that require re–positioning the control rules. Since this processing requires substantial computation power its execution should be optimised with the aligned views of the overall resources utilisation of the wider control and production system, see Section 8.8. Consequently, the decision blocks form and design should be simplified to bare minimum, so it does not allow on more complicated, elaborately processed control steps, and support fast business context request processing, under high load, to satisfy high throughput and minimal latency NFRs, especially important in context of real–time SLA guarantees. In general, the control block deals with tactical control decision making, whilst the evaluating process is responsible for more strategic planning of most probable eventualities [20].

8.11 Summary and Contribution of the Chapter

In EIS high load situations and requirements for stable service at times of peak demand are typically dealt by increasing computational power (i.e. auto–scaling) or by releasing load (i.e. terminating incoming, rescheduling planed activities). This chapter was focusing on non–acquisitive control methods, where establishing the right priority of control plays key role, that in this work is derived from SLA definitions, evaluation of system states and observation of the current situation.

The experiments identified many benefits, qualities, but also implications of the termination control and rescheduling batches when it is applied to different action types of distinct run–time characteristics under dedicated service level objectives by SLAs.

The limitations and potential of application to the ASM tools and industry with a special focus of a context of non–acquisitive control techniques will be discussed further in the

[20]This design foundations remind a situation of (a) conscious and (b) subconscious and sleep processes of a human brain, which is of course far more complex, but in principle, give some ground of analogy to above assumptions. The subconsciousness, which is the part of the human mind that is not currently in awareness and operating associations and impulses outside of consciousness, is analogical to the processes of systems states monitoring, collection, persistence and evaluation. The conscious thinking, when mind is aware of senses, cognitive processes and perception is active, is comparable to control actuation as a tactical operation of the controller in the system. The need of even immediate (when global resource permit) execution of the evaluation process (full or partial), which is inferring better control actions as a foundation for adaptivity principle, is similar to human ability to imagine that is basically a capacity of human brain of simulating events consciously with use of earlier gained experience.

next Chapter, see Sections 9.5.1 and 9.5.15.

Chapter 9

Applications

"The important thing in my view is not to pin the blame for a mistake on somebody, but rather to find out what caused the mistake."
— Akio Morita (1921 — 1999)

"You've got to start with the customer experience and work back toward the technology – not the other way around."
— Steve Jobs in 1997 (1955 – 2011)

This chapter reviews the research findings in the background of market and industry needs. It addresses questions where this work can be applied, incentives for adaptation, but also discusses open problems and the approach limitations.

The industry application realm was guiding the research of ASM control in area of enterprise systems since the very beginning. Thus, the every method tested was discussed in earlier chapters under pragmatic spotlight. Elements of the approaches were implemented in ready to be employed state, easily applicable to proof–of–concepts suites in the commercial field. Nevertheless, despite the effort, it is worthy to explain the general various needs in Sections 9.1, 9.2, 9.3 and 9.4, to highlight potential of the non–acquisitive control methods in Section 9.5, and to discuss limitations from industry stand point of view in Section 9.6.

9.1 Enterprise Systems Development

ASM systems contain definitions of artefacts and their connections explaining the architecture and collect run–time data of system states, **S**, in order to explore unknown execution characteristics of systems under observed clients activities and given control interventions, see Chapter 4. Building the total system model, **C**, is difficult due to vast problem space and many uncertainties, see Chapter 5 and 6. All control methods were designed and investigated in a context of targets gaining better financial performance, P, and customers satisfaction defined by SLAs functions, f_{SLA}, so driven by truly business directions, see Chapters 7 and 8.

Although, the background of interactions dynamics between customers, software and deployment architectures are deep and complex, the nature and emerging standards of key elements of EIS requirements are certain and commonly observed. Let's review them from three separate perspectives: (1) architecture, (2) deployment, (3) systems development.

1. Architecture:

   (a) Systems clients' requirements shift towards subscriber model of service, thus architecture paradigms align around high scalability principles.

   (b) Microservices and many smaller integrated systems following promises of SOA paradigm under orchestration form wider often distributed systems (Lewis and Fowler, 2014; Dragoni et al, 2017a,b; Zimmermann, 2017).

   (c) Enterprise architectures and foundations of web, mobile and services exposing systems functionalities need to be flexible to allow them to last for longer; the redesign and implementation are part of operations, and refactoring is a continuous effort (Fielding, 2000; Coplien and Bjørnvig, 2011; Fowler, 2018; Humble and Kim, 2018).

   (d) Systems require communication of more and more distributed geographically services providing computation and storage amongst many locations forming Internet of Things (IoT) under concepts such as edge or fog computing (Bonomi et al, 2012; Singh et al, 2021b).

   (e) Applications are progressively seen as business logic packages that are just one of the system parts; others are supporting more NFRs like monitoring, access control, audit, performance logging are more and more often extracted in a form of external and reusable services.

2. Deployment:

   (a) Recent years of software development highly utilise deployments using cloud computing (Briggs and Kassner, 2016; Ford et al, 2017) that is heavily reliant on virtualisation and containerisation (Goldberg, 1974; Rosenblum and Garfinkel, 2005; Smith and Nair, 2005; Watada et al, 2019).

   (b) Many multi–tier and cross–platform systems form larger ecosystems of loosely–coupled enterprise applications (Hohpe and Woolf, 2004; Fowler, 2012; Humble and Kim, 2018).

   (c) PaaS and Application Platform as a Service (aPaaS) powered by containers deployments turns more IaaS, i.e. AWS ECC and Azure containers (Watada et al, 2019).

   (d) FaaS and serverless computing significantly simplify and lower costs of fine–grained services deployment (Baldini et al, 2017).

   (e) There is a progressing business–driven Separation of Concerns (SoC) on coarse grained system level provided by SOA that is powered by cloud computing especially under SaaS but also FaaS model; where different decoupled systems are introducing targeted functionalities are closely integrated with each other (Sharma et al, 2011; Lewis and Fowler, 2014; Pautasso et al, 2017a,b; Dragoni et al, 2017a).

   (f) Business development and operations teams collaborate closely declaring requirements using modelling languages or notations like Unified Modelling Language (UML), Systems Modelling Language (SysML), Entity Relation Diagram (ERD), Service–oriented modelling framework (SOMF), C4 and Business Process Model and Notation (BPMN) offering common ground for widely understood documentation that can be integrated with service management systems – cloud computing vendors (AWS[1], MS Azure[2], IBM Cloud[3], Google Cloud[4]) offer specific dialects of graphical modelling to define required architectures notation (Fehling et al, 2012; Norton, 2016).

3. Development:

   (a) Data collection and processing of performance metrics are intertwined with business targets; monitoring of business and operational metrics is essential not only for SLAs, but for more aggregated, coarse–grained KPIs and other economic factors (Reis, 2011; Humble and Kim, 2018).

   (b) Financial performance, P, is incorporated into projects and maintenance work–streams giving insights for budgeting and Return of Investment (RoI) calculations; experimentation, design, development, deployment and long term maintenance are part of iterative phases proceed under CI/CD regimes, where infrastructure expenditure is just one part of the larger cost equation (Poppendieck and Poppendieck, 2003; Reis, 2011; Humble and Kim, 2018).

   (c) Cycle of business requirements building, starting from domain experts exploring the business areas and subsequently involving in tight team collaboration of specialist like analysts, designers, architects, developers, dev–ops, testers (that have wider vision and overlapping domain expertise, but separately defined roles), ending the feedback–loop on customers and domain experts interacting with the system (Beck et al, 2001; Fowler and Highsmith, 2001;

[1] https://aws.amazon.com/architecture/icons/
[2] https://docs.microsoft.com/en-us/azure/architecture/icons/
[3] https://github.com/ibm-cloud-architecture/ibm-cloud-stencils/
[4] https://cloud.google.com/icons/

Poppendieck and Poppendieck, 2003; Highsmith and Highsmith, 2002; Cockburn, 2004; Cohn, 2004; Cockburn, 2006; Janes and Succi, 2014; Stellman and Greene, 2014).

(d) Validating whether a change meets NFRs needs to be as short as possible; modern standards of CD require the team to mark the change as ready in a business day (Humble and Farley, 2011; Humble and Kim, 2018).

(e) RCAs of problems found and issues reported in fast feedback–loops need to be very fast; ideally starting and finishing in one business day.

(f) Fast adaptation needs are present on many organisation levels driven by such factors like: market–business niche shifts, clients behaviour, organisation structure, technology changes, methodology, and finally day–by–day fluctuations of load (Beck et al, 2001; Poppendieck and Poppendieck, 2003; Reis, 2011).

(g) Empowered and multi–skilled development teams craft and maintain products in a lean and agile fashion following various levels of feedback, see Figure 9.5; from PDLC, SDLC, via agile ceremonies, to micro feedback loops organically improving development team efficiency (McBreen and Hendrickson, 2002; Ambler and Lines, 2012; Hoover and Oshineye, 2009; Martin, 2009; Bass et al, 2015; Sussna, 2015; Kim et al, 2016; Larman and Vodde, 2016; Leffingwell, 2018; Humble and Kim, 2018).

All above presented EIS development trends state crucial requirements for the modern enterprise systems, which define the specification space of new class of ASM control systems. The ASM suite needs to react to holistic needs that drive optimisation from perspective of computing resources, organisation, software, and market interactions, see Figure 9.1.

Proposed novel approaches to non–acquisitive control methods like actions termination and rescheduling explained in Chapters 7 and 8 offer another level of service adaptation that typically reduces the need for computation power at times where the given service is impacted by higher than anticipated load.

In order to prepare the ground for discussion of specific applications and potential of the methods discussed further in Section 9.5 it is important to explain the rationale of market and industry needs in a background of crucial limitations the industry faces in Section 9.2, review of other, often non–technical requirements related to organisations and operators needs in Section 9.3, and discuss constraints of software–level control in Section 9.4.

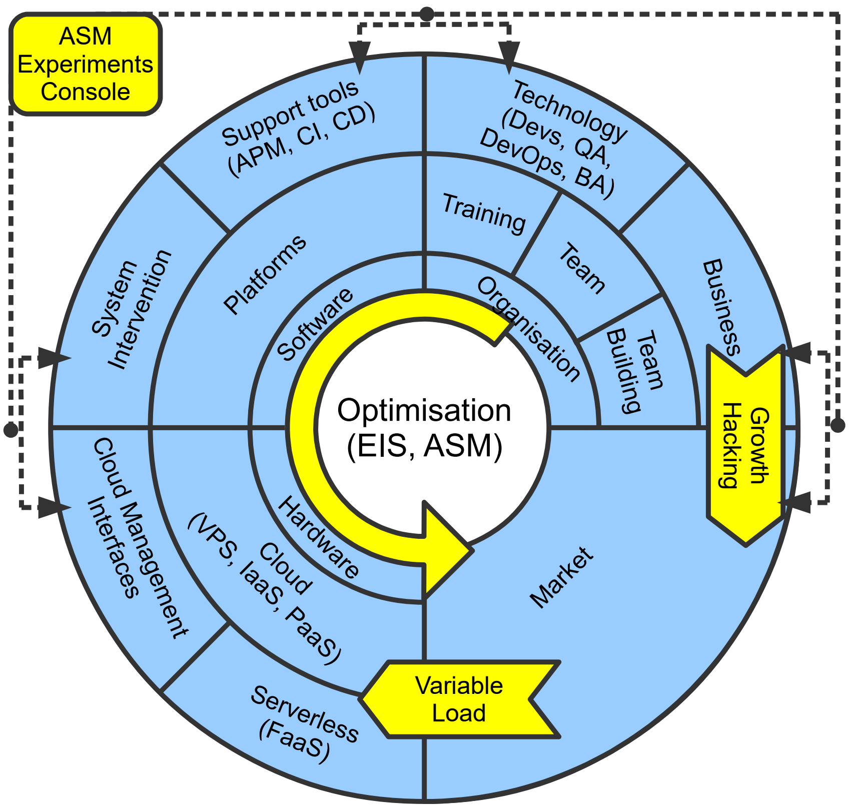


Figure 9.1: Circle of organisation, software, computing resources and market interactions governed by ASM suite for holistic optimisation needs. Platforms and architecture in showed in bottom–left corner have been outlined in this section, for bottom–right "Market" see Section 9.2, for top–right "Organisation" see Section 9.3, for top–left "Software" see Section 9.4.

9.2 Market and Industry Needs

Over 20 years ago a great enthusiasm related to the new economy based on information technology available to everyone who could open a web browser started widespread judgements that foundations of economic principles were no longer applicable to the new world supported by internet technologies. That false thinking was a foundation of the dot–com boom, bubble, and finally, the financial crash of 2000–2001 (Buenstorf and Fornahl, 2009). Although many technology companies were able to endure the crisis (it is estimated that 48% of dot–com companies founded since 1996 survived through till 2004) their valuations were small fraction of maximal values around their peak (Goldfarb et al, 2005). The global acceleration of e–commerce and new trade strategies (Plant, 2000), economies of scale and international technology–integrated industry (VanHoose, 2011) introduced new means of differentiation of goods (Reynolds, 2000). As the Information and Communications Technology (ICT) sector stabilised and technology companies consolidated, many larger players such as eBay, Amazon, Google and IBM naturally gained more market share and spread domination in their market niches of more specialised services.

As early as in 2002 European Commission through the Directorate General Information

Society activities triggered discussions around factors improving Small and Medium–sized Enterprises (SME) in area of business and e–commerce needs. Those led to building conceptual framework of digital business ecosystems as an advanced networked organisation models for ICT companies. The discussion identified several impediments such as lack of technological solutions and interoperability, or high costs of the introduction of e–business practices to live systems, but also highlighted the shortage of knowledge and technical skills in SMEs that are backbone of western economies, as a great deficit slowing further local development of the information era (Nachira, 2002). United States agencies were tackling similar problems by many initiatives researching and improving e–business adoption (Korchak and Rodman, 2001). Further socio–economic development of 2000s catalysed by dynamic changes of ICT structure and new technical means required even more emphasis on building the co–evolution between the business and the technology forming the digital ecosystem; as a complex system established on such pillars as free market economy of services, formal languages that evolve and proliferate, open source movements, common (possibly regulated) practices, and communities found on open knowledge and open governance principles (Nachira et al, 2007a,b).

In order to give a commercial response alleviating those problems and optimise internal capital and operation expenditure larger organisations began providing their infrastructure for other enterprises. From 2006 such organisations like Amazon, Google, Microsoft or IBM[5] started sharing their excellent computational power and resilience potential under platforms and the term cloud computing was coined[6]. In 2008 Gartner (Plummer et al, 2008a,b) outlined cloud computing as an emerging delivery model for wide ICT sector, that offer massively scalable, resilient and elastic computing services. The report defined the field as "a style of computing where massively scalable IT–enabled capabilities are delivered 'as a service' to external customers using Internet technologies.". Thanks to decisions which led to focusing on commodity parts cloud providers were able to quickly scale the model to growing demands of clients small and bigger enterprises.

More on the historical background the reader can find in the documentary Appendix Section B.1.

Following the software improvement review the explanation will now move to analysis of human and organisation requirements in Section 9.3 followed by software execution control constraints discussion in Section 9.4.

[5]In 2006, Amazon created AWS and introduced its AWS EC2. In 2008, Google started Google App Engine (GAE). In 2010, Microsoft released Microsoft Azure. In 2011, IBM opens the IBM SmartCloud. In 2012, Google releases Google Compute Engine (GCE) as IaaS response.

[6]The first wide publicity use of "cloud computing" in the currently understood context occurred on Search Engine Strategies Conference stated by Google CEO Eric Schmidt on August 9, 2006.

9.3 Human and Organisations Requirements

This section will be discussing context of other non–technical requirements that are necessary to understand the wider picture of EIS systems working under ASM frameworks.

In order to define areas of specific applications of researched non–acquisitive control methods, see Section 9.5, it is important to highlight specific needs that are present in organisations employing ASM suites in a background of: (1) limitations of full autonomous control – Section 9.3.1, (2) risks of dispersed knowledge, where no single agent has a full information – Section 9.3.5, (3) low transparency and accountability tracking – Section 9.3.5, (4) psychology of individuals and groups, exhibiting certain human oriented limitations – Section 9.3.4, (5) distributed teams of many roles, especially visible in larger organisational context – Section 9.3.5, (6) human cooperative control, and augmented intelligence.

9.3.1 Automation Limitations

It has been highlighted on several occasions in this study that the adaptive controller, alone, without human interaction faces several challenges impacting the control performance. The organisational, commercial and technical optimisations are main fields of exploitation of this work. Of course there are also philosophy, ethics and safety limitation aspects related to autonomous controllers which will not be tackled in this thesis (Russell and Norvig, 2002).

First of all, the space of the control problem is very complex and learning with interaction with the plant consumes significant time. Solution speeding and intensifying the process is to allow the controller to take actions since the beginning. However, this is coming with certain risks of significant overreactions, as discussed in Chapter 7.

Every EIS system is different and the ASM suite need to be deployed and installed in a context of a complex ecosystem of interaction and dependencies. Thus, the system needs to be parameterised and allowing flexible setup, so the coordinated by by human operator tool can be further exposed to other teams. It is a clear observation that ASM suites equipped with adaptive controllers require larger amount of preparations. Therefore the performance of the ASM suite effectiveness depends on highly skilled operators.

9.3.2 Joint Human and Machine Interactions: the System Under Control

Adaptive algorithms, some of them tested and discussed in earlier chapters, require tuning and wider context of the system. Many data are collected directly from the live systems still the certain details and integrations need to be setup by domain experts.

The emerging discipline allowing human to interact with learning systems is Interactive Machine Learning (IML). That tackles is the design and implementation of algorithms and Intelligent User Interface (IUI) frameworks facilitating better joint cooperation of human with systems. IML seeks to complement human perception and intelligence by tightly integrating these strengths with the computational power and speed of computers (Dudley and Kristensson, 2018).

Positioning human operator in the loop of learning systems to orchestrate the data input and decide on the qualities of data should be a central goal of the design principles of future ASM. There are several broadly accepted guidelines for the designers of applications that use elements of data–driven control and autonomous online–decision making. Selected solution principles identified based on work of Dudley and Kristensson (2018) conclusions, are:

1. Make task goals, performance and SLA constraints explicit.
2. Support operator understanding of model uncertainty and control confidence – especially important at the beginning of ASM execution or after releases.
3. Capture intent rather than input – i.e. define performance goals for both the controller and development teams, set SLAs changes for re-negotiations.
4. Provide effective data representations and visualisations for insights.
5. Exploit interactivity with user and other departments promoting rich interactions.
6. Engage human operator to manage the right level of concentration, encouragement and arousal.

Figure 9.2 presents an example of sequences of joint effort of human driven modifications and adaptive controllers in a context of SDLC release cycles. The figure shows cycles of human and adaptive controllers engagement for developers, the business representatives, DevOps and AIOps.

9.3.3 Human Cooperative Control

AIOps as a new emerging interdisciplinary field (Prasad and Rich, 2018) tackling needs of systems operations where complexity is steadily increasing to a level where the manual operations around such tasks as system monitoring, anomaly detection, root cause analysis, overload–prevention and recovery become infeasible. The field is in a dynamic growth to become a de–facto standard set of techniques for IT operation management (Dang et al, 2019; Bogatinovski et al, 2021; Arya et al, 2021).

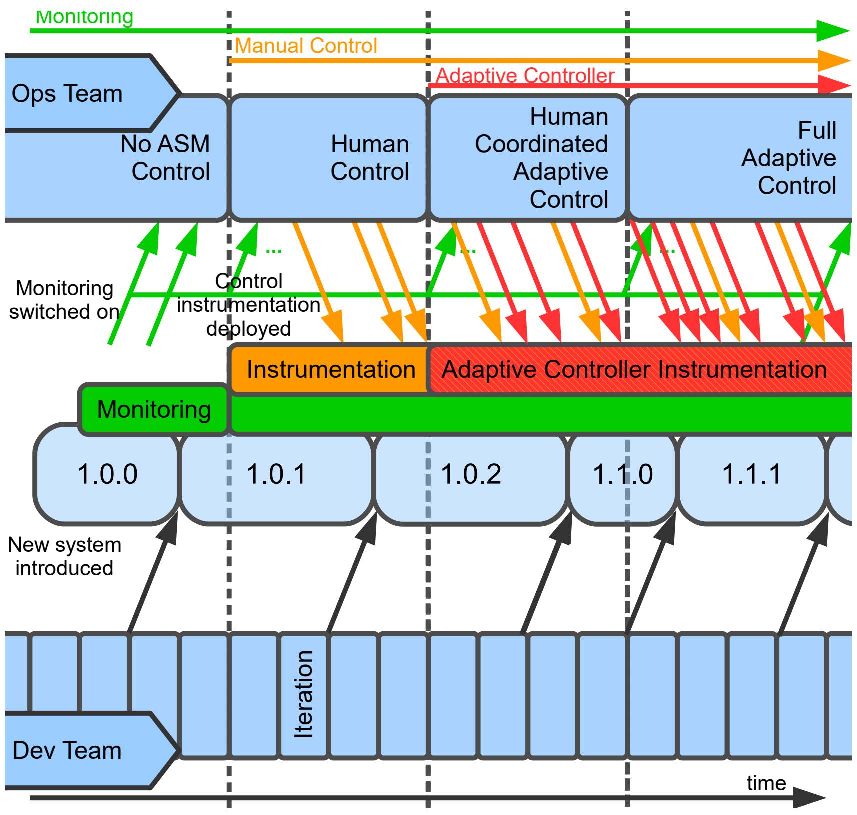


Figure 9.2: Sequence of human–guided control, autonomous and adaptive control.

Building ASM suite to support AIOps features requires holistic design and wide consideration of the whole problem space, from business initiatives, value and constraints, collected data, trained models, to system and process integration, see Figure 9.1.

Manual administrators, DevOps and operators tuning interactions with the system should be recorded, extending system run–time model and define a knowledge–base for future controller actions or just suggested actions performed manually by the staff as extension to used in manual resources management (for not–autonomous ASM goals), see architecture proposed by Hasselmeyer et al (2006).

Cooperation between manually managed ASM system by administrators, DevOps or AIOps decisions, autonomous control and adaptive control is an intriguing aspect. Mixture of interactive and automatic approach can lead to interesting results. Interactive Evolutionary Computation (IEC) use evaluation techniques that rely on human innovation (Takagi, 2001). Usually human evaluation is necessary when the form of fitness function is difficult to determine or not known. In the ASM field administrators' decisions and taken actions can lead to new situations recorded in metrics knowledge repository, which is later explored by evolutionary optimisation methods. Such a cooperative adaptive control with service termination methods has been discussed earlier in Chapter 8.

9.3.4 Human Psychology and Concentration Limitations

Since human operators of the control system are part of the critical element of the wider ASM suite, let's discuss several characteristics that would undermine the performance of the tandem human–machine in the business optimisation realm; so the future designer of the suite could consciously explore the area to take further details into account.

There are several widely acknowledged cognitive biases impacting perception of a problem and decision making. Thay may lead to sub–optimal decision making in software development (Fleischmann et al, 2014; Chattopadhyay et al, 2020), systems architecture (Zalewski et al, 2017) and DevOps fields (Davis and Daniels, 2016), leading to diagnostic errors in software operations (Mohanani et al, 2018). Great source of experience for that purposes can be taken from medical research (Croskerry, 2003). Let's name a few biases: (1) "curse of knowledge" (curse of expertise) – is present when an individual, communicating with others, unthinkingly assumes that respondents have the full background to understand the message, (2) "einstellung" – as an negative effect of previous experience when solving new problems in exactly the same way as earlier – in engineering practice called also a golden hammer antipattern, (3) "anchoring" (focalism) – is a group of biases where an individual depends too heavily on an initial piece of information offered to make further judgements during later decision making, impacting by high assimilation to all future negotiations, implementations, operations, and estimates, (4) "confirmation bias" – estimating the expected value, cost of a solution, or time needed to complete a task wrongly, because of the intentions placed on the team by the business, rather than the true complexity of the problem, (5) "planning fallacy" – an underestimating of a problem by assuming that it could be solved by simple solution without any plan of the implementation, (6) "law of the instrument" – happening when the design patterns or solutions are applied that were well understood even if it becomes clear are not the best for the specific problem faced; Wirth (1995) argues that people are increasingly misinterpreting complexity as sophistication.

Contrary to above stated risks ASM suite should take advantage of human strengths and exploit them, like: (1) experience, (2) imagination as an ability to inspiration, (3) retrospective views with use of analogies.

It is a fairly common situation in large organisations where even very clear and consolidated targets lead to complex management with many sub–programmes that are not aligned, or more, conflicting with each other, e.g. faster delivery versus technical debts removal first, or development–level performance optimisation of a function or adding threshold on time operation. Teams need to deal with the conflicts trying to resolve them in a single wider accepted solutions. Such situation introduce additional communication, decision pressures and higher stress level that should be considered in the ASM suite designs. One of best ways in this MOO problem and to resolve cognitive dissonance (Festinger, 1962) is to convert all discussion to the same unit; calculate the business value added, valuate technical risks to costs or lower revenue. Clients of the suite, such as, business champions, administrators, DevOps, developers, and operators dealing with automated controllers within EIS, can see conversion of business targets, expected value additions, options of tackling and implemented actions, contributing to wider organisation knowledge tracking and building stronger control models supporting the EIS, see discussion of a control based on evaluation with model–driven prediction, Section 10.3.2.

Finally, it is important to focus on engagement factors of human operator to balance the right quality of concentration on communication needs and attention to the EIS under operation. It is critical to design ASM to allow full cognitive orchestration of the control activities, review of data, and decision making based on statistical factors, avoiding stress, unnecessary risks–taking in a context of productive level of arousal. Figure 9.3 presents a diagram of human performance under pressure (Yerkes et al, 1908; Teigen, 1994), depicting performance simply as an attention to detail and ability of decision making in function of pressure as a level of stress and amount of stimuli. Yerkes–Dodson model states; when the stress level is low, the performance is also low. Although this increases as the stress levels also increase up, but only to a certain optimal level beyond which the performance reduces. Also, Gigerenzer and Todd (1999) demonstrate that less information, simpler heuristic demonstrated (Todd and Gigerenzer, 2000; Gigerenzer and Gaissmaier, 2011) can lead to better performance; where emotional arousal, according to Easterbrook (1959), limits the amount of information operators can assimilate or process. Thus, difficult or intellectually demanding tasks may require a lower level of arousal (to facilitate concentration), whereas tasks demanding stamina or persistence may be performed better with higher levels of arousal (to increase motivation). The typical Yerkes—Dodson law illustrated by U–shaped curve chart (at the bottom) was extended by hypothetical a bow–shaped curve (the upper) as an improvement in the performance achieved in ASM human–guided autonomous control, which is one of the goals of human cooperative control. It can be achieved by two directions: (1) "Shift up" – with large expected potential of improvement, especially on the right hand side of the model – higher the general performance by reducing need of interpreting and coordinating many complex details (stop a certain function, selected actions parameters to operate, tune conflict resolution regulator of training for actuator blocks, etc) to simpler and more aggregate control directives that are given based on experience. The ASM UI should be offering better condensed and clear visualisation features, converting plethora of simple actions to complex intellectual tasks, reducing amount of nonconstructive stimuli, wide introduction automation, and supporting organisation with consolidated work–load–distribution adapted to the decision making balancing stress level and fatigue limits. (2) "Shift left" – with average improvement expected – tougher decisions consisting wider context, new initiatives or experiments consisting wider range of impact should be performed in context of lower intensity tasks.

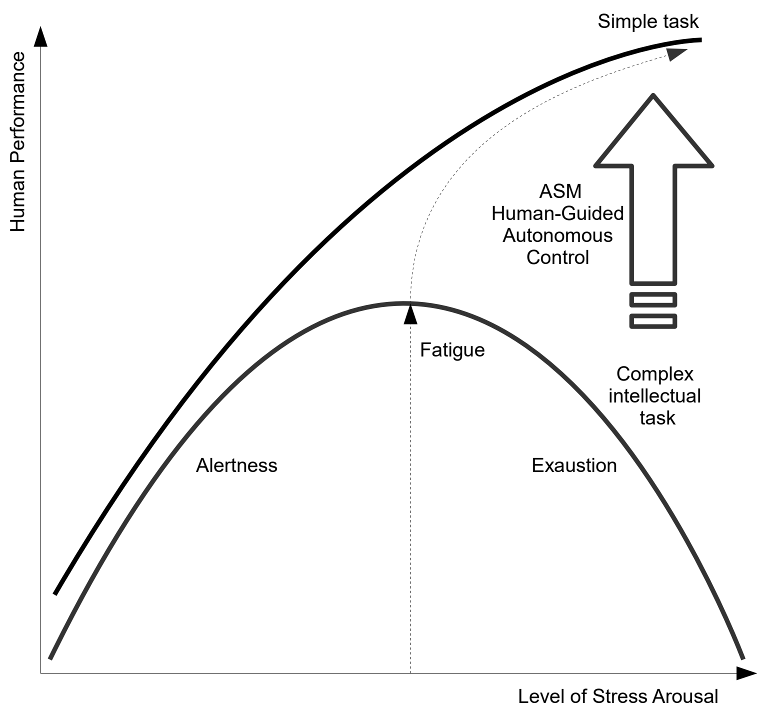


Figure 9.3: Human performance under pressure.

9.3.5 Scaling up the Enterprise and Large–scope Software Development

Modern EIS are being analysed, designed, developed, tested and put into operation often by hundreds and thousands of engineers, who deal with hundreds of systems, thousands of services using many platforms and heterogeneous fashion of deployment. Number of potential one–to–one communication channels grow with binomial expansion $\binom{n}{2} = n(n-1)/2$ with organisation size or n people. Figure 9.4 illustrates the relation between size of an organisation, structure, division to subteams in scaled scrum methodology (Cohn, 2010; Hanoch and Vitouch, 2004; Ambler and Lines, 2012) and impact on scale of the communications. The figure presents the cognitive limits of individuals (Miller, 1956) and impacts to cooperation (Dunbar, 1992) and group networking behaviour (Bernard et al, 1987).

The work following the whole cycle of SDLC must be rigorously checked in terms of NFRs of EIS were APM and ASM play critical role. Brooks Jr (1995) highlights that all the teams working on a project should remain in contact with each other in as many ways as possible. However, huge amount of possible interactions in larger organisations of flat structure quickly exceed human cognitive abilities, on the other hand, management following too strict divide–and–conquer or hierarchical approach would be leading to creation of competence silos (Javidan, 1998; Rodríguez et al, 2013; Lwakatare et al, 2016).

Although small teams productivity is usually greater (Cohn, 2010), in order to extend software development capacity, larger organisations face challenges related to massively

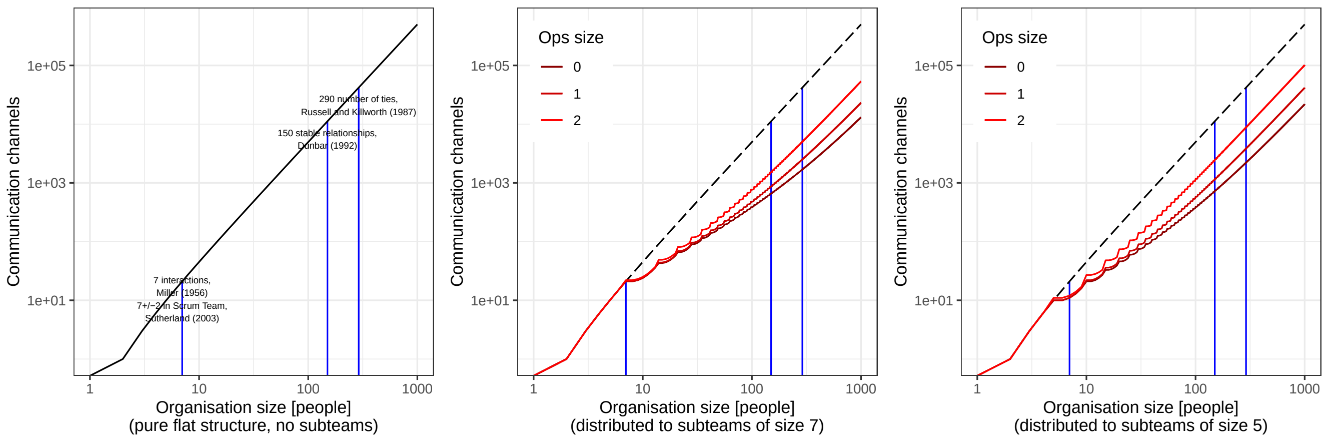


Figure 9.4: Organisation sizes, subteams and impact on scale of the communication.

scaled up teams of not collocated groups, thousands of people working on distributed architectures, focusing on different parts of EIS applications. In this context three general areas may be outlined: (1) organisation – consisting structure, people and culture dimensions, (2) process – used methodologies of needs identification, ideation, analysis, design, development and deployments standards, and (3) tools – systems assisting the earlier areas also with APM and ASM capabilities.

In order to define needs of tools (3) and ASM applications we need to review shortly the scope of organisations (1) and processes/methodologies (2). Majority of medium to large organisations can follow the model of split to strategic, operation and tactical/implementation layers (Baker and Orton, 2010). All later support earlier mentioned with lower–level of granularity. Strategy declares the organisations' vision, mission and goals. Operation layer defines campaigns, programmes and projects or work changing systems, platforms but also required integrations. Tactical level formulate work across different competence groups, maintaining know–how, quality control, delivery in cycles of functional/non–functional changes.

There are several methodologies introduced to alleviate these issues (Alqudah and Razali, 2016; Conboy and Carroll, 2019), i.e. Disciplined Agile Delivery (DAD) (Ambler and Lines, 2012), Scaling Agile at Spotify (Kniberg and Ivarsson, 2012), Large–Scale Scrum (LeSS) (Larman and Vodde, 2016), Scaled Agile Framework (SAFe) (Leffingwell, 2018), Nexus (Bittner et al, 2017). Although, all of them are built on foundations of Scrum (Cohn, 2010) they differ in the greater organisation culture element and introduce slight differences in roles and teams formation, but each of them introduce a notion of Product Ownership (PO) responsible for team backlog of work delivery aspects. Growing importance of operation roles in the teams is a subject of a debate and stabilisation of how DevOps, growth leads, AIOps or admins will be coordinating tasks within a wider community of care delivering development support solutions (Humble and Kim, 2018). One thing is certain, all those professionals will be far more effective when using the same tool–set allowing distributed collaboration of using ASM suite. Figure 9.5 depicts the feedback loops present in the larger cycle of processes defined by SDLC and PDLC.

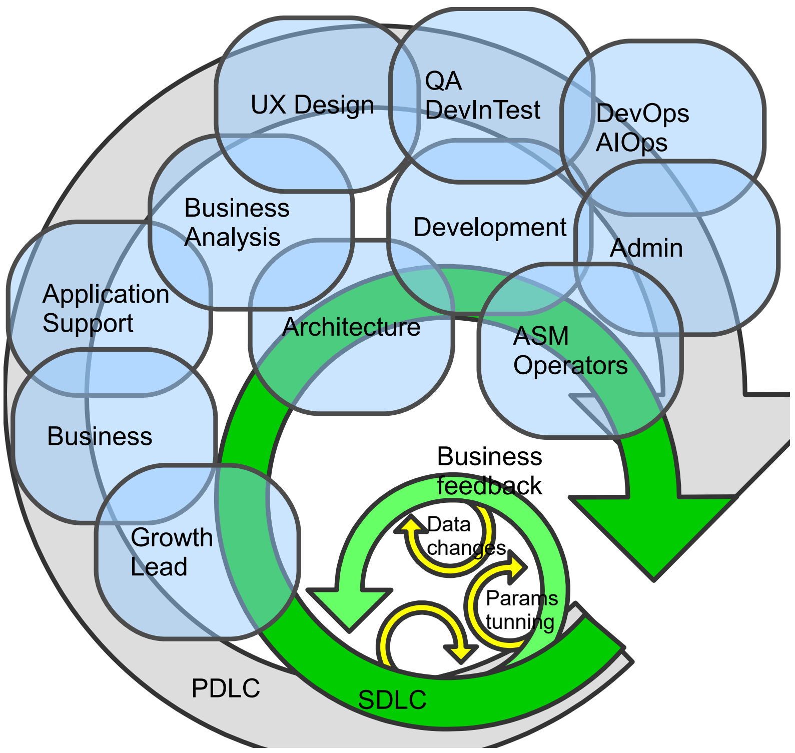


Figure 9.5: ASM suite client roles that support holistic organisation wide feedback loops.

In these conditions, supported by ASM, effective teamwork must exhibit certain characteristics that an organisation must have in order to operate in changing market conditions. These attributes are tightly interrelated, e.g.: (a) strong group cohesion of the team, with positive relationship – commonly used tools and culture of trust in high–levels of interdependence tasks are increasing collaboration to maintain high performance, level of trust, and reduced need of risk–taking, (b) effective and open communication that support arising conflicts resolution – visualisations and Instant Messaging (IM) integrations, (c) dynamic coordination of projects, experimentations, and strategic initiatives, with precisely defined targets and a clear team purpose of common goals transparent to all involved – common goals will improve team coherence, mitigate risks of dispersed knowledge and secure areas of overlapping impacts (Putnam, 1978), (d) clear accountability, measured progress, and visible commitments and traceability around taking actions, (e) eagerness to follow performance optimisation and resolving conflicts advised in internal work and by Information Technology Service Management (ITSM) systems integrated with ASM–suites (Liu et al, 2016).

9.4 Software Execution Control Constraints and Requirements

EIS and their ASMs are growing bigger and bigger, with more interactions with other systems, working under different platforms, deployment standards and communication approaches such as IoT for edge and fog computing (Aslanpour et al, 2021; Singh et al,

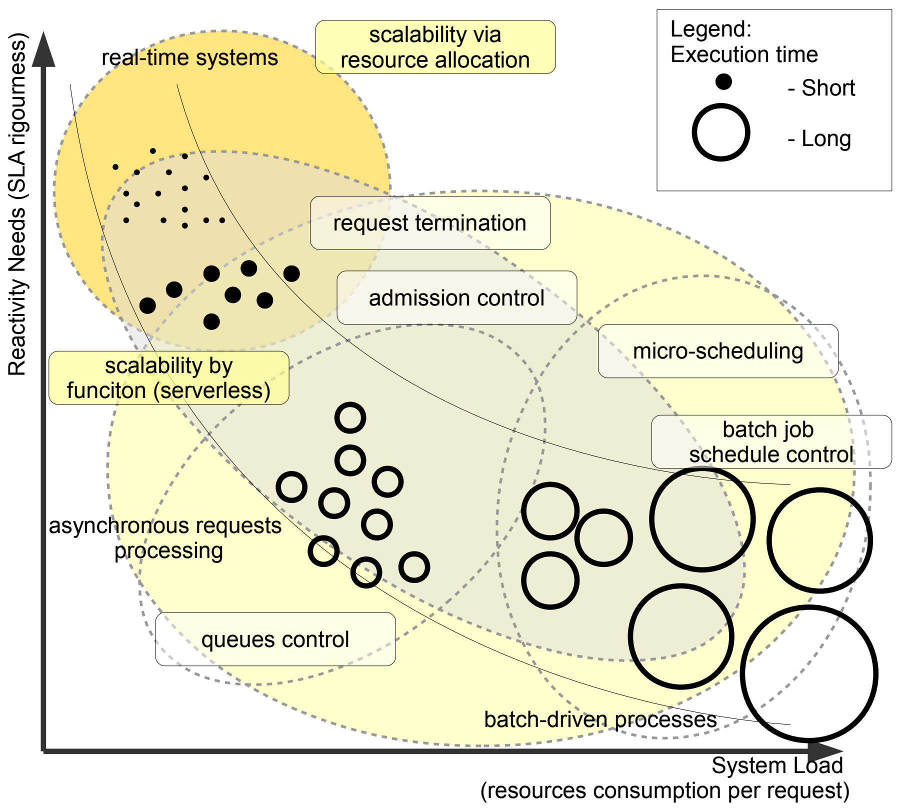

Reactivity Needs (SLA rigourness)
real-time systems
scalability via resource allocation
Legend: Execution time
- Short
- Long
request termination
admission control
micro-scheduling
scalability by funciton (serverless)
batch job schedule control
asynchronous requests processing
queues control
batch-driven processes
System Load (resources consumption per request)

Figure 9.6: Reactiveness needs and SLA harshness in load and resources utilisation, vs execution times.

2021b). In order to see the whole and build conceptual integrity (Poppendieck and Poppendieck, 2003), as a complex structure of architecture, resources, executing software, development and deployment standards ASM design need to react not only to, as discussed earlier in Section 9.2, technical and market needs, nor organisations requirements explained in Section 9.3, but also to specific constraints to be reviewed in this section.

The economics of data centres, see Section B.1.1, which offer cloud computing services, have pushed hardware and software to the limits, Section B.1.6, leaving only very little performance overhead before systems get into saturation. As a result, the risks of impacting execution and communication times grow rapidly.

In order to deliver a stable service at a time of high demand on computational power, cloud providers must implement fast and robust control mechanisms to adapt to changing operating conditions. The control can be enforced manually by operators and automatically by autonomous systems. There are normally two methods for dealing with increased load, namely increasing computational power or releasing the load. That is typically realised either by (a) allocating more machines, which must be available, waiting idle, to deal with such a situation, shifting VMs, redistributing load, or (b) throttling, terminating incoming actions that are less important to the new activity demand pattern.

Reactiveness needs of control systems to react on specific execution conditions have been compared with systems load on Figure 9.6. The comparison rely on the research conclusions of Chapters 7 and 8. More details of the practice of software execution control and its constraints has been added to the documentary appendix Section B.2. Those sections provide a discussion of constraints observed in the current market and technology conditions, moving to the specific ASM–driven requirements considerations of non–acquisitive control methods (see point (b) above).

9.5 Summary of Non–acquisitive Control Applications

All of the aforementioned needs require control actuation to tune the service level, where the non–acquisitive control is a central method group for the thesis, see Section 2.5. Let's please collect and summarise all findings listed so far.

As stated earlier on various occasions, one of the most critical NFR of EIS is scalability, which in cloud computing comes with elastic compute capabilities and appropriate design principles. Very often applied practice helping to keep stable level of service under highly volatile load is to keep more resources in reserve than it is needed. As signalled earlier in Section B.1.6 over–utilised and under–utilised servers situations are undesired. Additional source of instability and waste that can significantly limit profitability of the financial performance is a situation arising in a context of negative SLA values, see Chapters 8 and 9, where non–acquisitive control can be applied to prevent intermittent and even long term high–load securing capacity of the system. Short– or medium– term load fluctuations can be managed and mitigated much cheaper by intelligent controlled administration using non–acquisitive control, see Section B.1.6. Adding more computational power just to remove the risk of saturating one of the crucial resources may not be always necessary (Sikora and Magoulas, 2013b; Rosati and Lynn, 2020).

In order to understand non–acquisitive control against other commonly used methods review Table 9.1, which presents comparison of main control approaches for cloud computing and their applicability to service delivery models. Note, FaaS software framework constraints were highlighted in Section B.2.2 and expected ASM suites requirements in Section B.2.3.

Figure 9.6 demonstrates comparison of various control techniques applicability in a background of systems service reactivity needs defined as a function of load.

9.5.1 Main Characteristics of Termination Control and Applications

Action termination as a control approach may be eminently suitable for systems with harsh execution time SLAs, such as real–time systems, see Section 8.5.1, or systems running under conditions of hard pressure of power supply or constraints set up by the green computing paradigm, see Sections B.1.3, B.1.5 and B.1.6.

Table 9.1: Comparison of main control approaches for cloud–based enterprise information systems and their applicability to different service delivery models.

Control approach		Main cloud features support			Applicability to cloud service layer			
Name	Architecture	Load	Scalability	SLA	IaaS	PaaS	FaaS	SaaS
Server Consolidation[a]	Hypervisor[c]	Strong[d]	Strong	Weak[f]	Strong	→ (Strong, often given via IaaS)	→	→
Virtual Machines Migration[b]	Hypervisor[c]	Strong[d]	Strong[e]	Weak[f]	Strong	→ (Strong, often given via IaaS)	→	→
Containerisation[g]	OS–level processes isolation[h]	Strong[i]	Strong[j]	Weak[f]	Medium	Strong	→ (Strong, often given via PaaS)	→
Auto-scaling	IaaS Service Manger	Strong[di]	Strong	Weak[f]	Strong (Client-driven control mechanism)	—	—	—
Adaptive micro-scheduling, Asynchronous Messaging	Application tier, Message Queue Systems	Medium[if]	Medium[k]	Medium	N/A[n]	Strong (by event-driven platform framework, Application Server, MQ)	Strong	Strong
Request Termination	Application tier or Platform.	Medium[ifl]	Weak[k]	Strong[m]	N/A[n]	Medium (by admission control framework, AOP, App. Server)	Strong	Strong

[a]Server Consolidation is provided by multiplexing physical resources over virtualised environments adapting the assignment to the current system workload (Vogels, 2008). Such an optimisation can be formulated as a bin–packing problem which is NP–hard (Padala et al, 2007b).

[b]Dynamic VM placement to assign/reassign virtual resources to applications on-demand (Ahmad et al, 2015). VM migration is often used to consolidate VMs residing on multiple under-utilised servers onto a single server, so that the remaining servers can be set to an energy-saving state (Zhang et al, 2010).

[c]Amongst other most notable technologies are Xen, VMware, Libvirt.

[d]Applicable to all load scenarios, with lower efficiency for sudden spikes in load.

[e]Secures scalability requirement is fulfilled, uses proactive/reactive control to get resources before/on demand.

[f]It is difficult to build a model that provides the mapping between request-level SLA and resources.

[g]Containerisation by Linux kernel cgroups (control groups) has lightweight nature, but containers have to share a common kernel. It can be managed by technologies like Docker, DockerSwarm, Kubernetes, LXC, CoreOS.

[h]Linux kernel cgroups, Hyper-V for Windows

[i]It does not introduce the overhead of full virtualisation neither on the hypervisor nor the guest operating systems.

[j]Without hypervisor overhead the container capacity auto-scales dynamically with computing load.

[k]Supports scalability requirements indirectly; works better in a composition with other control approaches.

[l]Very effective under high load and strict SLA function definitions. The approach is dealing well with run-times that are a mixture of different load profile functionalities or parameters sensitive functions.

[m]Just-in-time style of control is able to explore and find the mapping between SLAs and resources needs.

[n]Strong support if the control is applied through weaved-in block to the application code directly.

It is a method that can secure operations of working system under sudden spike of load, immediately reflecting the higher demand to lower–priority actions cancelling, see Sections 7.6 and 8.7. This is possible and economically viable even in cases where the system is prepared to increase the computational power of the working solution to balance the higher demand, but since the elasticity has far higher inertia (in IaaS vertical scaling requires restarts, horizontal waits for new instance spin–up, whilst in PaaS and FaaS the issue is known as "cold–start problem"), the time the business suffers may be limited by termination control support. The time may be especially long in case of memory–bound systems where in–memory structures need to be reestablished, or simple uncertain due to other factors loading the cloud operator data–centre and general usage in the IaaS availability zone.

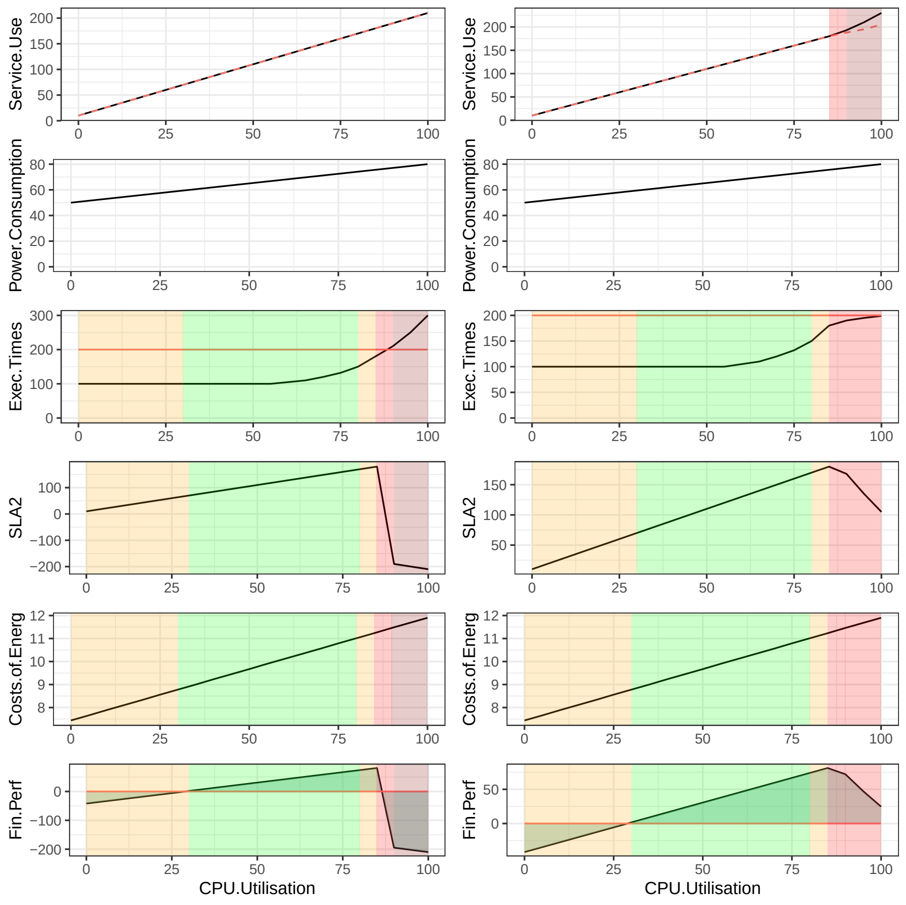


Figure 9.7: Theoretical financial performance in function of core resource utilisation (CPU) in a context of load, execution times, energy costs and SLA revenue; comparison of no termination (left) and with termination (right).

The method can significantly reduce the effects of SLA costs footprint, as presented on Figure 9.7. Termination actuator is allowing higher load to be consumed, still the same maximal acceptable per SLAs quantity of calls can be maintained.

Cloud computing, hybrid clouds, but also on–premises systems demand more and more autonomous control integrated within cloud deployable EIS. Adaptive controller can work in conditions of many overlapping SLA definitions and far more complex functionalities not only heavy CPU bound, as presented in the example for high CPU utilisation remedy on Figure 9.7, but also with mix of CPU, IO and Memory footprints.

Furthermore, not all systems can be easily decomposed to smaller services (Fritzsch et al, 2018), nor be refactored to be scalable. Termination control by applying application server level configuration or with use of AOP weaved–in actuators, see Sections 9.5.4 and 9.5.2, can be interesting option for legacy systems, often of monolithic architecture with not clear separation of independent chains (Roberts, 1999) and heavy reliance on rigid data model working with central RDBMSs impeding the redesign and decomposition works.

Table 9.1 presents a qualitative comparison of main control mechanisms for cloud–based EIS, where requests termination is at the bottom of the list. Table 9.2 demonstrates details of the termination control applicability in a wider context of various technologies, system tiers and cloud models split to perspective of the enterprise as a client of the cloud services and the cloud computing provider.

Table 9.2: Comparison of termination control actuation placement for cloud services and cloud–deployed enterprise information systems applications.

		Actuator type				Tier applicability & Agent ownership[a]									
						Enterprise, client of a service					Cloud service provider				
		Framework		System		Non cloud	Cloud model				Cloud model				
Actuating layer	Vendor, System, e.g.	AOP	API	CLI	Term. elem.[b]	On–prem.	IaaS	PaaS	FaaS	SaaS	IaaS	PaaS	FaaS	SaaS	
		9.5.2	9.5.5			9.5.9	9.5.10	9.5.11	9.5.12						Sec.[c]
Front-end		Poss.[d][e]	—	—	Met.[f]	↗[g]	↗	↗	↗	↘	—	—	—	↗	9.5.13
Facade, CAC		Yes	Yes	—	UI, WS	↗	↗	↗	↗	↘[h]	↘	⇄[i]	↗	↗	9.5.2, 9.5.14
Thread control	Java, .Net, Python, Ruby, Java Script	Yes[j]	Yes	No	Thread	↗	↗	↗	↘	—	—	↗	↗	↗	9.5.6
App. Server		Poss.	Yes	Poss.	UI, CAC, WS	↗	↗	⇄	↘	↘	—	↗	↗	↗	9.5.4
Full-stack framework		Yes	Yes	No	Met.	↗	↗	↗	↘	↘	—	↘	↗	↗	9.5.14
Application[k]		Yes	Yes	No	Met.	↗	↗	↗	⇄	—	—	↘	↗	↗	9.5.2
DAO		Yes	Yes	—	Met.	↗	↗	↗	⇄	—	—	↘	↗	↗	9.5.2
Batch frameworks		Poss.	Yes		↗	↗	↗	↗	—	—	↘	↗	↗	↗	9.5.17
RDBMS	Oracle, MSSQL, DB2, Postres, MySQL	—	Yes	Yes	Sess.[l], Oper.[m]	↗	↗	↗	⇄	—	↗	↗[n]	↗	↗	9.5.8
NoSQL	Mongo, Neo4j	—	Yes	Yes	Sess., Oper.	↗	↗	↗	⇄	—	↗	↗	↗	↗	9.5.8

[a]Who setups the control?
[b]Term. elem. = Termination element
[c]Sec. = Section number
[d]Poss. = Possible
[e]Yes, for native clients
[f]Met. = Method
[g]↗ = Strong
[h]↘ = Weak
[i]⇄ = Medium
[j]Excl. JavaScript
[k]Full–stack, incl. UI
[l]Sess. = Session
[m]Oper. = Operation
[n]DBaaS

Figure 9.8 presents a potential of integration to several layers of the enterprise system, namely, from top: front–end, facade of application services, web–services, application core code, data-access layer, Object-–relational mapping (ORM) and databases. Each of those layers offers different prospects of control, but also introduces specific challenges that require careful consideration. Every layer allows to deploy a control block consisting

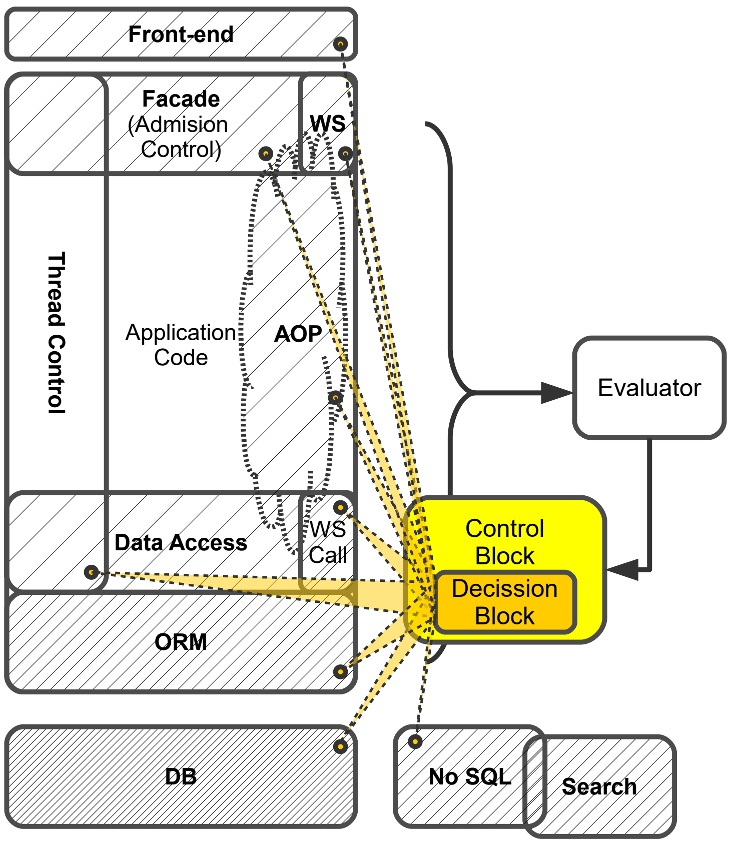


Figure 9.8: Termination actuators range of possible setup and ASM suite integration.

decision element allowing to take an actuator action in real–time based on the model outputs. The model is defined by the evaluator findings that should consider wider scope, ideally the entire stack of the system.

To this end, the presented approach is driven by SLA requirements, defined in a context of action performance characteristics. The design explained in Chapters 7 and 8 using the termination actuator pointed to the NN adaptive controller with the goal to satisfy the quality of the services provisioned and the financial performance P is flexible enough to be applied to many tiers of EIS deployed to the cloud models. Table 9.2 shows different applicability of the actuation to the various cloud models, such as IaaS, PaaS, FaaS but also SaaS. More details can be found in further sections of the chapter.

Cloud service is nothing else but an EIS as well, where SLA can be optimised in a context of globally seen financial performance P. Thus, can also take advantages of termination control, however its nature is different than from the client perspective. Services offered by providers have far different set of operating features and potential termination dynamics drive specific requirements.

In both all cases metrics signals gathered during normal operation of the system are used to build a database of time sequences that effectively contains definitions of hidden causality chains present in the system. These data are used to train control blocks on how calls should be managed and what is the most promising decision to feed the actuator blocks at each time step. ASM suite integrate the monitoring and control instrumentation to expose those characteristics to operators that are supervising the system under adaptive control.

9.5.2 Termination Actuators Weaved—in With Use of AOP

As stated earlier in Section 4.7.4 there are several means of deploying the monitoring and control instrumentation into the application. In a typical implementation actuator objects wrap a method call for its control and in the event of termination throw an error or exception. Thus additional coding or configuration is required to allow the termination to be effective in the depth of the call stack processing as explained in Chapters 7 and 8. AOP seems to be the most flexible out of others to be explained further in below sections.

There are several benefits of the weaving–in the termination controllers to EIS with use of AOP, such as: (a) easy reconfigurability offered by AOP providing libraries for dynamic interception, (b) when it is preferable to maintain loosely–coupled functional code and control mechanisms (AOP very well supports Separation of Concerns (SoC)), (c) when client code is prepared for control driven exceptions handling (there is no need to do any coding plugging in the application), (d) the method can be applied to any layer, i.e. admission control, selected methods used by UI, facade of exposed WSs or internal APIs, Data Access Objects (DAO), or even ORM (AOP configuration can be minimal if the naming conventions allow on that), (e) AOP allows deeper instrumentation which is widening the operating region of the control (termination can be executed not only just before entering a function[7]), (f) precise injection of controllers to even deep code methods of the solution in isolation to the rest of other instrumentations.

Naturally, this approach can be applied by cloud service providers managing PaaS, FaaS and SaaS, but also to client systems deployed on IaaS, PaaS and even FaaS where the control framework is weaved–into the application.

Parameters are key dimensions that drive performance footprint of an operation. Monitoring that is taking into account parameters build far stronger context and chiefly improves the potential termination control actions. AOP style of instrumentation is able to collect the entire properties and stack trace details.

As of now most of OOP languages either directly by the platform, extension, with libraries or frameworks support AOP paradigm, where cross–cutting concerns code can be weaved in with use of configurations and annotations. Java AOP methods were area of earlier experimentation explained in Chapter 7. Java was the first platform where such methods have been applied by Kiczales et al (1997), later many works extending the approach (Popovici et al, 2003; Debboub and Meslati, 2013), as of now Java enterprise mainstream takes advantage of AOP in plethora of use–cases such as transaction propagation, authorisation, dynamic configuration or monitoring, where main implementations are Spring

[7]In certain conditions delayed termination during processing can be beneficial.

(Johnson et al, 2004; Sirbi and Kulkarni, 2010), Aspect–J (Soares et al, 2002; Shao et al, 2010; Przybyłek, 2018), JBoss–AOP (Janssens et al, 2007). AOP paradigm was quite quickly adopted to .Net framework (Frei et al, 2004; Blackstock, 2004) there are currently many implementations (Wen and Zhang, 2014) such as AspectSharp, PostSharp[8], also introducing i.e. Spring for .Net[9] or Autofac as a Inversion of Control (IoC) container for .Net Core. Python AOP can be applied with use of decorators to extend a class using composition at runtime, (Matusiak, 2009) still there are libraries like aspectlib[10] and frameworks like Spring for Python that introduces full IoC and AOP capabilities (Turnquist, 2010). With Ruby 2.0 and higher language supports module mixins that provide control of adding functionality to classes (Thomas et al, 2004) still there are libraries like Aspector[11] or Aquarium[12] (Zambrano et al, 2009; Günther and Sunkle, 2009).

9.5.3 Termination Actuators in Custom and General Use and Web Frameworks

Software frameworks are ubiquitous part of the software development scenery. Every larger system contains certain list of specifics that are commonly used throughout the modules code–base, including: user account authorisation and management, logging, DAO, ORM, or admission control. Many cross–concern NFRs such as audit, custom profiling, passive monitoring, function SLA validation and usage accounting can be applied within the frameworks using natural for the used platform coding paradigm.

Every larger application in order to be maintainable needs to follow general guidelines of good practices. There are many widely adopted object–oriented programming patterns and application architecture standards such as: GOF design patterns by Gamma et al (1995), GRASP (Larman, 1998), SOLID (Martin, 2000b,a; Martin et al, 2018), Enterprise Integration Patterns (EIP) by Hohpe and Woolf (2004)[13]. The design patters introduced to better manage capabilities and pitfalls of programming languages and platforms are accepted solutions to recurring problems. Similarly to typical common services code addressing cross–cutting concerns with use of AOP (Monteiro and Fernandes, 2004; Garcia et al, 2006) implementations of such general patterns like GOF or EIP in custom areas of the system code offer a suitable ground for non–acquisitive control agents.

General use software frameworks allow rapid development. They foster good practices for software architecture modelling and provide stronger conventions of typical components definitions that significantly reduces the code complexity and helps in the maintenance. Due to large open collaborating community supporting insights and good–practices spread

[8]https://www.postsharp.net/
[9]http://www.springframework.net/
[10]https://python-aspectlib.readthedocs.io/
[11]https://github.com/gcao/aspector
[12]https://github.com/deanwampler/Aquarium
[13]https://camel.apache.org/

open source frameworks gained most of the EIS market–share. Open source can be fairly easily customised and extended by plugin additions of specific to business needs NFRs, including instrumentation of adaptive controlling actuators offering good connection to ASM suites. General use full-stack (back–end and front–end) Model–view–controller (MVC) pattern oriented frameworks in the main OOP platforms such as Java Spring (Johnson et al, 2004; Perez et al, 2019), Apache Groovy with Grails (Rocher and Brown, 2009), Ruby on Rails (Ruby et al, 2020; Hartl, 2015), Python Spring or Django (Forcier et al, 2008), .Net Core Framework (Freeman, 2016), and more specific to local features web– or micro– frameworks like Python TurboGears (Moore et al, 2008), flask (Grinberg, 2018), web2py (DiPierro, 2009), older but still in wide use Java Vaadin (Grönroos, 2011), Struts (Wojciechowski et al, 2004), Ruby Sinatra (Harris and Haase, 2011) allow on significant level of extensibility and configurability. Also modern and powerful JavaScript frameworks such as Angular (Fain and Moiseev, 2017), React.js (Staff, 2016), Vue.js (Kyriakidis and Maniatis, 2016) (those three frameworks have been compared by Koetsier (2016); Satrom (2018); Wohlgethan (2018); Saks (2019)), Next.js (Konshin, 2018), Ember.js, Knockout, Backbone (Delcev and Draskovic, 2018), Express.js (Hahn, 2016) all offer great flexibility of being extended with client–side or supported by facade–layer instrumentation, see Front–end (FE) in Section 9.5.13 and CAC in Section 9.5.5.

9.5.4 Termination Actuators in Web– and Application– Servers

Web servers are true pillars of EIS, where UI of web browsers and client applications communicate over HTTP to respond to users' requests. Web servers allow configuration level injected instrumentation on Application servers and web containers levels. Depending on the technology and platform they can be called modules, filters, or interjections. It is also possible to apply custom extensions to the servers code–base since most of the modern servers are open–source. This class of software offers support for security enforcement, but also clustering, fail–over and load–balancing, so vital elements of alternative load orchestration to non–acquisitive control methods. This tier delivers an entry element in scalable EIS architecture to authorisation, CAC, allowing requests–responses coordination and present in PaaS, FaaS and SaaS models, see Figure 9.8.

In order to position more context of the implementation let's review most notable web servers. According to Netcraft report[14] (Netcraft, 2021) the market share of web servers of all sites is following: Nginx 35%, Apache HTTP Server 25%, Google Web Server (GWS) 4%, Microsoft IIS 4%. Of course the following short overview should be considered only as entry–level attempt providing a few very limited examples about some approaches that may offer implementation under a web server or application server layer.

Nginx (Nedelcu, 2013) architecture provides "dynamic modules"[15] allowing components to

[14]https://news.netcraft.com/archives/2021/10/15/october-2021-web-server-survey.html
[15]ttps://docs.nginx.com/nginx/admin-guide/dynamic-modules/dynamic-modules

be plugged–into the web server processing flow or a load balancer that nginx often function plays. It is possible to build a custom component following the interface guidelines specified[16].

Apache HTTP server (ASF, 2015) remains the most commonly deployed web server for over two decades. Apache web server also supports the first– and third–party modules available e.g. `mod_log_config` that provides flexible logging of client requests in a customised format, `mod_qos` by Pascal Buchbinder that implements control mechanisms to provide different priority to specyfic requests and allowing controlled access to the web server in order to avoid resource over–subscription, also `mod_throttle`, `mod_ratelimit` introduce preprocessing to improve security and tackle performance NFRs. The modules rely on request handler implementation in C code following the guidelines by Apache[17].

Microsoft IIS offers a programmatic interface for modules implementations[18] that can be either a native module Win32 DLL or a managed module of .NET 2.0 type contained within an assembly. All IIS modules can be removed or replaced by custom implementations developed using the IIS C++ or .NET APIs. `RequestFilteringModule`, `HttpLoggingModule` or `TracingModule` are examples of modules that can provoke giving a ground for control implementations.

Number one in Java world of web containers is Apache Tomcat server that offers filters interfaces[19] according to catalina implementations `org.apache.catalina.filters.FilterBase` and general interface `javax.servlet.Filter` and `javax.servlet.GenericFilter`. Filter implementation is a very convenient area for interception calls and termination actuation. Example of `ExpiresFilter` as a Java Servlet API port of Apache `mod_expires` that controls the setting of the Expires HTTP header and the `max-age`, see RFC–2616 by Fielding et al (1999), would be a good illustration of the instrumentation capabilities in area of changing request processing. Both Eclipse Jetty and JBoss WildFly servers allow the same level of integration due to compliance with the same standards.

9.5.5 Actions Termination in Admission Control and API Access Management

The facade–layer of exposed EIS interfaces that typically consists implementations of many NFR features allowing to authorise, audit, account check and do full performance check of Turnaround Time (TaT) is a very suitable area for termination control. It is also a convenient field for additional instrumentation of termination actuator that could work with addition to admission control (Poggi et al, 2009; Muppala et al, 2014) features, see

[16]https://www.nginx.com/blog/compiling-dynamic-modules-nginx-plus
[17]https://httpd.apache.org/docs/2.4/developer/modguide.html
[18]https://docs.microsoft.com/en-us/iis/get-started/introduction-to-iis/iis-modules-overview
[19]https://tomcat.apache.org/tomcat-9.0-doc/config/filter.html

Chapters 7 and 8.

Termination actuators can be offered as part of FaaS model by service provider, but certainly can be a component of a general framework or custom system deployed on any cloud model or on premises. SaaS and FaaS systems providers can take special advantages of the method for internal optimisations, allowing the service provider to secure the elasticity of the services with tight relation to clients' account details, like services and data usage, billing, freemium models, SLA etc.

Tools which are part of FaaS and SaaS suites of Identity ad API Access Management such as Okta[20] providing Identity–As–A–Service (IDaaS) for enterprise, or cloud computing admission control management like Google [21] or access management offered by AWS[22] can be extended by control mechanisms providing another layer of overload prevention.

Furthermore, as far as PaaS model is concerned, as mentioned in Section 4.7.4, container cluster technologies like Kubernetes or Docker–swarm can work with admission controller which can refiew, validate and reject if needed exposed load balancer services of the cluster[23].

Lastly, as the action level framework exposes the details of the financial and computational performance of an application, it could become part of ASM suite as an augmented operation–toolbox to support human administrators coordinating autonomous control agents that operate in high–dimensional enterprise environments. This audit and visualisation, built on the top of the data, can offer insights into the dynamics of the system and of the algorithms employed. This is a specialised domain where human administrators require additional support mechanisms as they frequently experience problems when analysing high–frequency sampled time–series data from weeks and months to arrive at a judgement on a particular control decision.

9.5.6 Termination Actuators With Use of Platform Thread and Event–loop Control

There are situations where the code stack of an application needs to be minimised or larger software framework is not intended to be used and AOP may be considered as too intrusive. In such cases it is still possible to apply the control execution with termination actuators by using lower–level platform thread or function execution mechanisms. Thus, the actuation model can be used (a) on–premises and in cloud deployments under IaaS, Container–as–a–Service (CaaS), PaaS models by EIS applied by enterprises, and (b) introduced by cloud computing vendors in PaaS, FaaS and SaaS models.

[20] https://www.okta.com
[21] https://cloud.google.com/service-infrastructure/docs/admission-control
[22] https://aws.amazon.com/iam/
[23] https://github.com/giantswarm/aws-admission-controller

Depending on specifics of the platform there are different systems of thread control. In Java there is no supported JVM mechanism allowing to gracefully kill a thread. This is since `stopThread` or `ThreadDeathException` to a specific JVM thread are deprecated in Java. However, it a fairly simple technique to apply a standard java concurrency practice to create a `Thread` class object with custom thread–stop methods[24]. The same problem exists in Python and .Net (including .Net Core) where code cannot kill or abort a `Thread` directly, and the same as above highlighted solution can be used.

Of course whenever multi–threading code is used a special attention needs to be paid to resolve situations of termination of one of the threads as it severely may impact the computation process.

Out of Memory Error thrown under VM of Java and .Net platforms is an interesting form of a termination control which however does not provide enough stability to be classified in full as part of the non–acquisitive control methods.

In JavaScript platforms space the situation is particularly different. In browser engines and Node.js on server–side (Tilkov and Vinoski, 2010; Cantelon et al, 2014) where the event loop[25] is introduced, to secure requirements of non-blocking I/O access, asynchronous processing allowing concurrency under single–threaded execution model, the EIS code must rely (Prusty, 2015) on callbacks (Zakas, 2016) and more recently on promises and async/await (Madsen et al, 2017), see ECMAScript6 2015 (ECMAScript, 2015). The even loop offers a high performance and scalable approach for highly utilised web solutions (Lei et al, 2014). The termination controller could be introduced in the event loop processing[26] without a need of changing the order of the specific operations phases in a sequence of `timers`, `pending callbacks`, `idle`, `prepare`, `poll`, `check`, `close callbacks`. Of course the actuator code will not terminate the thread it will execute the callback routine accordingly to the control action decision.

Since PaaS vendors can control the open–source of servers where the client code is deployed and other lower details of platform related tools, they would be in a position to extend those by aforementioned mechanics injected with control actuators as per the delegator or proxy patterns, with no change to the style of deployment and use of the platform by client. The same situation apply to FaaS model.

The exposed actuator APIs could be used by cloud provider (with higher priority) or by client via appropriate ASM consoles. Also, under reactive programming paradigm where the focus is on developing asynchronous and non–blocking components, with data streams and the propagation of change, proposed approach to handle near–real–time systems that are responsive, resilient, elastic and message driven[27] (Bonér et al, 2014), following reactor

[24]Such as sending a “poison pill” signal or value to stop processing.
[25]`https://nodejs.dev/learn/the-nodejs-event-loop`
[26]`https://github.com/nodejs/nodejs.org/blob/main/locale/en/docs/guides/event-loop-timers-and-nexttick.md`
[27]Responsive: systems should respond in a timely manner. Resilient: systems should stay responsive when some components fail. Elastic: systems should stay responsive under high load. Message Driven:

and observer patterns (Bainomugisha et al, 2013; Salvaneschi et al, 2015; Allen, 2017; Debski et al, 2017).

9.5.7 Termination Control For Event–based Systems: Message Broker Queues and Event Streaming Platforms

Event processing is an approach that allows loosely–coupled modules to process communication asynchronously often with use of broker middleware. The asynchronous style of processing lets systems to lower cumulative latency by avoiding over–utilisation and higher the throughput of processing employing queuing and topic style redistribution. Event–based systems are key components of EIS architectures, using Message Queue (MQ) broker, Enterprise Service Bus (ESB) and utilising Enterprise Integration Patterns (EIP) (Hohpe and Woolf, 2004; Tarkoma, 2012), compliant with such protocols as Java Message Service (JMS), Advanced Message Queuing Protocol (AMQP) or more lightweight suitable for distributed, smaller devices and telemetry MQ Telemetry Transport (MQTT). This section will review termination potential on the ground of capabilities amongst the aforementioned approaches.

Message queue paradigm follows the publish–subscribe pattern (Eugster et al, 2003), which can be following quite different queue or topic methods. Queue driven designs allow to disseminate the message to only one consumer that is available to process the work. Whilst topic oriented setups promote the message to be accessible to get delivered to all subscribers.

Termination actuation may be allocated in the form of an additional processor embedded in the producer–subscriber chain. The actuator can be implemented as a Selective Consumer, Message Router or Dynamic Router patters of EIP. Selected requests types that are linked with a topic for broadcasting purposes or a queue are filtered and processed using controller rules or models. Whenever there are indications leading to conclusions of a control to prevent overloading or impacting defined SLAs the message can be cancelled, delayed or prioritised. Another solution would be to wrap the message in a special frame or associate with a flag and leave the decisions to terminate of the agent deployed in the runtime of the subscriber.

Throttling on delivery time, Time–to–Live (TTL) quotas or other forms of delaying or distributing the subscribed messages are typical additional forms of control in cloud brokers (Wilder, 2012). Thus the control style is a natural in the cloud providers implementations to prevent abuse, defend from floods of messages and to optimise the overall network, and key resources efficiency.

Most often used message–oriented middleware and software that routes information between applications, devices and user interfaces are open source. The software appliances

systems should use async message–passing between components to ensure loose coupling.

are deployed on–premises or used by cloud providers. Since they source code is available they can be extended by instrumentation allowing embedded control mechanisms. Let's name a few most important MQ brokers: Active MQ[28], RabbitMQ[29], JBoss Enterprise Application Platform (EAP)[30], and ESB suites: Mule ESB[31], WSO2[32], JBoss Fuse ESB[33], and Apache Camel[34] introducing EIP implementations. I omit here all proprietary implementations. As far as FaaS providers offer is concerned the serverless event–bus features support widely accepted protocols such as JMS or Advanced Message Queuing Protocol (AMQP) (Adzic and Chatley, 2017), where amongst many most important are Google Pub/Sub[35] AWS EventBridge[36] MS Azure Service Bus[37] , Oracle Messaging Cloud Service (MCS), IBM Cloud MQ[38]. Amongst many SaaS providers most notable delivering queue and messaging capabilities are: Google Firebase Cloud Messaging (FCM)[39], Amazon Simple Queue Service (SQS)[40], Apple Push Notification (APN) service[41], CloudAMQP[42] offering RabbitMQ as a Service, or Solace[43].

Another separate group of asynchronous messaging define stream–processing platforms that also bring software–bus and allow distributed event streaming (Cherniack et al, 2003). These software platforms can implement analogous control to discussed above but due to more continuous and fine–grained level the termination actuation will follow more stream–oriented approach (Hirzel et al, 2014). Similarly to above the most important implementations are open–source, such as: Apache Kafka[44] (Narkhede et al, 2017), Akka[45], Apache Storm[46] Apache Flink[47] allowing statefull computations over streams of data and finally Apache Spark[48] All above are also offered by PaaS and SaaS models under services marketplace or directly by such vendors as AWS, GAE or MS Azure.

[28]https://activemq.apache.org/, https://github.com/apache/activemq
[29]https://www.rabbitmq.com/, https://github.com/rabbitmq/rabbitmq-server
[30]https://www.redhat.com/en/technologies/jboss-middleware/application-platform, https://github.com/jboss-eap
[31]https://www.mulesoft.com/, https://github.com/mulesoft/mule
[32]https://wso2.com/, https://github.com/wso2
[33]https://jbossesb.jboss.org/, https://github.com/jboss-fuse/fuse
[34]https://camel.apache.org/, https://github.com/apache/camel
[35]https://cloud.google.com/pubsub
[36]https://aws.amazon.com/eventbridge
[37]https://azure.microsoft.com/en-us/services/service-bus
[38]https://www.ibm.com/cloud/mq
[39]https://firebase.google.com/docs/cloud-messaging/
[40]https://aws.amazon.com/sqs/
[41]https://developer.apple.com/notifications/
[42]https://www.cloudamqp.com/
[43]https://solace.com/
[44]https://kafka.apache.org/, https://github.com/apache/kafka
[45]https://akka.io/, https://github.com/akka/akka
[46]https://storm.apache.org, https://github.com/apache/storm
[47]https://flink.apache.org, https://github.com/apache/flink
[48]https://spark.apache.org, https://github.com/apache/spark

9.5.8 Session Termination Control For Database Systems

Heavy queries and data modification requests may lead to colossal resources usage and long execution times. Both of the situations can be substantially secured by termination control that is available by the design of DB systems.

The query structure, next to the type of requests, such as data query, data modification or index building, are key dimensions helping to differentiate the load to the system. Following the nomenclature of the thesis they are actions a. Most of the DB systems, both RDBMS and NoSQL hold statistics, known as execution plans or query plans that support the data engine uses to optimise processing given the available indexes. Many DB systems give interfaces that ASM suite could use to extract the statistics vitally improving ability to coordinate the execution regimes. Moreover, parameters are dimensions that require special attention in the monitoring and control considerations. This is mainly due to the DB and ORM caching, thus, the sequence of use those parameters is important too. Also, the use–case context of the SQL executed from DAO layer may bring decisive information about the control. Therefore, the complete ASM solution contains monitoring on as many tiers of the application as possible.

All of the most commonly used RDBMS such as Oracle[49], Microsoft SQL Server[50], MySQL[51], PostreSQL[52] allow management of established sessions, including termination of selected heavy processing SQL queries, Data Manipulation Language (DML) and procedural calls. Typically every DB role with superuser rights is permitted to terminate database connections; it is given to Database Administrations (DBAs) in order to control the client apps.

RDBMSs are well established in PaaS for many years (Hacigumus et al, 2002), key cloud computing providers offer DBs under the model with extension wrapping and abstracting DBA consoles with appropriate management access APIs, vendors such as Amazon[53], Microsoft Azure[54] allow session or statement execution control. As of now, Google Cloud

[49]Oracle offers several ways of killing a long executing session, i.e. invoke `SQL*Plus`, or use `execute immediate` to query `V$SESSION` supplying the username (subsystem connection) for the session to terminate, receive unique identifier of the session and serial number and execute the `ALTER SYSTEM KILL SESSION '<sid , serial#>'` command to terminate the session. Another method to mention is the `ALTER SYSTEM CANCEL SQL` command that was introduced in Oracle 18c to cancel a SQL statement in a session, providing an alternative to killing a bad performing session.

[50]In MS–SQL Server run a command to show full process list; to get the process id with status and query itself which creates high load to the DB, select the process id and run a command `KILL <pid>;` to terminate that process.

[51]For MySQL the query `SELECT concat('KILL ',id,';') FROM information_schema.processlist WHERE user='user'` simply prints all processes with the command to terminate.

[52]PostreSQL allow to kill a connection by using `pg_terminate_backend()` function available to a superuser regardless of OS in use. `SELECT pg_terminate_backend(pid) FROM pg_stat_activity WHERE pid <> pg_backend_pid() /* don't kill this connection */ AND datname = 'db_name' /* don't kill the connections to other databases */`. There is also a way to cancel a running query. A very useful technique since it is not always desired to abruptly terminate an existing database connection: `SELECT pg_cancel_backend(procpid) FROM pg_stat_activity WHERE usename = 'usr_name'`.

[53]RDS method: `rdsadmin.rdsadmin_util.kill`

[54]Azure SQL Database provides a command to kill a specific session on a server and roll back all

SQL[55] does not offer methods for terminating sessions, still, this area is a subject of dynamic change.

As far as NoSQL field is concerned the situation is less clear. This is partially due to the nature of NoSQL DBs that typically offer higher performance get and persist functions (Čerešňák and Kvet, 2019; Martins et al, 2019). Most of the DBs introduce configurable timeout mechanisms, but only few give full control over the session and job management like in case of document–oriented MongoDB[56] or Graph DB Neo4j[57]. Often, due to architecture of high–resiliency features, with eventually consistent paradigm and distributed architecture, clients nodes have special functions coordinating access to clusters, so despite the fact there is no native support to kill large queries it is advised to kill the entire process on OS level if there are signs of overload, i.e. Cassandra[58].

The list of modern DB systems is very long and exploring the whole area is far exceeding the scope of the research. Still, as it has been shown above, there is a very good potential of support on this tier of EIS. Nevertheless, in a case of any limitation or long performing termination requests, with TaT under a specific DB system, the application can have a built–in mechanism terminating the call on the upper layer, see Figure 9.8, i.e. from a component that interacts with DB directly in DAO or ORM layers.

9.5.9 Termination Control For Systems Deployed On–premises

Software deployed on–premises is still commonly used across many organisations. Although the modern trends create significant pressure to install more and more software in distributed way using cloud computing there are several reasons stopping corporations from migrations. The most often seen scenarios are limited value of the software migration, risks of the platform shift and substantial costs of migration, but also limitation of the legacy systems redesign, or enterprise location policies forcing to use established within the organisations' internal system and hardware infrastructure. Due to the limitation organisation often concentrate the migration focus only on components that require the most of the advantages of cloud computing, thus hybrid cloud model is applied.

On–premises deployments is the area of use where enterprises are able to exploit full spectrum of advantages of the non–acquisitive control model in general. In particular,

transactions that are associated with it. To find session id (spid) to end select session listing from all sessions on the server with the query: `exec sp_who`, later execute command `kill <spid>`.

[55]Google Cloud SQL offers MySQL, MS–SQL Server and PostgreSQL, where i.e. in case of the last one suggest to use role of `pg_signal_backend` to cancel or terminate any user session.

[56]MongoDB by default use expiration timeout of 30 minutes, the `killAllSessions` command terminates all sessions for the specified users, and `killOp(opid)` cancells an operation as specified by the operation id.

[57]Neo4j driver offers API for session management and programmatic way to terminate (abort) a long–running transactions from another thread by calling `terminate()` method of `Transaction` object that is established by `GraphDatabaseService`.

[58]In Cassandra there is no API of Command Line Interface (CLI) way to stop individual long running CQL query. To overcome the problem it is advised to locate that query's coordinator node, and restart it.

the termination control can be applied to every tier and platform model as explained in Table 9.2. Also, the fact that on–premises setups often relatively quickly (especially under higher load condition) face limited computational power capacity gives additional potential of using non–acquisitive control.

9.5.10 Termination Control For Systems Deployed in IaaS

As soon as the organisation is able to perform "lift–and–shift" type of migration to the cloud computing, the IaaS model still allows to keep in use most of the established configurations and instrumentation points of ASM. The natural improvements can be made in area of virtualisation, containerisation, distributed deployment and CD, but in most cases those are not impacting the control instrumentation. Those improvements require more monitoring, both active and passive, which should only improve the controllers accuracy and operators ability to see–the–whole in the long term.

As mentioned in Section 9.5.1 termination control helps in scaling capabilities of the systems. Therefore the action termination method can be an especially interesting option for cloud service providers that build their upper models PaaS, FaaS and SaaS on top of IaaS, see Figure 9.9, whenever the service provider would like to mitigate the risk of the longer VM spin–off in situations of a sudden load increase. In such cases, they need to pay additional attention to prevent overload when auto–scaling is too costly (reached certain threshold), and responses times (execution times) are strictly defined in SLAs.

9.5.11 Termination Control For Systems Deployed in PaaS

After an organisation has built the capability to move to IaaS and see more and more options to improve the scalability and reduce Total Cost of Ownership (TCO) of hosting servers it is time to revise the offer of PaaS model services.

The systems decomposition and migration can be performed in steps, service by service. Again, most of the implementations done by the organisation can remain, the only areas that change the deployment style would require reconfiguration. Since the migration is performed steadily, the wider team using ASM suite will be able to see and coordinate the change in the background of observed changes to the runtime characteristics. The joint model of human–machine control will be able to adjust the new elements of knowledge on the basis of collected evidence.

After migration the termination actuator can be useful in PaaS as an embedded component (library in the tech stack of an app). Also, many PaaS CLIs could take advantage and adjust the commands integrating the platform functions with ASM.

For cloud service providers PaaS is the first tier that can bring significant improvements, as stated in the Sections above. Furthermore, it is expected to see the scale and quantity of this control use–cases growing exponentially with number of the service providers building function and systems services on the top of the platform model, see Figure 9.9.

9.5.12 Termination Control For FaaS and Serverless

Serverless computing creates completely new opportunities for termination control. As discussed earlier in a context of services cost optimisation under harsh SLA in Chapter 8 when action termination can substantially improve the financial performance P. Serverless is not suited for certain computing workloads, such as long, computation intensive, memory hungry, or any of wider High Performance Computing (HPC) requirements because of the resource limits imposed by cloud providers. It is a very convenient model, however, in use cases of integrations, especially event–driven, often changing functionalities (frequent redeploys), wherever a micro–billing and high scalability are important properties.

In this context, the termination control model offers several valuable features apart of the typical execution time control: (1) functions over–usage prevention – especially in situations where functions are split to use power of parallelisation and the billing model accounts per action, (2) malicious usage or services misusage at clients systems, (3) SLA–driven prioritisation of calls under a given client account, (4) freemium model,

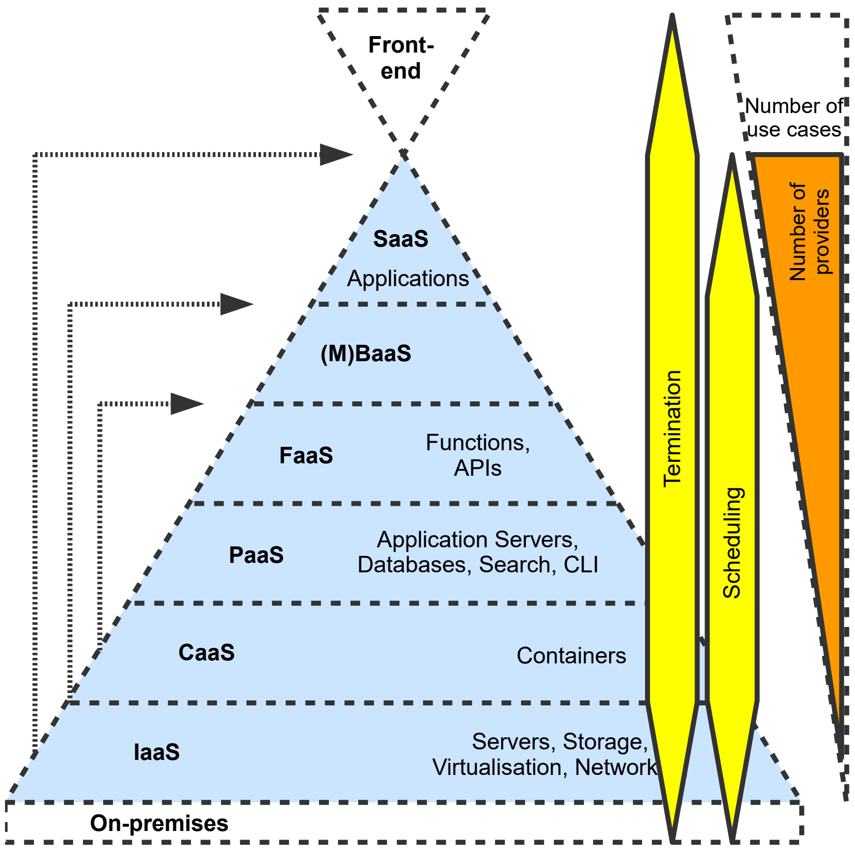


Figure 9.9: Triangle of cloud model stack and non–acquisitive actuations applications.

Section 9.5.5, and fast deployments supporting growth initiatives (Ellis and Brown, 2017), pretotyping (Savoia, 2011) and experimentation (Fabijan et al, 2018; Troisi et al, 2020), Section B.2.5.

9.5.13 Termination Control on Client–side (Front–end)

UI as the highest in the call processing layer of the stack can give fast and seamless termination control integration, see Figure 9.8 and 9.9. Modern web applications contain more and more complex browser side code base. It would be expected to see specialised frameworks that would support developer in designing the web app to adapt UI level behaviour when some of the requests to services (actions) are cancelled. Such an adaptation would be performed in a fully controlled way according to configuration or code. In result, in a properly constructed UI under well designed UX a user would not notice a massive difference in conditions of higher load.

Java Script and Node.js lead FaaS adoption (Adzic and Chatley, 2017). Java Script frameworks using that approach could find a very important use–case niche. Of course the code could be executing on server–side to if the implementation uses Node.js, e.g. deployed in FaaS. Such commonly used Java Script libraries which are de–facto standard of web UI development like Vue.js, React,js, Angular, can be wrapped or extended to allow the ASM data collection and control instrumentation directly on front–end side.

Flutter[59] (Windmill, 2020; Faust, 2020) and strictly mobile first cross–platform approaches, such as Xamarin[60] (Hermes, 2015) or Appcelerator Titanium Software Development Kit (SDK)[61], converting features to native platform code due to their strengths can be a very powerful area of such a control instrumentation.

9.5.14 Other Use–cases of Actions Termination

Termination control can be an additional measure of prevention against Distributed Denial–of–Service (DDoS) attacks (Nguyen and Choi, 2010). Since it is a method that secures momentum of working system under sudden spike of load it can be part of destination–based mechanisms prevention (Yan et al, 2015). Especially when the attack is coordinated in a gradual moderate rate load increase, slow HTTP request attack (Suroto, 2017), or executed by large group of hostile hosts the typical systems prevention mechanisms and intrusion detection may classify the additional load as a normal business activity. This could be applied as part of CAC or as a framework–level addition to cloud–provider–based Web Application Firewalls (WAF) offering additional data scanning of such attacks.

[59]https://flutter.dev
[60]https://dotnet.microsoft.com/apps/xamarin
[61]https://titaniumsdk.com/

Energy reduction goals can be another area of interest, see Section B.1.5. Under certain circumstances when scaling up can be limited by harsh green computing targets the termination control can offer significant level of preservation of critical system features.

9.5.15 Adaptive Jobs Scheduling Applications

Long executing, often sequential processes, that are heavily dependant on other computation or input data are critical in most of larger organisations, see Section 8.8. Batch applications are in large part of EIS because many common business processes are amenable to data–intensive operations and batch processing. The large bulk RDBMS queries, transaction processing, functional calculations and Extract–Transform–Load (ETL) routines feeding other structures of DBs, indexing of full–text search engines, offline model training processes, in–memory cache refreshes, long performing CRM, Data–warehouse (DW) or Data–lake (DL) data–feeds are inherently batch processes.

Low–interactive computation is, and will remain, very important for EIS. Batch processes allow the interaction to be limited to bare minimum and the setup effort is shift for the entry configurable step, so it is still a fairly human–oriented area. As stated on several occasions, see Sections 8.8, the efficiency of batch jobs is a key element of performance monitoring and control done in collaboration of many teams that could be using common toolset of ASM suite.

Adaptive jobs scheduling (intelligent scheduling) uses the data of required times of execution, preferences and is able to assist human operators with taking decisions of changes in the schedule to optimise the system as a whole under regime of SLA and financial performance P objectives. As explained in Chapter 8 the adaptive scheduling control is robust enough for corporate data processing, but is still operated by human. Therefore, apart from the normal pervasive (active and passive) monitoring of batch subsystem dynamics ASM suite must be integrated with access to time schedules meta–data or directly with interfaces of batch schedulers.

There are many successful batch frameworks offering tight integration with EIS including comprehensive support of managing jobs via APIs, start/stop/restart, processing statistics, cluster and scalability features, error handling, dependency and re–execution engine, and robust persistence of batched meta–data. The most prominent Java implementations are Quartz[62] (Cavaness, 2006) and Spring Batch[63] (Minella, 2011) that also integrates with Quartz (Harrop and Machacek, 2005) and has .Net framework[64] implementation. Python standard library offers scheduling event class, `shed.py`[65], there are also specialised libraries i.e. Prefect[66] for sophisticated workflow engine for data–flow automation, Ad-

[62] http://www.quartz-scheduler.org/
[63] https://spring.io/projects/spring-batch
[64] https://www.quartz-scheduler.net/
[65] https://docs.python.org/3/library/sched.html
[66] https://www.prefect.io/, https://github.com/PrefectHQ/prefect

vanced Python Scheduler[67], and Luigi[68] package allowing to build complex pipelines of batch jobs in workflows with dependency resolution, handling failures, CLI integration, and directed to run Python mapreduce jobs in Hadoop, as well as Hive, and Pig. Another Python tool that must be mentioned is Apache Airflow[69] that is a platform to programmatically create, schedule and monitor workflows, that is more and more often available directly under PaaS model.

From the EIS implementer perspective as a client of cloud services the schedulers can be applicable directly to on–premises, IaaS and PaaS. Whilst, from cloud provider side, the main area of focus is PaaS, FaaS and SaaS.

At IaaS level all can use OS powered scheduling mechanisms, for Linux and Unix crontab[70] and for Windows Task Scheduler[71] offering API management. Although this tier of scheduling software allows limited instrumentation the control integration of ASM is possible by configuration and scripting, and monitoring can be delivered by applying agents into the objects that are executed.

Another area of interest would be an application of Batch–Scheduler–as–a–Service (BSaaS). Several cloud computing vendors offer also PaaS and FaaS features like Azure Batch[72], AWS Batch[73], and Google Cloud Scheduler[74] as a fully managed cron job services accessible by CLI and API, so the integration with ASM is fairly easy.

9.5.16 Micro–scheduling Control Applications

The action termination may be the last resort that a service operator or an enterprise want to introduce. Not to mention the viewpoint of the client, of the API or UI service, that has to prepare to adjust and reflect the consequences of the request cancellation. Although, the client may be reassured or even attracted by the provider offering high penalty SLAs that will be accounted to the customer profile in a case of termination, the market response may be negative.

In these conditions the scheduling may be seen as an interesting alternative. The same principles driving scheduling control of large batch jobs can be applied to shorter calls.

This type of non–acquisitive control method is better reflecting financial and operational needs than termination in a case when SLA penalties for cancelling a call are higher than

[67]https://apscheduler.readthedocs.io/en/stable/, https://github.com/agronholm/apscheduler/
[68]https://luigi.readthedocs.io/, https://github.com/spotify/luigi
[69]https://airflow.apache.org/
[70]https://man7.org/linux/man-pages/man5/crontab.5.html
[71]https://docs.microsoft.com/en-us/windows/win32/taskschd/task-scheduler-start-page
[72]https://azure.microsoft.com/en-us/services/batch/, https://docs.microsoft.com/en-us/azure/batch/batch-technical-overview/
[73]https://aws.amazon.com/batch/, https://docs.aws.amazon.com/batch/
[74]https://cloud.google.com/scheduler/

proportional positive impact coming from release of the resources utilisation.

Event–based systems, due to their asynchronous nature and relaxed TaT SLAs is a natural niche and can be a very good ground for micro–scheduling control that delay the main function execution for time where the required resources are more available.

Promising perspective create an association of micro–scheduling and termination control. Both methods require the same level of actuation instrumentation, so can be deployed together. Micro–scheduling method can be applied earlier within the chain of control, before the main call execution or in the middle of a medium execution time routines depending on the level and application tiers the instrumentation was introduced. In a case when the outlook on the execution is not propitious, the termination control can be executed. Both methods can be supporting each other in a mission to reduce the current load and distribute the executions to maximise the affected resources for best SLA footprint and optimising financial performance, P.

9.5.17 Joint Termination and Scheduling Control

Termination control can be applied together with larger batch jobs rescheduling. This approach joins strengths of both key non–acquisitive control approaches. Whenever there is a strong need, and SLA permit on that, batch job can be terminated. Even in the middle of the execution, which can be still beneficial to the wider picture system, see Section 8.6.

Batch frameworks APIs generally contain functions to stop the execution job. The termination can be also done within the application code depending on level of applied instrumentation. As discussed in Section 9.5.15, since many scheduling frameworks are open source and running platforms allow deeper code instrumentation, the job classes can be additionally instrumented with termination actuators.

9.6 Discussion of Identified Limitations of Non–acquisitive Control

After the review of potential applications and implementations it is time to discuss further the applicability seen from constraints and limitations perspectives. Let's begin from commercial aspects moving to those more technical issues.

Again, the biggest, one could state, critique of the non–acquisitive control approach lies within its core philosophy, so in the nature of cancelling the requests (termination) or change of its execution (rescheduling). Also relatively narrow operating region of control, when a critical resource is close to challenging conditions of saturation, makes this approach just an auxiliary method to other commonly used control techniques, cf. Figures B.6 and 9.7. Without a strong motivation of resources optimisation or SLA conservation, both contributing on the global financial performance P, applicability of the methods may be seen in many cases as economically non–viable option, simply too expensive for the gains to be achieved.

Although many limitations, challenges and adoption barriers were listed already in previous Sections, it is important to collect those and place in a background of the ASM suite design.

9.6.1 Application–level Control Intrusiveness

Control methods are intrusive by their nature. They require computational power to process the requests. Although the design of control approaches presented and tested in Chapters 7 and 8 was taking into account the issue it is important to highlighting that additional processing is required at times where likely the CPU time is at high demand. Still, in cases of other resources shortage additional CPU oriented processing may be perfectly feasible.

Every weaved in layer of monitoring or actuation agent to the thread stack requires larger thread memory footprint. In heavy recursion method use that can cause further slowness in processing or even unexpected `StackOverflow` errors. The deeper instrumentation the more promising possible control benefits may be seen, however, this needs to be carefully investigated and balanced with close supervision of EIS administrators and ASM operators.

9.6.2 Organisation Challenges Slowing–down Adoption

Non–acquisitive control methods must be always clearly stated in the contract of the service under Terms of Service (ToS) or Terms and Conditions (TnC). Both termination and rescheduling can change sequence of execution at the client–side. The UI or client application functionality need to be carefully prepared, and in case of workflow constraints, specific adjustments need to be applied.

The long learning process and human coordination is an additional level of complexity for the ASM introductory training and can be expensive from organisational perspective.

9.6.3 ASM Suite Design and Implementation Challenges

Organisational and human oriented challenges reviewed in Sections 9.3 and 9.4 shape a complex and long backlog development project to fulfil in the ASM suite design. Control actuators (a) weaved into the application code or (b) given as part of the control framework or (c) set in platforms offered under or PaaS or (d) given in FaaS framework software allowing serverless computing processing is another source of additional complexity that must be well tackled in the ASM suite design, integrated and tested.

ASM–suites requirements are wide, so in a consequence the project assumptions and general market principles need to bring flexibility as a priority of the platform design. The additional level of flexibility always comes with larger complexity and in a result, the bigger costs of the solution. The costs are twofold: (a) for the implementer – longer production and larger product maintenance and support costs, (b) for the customer – longer adoption, configuration and training.

Limitations of ASM integrations and their high prices shape the current practice to recruit data scientists supporting DevOps (AIOps and MLOps) that will operate adaptive deployment and control with a direct access to EIS components without ASM–class system. Such a team formation offers promising flexibility to the organisation, however, is introduced an ASM toolset may be disturbing established by the team throughput of the solution delivery.

Additional advantages of ASM–level orchestration, defined in Section 9.3 and 9.4, bringing all the parties closer that cumulatively contribute to better global optimisation of the organisation resources within the market realms, see Figure 9.1, but can be difficult to quantify upfront. So the investment decisions can be impacted by no clear TCO and benefits rationalisation. Even though all of these commonly accepted standards are widely applicable and it is clear that a suite integrating tool–sets would bring another level of collaborations synergy.

9.6.4 Adaptive Frameworks Method Constraints

Adaptive control in a complex problem space, **C**, is difficult to be built due to many unknowns and uncertainties. ASM systems collect run–time data of system states, **S**, collections. In order to improve its actions, for the assumed adaptive ASM framework, the control system has to experience many different situations. Those include applied control successes and failures. In certain business conditions this behaviour can be unacceptable. Although adding limits of amount and scale of new control rules inferred could reduce the risks of inappropriate control applied, this would significantly reduce the speed of gaining the knowledge and therefore lower the adaptability of the system. Natural solution would be to add a human validation step to the rules induction mechanism, or a set of predefined system state regions where adaptive control is forbidden, see Section 9.3.2.

Another business–wise limitation is the nature of the actuator type used. Due to functional constraints termination actuators can not be used in every action of the system. On the other hand termination actuators produce the most benefits in terms of reducing work-load, which normally had to be served.

However the biggest weakness of the method proposed in this work is dependence on SLA definitions and potential impacts of wrongly set SLA. SLAs are defined by humans, often without a full awareness of the system run–time characteristics (see risks of disengaged organisation teams discussed in earlier sections). The controller framework "blindly" interprets the input and output system dimensions and by evaluating the effectiveness of earlier control, without any SLAs validation. Wrongly set SLAs can cause significant degradation of the system operation; especially those which depend directly on low level of resources or setting high penalties of often occurring system states. Naturally the control system will tend to terminate as much actions as possible, to limit resources utilised. As already mentioned in Chapter 8 the control framework setup with cheaper SLA termination penalties should use termination more often.

9.7 Summary and Contribution of the Chapter

This chapter explained implications for practice of development and operations teams, in a context of tested control approaches and reviewed theoretical concepts.

The clear tendency is to bring cross–discipline teams of specialists working together building a new ways of synergistic alliances tackling increasing levels of complexity and systems design detail. More development is needed to bring hardware and software with new tools and methods, carrying on started direction of introducing more and more abstractions as observed in development of migrations from IaaS to PaaS to serverless computing and SaaS systems setups where AIOps will have greater contributions.

The last chapter will summarise contribution, review open problems, tackled and non–tackled challenges, but also give directions for the future as part of the global conclusions of the thesis.

Chapter 10

Conclusions

"Truth therefore and utility are here the very same thing."
— Francis Bacon (1561 – 1626)

"Controlling complexity is the essence of computer programming."
"Software Tools" (1976),
— Brian Kernighan (1942 –)

This is the final chapter that concludes and summarises the main findings of the research; statements about the main contributions of the research, discussion of open areas and recommendations for the future work.

10.1 Summary of Research and Findings

The area of adaptive control with non–acquisitive control methods, such as action termination and jobs scheduling, is a significantly underutilised subject of Application Service Management. Cloud computing, hybrid clouds, but also on–premises systems demand more and more autonomous control integrated within deployable Enterprise Information System. This approach may be eminently suitable for systems with harsh execution time SLA, such as real–time systems, or systems running under conditions of hard pressure of power supply, or constraints set up by the green computing paradigm.

The main goal of the research was to build an approach of a conceptual framework that is a ground for further extensions and can be demonstrated with example implementations of self-–adapting control systems being part of ASM suites. In order to achieve above the author took series of steps:

- He reviewed the scale of ASM control with special focus on techniques that optimise the performance of the systems working under SLA objectives in modern EISs – Chapters 2, 3, 4, 7, 8 and 9.
- He introduced novel ASM metrics data processing methods that support the controller and improves performance of the suite in an organisation context – Chapters 5, 6 and 9.
- He proposed novel approaches to non–acquisitive control methods and applied several techniques that bring adaptive features to the control in ASM environment – Chapters 5, 6, 7 and 8.
- He identified the importance of joining algorithms and approaches from many disciplines such as control theory, cloud computing of all main models, software engineering, and Machine Learning (ML) methods, e.g. PSO, GA, NN – Chapters 5, 6, 7 and 8.
- Starting from the assumption that synergy of the specific technologies and ML methods introduce value for understanding applicability of non–acquisitive control approaches, author tested the methods in real application and simulation test–beds – Chapters 5, 6, 7 and 8.
- In a consequence author found several barriers of the methods in the ASM field and highlighted adaptive control limitations. As a result he stressed the necessity of integrated human and adaptive controllers – Chapters 7, 8 and 9.

10.2 Effect of the Original Contribution

In a contrast to the approaches found in literature and discussed in published reports, no system model of any kind is present in the discussed in Chapter 7 approach and a knowledge base is incorporated as the main system metrics and control actions store (Sikora and Magoulas, 2012). This allows to persist all the metrics and SLA values before any control is taken, and after, to reevaluate taken control actions for changed system characteristics, and retrain NN actuators with new control rules on–line. Such evaluation of control actions does not require direct settings of control action to the SLAs defined and later delivered. Also, in terms of SLAs, this approach provides a flexible way of defining functions, allowing the selection and join of all metrics present in the knowledge base.

Moreover, the termination actuator used extensively in this research is not yet very often applied technique to reduce the resources utilisation. The biggest advantage of this approach is a way of removing unintended load coming to the system when more important functions cannot operate and violate more expensive SLAs declared. Of course this control of actions is autonomous, and the control system, as part of ASM–suite, can adaptively define the scale of the interaction based only on measured impacts to SLAs.

Based on thorough accessible literature review, author believes, no one earlier used this type of actuator weaved in the application runtime using AOP and governed by a neural control block, see more details in (Sikora and Magoulas, 2013b) and Chapters 7 and 8.

Preliminary research, done during the problem formulation phase, highlighted a need of rule–based similarity criteria approaches (storing system state after taken control decisions) in adaptive control. Many QoS/SLA and resources bounds requirements can be in conflicting states, applying conflicts resolution strategies improved performance of the controller. Such rules generation have not yet been found in research reports published in the ASM field (Sikora and Magoulas, 2013b).

Dimensionality reduction is necessary in complex ASM systems, where hundreds to thousands dimensions may be present explaining EIS characteristics. Such investigations and applications of causality models have not been found in the earlier ASM literature. The proposed SCIC and SDCIC (Sikora and Magoulas, 2014a) algorithms help to solve the problem. To improve the methods further ASM signal deconvolution (ASM-SD) techniques have been studied as a search problem, see Chapters 5 and 6. The proposed SDCIC algorithm can be improved in many ways, one of them is to apply signals deconvolution, ASM-SD, or uncovering statistics of non–linearities under saturation, and enhanced similarities match including dependency direction in distances check in time. Activity signals extraction as a signals deconvolution technique applied to ASM control (Sikora and Magoulas, 2014b) is a novel technique studied using many metaheuristic search approaches.

Since typical real–time services must guarantee response within specified time limits, the service provider can avoid contract violations by terminating the execution before a hard deadline, just before processing the requested function but also during execution. This scenario is the main focus of this chapter, where the assumption is that already allocated resources are not scaling up or down due to control decisions but can be released by terminating incoming actions requests. This is a technique widely used in practice by human administrators. Nevertheless, it has not attracted considerable attention in cloud computing, cloud–based enterprise systems or operating systems implementations yet. Sill, it is foreseen the method will be introduced to FaaS and SaaS soon, see Chapter 9.

10.2.1 Literature Referring to the Original Research Contribution

- A self tuning regulator for nonlinear time varying control systems based on evolving linear models – A Kalhor - 2014 IEEE Conference on Evolving and Adaptive Systems, 2014

- A self tuning regulator design for nonlinear time varying systems based on evolving linear models – S Jahandari, A Kalhor, BN Araabi – Evolving Systems, 2016

- Service Level Agreement-based adaptation management for Internet Service Provider (ISP) using Fuzzy Q-learning – AK Bin Ramli, 2018
- Bouncer: A Resource-Aware Admission Control Scheme for Cloud Services – AA Abbasi, MAA Al-qaness, MA Elaziz, HA Khalil – Electronics, 2019 (Abbasi et al, 2019)
- Adaptive monitoring for autonomous vehicles using the HAFLoop architecture – E Zavala, X Franch, J Marco – Enterprise Information, 2021 – Taylor & Francis (Zavala et al, 2021)
- A pre-check operator for reducing algorithmic optimisation time in image processing applications – Z Han, K Liu, Z Li, P Luo – Enterprise Information Systems, 2021
- A novel service level agreement model using blockchain and smart contract for cloud manufacturing in industry 4.0 – W Tan, H Zhu, J Tan, Y Zhao, LD Xu – Enterprise Information Systems, 2022

10.3 Open Problems Identified for Research and Foreseen Development

This section will list a few still open problems that could satisfy motivations of future researches taking as a starting point areas discussed in this study.

ASM suite requirements set an extensive range of challenges for functionalities, visualisations and human exposed controls that require further more specific research and prototyping investments. Wide scope of different platforms require independent approach to instrumentation and building new frameworks or embedding in to existing PaaS, FaaS and SaaS models.

10.3.1 Open Problems Linked to Application Limitations Identified

Control systems intrusiveness declared in Chapter 4 is a significant threat to the control stability. As mentioned in Chapter 9 the ASM suite will need to balance are the impacts of monitoring intrusiveness and over–control (too often executed control, too expensive evaluations, etc.).

Another two situations where additional attention needs paid is the intrusiveness of the control actuation instrumentation and control decisions processing. Both are present where controller actuators use too much computation jeopardising efforts to reduce resources utilisation and optimisation is not effective anymore.

As discussed in Chapters 7, 8 and 9 those are important limitations and further research identifying modifiable and adaptive instrumentation methods coupled with controller setup as part of wider ASM suite would bring significant benefits to the performance of the control.

10.3.2 Review of the Open Challenges Reported

Below author presents a list of several open problems in the research area; most of them were outlined:

- High dimensionality (the curse of dimensionality effect): This was the main risk area of the first two years of the research. Author has introduced an approach based on metrics data time series analysis, which was tested using real and synthetic ASM data (Sikora and Magoulas, 2014a). Still, further development of the methods visualising the most significant dimensions and allowing interaction wit human operator would be beneficial.

- Instabilities found during high utilisation and saturation of resources: Tackling the control in the background of instabilities is one of the most difficult problems which has been faced in the research. The phenomena was studied earlier by Hellerstein et al (2004). Even though a resource signal transformation method used as a preprocessing in signal deconvolution search has been proposed and tested successfully, it would be desired to review the signals of different resources types like IO or Memory bound computations under a risk of saturation. Furthermore, the interesting effect of 3rd party processing could introduce similar issue which can be even more difficult to detect and decompose since the higher utilisation is not measured locally and contrary the effect may lead to lower resources utilisation due to wait times.

- Researching general level of autonomy and scope of autonomous control agents use in ASM: Monitoring environment can be also controlled, so the control system would be able to decide which system dimensions (resources as system parameters, and actions as instrumented application code) are most useful. Less important dimensions do not have to be considered; this would lead to limiting the observer effect and lowering metrics data store footprint on the control system. These aspects can be further researched after applying SCIC/SDCIC (Sikora and Magoulas, 2014a) as feature selection methods advising which metrics gathered from which elements of the architecture are most relevant for a given control.

- Optimal frequency sampling problem: The issue is to find the optimal sampling frequency in a given system, where the probes sampling too often is too intrusive for the monitored system and introduces chaotic component, but too seldom does not provide enough data. Using modulated aliasing and adaptive frequency of resources sampling and measure consequences on higher resources utilisation could be an important research area.

- The observer effect in applied monitoring and control agents instrumentation: Too many objects and methods under active– and instrumented with passive– and monitoring can be intrusive and can obfuscate results; it is crucial to understand potential effects of our inspection on all observed artefacts.

- Parameters driven control: For actions which load profile massively depends on parameters passed, controller could take into account parameters dimension as well.

- Hierarchical controllers: Focused on different operating regions agents work is managed by coordinators or agents can communicate with peers. Multi–agent based systems are natural candidates for applying control via distributed actuator agents instances, which could be governed (or cooperated) in hierarchical or more complex fashion. Agents could also independently execute control actions taking into account previous (historical) data and be rewarded and punished according to the real system situation and established SLA functions, moving and evolving in constantly changing environment.

- Time–series prognoses and forecasting in adaptive control, classification and pattern recognition, all present in knowledge extraction about the system usage, may lead to taking more accurate control actions.

- Standard statistical methods and clustering techniques can be used in knowledge discovery process, which should be applied in steps, such as: selection, pre–processing, transformation, data mining, interpretation and evaluation(Abonyi and Feil, 2007). Although this problem area has been covered with signal processing methods of SCIC/SDCIC and ASM-SD there seems to be an interesting research niche to cover related to selection of dimensions after control (intervention) that are measured from statistical significance perspective.

- Large amounts of data cause challenges for search in the systems states knowledge base. There is a need to research more efficient mining methods for patterns search and dimensions selection. Multivariate metrics search and analysis can be the basis for further predictors and classification research in the field.

- Queue–model–based controllers: The experimentation testbed can be a model for predictions as outlined in Section 10.3.2.

- Application of Causal Reinforcement Learning (CRL)[1] methods built from works of Bareinboim and Pearl (2016) in the area derived from intertwining domains of Reinforcement Learning (RL) and Causal inference: As mentioned in Section 5.7.2 RCT (Fisher et al, 1937; Chalmers et al, 1981; Splawa-Neyman et al, 1990; Rubin, 1990) and RL methods (Sutton, 1984; Sutton and Barto, 1998) are prone to limited

[1]https://crl.causalai.net/

efficiency especially in conditions of high–dimensional problem spaces, but when extended by causal models (Pearl, 2000; Pearl and Mackenzie, 2018) and Causal inference (Pearl, 2010) can support each other bringing interesting prospects to ASM control as well. Methods supporting control of systems under SLAs targets explained in Chapters 4, 5 and 6 and control methods discussed in Chapters 7 and 8 where data collection, model building as a curve fitting, and intervention as environment exploration are just two bottom layers of the causation hierarchy (Pearl and Mackenzie, 2018). Those are namely L1 and L2 according to the Causal Hierarchy Theorem (CHT) introducing tier model of 3 layers: L1 "Association" (seeing, observing), L2 "Intervention" (doing, intervening) and the highest L3 "Counterfactuals" (imaging, retrospection). Chapters 5 and 6 provide examples of mechanisms defending the position of RL in ASM control (Sikora and Magoulas, 2014b, 2015), nevertheless, more recent works of Bareinboim et al. seem to offer a tremendous potential of using CRL methods in the EIS services level control (i.e. non–acquisitive control) that has far lower granularity (so higher dimensionality) than conventionally used (Lu et al, 2002; Welsh and Culler, 2003; Abawajy, 2009; Li et al, 2010; Cheng et al, 2017) in adaptive scaling, scheduling and VM allocation. CRL tackles such areas as: (a) Generalised Policy Learning (GPL) as an attempt to systematically combine online and offline intervention approaches (Lattimore and Szepesvári, 2020; Gasse et al, 2021) , this is similar to the practical application of the approach mentioned earlier in Section that is combining offline and online methods of interaction; (b) deciding on when and where to intervene as a refinement process of the policy spaces, identifying on subset of offline policy spaces (Lee and Bareinboim, 2018, 2019, 2020); (c) Counterfactual Decision–Making (CDM) focusing on changing optimisation function based on intentional actions of human–agents, free will, and their autonomy, using optimisation criteria based on counterfactual reasoning (Forney and Bareinboim, 2019), the approach has been highlighted in Chapter 9; (d) Generalisability and Robustness of Causal Claims, under transportability and structural invariances shared across training causal models and deployment environments; (e) Learning Causal Models (LCM), methods discovering the causal structure (learning the causal graph) with systematic observation and performing experiments (Kocaoglu et al, 2017, 2019; Jaber et al, 2020); (f) Causal Imitation Learning (CIL), as a L2 policy learning with unobserved rewards or partially observable L1 data given by experts (Zhang et al, 2020).

10.3.3 Open Control Problems in Context of Cloud Computing Models

Chapter 9 listed possible applications in a context of EIS and cloud computing. Still, there are several open areas that could bring attention of future researchers and systems designers.

First of all, as we know, the application level adaptive control requires instrumentation. As of now author has not found an evident signal in the available literature to create a standard for the cloud control APIs, that can be applied to directly to FaaS model. Nevertheless, there have been instances of new standards raised as per Request for Comments (RFC) process[2]. Proposed RFC–7252 "Constrained Application Protocol (CoAP) Option for No Server Response" by Bhattacharyya et al (2016) introduces UDP–based lightweight binary protocol bringing the REpresentational State Transfer (REST) paradigm to constrained applications. RFC–8323 by Bormann et al (2018) extends Constrained Application Protocol (CoAP) over TCP, TLS, and WebSockets, and RFC–8768 by Boucadair et al (2020) removes risks of the infinite forwarding loops by detection and defining the hop–limit under CoAP. Californium[3], Java–CoAP[4], MjCoAP[5] (Cirani et al, 2015) are examples of open–source lightweight Java implementation of the protocol implementing server– and client– side interfaces. RFC–8516 titled ""Too Many Requests" Response Code for the Constrained Application Protocol" by Keranen (2019) introduces a special Hypertext Transfer Protocol (HTTP/1.1) (RFC–7230—RFC–7235 (Fielding and Reschke, 2013)) code `4.29` to CoAP that indicates the back–off period applying only to similar requests, where the similarity is context dependent. Joint to the above topics Hartke (2015) issued RFC–7641 addressing resource observation for CoAP. Although CoAP is specialised to support IoT demands of distributed devices of low–power, loosely coupled, lagged networks, low resources availability and other related constrains it can be an interesting field supporting EISs and cloud models PaaS and FaaS by the protocol extensions and implementations of non–acquisitive control.

Moreover, there is a strong need to define open standards of monitoring and actuators that can run under EIS. In the last two decades there were several RFC for monitoring but mainly for network and transport level, i.e. RFC–3919 by Stephan and Palet (2004), RFC–4148 by Stephan (2005), RFC–4149 by Kalbfleisch et al (2005), RFC–4502 by Waldbusser (2006), RFC–4710—RFC–4711 by Siddiqui et al (2003), RFC–8321 by Fioccola et al (2018).

Real–time FaaS or more generally function executing under serverless computing with harsh execution time SLAs can be considered as hard– or soft– real–time system, which due to multi–layer virtualisation and complex scheduling mechanisms are going to be extremely difficult to be addressed. RFC 7478 Web Real-Time Communication Use Cases and Requirements, MARCH 2015

[2]`https://www.rfc-editor.org/`
[3]`https://www.eclipse.org/californium/`
[4]`https://github.com/PelionIoT/java-coap`
[5]`http://netsec.unipr.it/project/mjcoap`

10.3.4 Cross–cloud Brokerage of Federated Clouds

FaaS due to heavy reliance on SOA and SLA is possibly a fruitful area for adopters of cross–cloud brokerage of services usage (Guzek et al, 2015). Multi–cloud and federated cloud (Kertesz, 2014) challenges that enterprise face (Paraiso et al, 2012) require higher level abstraction tools building bridges of integrations between cloud providers (Jamshidi et al, 2015).

ASM suites and adaptive control can introduce significant value to cross–cloud brokerage seen from three main perspectives: (1) cloud service inter–mediation, where intermediary introduces value by provisioning additional services, (2) aggregation, where a broker service combines many independent services into one coherent consolidated service of likely simplified interface allowing to manage the services of many sub–providers, (3) cloud service arbitrage, where a broker provide arbitrage service of choices of aggregated services maximising costs, workload requirements, specific SLAs, or functionality needs defined by the client. In all of the categories of the brokerage services non–acquisitive control can find a place under TnC of the broker and client contracts.

Dynamic B2B cloud marketplace and variances of SLAs and service prices across cloud providers (Paulsson et al, 2020) may introduce needs for additional level of non–acquisitive control methods to be applied. So that the controller can decide by adaptive switching method which cloud function should be fired under the current organisation and workload circumstances.

10.3.5 Adaptive Modelling of SLA Contracts, Pricing Models and Renegotiable SLAs

The adaptive control methods discussed in Chapters 7 and 8 shape designs for proactive frameworks allowing SLA contracts renegotiation, which can be another area of research interest. Such a meta–control could be applied to ongoing mediation of agreements over costs and prices, but also scope of the service and SLAs.

ASM suites data converted into business–driven KPIs would support clearer situational awareness allowing appropriate precautionary preparedness for the negotiations of services costs and SLAs.

By default open boundaries used by adaptive controllers in a normal work could be constrained in order to regulate the negotiated terms. Furthermore, the controllers with direct access to the contract negotiating features, i.e. on–spot or long terms price bidding, changing category of client account (gold, silver, bronze etc), tackling the business profitability optimisation function.

Adaptive SLA targets optimisation, supplemented by CapEx and OpEx monitoring, can be connected with other regular business domain regulating features of EISs such as: variable pricing, discounts, promotions, lead acquisition, user profiled recommendation engines tuning and digital marketing campaigns. Those are shaping market for the clients, giving fast–adaptive cycles of incentives, formulating impulsion and motivations for services adoption, and in the long term convert into better financial performance P.

10.3.6 Application Service Management as a Service (ASMaaS)

Most of modern APM and ITSM systems are available on SaaS basis, which is a preferred model by clients (Matchett et al, 2017) due to such factors as: fast operational readiness, minimised installation efforts, low costs of service introduction and flexibility of the subscriber pricing model.

Insufficient development in the area of open–standards of actuation, absence of general use control agents implementations that are easy to be applied, create an opportunity for a well designed set of implementation could become a de–facto standard, and the layer orchestrating the ASM could be moved to as a service model, Application Service Management as a Service (ASMaaS), linked with CAMP paradigm. Cross platform and application agnostic design could open the platforms to many clients.

Such an ASM control system could be fully aware of the costs and would be able to support those calculated under business targets, revenue gains and hosting costs can be converted to internal economical factors to be incorporated into SLAs for adaptive control goals and driving financial performance P optimisation, see Section 9.5.

10.4 Directions for Future Research

The core objective and the outcome of the research was to build a concept level foundation for creating a self–adapting controlling systems. Considering the fact that probably fully autonomous systems continuously adapting to changing conditions can have resistance of operators and marketing teams (via customers agreements) to be implemented in the practice, it is still important to begin scientific research which will face a problem of intelligent agents control in the enterprise sector.

- From Chapter 5:

  Methods discussed build sequences of Similarity, Interference and Clarity values, under SCIC – therefore provide details of interpreted data against time, which can be helpful for understanding the relationships evaluated. In cases when only some actions are controllable $a_c \in a$, it is possible to find out which controllable actions have the highest impact on resources and SLAs and include that in the rankings accordingly. This can be an interesting audit property of the method when applied in the engineering problems space.

Additionally the introduction of the Dependency that measure complements and enhances the Similarity measure, used in the original SCIC method, by applying further causality observations based on cross–correlation. The use of this measure leads to lower Clarity scores in situations when observed effects were delayed against the set causality chain. The evaluated scenarios confirm the practical potential of the proposed approach.

The research opportunities in the area of features selection for more focused autonomous control in the ASM field is far wider. In particular there are more methods investigated tackling various aspects relating to signal decomposition, unmixing hidden components, and incorporating knowledge about the system reactions delays, which were highlighted in the next point.

Finally the causal inference and counterfactual reasoning methods outlined in Section 5.7.2 can be a very fruitful future research area for means of checking the potential intervention and highlighting causal–impacts from already executed changes to the dynamics of the system under control.

- From Chapter 6:

  In the future it would be desired to continue enhancing the search process of ASM signals in the area of system states distributions driven search to tackle issues like system responses delays, signals noise, impacts of unmonitored blind spots, and observer effects, to better address needs of adaptive controllers, schedulers, but also decision support systems. Also, the other general optimisation methods are expected to be tested under the proposed and extended deconvolution model, with focus on industry applicability and parallelisation gains. This research area is also important for analytics and visualisation that is a key component of ASM tool–kits, extending our earlier research in the area of features selection.

  Moreover, additional research needs to be done in processing of signals acquired from more complex computing system models, containing many architectural units, components, network elements and communicating with other distributed services supplying the integration needs of present–day, distributed, hybrid– and cross– cloud computing systems.

- From Chapter 7:

  Further validation of the approach is needed and enhancements, particularly through designing more complex control blocks for scheduling in order to control specific operating areas. To this end, in the future the research could continue enhancing the framework with scheduling algorithms, Section 8.4.1, especially those that are dealing better with overloaded systems, and apply more complex system architectures, allowing to test memory allocation or network consumption as key resources for distributed systems.

Another promising area for further work would be to extend the control block and the neural model of action types. Such extension would support collecting and processing run–time data directly without evaluation that unavoidably comes with longer feedback time. Furthermore, using adaptive thresholds for defining high–mark and low–mark states, see Section 8.5.2, would be useful as it would allow the controller to regulate how aggressive or smooth the applied control is; for example, reducing the frequency of terminations when the system under control has not reached certain operating regions yet, that should smooth the effect of the control in the declared areas of the system state space.

Lastly, there is a significant potential in an extension that would allow the controller to switch off a decision block the chosen action when the system is not significantly loaded. This will reduce the activity of the control block to sampling only, and would allow implementing a risk mitigation strategy for preventing potential terminations that do not bring additional benefit since the available resources are expanded to support a good level of operations.

- From Chapter 8:

  To improve the performance of the adaptive control and shorter the feedback loop by apply additional data sources, available in the local context of the neural decision block runtime require actions activity as input for local termination decision making.

  Models of queues could be used for advisory component of a decision block, see Section 10.3.2. Although the neural control block is faster the control decisions that are just below confidence levels of NN model deployed to the actuator can slightly delay the execution by additional check with an additional decision block that perform simulation of the current conditions. Such mixed approach open the area of the research of hierarchical control blocks; that perform decision making in a sequence of: (a) simple if–else rule–based, (b) NN, (c) queue based, (d) block that could execute simple evaluation with access to the knowledge–base of system states. Where the blocks are arranged from those of the lowest computation footprint and with the highest certainty of control decision gradually introducing more complex models.

10.5 Concluding Remarks

Although ASM control is gaining more and more attention in the enterprise, there is still lack of well researched features selection methodology, which could be used to improve autonomous control agents performance and support ASM operators.

As mentioned earlier, so far author has not found ASM solution using non–acquisitive control techniques on wider scale in adaptive control in the ASM field. Author believes that the main contribution in the area would be researching best ways of applying those methods to the field for particular use–cases.

Higher energy costs footprint in TCO of hosting, greater CO2 reduction pressures and stricter overspending policies should stimulate the research of non–acquisitive control area.

Further industry implementation that base on the research results could help in daily work of performance engineers, DevOps, and the system administrator staff. The problem area is very wide to accommodate many further research programmes working on various adaptive techniques which can be used for adaptive controllers. The proposed conceptual framework and architecture is scalable and implementations can be use in EIS practice. Cloud computing trend emphasises needs for the ASM control to optimise–the–whole (Poppendieck and Poppendieck, 2003) in big computation centres and in firms delivering their EISs.

Appendix A

Publications

This Appendix lists author's publications shortly describing the research results and original findings, that have been incorporated in the Thesis and were elaborated in other chapters accordingly.

A.1 Neural Adaptive Control in ASM

The paper was published in proceedings of the 13th International Conference on Engineering Applications of Neural Networks (EANN'12), which took place in London between 20th and 23rd September 2012. In the report Sikora and Magoulas (2012) describe a neural network controller for ASM. The work presents developed software framework equipped with a simple statistical methods for evaluating the system states, simulations results and explore the potential of this approach. The control system checks current and historical system states information and generates rules for the controller. Rules definitions are approximated by a trained pair of NNs, that are weaved in the application run–time. The software framework is able to change internal elements of runtime execution, by taking control actions in background of flexible SLA definitions and resources as part of the current system state. In this study the controller is only equipped with a termination actuator eliminating expensive actions, and can adapt to changing conditions according to modifiable SLA definitions, still without a model of the system. The general objective is to optimise declared SLA functions values. No predictors and no forecasting of services demands and resources utilisation have been used.

The material from this paper is used mainly in Chapter 7.

A.2 Extended Neural Adaptive Control in ASM

The previous conference paper was selected for inclusion in a special issue. The paper is an extension of the earlier results published and contains additional findings gathered during the research. The submitted manuscript has been published in *Evolving Systems* Journal[1]. The paper (Sikora and Magoulas, 2013b) contains material extended of a detailed description of the evaluator and rules generation engine working with different conflicts resolution strategies. The control system "learns by observation", building a Knowledge Base and solving the optimisation problem under SLA definitions. The use of a reward and punishment method in order to find control rules according to the specified in SLAs direction was explained. The idea of exploiting background knowledge with positive and negative examples in order to formulate a rule–based hypothesis about the accuracy of an action was researched in the context of inductive reasoning (Muggleton and De Raedt, 1994; Muggleton, 1999).

The material from this paper is used mainly in Chapter 8.

A.3 Feature Selection based Dimensions Relevance Search

The paper (Sikora and Magoulas, 2013a) contains results of the third year of the study, and was presented in proceedings of the IEEE Science and Information (SAI) Conference, which took place in London on 7 to 9th of October 2013. It describes a new approach based on metrics data time series analysis, which was tested using real and synthetic ASM data. Statistical methods and causality model assumptions (on the system structure) were applied to derive the system interdependencies using time series analysis and a simple causality model of the system definition. Specifying the causal model (Pearl, 2000) in the controlled system significantly simplifies evaluation of defined elements utilisation dependencies. This allows more efficient time series similarities search processing, selection of most relevant dimensions, and easier control in the reduced space, which would ultimately reduce maintenance effort. This paper poses the feature selection method based on metrics time series analysis.

The material from this paper is used mainly in Chapter 5.

[1]Evolving Systems – An Interdisciplinary Journal for Advanced Science and Technology, Editors-in-Chief: P. Angelov; D. Filev; N. Kasabov, ISSN: 1868-6478 (print version), ISSN: 1868-6486 (electronic version), Journal no. 12530

A.4 Finding Dimensions Relevance in ASM Control

The previous paper was well received by the reviewers board, and extended material containing improved method SDCIC was published as a book chapter (Sikora and Magoulas, 2014a) in Intelligent Systems for Science and Information, Studies in Computational Intelligence, Springer, 2014. The main area of the improvement was an additional measure of Dependency between signals greatly improving stability of Consequence and strength of causal relationship by checking result of cross-correlation (positive lags) following the key assumption of causation theory that in a causal system a response never precedes an input. In this study I have developed better understanding of the signal unmixing/extraction problem and complexities involved.

The material from this paper is used mainly in Chapter 5.

A.5 Search–guided Activity Signals Extraction

As outlined in (Sikora and Magoulas, 2014a) conclusions and future work section SCIC/SDCIC do not address well signals of resources that contain impacts of many other actions running. This paper discusses an approach that can be applied as a deconvolution technique that can greatly help to uncover hidden run-time relations between observed signals in the ASM field. This is further work on features selection methodologies, which could be used to improve autonomous control agents and support ASM operators. In order to analyse extracted metrics ASM signals need to be decomposed to find relations between system utilisation and effective activity. Introduced a metrics signal deconvolution method that can be used to support human administrators and/or be incorporated into feature extraction preprocessing of decision blocks of autonomous controllers. Special attention was paid to highly utilised resources and saturation effects.

The method considers the decomposition problem as a search problem that is solved using heuristics and metaheuristic strategies. Quantitative and qualitative relations between activity and system resources signals were searched with use of a model that is based on similarity and variability of the changes, where minimal assumptions about the system architecture and design are present. In this study four different groups of optimisation techniques have been tested starting from gradient-based method Nelder-Mead(Nelder and Mead, 1965), through SANN (Bélisle, 1992), PSO (Clerc et al, 2011), and GA (Tenorio and Tenorio, 2013). Findings of this part of the research have been published in the proceedings of 4th UK Workshop on Computational Intelligence, that took place in Bradford on 8 to 10th of September 2014.

The material from this paper is used mainly in Chapter 6.

A.6 Evolutionary Approaches to Signal Decomposition in an Application Service Management System

Activity signals extraction as a signals deconvolution technique applied to ASM control (Sikora and Magoulas, 2014b) has been found as a promising field by UKCI2014 reviewers, and this work has been selected out of the best papers to be extended and proposed to be a part of a special issue of *Soft Computing* Journal[2] in en expanded form.

The material from this paper is used mainly in Chapter 5.

A.7 Neural Adaptive Admission Control Framework: a SLA–Driven Action Termination for Real-Time Application Service Management

Adaptive control features that have been tested with use of a modelled system (Sikora and Magoulas, 2021), applying "grey–models" and reduced dimensionality spaces. Experiments have been conducted with use of a modelled application, that has been instrumented with termination actuators. The simulations contain far more complex SLA conditions. Apart of action termination actuators there is job scheduling control discussed further. All the work was published under Enterprise Information Systems[3] Journal.

The material from this paper is used mainly in Chapter 8.

[2]Soft Computing – A Fusion of Foundations, Methodologies and Applications, Editor-in-Chief: Antonio Di Nola, ISSN: 1432-7643 (print version), ISSN: 1433-7479 (electronic version), Journal no. 500

[3]Enterprise Information Systems – Publishes on the theory and applications of enterprise information systems, from architecture and design to modelling and integration, Editor-in-Chief: Andrew W. H. Ip, Hong Kong Polytechnic University, Hong Kong & University of Saskatchewan, Canada, Print ISSN: 1751-7575 Online ISSN: 1751-7583.

Appendix B

Documentary

This documentary appendix discusses a general picture of the technology, industry, psychology, economic, and social perspectives describing the background of the research area and highlighting the motivations of use and industry applications.

B.1 Background of the Market and Industry Needs

The author believes it is important to collect the historical details describing the context of the market and industry needs for a better understanding of the applications, conclusions and future work discussions in Chapters 9 and 10 of the thesis.

B.1.1 Enterprise and Cloud Economics

With those motivations and objectives the new model offering additional level of abstraction tools, elastic compute scalability, high frequency of deployments, and great monitoring capabilities, cloud computing allowed development and operation teams collaborating even tighter reducing overspending, limiting communication waste and reducing SDLC inefficiencies of long processes, team–work, business–needs and technical challenges (Beck et al, 2001; Poppendieck and Poppendieck, 2003; Reis, 2011; Humble and Kim, 2018), see Section 9.1 Points 3f–3g.

KPMG study by Bishop et al (2015) suggests that enterprises TCO may be reduced to even 45% for private and 10% for public cloud in comparison to traditional IT TCO. Apart of the lowered service hosting costs (Rosati and Lynn, 2020) for further maximisation of RoI it is critical to exploit the scaling flexibility and reduced time of deployments in cloud–ready EIS architectures (Jackson and Goessling, 2018).

Value chains of cloud computing have been thoroughly studied in last two decades (Qian et al, 2009) first focusing on elastic managed services support for distributed and SOA driven designs (Neto, 2011), subscription models, spot (Agmon Ben-Yehuda et al, 2013) and bid pricing (Li et al, 2016) and later moving to more serverless deployment models (Eivy and Weinman, 2017). Public clouds benefit from economies of scale (Etro, 2015) and so still the demand on the services grow (Sastry, 2020) the observed business value for both clients and providers continuously progresses (Lynn et al, 2020).

Nevertheless, despite the huge potential of main cloud computing models, those motivations for the further progresses are limited by many practical challenges (Llorente, 2017). The slowdown (impediments) introduce the need for more mature management and service optimisation of EISs (Briggs and Kassner, 2016).

B.1.2 Moore's Law and Cloud Computing Challenges

Moore et al (1965) predicted that "the number of transistors in CPU chip will double around every two years". The projection based on early observations appeared to be remarkably precise establishing constant historical trend, see Figure B.1 (Rupp, 2020)[1]. While Moore's Law was offering gains in performance every two years it has been a basis for a regular boosts in speed of general–purpose CPUs. Consequently enterprise software development tend to focus on reducing the time to develop an application rather than the total execution time it takes to run applications. Furthermore, this effect introduce situations where no incentive to design and develop chip architectures designed to a specific task under a risk that in two years a general–purpose chip would outperform the initiative.

Since reaching the CMOS[2] limitations and power dissipation issues (Etiemble, 2018), the clock frequencies largely levelled off around 2005 at around 3.7 GHz in 2008 (Danowitz et al, 2012), see Figure B.1. In effect many enterprises and academic institutions were forced to adopt systems to be designed in order to exploit multi–core resources (Stewart et al, 2017). Additionally, there are more constraints preventing further scaling of cores and slowing down production of larger CPUs (Taur, 2002). Still the demand forces the industry to overcome the impediments (Yeap et al, 2019) including 5nm barrier (Lee et al, 2018), and carry on further CMOS scaling towards 3nm and beyond (Mocuta et al, 2018). There are many other factors that need to be considered while comparing the performance of CPUs, i.e. the width of the data bus, the response time of the memory, and the cache architecture, but all of those are quite related to the CMOS limitations.

The current technology state shapes generally two ways of further development outlooks:

1. The wider use of parallelism with many cores of CMOS technology, with many

[1]Chart and data hosted at `https://github.com/karlrupp/microprocessor-trend-data`, reviewed and published as well at `https://ourworldindata.org/technological-progress`.

[2]Complementary Metal—Oxide-–Semiconductor (CMOS), as a type of Metal—Oxide—Semiconductor Field–Effect Transistor (MOSFET) creation process.

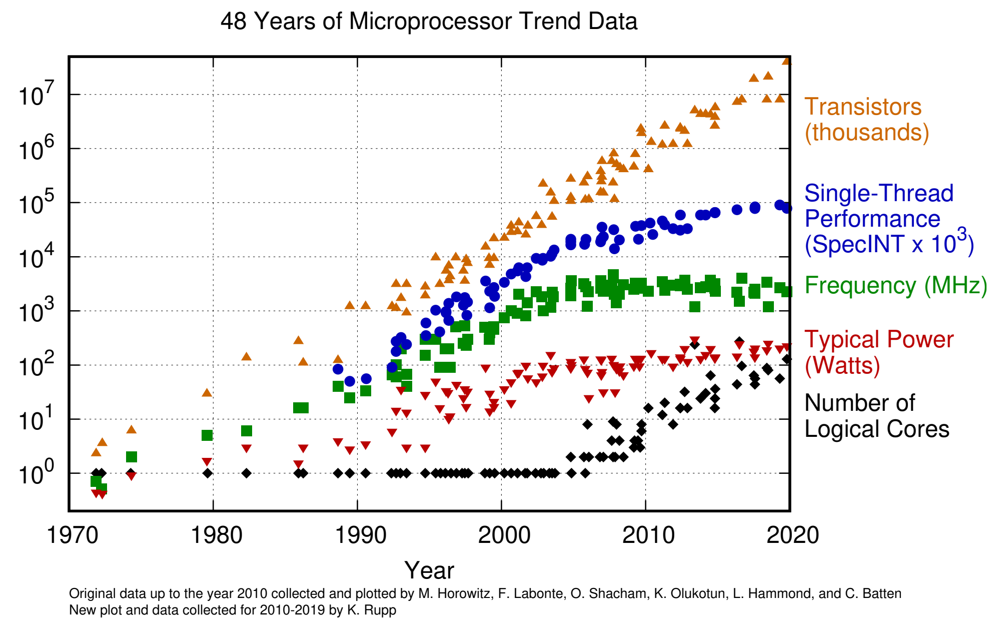


Figure B.1: This chart compares the transistors, single-thread performance, clock frequency, thermal design power, and number of logical cores of microprocessors from 1971-2020 (Rupp, 2020).

limitations (Haensch et al, 2006) and ways to improve the performance of CMOS planned (Bohr and Young, 2017; Razavieh et al, 2019) (a) employing larger groups of general–purpose CPU (or more specific GPU) cores on the same chip[3] (González, 2019), (b) implementing larger systems that use parallel processing (Shukur et al, 2020; Behnezhad et al, 2021) often following heterogeneous computing models (Shalf, 2020; Zou et al, 2021).

2. The introduction of other technologies, where many are still on very theoretical level, tackling both logic elements and memory improvements (Chen et al, 2019) such as: 3D transistor, Spintronics (Manipatruni et al, 2018), Tunnel Field–Effect Transistor (TFET), Carbon Nanotube Field–Effect Transistor (CNTFET), Single–Electron Transistors (SET), Quantum dot as Quantum Cellular Automata (QCA), Photonics and optical computing, Reversible computing, DNA computing, Quantum computing, and many more, with significant challenges as outlined by Wu et al (2013) and often requiring new approach to software design and development.

Still, typical EIS applications bottle necks are more I/O related, with disk and memory access, since they very much operate within feature–sets that are requiring more On–Line Transaction Processing (OLTP)–class or online–event processing with often access to persistence mechanisms. Also, the tendency to serve higher load of incoming requests (Lomet, 2018) often under SLA drives a demand on larger software cache subsystems, such as full text search indexes, data dictionaries, web caching, ORM and DB–level caching. Wider application of those often expose limitation in a form of low amount of RAM

[3]See processor offer and plans of AMD https://www.amd.com/en/processors/, Intel https://www.intel.com/content/www/us/en/products/details/processors.html, and IBM servers and processors https://www.ibm.com/it-infrastructure/power/.

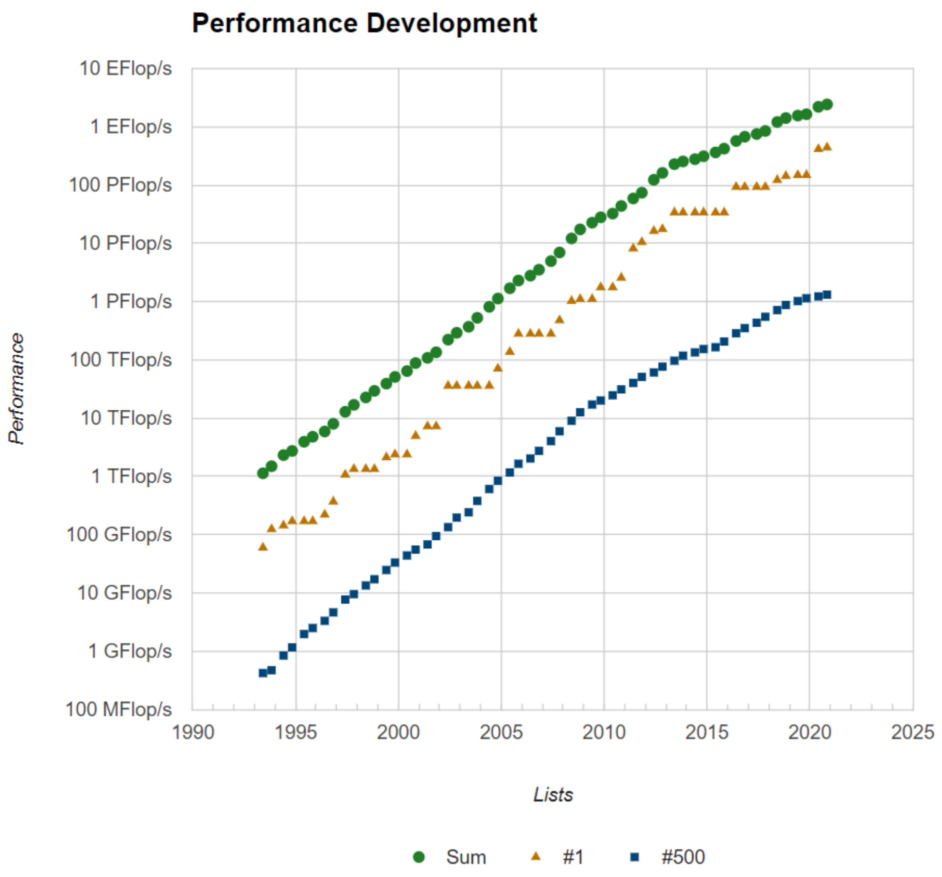


Figure B.2: Performance development of top500.org supercomputers, 1997–2020. Total computational power vs the best and the last in the ranking (Strohmaier et al, 2020).

memory available (Paschos et al, 2018).

Another critical requirement creating limitation of large systems is need to distribute computation amongst many locations. The wide well established in literature extension to the cloud computing facilitating the rapidly spreading paradigm of IoT is a concept of fog computing (Bonomi et al, 2012; Yi et al, 2015; Atlam et al, 2018; Singh et al, 2021b). In the fog the services are instantiated across many logical nodes distributed geographically and that implies distribution of not only the computation, but also the communication, storage resources, and other services on closer to services of the end–devices (edge devices) and systems in the control of end–users.

Another group of cloud computing challenges is lined to the the internal for cloud clients limitations is the inadequacy of team resources and expertise to perform migrations. Employers are still finding it difficult to find employees with all the necessary experience to set up adequate cloud deployment pipeline and services hosting setup. Complete governance over corporate services in hybrid cloud, compliance and cost management are another key difficulties organisations face which are slowing down the adoption phase. Further adoption may also bring risks related to the multi–cloud environments and vendor lock–in issues.

B.1.3 Energy Consumption of Cloud Computing

Since the demand on cloud computing remains extensive (Jones, 2018; Kamiya and Kvarnström, 2019) and regardless of the above mentioned limitations the growth of computational power deployed in the industry continuous (Shehabi et al, 2018). It is important to review other globally observed risks that are just another type of limitations to overcome. Where growing energy consumption is a key area of concern (Kamiya and Kvarnström, 2019; Kamiya, 2021).

Cloud computing is already a valuable response to global ICT emissions (Pesce, 2021), which is extended by raising transfer electricity footprint (Aslan et al, 2018).

Still, server power supply and processors are accountable for only 14% and 15% total energy demand in data centres, while cooling accounts for 38% of total energy demand in data centres. Cloud computing due to computation resources consolidation that significantly reduces the gross energy use (Verma et al, 2009), with monitoring and profiling (Govindan et al, 2009) improves the general performance and SLAs (Gandhi et al, 2011), tackling conditions of overloading (Beloglazov and Buyya, 2012a), and network usage (Zheng et al, 2014), as opposed to standard enterprise on–premises systems that often are highly under–utilised (Koomey et al, 2011).

Ultimately the greater computational power of cloud data centres the greater energy usage. Electricity costs are a dominant operating expense and account for 10% (Koomey et al, 2007) to even 20% (Rasmussen, 2011) of the TCO of a high availability computing data centre. Thus the energy efficiency optimisation become a critical element of design and costs implications for cloud providers (Beloglazov, 2013). At the end, data centres are not only as valuable as the processors but the rest of the hardware and power setup they utilise.

According to Koomey (2008); Koomey et al (2011) the number of operations a computer can perform with each kilowatt hour [kWh] of energy has doubled around every 1.6 years for peak performance and every 2.6 years for the average performance. That's a 10-billion-fold improvement over 50 years. Although, the rate of the increase has slowed since mid 2000's but not stopped, the current generation of computing will face physical limits of the transistors functions due to the heat dissipation challenges and non proportional power increase in function of computational power. Further comparable efficiency gains require revolution in electronic circuits design.

As discussed above the observed slow down of single thread performance related to 2000's slow down of Moore's predictions due to frequency limits, see Figure B.1, the power efficiency [GFlops/Watt] of large datacenteres is constantly growing (Khan et al, 2021) as shown in green500 ranking[4], see Figure B.2. According to the study of IEA[5] by Kamiya

[4]`https://www.top500.org/lists/green500/`

[5]IEA – International Energy Agency, `https://www.iea.org`.

and Kvarnström (2019) the global data centre energy demand has stabilised between 2015 and 2021 not reaching 200 TWh yearly, with steady reduction of power consumed on infrastructure from 84 in 2015 to 63 TWh in 2020 giving proportionate gains for servers consumption with comparable network and storage needs. The same study reports even bigger change in the share of hyperscale data centres from 34 TWh in 2015 to 76 TWh, so 117% increase, where cloud non–hyperscale raised only 17%, whilst traditional lowered proportionally. Forecasts of energy consumption of servers and data centres worldwide vary from 800 up to 3000 TWh per annually in 2030 (Hintemann and Hinterholzer, 2019).

Ultimately the growth of power consumption will be related to greater costs (Andrae and Edler, 2015). Even if the energy footprint stabilises the cost of necessary infrastructure needed to optimise the energy consumption and minimise the waste will require additional investments.

It is expected that with further exponential growth of computing power of data centres and cloud computing in general the component of energy costs will be wider. This effect will be stronger when the costs of electricity production will continue to grow (Lorenczik et al, 2020).

Table B.1 presents summary of "forces" that slow down but also speed up the further development of computational power of data centres and their applicability to deliver solutions to EIS problems.

B.1.4 Supercomputers

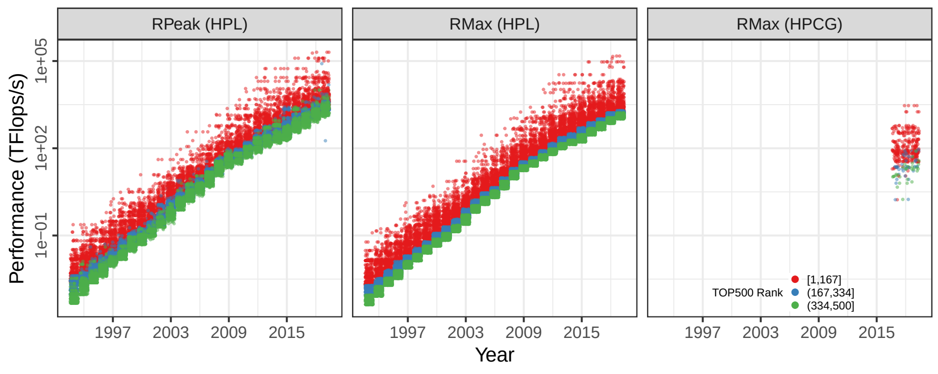
Interesting perspective on the future of cloud computing that currently is more reliant on commodity hardware may bring review of HPC and development of supercomputers regularly investigated and benchmarked by top500.org[6].

Although supercomputers create a separate niche there are numerous similarities that help to conclude on trends well correlating to the cloud computing scale of development.

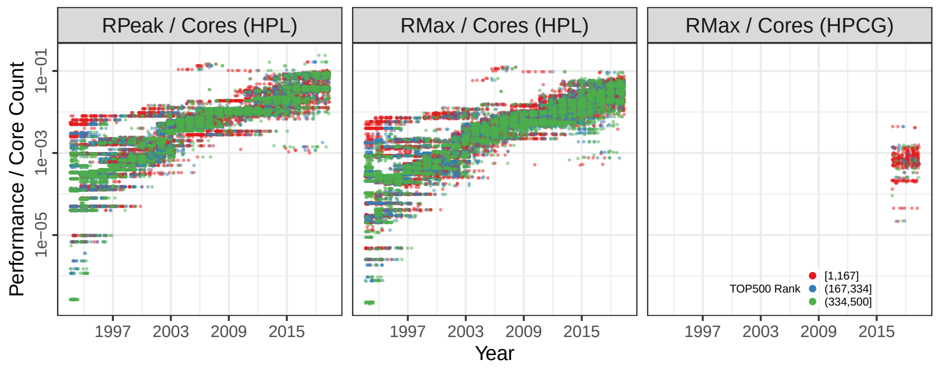
In 2000's cloud computing was seen as still not yet mature enough for HPC computations. Napper and Bientinesi (2009) acknowledged that although cloud provides an extensible and powerful computing environment for web services, still the experiments conducted indicated that the cloud (or leading provider like Amazon's EC2, at least) is not meeting the research requirements. Almost a decade later Mohammadi and Bazhirov (2018) reported that leading public cloud providers deliver the single computing core performance comparable to modern traditional supercomputing systems.

Let's review the data collected. Figure B.3 contains the latest data of supercomputer performance statistics provided by top500.org (Lenski, 2020), where RMax is maximal Lin-

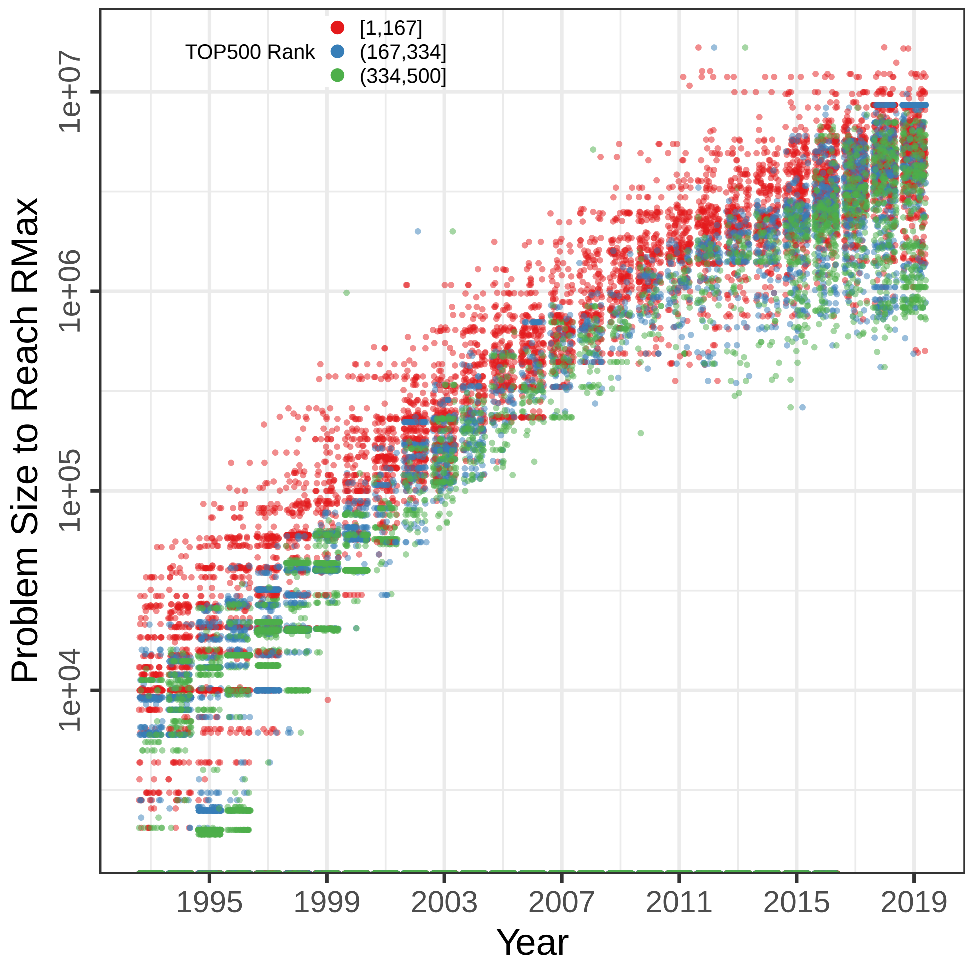
[6]https://top500.org



(a) top500.org RPeak



(b) top500.org RMax cores



(c) top500.org NMax

Figure B.3: Selected statistics of top500.org supercomputers from 1993 to 2020.

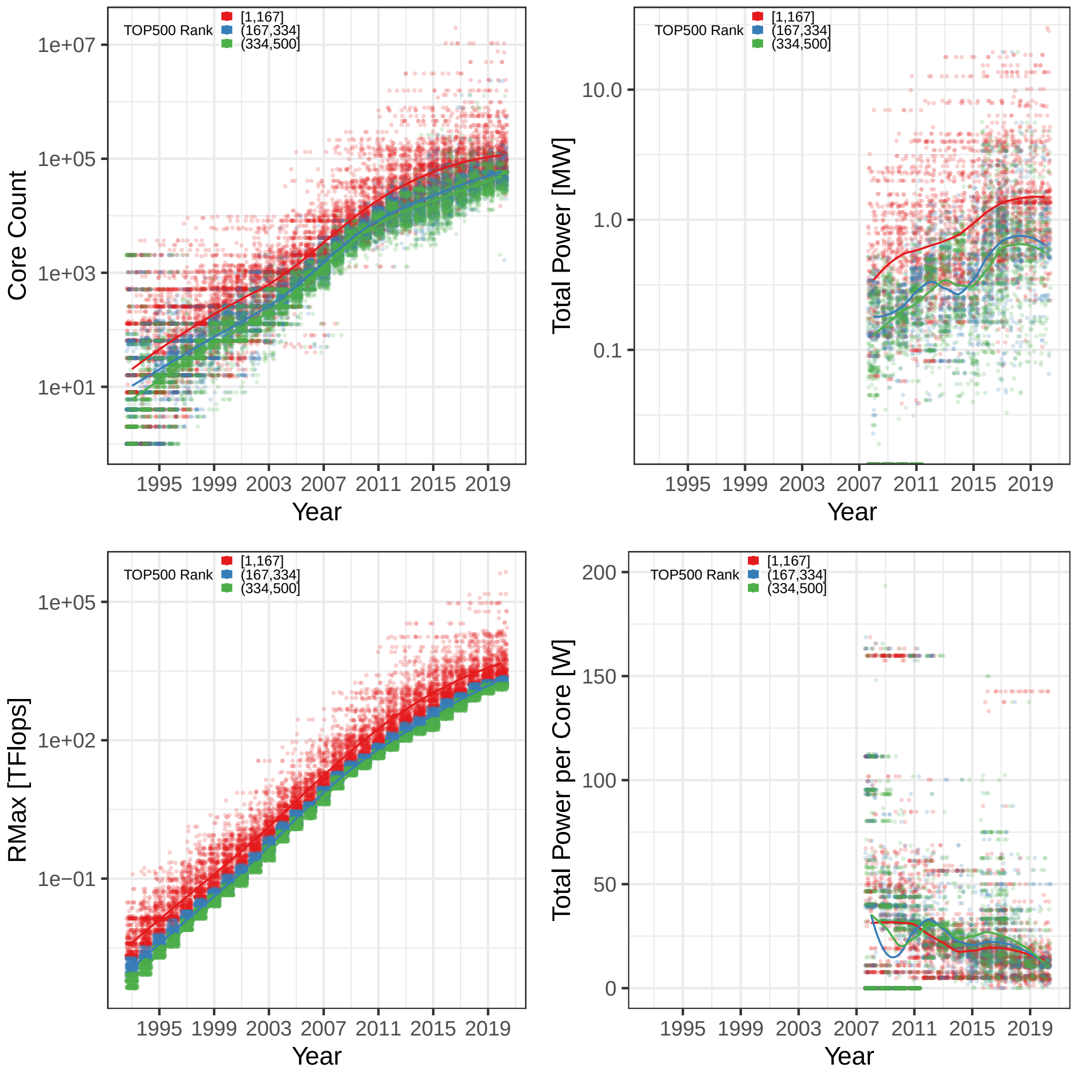


Figure B.4: Top500.org supercomputers development of performance, cores and total power per core from 1993 to 2020.

pack Benchmark[7] performance achieved, RPeak is theoretical peak performance, NMax is the problem size for achieving RMax, where RMax and RPeak values are in GFlops. Although clock frequencies stagnated after 2008 the total processor cores is increasing exponentially with almost the constant rate for last 25 years, and only after 2011 the NMax distribution of 500 biggest supercomputers is broadening, see Figure B.3c. The increase of performance per core in last 25 years is exponential too, due to advancements introduced in the architectures of CPUs.

The power efficiency has been significantly improved in last 10 years, see Figure B.4. Although the total power is raising exponentially (top right) the rate is slower than RMax (bottom left) or count of cores (top left). Also, proportional total supercomputer power per core is reducing in years (bottom right).

[7]The Linpack Benchmark used by top500.org `https://www.top500.org/project/linpack/` is a set of algebra routines widely used in solving dense multidimensional linear equations which is a widely used measure of a system's floating-point computing power. Linpack has been adopted to be executed in CPU and GPU computations (Dongarra, 1987; Dongarra et al, 1993, 2003; Wang et al, 2011; Mohammadi and Bazhirov, 2018) .

B.1.5 Green Computing

According to IEA the entire ICT energy demand accounts for approximately 200–250 TWh, that is around 1% of global electricity consumption (Kamiya, 2021). The electricity sector generation of Carbon dioxide (CO2) emissions varies depending on a source between 25% (EPA, 2021) to 38% (Yoro and Daramola, 2020), but it could be estimated that ICT sector should account to around 0.25% to 0.4% of global CO2 emissions.

The low–carbon generation is becoming a cost competitive target (Lorenczik et al, 2020) and renewable energy sources are being more and more reliable (Infield and Freris, 2020).

The accessible and widely accepted by governments and public opinion trend leads another level of optimisation needs. The cloud computing providers also received this pressure to not only focus on the energy–efficiency (Mytton, 2020), investing in hardware and software that reduces the energy footprint (Berl et al, 2010), but also use of waste heat and implement wide use of renewable energies.

The green computing and green cloud (Liu et al, 2009) are already well established trends shaping the data centres architectures, operations, costing and pricing where environmental perspectives are pushed to the agenda of highest priorities. The same sustainable computing standards have settled in HPC sector as well (Sharma et al, 2006; Feng and Cameron, 2007). The cloud computing data centres globally invest more and more and focus transformation strategies on green computing KPIs such as CO2 footprint per service unit, service quality, resource utilisation (Shehabi et al, 2016; Malmodin and Lundén, 2018; Laganà et al, 2018; Masanet et al, 2020; Xu and Buyya, 2020).

The promise of better resources utilisation and energy conservation by consolidation of servers (Kim et al, 2009; Ismaeel et al, 2018) applications (Srikantaiah et al, 2008), VM allocation (Beloglazov and Buyya, 2010a, 2012a), CPU voltage and frequency scaling (Fan et al, 2007), providing flexible auto–scaling (Wu, 2014b), and finally introducing holistic control of the entire development stack (Baldini et al, 2017; Adzic and Chatley, 2017; Castro et al, 2017; McGrath and Brenner, 2017; Jonas et al, 2019) is very widely adopted in the cloud computing. These efforts also create pressure on vendors to invest in the marketing, encouraging the use of additional services in their ecosystems and education efforts to popularise serverless computing and FaaS models (Shahrad et al, 2020).

B.1.6 From Energy Efficiency and Resource Optimisation to Over–utilised Shared Services and SLA Violations

Under–utilised servers is an obvious source of waste for service providers who could have over invested in too big infrastructure, so both CapEx was too high and required OpEx, i.e. costs of staff for operations and energy[8], is still needed. Of course this can be an intermittent or temporary situation, but in an ideal scenario the machines are fully utilised.

Similarly, over–utilised servers case can be considered as a source of waste too since it is impacting quality of provided service. This balancing act is difficult since many applications are CPU and memory hungry and more and more often this resource is a bottle neck in computing allocation for scalable systems. Moreover, access to disks for I/O intensive applications can cause serious performance impacts when heavy read/write processing is performed leading to persistence layer contention. Thus, the normal response of cloud providers is to offer specialised services under PaaS, FaaS, or SaaS models that allow to optimise the resources usage by dedicated and more predictable characteristic even under unexpected/not–manageable work–load (Lynn et al, 2017; Bhamare et al, 2017; Fazio et al, 2016; Hawilo et al, 2019).

Therefore, servers should utilise maximum capacity that is still allowing meeting throughput and latency requirements given by SLA that clients expect guaranteed computational power and low–latency executions. Unfortunately, as a consequence to the optimisation pressures explained in the first paragraph of the section (escaping from under–utilised resources) that is typically very close the point of saturation[9] which causes huge impact on SLA, as discussed in Chapter 5.

The economical and energy conservation pressures forcing usage of the available resources to the maximum, high inertia of elasticity (auto–scaling) and unpredictability of load are typical forces pushing cloud providers out of the stable operating regions. It is especially visible in IaaS and PaaS models powered by stacks of virtualisation layers. Switching off machines is saving the percentage of costs due to energy conservation, but is introducing the cold start problem and reducing the standby reserve, so raising the inertia of elasticity furthermore. Pushing the responsibility of costs optimisation to the client in IaaS is being challenged by market and balanced by more PaaS and FaaS offers available (Van Eyk et al, 2018).

In not well maintained cloud machines rented to support EIS, a situation where servers are idle or overloaded can be common. As shown on Figure B.5 similar to presented by Beloglazov (2013, p.25) demonstrates wide scenario example of issues related to poor balancing of CPU utilisation distribution of 100 most loaded PlanetLab hosts[10] (Chun et al, 2003; Peterson et al, 2006; Park and Pai, 2006).

To illustrate the problem that is more common to on–premises IaaS for clients but also to service providers of FaaS and serverless computing models let's consider a simple and

[8]It is estimated that even 20% of TCO of a data centre is spent on electricity (Rasmussen, 2011).

[9]Servers that are close of reaching maximum utilisation of a given resource under long–term load are prone of getting in the state of saturation which case load measurements to raise exponentially.

[10]PlanetLab workload traces collected in March and April 2011 `https://github.com/beloglazov/planetlab-workload-traces`

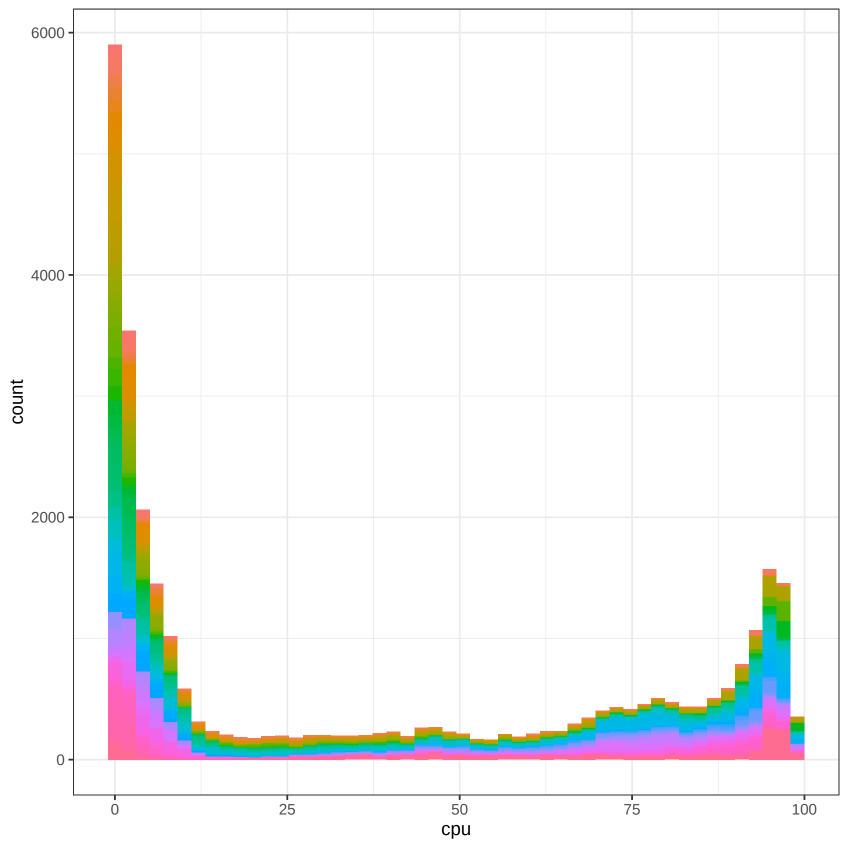


Figure B.5: Example of CPU utilisation histogram of 100 most loaded PlanetLab servers.

idealistic scenario. This is merely approximating observed phenomena for demonstration purposes and not taking into account many underlying processes that will in normal conditions cause impact on the resources utilisation time and obfuscate the results. First assumption is that the load is generated by long–term activity using highly CPU–bound functionality with certain always the same CPU time executing in a capacity constrain system in order to limit example complexity. The model was built following conclusions of earlier experiments explained in Chapter 5 relations between incoming requests, utilisation, system load, and SLA, confirmed in real–application and simulated in test–beds explained in Chapters 6 and 7. See Figures 5.2 7.5 and 8.7 demonstrating relation of activity inputs, CPU and processor queue (load) to SLA. Figure B.6 shows various metrics in function of CPU utilisation[11], such as: quantity of requests (Service Use), power consumption[12], execution times, example of SLA that contains no cost penalty[13] for double execution that are longer than twice as expected execution time, costs of energy, and total financial performance P that contain revenue reduced by CapEx, energy, other OpEx

[11]The figure is presenting normal operating conditions from 0 to 100% utilisation; after CPU is saturated under constant usage the load is quickly increasing exponentially, forcing server to stale condition (late signal to internal scale–up IaaS that powers FaaS layer) which were not presented here.

[12]Each processor will a specific energy consumption characteristic, so it is impossible to give generic factors, but in order to illustrate the general pattern let's assume the energy usage performance in a function of load under the same clock frequency and voltage is well defined by a linear approximation starting from P_{idle} for no use to P_{max} for 100% utilisation (Fan et al, 2007), which varies from 50% to 70% depending on CPU architecture and use–case (Zhang et al, 2013; Ali et al, 2016; Brünink et al, 2011).

[13]The scenario was extended in Section 9.5 where SLA contains cost incurring penalties for service provider.

costs. In colours were depicted operating regions; in green – positive financial performance, in orange – no profit in FaaS conditions, in red – signals to scale up, in dark–red – loss of revenue in a situation of high demand on services.

Quality and performance of cloud services control should focus on widening the region of profitable operation, as highlighted in green on Figure B.6. Ultimately, the main point of the control is to widen the curve of financial performance, P, to left, by scaling–down and switching off not needed resources, offering on–request prices modulating the demand, and, to right, by scaling–up and using non–acquisitive control methods, see Section 9.5. Furthermore, cloud service providers being attracted by the fine grained management of service of machines (IaaS) and VMs (PaaS) shared provisioning drive the industry to incentivised usage of serverless approaches (FaaS) and features exposure (SaaS).

B.1.7 The Software Factor

Lots have been said about how computing resources are offered to the enterprise use i.e. all cloud computing approaches and how computing improved in last 20 years, but many argue more has been done in the area of software.

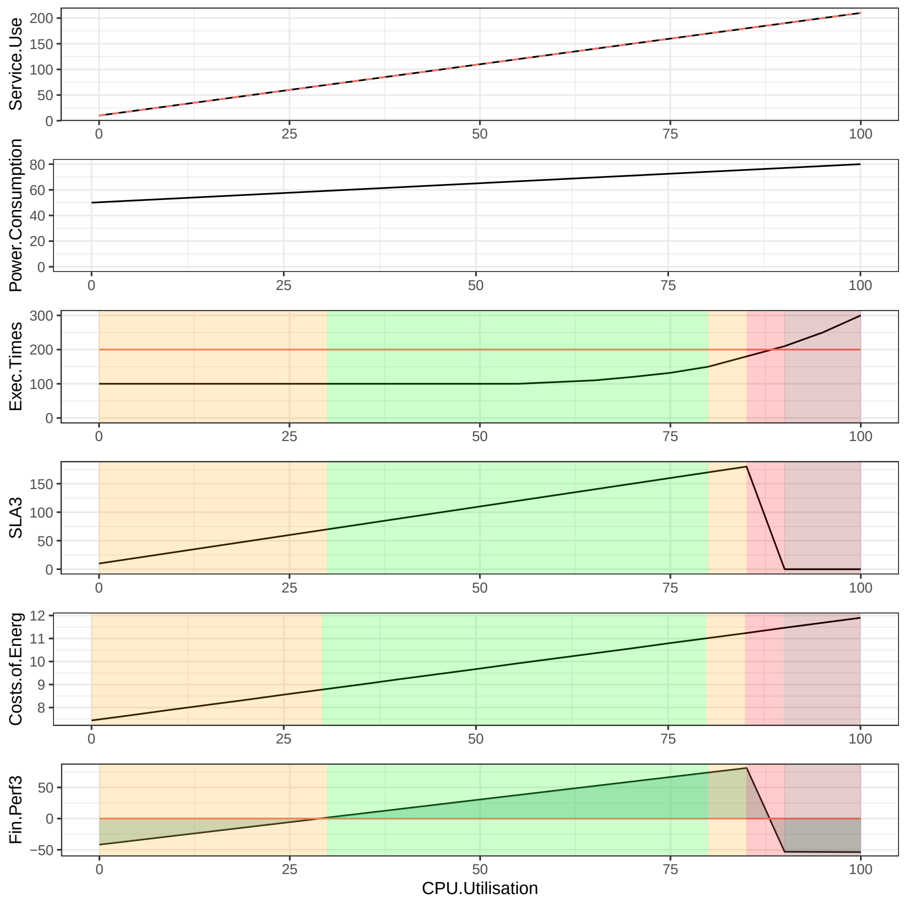


Figure B.6: Theoretical performance of function driven cloud service that is capacity constrain in context of energy costs and SLA.

Despite the view of Wirth (1995), who observed that in the pursue of quick business requirements delivery the software expands quickly to fill available resources and is getting slower more rapidly than hardware is becoming faster, last 20 years is a massive improvement in software development practices; note agile, lean, and scaled teams software development (Cockburn, 2006; Poppendieck and Poppendieck, 2003; Hanoch and Vitouch, 2004).

The progress was also possible due to enormous work of the wider, global community (information society age), building open source software for completely new value of synergy in collaboration, in the era of internet declaring good practices and list of software, organisation and architectural anti–patterns.

Algorithms, software approaches and systems architectures drive further improvements. Software delivered in forms of open source, reusable packages, libraries and services help to speed up further development. Software is also more efficient due to algorithmic advances. Thompson et al (2021) argue that the progress made in algorithms field, is more important than previously realised and sometimes even orders of magnitude more important than hardware improvements offering the raw computational power increase.

Therefore, the software tier of cloud computing service providers but also clients shall be utilised for the control needs further.

Table B.1: Table of “forces” shaping further development of computing, data centres development and their applicability to deliver solutions to enterprise.

	Force name	Response to	Type	Description
⇊	Frequency limits	—	Slowing force	Observed inefficiency of CMOS based transistors power consumption and heat dissipation challenges.
⇈	Open source	Vendor lock–in, Internet, Demand on solutions to common problems	Pushing force	Software development model with source code that is freely available for possible modification and redistribution open new ways of collaboration of software engineers greatly improving the software by sharing the changes within the global community.
⇈	Common algorithms	Demand on solutions to common problems	Pushing force	Advantages of common algorithm progress, may be seen as far more important than hardware (Thompson et al, 2021).
⇈	Multi–tenancy	Demand on software distribution	Pushing force	An approach relaying on a policy introducing segmentation, isolation and governance over service levels, and billing models for different consumer accounts.
⇈	Dynamic provisioning	Demand on computation power	Pushing force	A solution that is allowing further reduction of wasted computing resources through delivery of shared servers capacity to clients for the actual demand.
⇈	Moore’s Law progress	Demand on computation power	Pushing force	The observed phenomena where the number of transistors in a dense integrated circuit doubles every two years greatly extending the scope and performance of computational units.
⇈	Beyond CMOS	Demand on computation power	Slowing force (impact uncertain)	Challenges and uncertainty related to research and development of new technologies beyond the CMOS scaling limits, and further clock scaling trends.
⇈	Edholm’s law progress	Demand on connectivity and distributed systems	Pushing force	Observed development of the bandwidth of telecommunication networks (including the Internet) is doubling every 18 months.
⇈	Koomey’s law progress	Costs of power consumption	Pushing force	Observed the number of computations per joule of energy dissipated doubled about every 18 months.
⇊	Impeded scalable parallelism	Demand on parallel processing	Slowing force	As per Amdahl’s and Gustafson’s law parallel processed may introduce dramatic improvements in throughput of multiprocessor–, multi–core– computing, nevertheless, not all parts of programmes can be truly parallel, sequential parts are significant part of the computing paradigm, see Section B.2.3.
⇈	Parkinson’s law	Demand on computation power	Pushing force	In computing context the law can be generalised further as a statement where the demand upon a resource lead to expand to match the supply of the resource availability. In result we observe, data centres growth, further consolidation and hyperscale. Costs of new data centres will be balanced by demand.
⇊	Energy conservation	Demand on computation power	Slowing force	There is no hard barrier to be faced in the near future, however, with growing focus on energy conservation, CO2 reduction pressures and raising costs of power the green computing should see rapid development but also will be slowing down the pure computational performance increase.
⇊	CO2 emission reductions	Stabilising the climate change, demand on computation power	Slowing force	In order to fasten the transition, rather than rely on national grids migrations, large computation centres will require to invest in renewable energy sources. This requires substantial founding in redevelopment of main sources, backups (largely diesel), energy storage.

B.2 Practice of Software Execution Control Constraints and Requirements

EIS are growing bigger and bigger, with more interactions with other systems, and different deployment approaches. In order to see the whole and build conceptual integrity (Poppendieck and Poppendieck, 2003), as a complex structure of architecture, resources, executing software, development and deployment standards ASM design need to react not only to, as discussed earlier in Section 9.2, technical and market needs, nor organisations requirements explained in Section 9.3, but also to specific constraints to be reviewed in this section.

The economics of data centres, Section B.1.1, which offer cloud computing services, have pushed hardware and software to the limits, Section B.1.6, leaving only very little performance overhead before systems get into saturation. As a result, the risks of impacting execution times grow rapidly.

In order to deliver a stable service at a time of high demand on computational power, cloud providers must implement fast and robust control mechanisms to adapt to changing operating conditions. The control can be enforced manually by operators and automatically by autonomous systems. There are normally two methods for dealing with increased load, namely increasing computational power or releasing the load. That is typically realised either by (a) allocating more machines, which must be available, waiting idle, to deal with such a situation, shifting VMs, redistributing load, or (b) throttling, terminating incoming actions that are less important to the new activity demand pattern.

This section begins from discussion of constraints observed in the current market and technology conditions, moving to the specific ASM–driven requirements considerations of non–acquisitive control methods (see point (b) above).

B.2.1 Software as a Control Actuation Layer

As highlighted in Section B.1.7 there is great value of software advancements often underrated in comparison to hardware improvements of last 20 years.

Application–level control offers several benefits and angels of flexibility such as:

1. faster actuation mechanisms than those that can be applied to infrastructure,
2. insight to run–time details of the application,
3. tightly integrated control based on business driven metrics collection,
4. control can be applied for specific functions of a wider code–base,

5. control over interactions between layers of the application or interactions between systems,

6. control over heterogeneous hardware, platform and function service providers that may introduce fluctuations of random characteristics (Malawski et al, 2017; Figiela et al, 2018) or even expected situations, i.e. cold–starts or call–time–outs in FaaS (Shahrad et al, 2020; Manner et al, 2018),

7. access to input parameters and deeper characteristics of persisted data that can significantly change run–time profile of a functionality.

The approach can be applied by cloud computing vendors to regulate FaaS and event–based software interactions, but also in on–premises model, see Section 9.5.

B.2.2 Software Framework Constraints

There are many different constraints software frameworks exhibit under different cloud model circumstances (Furda et al, 2017; Krancher et al, 2018; Mohanty et al, 2018).

Seeing matters from IaaS and PaaS perspectives, a control software framework needs to be either weaved–in to the application or be part of the platform the application is deployed on. Several PaaS providers that allow deployment of custom EIS but also SaaS applications (Walraven et al, 2014; Gey et al, 2015; Truyen et al, 2016) such as Amazon Elastic Beanstalk[14], Atos apprenda[15], Azure App Service[16], CloudBees[17], Engine Yard[18], Google App Engine[19], Heroku[20], Jelastic[21], Oracle Cloud Platform[22], and PythonAnywhere[23]. There are several open–source projects, such as: Apache Mesos[24], Google Kubernetes[25], CoreOS[26], Red Hat OpenShift[27], and Cloud Foundry Foundation (Apache 2.0)[28] (Kratzke, 2014) supporting such run–time platforms as Java (Spring), Ruby (Rails, Sinatra), JavaScript (Node.js), .NET (.NET Framework, .NET Core Framework), Python, PHP, Go an can be deployed to IaaS of such vendors like Amazon Web Services, Microsoft Azure, IBM Cloud, Google Cloud Platform, VMware's vSphere, OpenStack.

[14]https://aws.amazon.com/elasticbeanstalk
[15]https://apprenda.com
[16]https://docs.microsoft.com/en-us/azure/app-service
[17]https://www.cloudbees.com
[18]https://www.engineyard.com
[19]https://cloud.google.com/appengine
[20]https://www.heroku.com
[21]https://jelastic.com
[22]https://cloud.oracle.com/paas
[23]https://www.pythonanywhere.com
[24]http://mesos.apache.org
[25]https://github.com/GoogleCloudPlatform/kubernetes
[26]http://coreos.com
[27]http://www.openshift.com
[28]http://cloudfoundry.org

Software in order to operate requires a common framework deployed to all parties, especially in case of hybrid clouds, FaaS, and wider event–driven and serverless computing are great areas of future development (Lynn et al, 2017). Although there are many mature FaaS approaches (Leitner et al, 2019), various platforms differ in focus, target domain, assumed model, and architectural decisions (Van Eyk et al, 2018). Since there are clearly observable certain specifics of given vendor solution characteristics of how different cloud providers deliver their function–driven services; this is magnified by no clear standard adoption the market tend to allow on vendor lock–in practices (Opara-Martins et al, 2016) significantly impeding migration efforts (Yussupov et al, 2019). Apart from the closed source FaaS platforms, there are several open–source platforms that have been emerging (Mohanty et al, 2018), including Apache OpenWhisk[29], Fn project[30], Kubeless[31], Funktion[32], Knative[33], Nuclio[34], and OpenLambda[35] (Hendrickson et al, 2016) that are partially addressing the lack of insight and vendor lock–in (Palade et al, 2019), supporting most of the aforementioned platforms and remove migration barriers, but typically require higher start–off investment for adoption.

Reactiveness and rigorous needs of control systems to react on specific execution conditions have been compared with systems load on Figure 9.6. Serverless computing due to its function driven nature is more suitable for short execution times[36], still where requirements around execution times are not critical. Whilst, PaaS and IaaS are more suitable for transactional systems with medium execution times, acceptable for UX, and long executing functionalities, batches–driven processes, like aggregation tasks, indexes rebuild etc. All of those can be areas of non–acquisitive control exploitation, see Section 9.5, such as request termination, admission control, micro–scheduling and batch jobs schedule control. When the requirements for execution time and reactivity are high, costs– or deadline– constrained (Malawski et al, 2015), and termination actuation is not accepted the system must adaptively allocate resources (Ran et al, 2015) often securing a headroom for additional requests avoiding resource congestion and allowing scalability mechanisms to regulate the allocated resources (Thanasias et al, 2016; Madni et al, 2016).

B.2.3 Development Cycle and Adaptive Control Introduction

Meticulous, stepwise refinement of requirements and code (Wirth, 1971) is an absolute essence of modern technologies, languages and methodologies used in software develop-

[29]https://openwhisk.apache.org, https://github.com/apache/openwhisk
[30]https://fnproject.io, https://github.com/fnproject
[31]https://kubeless.io, https://github.com/kubeless/kubeless
[32]https://funktion.fabric8.io, https://github.com/funktionio/funktion
[33]https://knative.dev, https://github.com/knative
[34]https://nuclio.io, https://github.com/nuclio/nuclio
[35]https://github.com/open-lambda/open-lambda
[36]Many FaaS vendors use hard cap on execution times, time–outs happen after typically up to 5–10 minutes of operation, preventing too much computational power being directed to function calls that should be as quick as possible or limiting bad–design issues like infinite loops or long waits for other services

ment. The produced solution is under refactoring, being improved continuously in stages (Fowler, 2018), with rapidly executed full SDLC iterations leading to production deploy (Fowler and Highsmith, 2001; Martin, 2002; Poppendieck and Poppendieck, 2003). Although, software performance must be verified as apart of the regression suite, which is an element of CD pipeline, very often the production system is the only place where certain market, performance and technology characteristics can be tested (Humble and Farley, 2011; Kim et al, 2016; Humble and Kim, 2018).

As explained in Section 9.3.2 the ASM framework needs to provide common platform for application and operators and be aware of orchestrating data collection of released software changes that can have a dramatic effect on the performance, see Figure 9.2.

High–performance code tends to be far more complicated to develop and maintain than code that is sequential (single threaded of low resources footprint and higher execution times). APM and ASM suites can respond to cross–team collaboration and training needs, so programmers can get more productivity and good value solutions being deployed, whilst DevOps would track the deployment and execution standards. Those standard practices can turn into Monitoring Driven Development (MDD) (Ford et al, 2017).

The hardware performance improvement is tremendous in last years, see Section B.1.2, offering high computational power distributed to many cores, so the potential throughput can be huge, but software design need to adapt to take advantages of the parallel style of execution, using reactive, event–driven and function oriented calls. Converting "embarrassingly parallel" (Shi et al, 2016) or adopting the problem solution to be used under high parallel scenarios (Wilkinson, 2006) regardless of cloud model, from IaaS via PaaS to FaaS, requires different development mindset, but also methods and tools (Fox et al, 2017a).

The ASM suites should be present and support development team interactions applying shift–left testing (Hutchison, 2013) and TDD principles of the SDLC (Zimmerer, 2018). Software performance oriented development requires different testing approaches and discipline in checking latency and throughput acceptance criteria in CI pipelines and monitoring of the features performance. Executed enough number of times and collecting enough data to achieve statistical significance, see Chapter 5. ASM suite should allow wider management of the performance testing leading to a release. Also, application–level adaptive controllers should be tested under such conditions too.

Furthermore, developers will face obstacles related to Amdahl's and Gustafson's laws, see Table B.1 preventing them to achieve possibly overestimated expectations (Hennessy and Patterson, 2011, p. 55). The ASM suite should allow to demonstrate the gains and assess the value delivered to the business, and support the re–negotiations of the given acceptable criteria and production–level SLA. The performance goals and intensity of usage can be converted to economical factors, incorporated into SLA, costs C and financial performance P optimisation as a target of the wider team, see Sections 9.3.3–

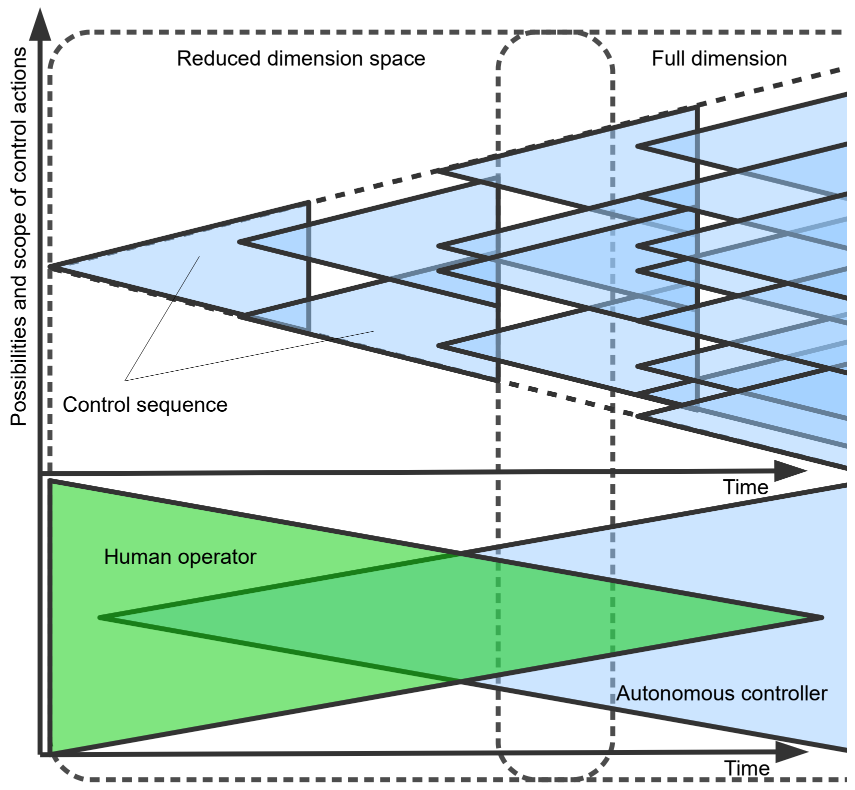


Figure B.7: Sequence of joint adaptive control of human and autonomous controllers.

9.3.5.

B.2.4 Sequential Run–time Evidence Gathering of Technology Execution and Business Domain

As depicted on Figure 9.1, the technology is executing in a business domain realm and ASM controllers operate accordingly to the market conditions. Even though, the treatments evidence gathering and model building is performed on live systems, the ASM suite should allow on integrated test runs in specifically prepared environments that are as similar to live systems as possible that can help the controller to uncover runtime characteristics of the system under synthetic load, preparing to release of a new version as a result of a single sequence of SDLC or a wider PDLC, see Figure 9.2.

Figure B.7 shows the progress flow of joint adaptive control also lead by human operator and autonomous control agents activities in a context of base setup, interactions, and further control. Each of the triangles (cones) represent an independent improvement cycle, which originates at the first control action and extends while the system and operators observe its effects. The whole situation described by all applicable dimensions are monitored specifically from the perspective of expected outcomes (targets). Widening cones in time demonstrate growing size of possible consequences of control actions, enlarging the search space, but on the other hand, constraining the positive and negative control system states changes S for declared operating areas and SLAs. As discussed in Section 5.5

steps of a control sequence consisting such activities as: (1) isolated run, (2) normal work, (3) dimensions selection, and (4) controlled run, should be present and executed at each of the improvement cycles. Of course, DevOps and AIOps teams may decide on other steps that better reflect the needs of specific company environment or EIS, i.e. Prasad and Rich (2018) mention 4 steps: acquire, aggregate, analyse, and act that are quite similar to more general Plan–Do—Check—Act (PDCA) cycle of continuous improvement introduced by Deming (1950) (Moen and Norman, 2006) supporting RCA of incidents (Li et al, 2000; Danovaro et al, 2008) and manual causal–chain search adapted from works of Ishikawa (1985), or kaizen approach explained by Masaaki (1986) adopted to complex manufacturing processes (Ohno, 1988).

Since the control action is an attempt of optimisation driven improvement of a dynamical system, that can be categorised and treated as an independent experiment. Further corrections, interventions, re–tests, and ensurement in a line of this experiment should be taken with extra consideration of additional effects. Other improvement cycles executed at the same time under the dimensions that are part of the active experiment should be flagged and negotiated accordingly to limit unintended interference of interconnected reactions in the complex EIS environment.

In the longer PDLC there are two phases (a) with reduced dimension space, where only key dimensions are taken into account for control which are setup arbitrarily by system operators from all collected by monitoring metrics, and (b) with wider system dimensions, where the controller containing greater adjusted models of system reactions can extend the awareness of deeper dynamics in the system. The frequency of control can grow allowing the adaptive controller to takeover gross of the activities from the human operator, who still should support the incremental model building in normal IML cycles and carry on the supervisory functions, as illustrated at the bottom of the diagram on Figure B.7. The longer the product is treated under ASM the wider data set and better fitting model can be established. The specific guidelines on the time of collecting enough data to achieve statistical significance or measuring strength of causality should be given by the ASM suite, see Chapter 5, however due to domain–expertise, intuition and experience, the final decisions should be taken by operators.

A new version release can bring significant changes to the run–time dynamics, thus, should be considered as an another improvement cycle, see Figure 9.2, where all the steps are just final micro–level sequences of the SDLC as depicted on Figure 9.5.

Note, the model of methodology is applicable to all of the non–acquisitive control methods described in Section 9.5. The wider picture, showing PDLC, of the sequence of joined human and autonomous control in a context of development activities is shown in Figure 9.2.

B.2.5 Growth Hacking and Data–driven Prioritisation for Optimisation and Development Actions

Since the global objective is very aligned with cost reduction C and financial optimisation P, it is impossible to undervalue growth hacking discipline (Reis, 2011; Ellis and Brown, 2017) as an integration field in the EIS development. Low prices, good support of stability under governed SLA build better market sentiment, customers satisfaction and new business growth opportunities. ASM suite must integrate means and methods like: A/B testing, in the experiments executed in the systems. Faster feedback loop, between the business, marketing and engineering teams working together, monitoring, and shared experiments metadata should be available to support better synergy building and quicker ideation of new approaches of raising conversions.

Additional sources of data (Troisi et al, 2020) supporting this kind of experiments should require data–layers embedded into such areas as: EIS DB centred analytics, web– and mobile– based analytics, social media and other economic analytics. Furthermore, FaaS and A/B testing are supporting each other very well together due to low starting costs (Adzic and Chatley, 2017). Thus, integration with internal data storage of applications, corporate data–warehouses and data–lakes, CRM, and other SEO, Conversion Rate Optimisation (CRO), conversion optimisation tools Optimizely[37], VWO[38], Hackle[39], Plerdy[40], user behaviour tracking Smarlook[41], HotJar[42], MouseFlow[43], and web analytics tools like Google Optimize[44], Google Analytics[45], Matomo (Piwik)[46] are highly desired.

B.2.6 Application Operations and Service Management Systems

ASM suites work within requirements of wider Enterprise Service Management (ESM) systems. Development, application support and operation teams cooperate following support and management frameworks of such general models as ITSM, IT Operations Management (ITOM), and Information Technology Infrastructure Library (ITIL) that define set of policies, processes and procedures for managing the services around customer needs (Wulf et al, 2015). Joint by deep integrations (Matchett et al, 2017) with ASM suites data–centre asset management tools of Data Centre Infrastructure Management (DCIM) and IT Asset Management (ITAM) class systems provide holistic view and control offering higher value of synergy built between development, DevOps, AIOps activities.

[37] https://www.optimizely.com
[38] https://vwo.com
[39] https://www.hackle.io
[40] https://www.plerdy.com
[41] https://www.smartlook.com
[42] https://www.hotjar.com
[43] https://mouseflow.com
[44] https://optimize.google.com
[45] https://analytics.google.com
[46] https://matomo.org

Selected popular ITSM frameworks are: (a) ITIL 4 – a framework of best practices for delivering IT services (Kneller, 2010; Agutter, 2020), (b) COBIT 5 (Control Objectives for information and Related Technologies): an IT governance framework (Mangalaraj et al, 2014; Bernroider and Ivanov, 2011), (c) FitSM – a family of standards for lightweight, streamlined service management framework[47] (Mora et al, 2021; Michael et al, 2019), (d) ISO/IEC 20000 – is the international standard for IT service management and delivery (Van Bon and Van Selm, 2008; Van Bon and Clifford, 2008; Pardo et al, 2016), (e) Six Sigma – framework introduced by Motorola as a data analysis driven framework to minimise product and service flaws (Kwak and Anbari, 2006; Antony, 2006; Schroeder et al, 2008; Pyzdek and Keller, 2014; Raisinghani et al, 2005), (f) Microsoft Operations Framework (MOF): a compilation of 23 documents that guide businesses through the entire life–cycle of an IT service, including creation, implementation and cost-effective management with an emphasis on Microsoft technologies (Pultorak and Henry, 2008), (g) The Open Group Architecture Framework (TOGAF) – created and managed by The Open Group as a way to provide businesses with structure when implementing technology with a focus on software, (h) Business Process Framework (eTOM): a framework designed for telecommunications service providers (Huang, 2005). Examples of ITSM systems: (a) SolarWinds Service Desk (SSD)[48], (b) ServiceNow IT Service Management[49], (c) Atlasian Jira Service Management[50], (d) TOPdesk Enterprise Service Management[51], (e) BMC Helix[52] and Control–M[53] for holistic SaaS–based application workflow orchestration, (f) AxiosSystems[54], (g) EasyVista[55], (f) Ivanti Service Manager[56] for the end clients service management, (g) IBM Control Desk[57] for best practice-based service desk capabilities, IBM Workload Automation[58] to automate complex workloads, and IBM Cloud Pak for Watson AIOps[59].

B.2.7 Application Deployment and Configuration Management Systems

DevOps orchestrate deployments using cloud and general purpose deployment systems offering many levels of automation capabilities and configuration management.

[47]FitSM was delivered by a project funded in the 7th Framework Program for Research and Technological Development by the European Union, https://www.fitsm.eu, thus it is aligned with ISO/IEC 20000 standard

[48]https://www.solarwinds.com/service-desk

[49]https://www.servicenow.com/products/itsm.html

[50]https://www.atlassian.com/software/jira/service-management

[51]https://www.topdesk.com/en/enterprise-service-management-software

[52]https://www.bmc.com/it-solutions/bmc-helix-operations-management.html

[53]https://www.bmc.com/it-solutions/control-m.html

[54]https://www.axiossystems.com/it-service-management-product-overview

[55]https://www.easyvista.com/products/cloud-based-it-service-management-software

[56]https://www.ivanti.com/products/ivanti-neurons-itsm

[57]https://www.ibm.com/products/it-service-management

[58]https://www.ibm.com/products/workload-automation

[59]https://www.ibm.com/cloud/cloud-pak-for-watson-aiops

The tool–chain should support all DevOps and AIOps stages of activities under SDLC: (1) "define and plan" – declare production and business metrics, feedback requirements, plan release plan, timing and business support and security policy compliance, (2) "create" – supporting development activities of designing, building, coding, controlling version and configuring under SDLC, (3) "verify" – ensuring the quality of the software release; activities designed to ensure code quality is maintained and the highest quality is deployed to production, (4) "package and release" – shipping the ready for deploy release, release coordination to staging and live system rollout, (5) "configure" – maintaining the software while it is deployed, for additional infrastructure provisioning and configuration activities (6) "monitor" – observing the software, infrastructure and organisation to identify specific issues of development team releases to conclude on the impact to the clients,

It is important to list several tools following Infrastructure as Code (IaC) automation approach that are primarily open–source and can find use in orchestrating hybrid–cloud infrastructure setups mainly under IaaS like: Chef[60], Puppet[61], Cfengine[62], Quattor[63], Juju[64] Salt[65], allowing event–driven automation, remote task execution and configuration management, or agentless Ansible (Red Hat)[66].

Systems for performance and capacity management such as TeamQuest[67], Red Hat Satellite[68], Oracle Enterprise Manager[69], Microsoft System Center Configuration Manager[70] or ZENworksa[71] suite of products like Service Desk, Asset Management, Configuration Management, Endpoint Security Management help the organisations to control the distributed IT assets.

As far as the package, release and deploy of new versions is concerned under CI and CD there are open–source tools Jenkins[72], Buddy[73], or CruiseControl [74] to name few only, but there are also proprietary delivered in a SaaS model tools such as Octopus Deploy[75], AWS CodeDeploy[76], Azure DevOps Services[77], CircleCI[78] and excellent TeamCity[79].

[60]`https://www.chef.io`, `https://github.com/chef`

[61]`https://puppet.com`, `https://github.com/puppetlabs/puppet`

[62]`https://cfengine.com`, `https://github.com/cfengine`

[63]`https://www.quattor.org`

[64]`https://github.com/juju/juju`, `https://juju.is`

[65]`https://saltproject.io`, `https://github.com/saltstack/salt`

[66]`https://www.ansible.com`

[67]`https://www.helpsystems.com/product-lines/teamquest`

[68]`https://access.redhat.com/products/red-hat-satellite`

[69]`https://www.oracle.com/enterprise-manager`

[70]`https://docs.microsoft.com/en-us/mem/configmgr`

[71]`https://www.microfocus.com/en-us/portfolio/zenworks-suite/overview`

[72]`https://www.jenkins.io`, `https://github.com/jenkinsci`

[73]`https://buddy.works/`, `https://github.com/buddy-works`

[74]`https://cruisecontrol.sourceforge.net`

[75]`https://octopus.com`

[76]`https://aws.amazon.com/codedeploy`

[77]`https://azure.microsoft.com/en-us/services/devops`

[78]`https://circleci.com`

[79]`https://www.jetbrains.com/teamcity`

Appendix C

Glossary of Terms, Abbreviations, and Acronyms

The following appendix defines an index of symbols, terms, abbreviations and acronyms, but also a glossary of terminology used in this Thesis.

List of Symbols

The next list describes several symbols that will be later used within the body of the document

$\mathbf{a}$	Vector of actions
$\mathbf{C}$	System model
$\mathbf{P}$	Vector of input parameters
$\mathbf{r}$	Vector of resources
$\mathbf{S}$	System states
a	Action
$a(t)$	Action metrics values in function of time
a_{term}	Action terminated
f_{SLA}	Service Level Agreement (SLA) function
$f_{SLA}(t)$	SLA values in function of time
r	Resource
$r(t)$	Resource values in function of time
t	Time
Financial	
C	Cost
P	Financial performance
R	Revenue
Number Sets	
$\mathbb{R}$	Real Numbers

Acronyms

AIOps Artificial Intelligence for IT Operations. 18, 20, 224, 225, 229, 254, 255, 294, 295, 297

AMQP Advanced Message Queuing Protocol. 243, 244

AOP Aspect–Oriented Programming. 64, 65, 135, 170, 171, 233–235, 237, 238, 241, 259

API Application Programming Interface. 12, 17, 52, 56, 58, 62, 64–66, 68, 97, 170, 174, 199, 235, 237, 242, 245, 246, 250–252, 264

APM Application Performance Management. 18–20, 26, 38, 54, 56, 87, 95, 129, 228, 229, 266, 292

ARM Application Response Measurement. 19

ASM Application Service Management. 2, 4, 5, 8–10, 13, 15–20, 24–30, 32, 36, 38, 40, 43–48, 52–54, 58, 60, 64, 66, 68, 71–73, 85–87, 89, 90, 93, 95–99, 102, 108, 114, 121, 125–130, 154, 156, 159, 195, 196, 198, 208, 214, 217, 220, 223–232, 236, 239, 241, 242, 245, 247, 249–251, 253, 254, 257–261, 263, 265–269, 271–273, 289, 292–295

ASMaaS Application Service Management as a Service. 266

ASM-SD Application Service Management Signal Deconvolution. 51, 86, 87, 89, 96, 101, 105, 107–112, 116, 118, 121, 125, 126, 157, 199–201, 203, 259, 262

AWS Amazon Web Services. 57, 222, 244

AWS EC2 Amazon Web Services Elastic Computing Cloud. 34, 222

BDD Behaviour Driven Development. 39

BPF Berkeley Packet Filter. 57

BPMN Business Process Model and Notation. 219

BSaaS Batch–Scheduler–as–a–Service. 251

BSS Blind Signal Separation. 86, 98, 102, 125

CaaS Container–as–a–Service. 241

CAC Call Admission Control. 65, 235, 239, 249

CAM Cloud Application Management. 17

CAMP Cloud Application Management for Platforms. 17, 266

CapEx Capital Expenditure. 61, 265, 283, 285

CD Continuous Delivery. 12, 39, 40, 73, 219, 220, 247, 292, 297

CI Continuous Integration. 12, 39, 73, 219, 292, 297

CLI Command Line Interface. 235, 246, 247, 251

CMS Content Management System. 56

CoAP Constrained Application Protocol. 264

CRL Causal Reinforcement Learning. 262, 263

CRM Customer Relationship Management. 35, 250, 295

CRO Conversion Rate Optimisation. 295

CSS Cubic Smoothing Spline. 75

CWV Core Web Vitals. 12, 60

DAG Directed Acyclic Graph. 33

DAO Data Access Objects. 235, 237, 238, 245, 246

DB Database. 56, 66, 245, 246, 250, 277, 295

DBA Database Administration. 245

DCIM Data Centre Infrastructure Management. 295

DDoS Distributed Denial–of–Service. 249

DE Differential Evolution. 116, 126

DES Discrete–Event Simulation. 163, 194, 212

DESMO-J Discrete–Event Simulation Modelling in Java. 108, 162, 163, 212, 213

DL Data–lake. 250

DML Data Manipulation Language. 245

DTW Dynamic Time Warping. 72, 75, 78

DW Data–warehouse. 250

EIP Enterprise Integration Patterns. 238, 243, 244

EIS Enterprise Information System. 1–3, 33, 40, 57, 58, 61, 67–70, 95, 201, 212, 214, 218, 220, 223, 227–230, 232, 234, 236, 237, 239–243, 246, 250, 251, 253, 254, 257–259, 263, 264, 266, 269, 275–277, 280, 284, 289, 290, 294, 295

ERD Entity Relation Diagram. 219

ESB Enterprise Service Bus. 12, 56, 208, 243, 244

ESM Enterprise Service Management. 295

ETL Extract–Transform–Load. 250

FaaS Function–as–a–Service. 5, 24, 35, 36, 40, 47, 57–59, 62, 65–67, 96, 127, 128, 156, 157, 159–161, 165, 166, 174, 175, 184, 208, 212, 219, 232, 233, 235–237, 239, 241, 242, 244, 247, 249, 251, 254, 259, 260, 264, 265, 283–286, 290–292, 295

FE Front–end. 239

GA Genetic Algorithm. 114, 126, 198, 203, 205, 208, 211, 212, 258, 273

GAE Google App Engine. 222, 244

GCE Google Compute Engine. 222

GOF Gang of Four. 65, 238

HPC High Performance Computing. 248, 280, 283

IaaS Infrastructure–as–a–Service. 34–36, 40, 60, 156, 160, 219, 222, 233, 235–237, 241, 247, 251, 255, 284–286, 290–292, 297

IaC Infrastructure as Code. 63, 297

ICT Information and Communications Technology. 221, 222, 279, 283

IEC Interactive Evolutionary Computation. 225

ILP Inductive Logic Programming. 130

IM Instant Messaging. 230

IML Interactive Machine Learning. 224, 294

IoC Inversion of Control. 238

IoT Internet of Things. 218, 230, 278

ITAM IT Asset Management. 295

ITIL Information Technology Infrastructure Library. 295, 296

ITOA IT Operations Analytics. 18

ITOM IT Operations Management. 295

ITSM Information Technology Service Management. 230, 266, 295, 296

IUI Intelligent User Interface. 224

JMS Java Message Service. 12, 131, 137, 243, 244

JMX Java Management Extensions. 12, 57, 134

JVM Java Virtual Machine. 12, 121, 134, 242

KPI Key Performance Indicator. 2, 56, 58, 219, 265, 283

LMF Linear Models Fitting. 102

LOESS Locally Estimated Scatterplot Smoothing. 149, 151, 153

MDD Monitoring Driven Development. 292

MIMO Multi–Input Multi–Output. 27, 47

ML Machine Learning. 258

MLOps Machine Learning Operations. 18

MLP Multi Layer Perceptron. 142, 170

MOO Multi Objective Optimisation. 154, 171, 226

MQ Message Queue. 56, 57, 64, 131, 194, 243, 244

MQTT MQ Telemetry Transport. 243

MVC Model–view–controller. 239

NFR Non–functional Requirement. 3, 6, 11–13, 15, 22, 28, 34, 195, 200, 214, 218, 220, 228, 232, 238–240

NLP Non–linear programming. 102, 199

NN Neural Network. 9, 26, 27, 29, 30, 33, 87, 128, 131, 136, 137, 141–143, 170, 173, 193, 194, 236, 258, 268, 271

NoSQL Non–SQL, Non–Relational Database Management System. 12, 235, 245, 246

OLAP On–Line Analytical Processing. 60

OLTP On–Line Transaction Processing. 277

OOP Object–Oriented Programming. 65, 237, 239

OpEx Operating Expense. 61, 265, 283, 285

ORM Object-–relational mapping. 235, 237, 238, 245, 246, 277

OS Operating System. 21, 57, 101, 121, 134, 233, 245, 246, 251

PaaS Platform–as–a–Service. 17, 35, 38, 47, 49, 57–60, 62, 65, 66, 128, 156, 159–161, 167, 173–175, 184, 212, 219, 233, 235–237, 239, 241, 242, 244, 245, 247, 251, 254, 255, 260, 264, 284, 286, 290–292

PDCA Plan–Do—Check—Act. 294

PDLC Project Delivery Life–Cycle. 12, 87, 220, 229, 293, 294

POJO Pure–Old Java Objects. 109

PSO Particle Swarm Optimisation. 114, 115, 126, 198, 203, 205, 208, 210, 212, 258, 273

QoS Quality of Service. 3, 5, 10, 17, 26, 50, 58, 61, 99, 158, 213, 259

RCA Root–Cause Analysis. 18, 220, 294

RCT Randomised Controlled Trial. 87, 262

RDBMS Relational Database Management System. 12, 50, 131, 234, 235, 245, 250

RFC Request for Comments. 264

RL Reinforcement Learning. 87, 130, 168, 195, 262, 263

RoI Return of Investment. 219, 275

SaaS Software–as–a–Service. 5, 16, 35, 38, 47, 49, 58, 59, 65–67, 96, 127, 156, 159–162, 165, 166, 173–175, 184, 212, 219, 233, 235–237, 239, 241, 244, 247, 251, 255, 259, 260, 266, 284, 286, 290, 296, 297

SANN Simulated Annealing. 198, 203, 205, 208, 209, 212, 273

SBSS Semi–Blind Signal Separation. 86, 98

SCIC Similarity, Consequence, Interference, and Clarity. 10, 68, 69, 74, 77–79, 82, 93, 259, 261, 266, 267, 273

SDCIC Similarity, Dependence, Consequence, Interference, and Clarity. 10, 51, 77, 83–85, 87, 89, 93, 157, 199, 203, 259, 261, 262, 273

SDK Software Development Kit. 249

SDLC Software Delivery Life–Cycle. 11–13, 39, 87, 220, 224, 228, 229, 275, 292–294, 297

SEO Search Engine Optimisation. 12, 295

SLA Service Level Agreement. 3–6, 9, 10, 12, 13, 15–20, 23, 25–28, 32, 33, 36–40, 43, 44, 47–56, 58–61, 64, 65, 68, 69, 75, 89, 93, 96, 99, 101, 128, 129, 131, 132, 137, 140–142, 146, 148, 154, 156, 157, 160–165, 168, 170–173, 175, 181, 182, 193, 195, 196, 198, 199, 202–204, 208, 212–214, 218, 219, 224, 232–234, 236, 238, 241, 243, 247, 248, 250–253, 255, 257–259, 262–266, 271, 272, 274, 277, 279, 284, 285, 292, 293, 295

SMA Simple Moving Average. 75, 76, 100, 104, 199

SME Small and Medium–sized Enterprises. 222

SOA Service Oriented Architecture. 2, 34, 218, 219, 265, 276

SoC Separation of Concerns. 219, 237

SOMF Service–oriented modelling framework. 219

SQL Structured Query Language. 132, 245

SQS Amazon Simple Queue Service. 244

SysML Systems Modelling Language. 219

TaT Turnaround Time. 240, 246, 252

TCO Total Cost of Ownership. 247, 254, 269, 275, 279, 284

TDD Test Driven Development. 12, 39, 292

TnC Terms and Conditions. 253, 265

ToS Terms of Service. 253

TTL Time–to–Live. 243

UAT User Acceptance Testing. 73

UI User Interface. 12, 17, 39, 57, 58, 64, 199, 227, 235, 237, 249, 251, 253

UML Unified Modelling Language. 219

UX User eXperience. 12, 17, 64, 249, 291

VM Virtual Machine. 21, 24, 27, 231, 242, 247, 263, 283, 286, 289

WAF Web Application Firewalls. 249

WS Web Service. 12, 17, 58, 235, 237

Glossary

adaptive control is such a control method that must adapt to a controlled system with parameters which vary in time during the normal work or are uncertain initially. Adaptive control does not require declared a priori information about the bounds on these time–varying or uncertain parameters. Adaptive control theory allows to deal with instabilities caused by system responses variations in a way where an adaptive controller can modify its behaviour in response to changes in the dynamics of the process under control and the character of the disturbances with adjustable parameters, but also with mechanisms to adjust the parameters. . 8, 10, 16, 21, 27, 28, 38, 40, 52, 54, 127, 128, 146, 155, 157, 159, 163, 225, 237, 254, 257–259, 262, 264–266, 268, 274, 293

admission control is a validation process applied mainly to communication but used in computing systems as well where a resource availability check is performed before a connection or request is established or serviced to verify if present resources are sufficient for the proposed request for connection, transfer or computation service, preventing over–subscription and over–utilisation.. 65, 237, 238, 240

application architecture describes how the application assets are deployed, interact with each other, form interfaces and communicate with the external parties like other applications or end–users. Architecture declares the composition of software libraries, modules and components that delivers expected set of behaviours, functionalities and non–functional requirements following established good–practices, software principles of well–structured systems and design patterns used appropriately to the problem the application is designed to solve.. 34

application server is a software that is deployed on the operating system and creates communications to external entities over network providing foundations to distributed systems and service oriented architectures to clients, users and other systems via leading protocols like HTTP creating APIs for SOAP or REST and web clients like browsers.. 65, 234, 239

autonomous control is a process that involves minimal external human assistance even under significant uncertainties in the environment for extended period of time, with a high degree of autonomy, even with the power of self governance, and should exhibit adaptation and abilities to compensate for failures without the need of external intervention. . 19, 26, 32, 49–51, 54, 58, 68, 73, 74, 93, 97, 126, 128, 213, 223, 225, 227, 234, 241, 257, 261, 267, 268, 273, 293, 294

C4 is a modelling standard as a graphical notation technique describing the architecture of software systems addressing four viewpoint levels of different abstraction of diagrams: context, container, component and code.. 219

causal inference aims to answer causal questions as opposed to just statistical ones through a process of determining the independent, actual effect of a particular phenomenon by allowing intervention over graph model mapping acyclic directions between cause and effect entities.. 28, 86, 125, 262, 263, 267

cloud computing is a system capability offering availability on–demand of computer resources, especially data storage and computing power, without direct active management by the user. The term is generally used to describe data centres available to many users over the Internet.. 1, 16, 17, 20, 23, 27, 34–36, 40, 50, 58, 62, 65, 67, 96, 97, 101, 128, 155–158, 160, 166, 195, 198, 218, 219, 222, 231, 232, 234, 235, 241, 245–247, 251, 257–259, 263, 269, 275, 278–280, 283, 286, 287, 289, 290

containerisation is a technique of packaging software with list of defined operating system dependencies and libraries allowing to execute the code in a lightweight run–time (container) allowing greater portability and reproducability of the setup on other infrastructures than originally developed. These properties allow development teams to work faster and more securely delivering increments between local, development, staging and production environments in fully automatic pipelines working on many platforms.. 37, 40, 63, 218, 233, 247

DevOps is a software development team role, or a set of practices, where information–technology operations (Ops–) closely support software development (Dev–) to achieve better stability of the delivery by applying techniques like continuous integration and delivery providing higher software quality and shorter systems development life cycle.. 15, 18, 19, 40, 224–227, 229, 254, 269, 292, 294–297

fog computing (also fog networking) is an architecture paradigm where focus is on building decentralised systems infrastructure allowing to perform substantial amount of computation, processing of data on many devices deployed somewhere between the data sources (outer edges), aggregating and storing data, communications routed over the Internet via gateways and the cloud services and other deployed artefacts. The business logic and computing power are close to devices and systems in the control of end–users and so highly distributed across variety of components, functions, endpoint collections and hosts.. 218, 230, 278

GRASP is an abbreviation of collection of General Responsibility Assignment Software Patterns wrapping nine fundamental principles of Creator, Indirection, Controller, Information expert, Low coupling, Polymorphism, High cohesion, Pure fabrication, Protected variations.. 238

green computing is a set of concepts allowing to build and optimise computing systems in order to minimise the environmental impact following guidelines lowering resources and power usage thanks to further advances of power management, virtualisation, containerisation and cloud computing adoption. Effectively the discipline focuses on maximising the efficiency of the required computation as a function of energy consumed.. 250, 257, 283, 288

grid computing is a design principle allowing to form an extensive possibly distributed computing systems leveraging multiple servers connected by network to accomplish large computation tasks in a coherent way interacting with each other.. 198

growth hacking is a process of seeking new opportunities and improving performance of organisation with rigorous, methodical experimentation and variants testing with use of new versions of software, new products, alternative means of advertisements and other ways of client contact.. 295

Java is a software offering a general–purpose programming language that is a high–level object–oriented with a design focusing on minimising scope of implementation dependencies and running on run–time allowing platform independence of the compiled code that can run without the need for recompilation on all supported platforms.. 131, 135, 137, 163, 205, 242

load balancer is a software component or device that distributes requests traffic to number of applications, nodes or servers. Primarily the component is used to introduce horizontal scalability and increase capacity for concurrent requests and reliability by redundancy of the systems.. 65, 241

microservices is an architectural style where the focus is on forming the software f losely couples components by breaking down into services that are independently deployable, that interact using mechanisms such as a web service request or remote procedure call, instantiated in separate run–times.. 23, 34, 37, 38, 40, 50, 59, 218

monetisation is a process of converting or establishing something into value given in money, or adapting non–revenue–generating assets to generate revenue.. 35, 36, 38, 51, 156, 181, 184

.Net is a software framework developed by Microsoft that includes built–in class library and offer multi programming language support. Applications developed in the framework execute in the virtual software environment named Common Language Runtime (CLR) that translates managed code into the host platform instructions.. 242

Node.js is a software platform delivering JavaScript everywhere paradigm that offers back–end and cross–platform runtime environment allowing developers to code in JavaScript on server–side to produce dynamic web page content before the response on a request is sent to the end users' web browser. The framework is open source and heavily emphasises event–driven architectures delivering asynchronous processing and I/O in order to optimise throughput and scalability.. 242

non–acquisitive control is a group of control methods that provide service adaptation by reducing the need for computation power at times where the given service is impacted by higher than anticipated load. 25, 33, 59, 61, 63, 96, 98, 155, 195, 214, 217, 220, 223, 232, 238, 239, 242, 246, 247, 251–253, 257, 258, 263–265, 268, 269, 286, 289, 291, 294

refactoring is an activity leading to better design in a disciplined restructuring of the existing code throughout a series of small but continuous transformations reducing a risk of wrong implantation.. 13, 292

scalability is an ability for a heterogeneous and/or distributed system to accommodate more or less load adaptively by adding or removing nodes or resources allocated to support the required features.. 3, 11, 22

serverless computing is an architecture paradigm that introduced ability of service provider to allocate sufficient computing resources on demand without a need of setting up specifics of machines seen from client perspective. The approach significantly reduces simplicity of a hosting layer of the system on customer side and speeds up development and deployment of further requested changes.. 16, 17, 24, 34, 35, 40, 49, 59, 62, 157, 164, 165, 167, 219, 248, 254, 255, 264, 283, 284, 291

SOLID is a collection of five core principles in object design such as: Single responsibility, Open–closed, Liskov substitution, Interface segregation, Dependency inversion.. 238

type–1 hypervisor is a software that runs virtual machines on a host machine to control the hardware and to manage the operating system, thus, called bare–metal hypervisors.. 63

virtualisation is a software method allowing to logically divide the system resources to spaces called virtual machines that act like a real computer with an operating system.. 35, 40, 63, 101, 175, 218, 233, 247

web container is a software component of a web server that interacts with code responding to HTTP requests (i.e. Java servlets). Since the web container manages the life cycle of requests processing and hosts code produced responses in order to decode the specific class/function it maps a URL to a particular asset validating access rights of the requester client.. 65, 135, 239, 240

Bibliography

Abawajy JH (2009) An efficient adaptive scheduling policy for high-performance computing. Future Generation Computer Systems 25(3):364–370 198, 263

Abbasi AA, Al-qaness MA, Elaziz MA, Khalil HA, Kim S (2019) Bouncer: A resource-aware admission control scheme for cloud services. Electronics 8(9):928 260

Abdelzaher T, Lu C (2000) Modeling and performance control of internet servers. In: Decision and Control, 2000. Proceedings of the 39th IEEE Conference on, IEEE, vol 3, pp 2234–2239 25, 29

Abdelzaher T, Shin K, Bhatti N (2002) Performance guarantees for web server end-systems: A control-theoretical approach. Parallel and Distributed Systems, IEEE Transactions on 13(1):80–96 25, 29

Abdelzaher T, Stankovic J, Lu C, Zhang R, Lu Y (2003) Feedback performance control in software services. Control Systems, IEEE 23(3):74–90 25, 29

Abdelzaher TF, Bhatti N (1999) Web server qos management by adaptive content delivery. In: 1999 Seventh International Workshop on Quality of Service. IWQoS'99.(Cat. No. 98EX354), IEEE, pp 216–225 29

Abeel T, Van de Peer Y, Saeys Y (2009) Java-ml: A machine learning library. The Journal of Machine Learning Research 10:931–934 136

Abonyi J, Feil B (2007) Cluster analysis for data mining and system identification. Birkhäuser Basel 262

Abraham A, Liu H, Zhang W, Chang TG (2006) Scheduling jobs on computational grids using fuzzy particle swarm algorithm. In: International Conference on Knowledge-Based and Intelligent Information and Engineering Systems, Springer, pp 500–507 198

Abrahao B, Almeida V, Almeida J, Zhang A, Beyer D, Safai F (2006) Self-adaptive sla-driven capacity management for internet services. In: Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP, IEEE, pp 557–568 26, 29

Adzic G, Chatley R (2017) Serverless computing: economic and architectural impact. In: Proceedings of the 2017 11th joint meeting on foundations of software engineering, pp 884–889 244, 249, 283, 295

Agarwal MK, Sachindran N, Gupta M, Mann V (2006) Fast extraction of adaptive change point based patterns for problem resolution in enterprise systems. In: Large Scale Management of Distributed Systems, Springer, pp 161–172 73

Agmon Ben-Yehuda O, Ben-Yehuda M, Schuster A, Tsafrir D (2013) Deconstructing amazon ec2 spot instance pricing. ACM Transactions on Economics and Computation (TEAC) 1(3):1–20 276

Agutter C (2020) ITIL® 4 Essentials: Your essential guide for the ITIL 4 Foundation exam and beyond. IT Governance Ltd 296

Ahmad RW, Gani A, Hamid SHA, Shiraz M, Yousafzai A, Xia F (2015) A survey on virtual machine migration and server consolidation frameworks for cloud data centers. Journal of network and computer applications 52:11–25 233

Aksanli B, Venkatesh J, Zhang L, Rosing T (2012) Utilizing green energy prediction to schedule mixed batch and service jobs in data centers. ACM SIGOPS Operating Systems Review 45(3):53–57 101

Al-Dahoud A, Al-Sharif Z, Alawneh L, Jararweh Y (2016) Autonomic cloud computing resource scaling. In: 4th International IBM Cloud Academy Conference (ICACON 2016), University of Alberta, Edmonton, Canada, IBM 25, 165

Alfakih T, Hassan MM, Gumaei A, Savaglio C, Fortino G (2020) Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on sarsa. IEEE Access 8:54,074–54,084 158

Ali FA, Simoens P, Verbelen T, Demeester P, Dhoedt B (2016) Mobile device power models for energy efficient dynamic offloading at runtime. Journal of Systems and Software 113:173–187 285

Allen J (2017) Reactive design patterns. Simon and Schuster 243

Almeida F, Miranda E, Falcão J (2019) Challenges and facilitators practices for knowledge management in large-scale scrum teams. Journal of Information Technology Case and Application Research 21(2):90–102 13

Almeida J, Almeida V, Ardagna D, Cunha Í, Francalanci C, Trubian M (2010) Joint admission control and resource allocation in virtualized servers. Journal of Parallel and Distributed Computing 70(4):344–362 158

Alqudah M, Razali R (2016) A review of scaling agile methods in large software development. International Journal on Advanced Science, Engineering and Information Technology 6(6):828–837 229

Alsarhan A, Itradat A, Al-Dubai AY, Zomaya AY, Min G (2017) Adaptive resource allocation and provisioning in multi-service cloud environments. IEEE Transactions on Parallel and Distributed Systems 29(1):31–42 30

Ambler SW, Lines M (2012) Disciplined agile delivery: A practitioner's guide to agile software delivery in the enterprise. IBM press 220, 228, 229

Andrae AS, Edler T (2015) On global electricity usage of communication technology: trends to 2030. Challenges 6(1):117–157 36, 280

Andrieux A, Czajkowski K, Dan A, Keahey K, Ludwig H, Nakata T, Pruyne J, Rofrano J, Tuecke S, Xu M (2007) Web services agreement specification (ws-agreement). Tech. Rep. 1, Global Grid Forum (GGF) 17

Antony J (2006) Six sigma for service processes. Business process management journal 296

Antsaklis PJ, Passino KM, Wang SJ (1991) An introduction to autonomous control systems. IEEE Control Systems Magazine 11(4):5–13 19

Ardagna D, Panicucci B, Trubian M, Zhang L (2010) Energy-aware autonomic resource allocation in multitier virtualized environments. IEEE transactions on services computing 5(1):2–19 30, 33

Ardia D, Boudt K, Carl P, Mullen KM, Peterson BG (2011) Differential evolution with DEoptim. The R Journal 3(1):27–34 112, 116

Ardia D, Mullen K, Peterson B, Ulrich J, Boudt K (2014) DEoptim: Global Optimization by Differential Evolution. URL `http://cran.r-project.org/web/packages/DEoptim/` 116

Arlitt M, Jin T (1998) 1998 world cup web site access logs. The Internet Traffic Archive, sponsored by ACM SIGCOMM, URL `http://ita.ee.lbl.gov/html/contrib/WorldCup.html` 183

Arlitt M, Jin T (2000) A workload characterization study of the 1998 world cup web site. IEEE network 14(3):30–37 183

Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, et al (2010) A view of cloud computing. Communications of the ACM 53(4):50–58 160

Arpaci-Dusseau RH, Arpaci-Dusseau AC (2015) Operating Systems: Three Easy Pieces, 0th edn. Arpaci-Dusseau Books 164

Arunarani A, Manjula D, Sugumaran V (2019) Task scheduling techniques in cloud computing: A literature survey. Future Generation Computer Systems 91:407–415 195, 198

Arya V, Shanmugam K, Aggarwal P, Wang Q, Mohapatra P, Nagar S (2021) Evaluation of causal inference techniques for aiops. In: 8th ACM IKDD CODS and 26th COMAD, Association for Computing Machinery (ACM), Washington, DC, USA, pp 188–192 224

ASF (2004) Apache license, version 2.0. Tech. Rep. Version 2.0, The Apache Software Foundation, URL `http://www.apache.org/licenses/LICENSE-2.0` 133, 142, 162

ASF (2015) Apache http server documentation. URL `https://httpd.apache.org/` 240

Aslan J, Mayers K, Koomey JG, France C (2018) Electricity intensity of internet data transmission: Untangling the estimates. Journal of Industrial Ecology 22(4):785–798 279

Aslanpour MS, Toosi AN, Cicconetti C, Javadi B, Sbarski P, Taibi D, Assuncao M, Gill SS, Gaire R, Dustdar S (2021) Serverless edge computing: vision and challenges. In: 2021 Australasian Computer Science Week Multiconference, pp 1–10 230

Aston P, Fitzgerald C (2005) The grinder, a java load testing framework. URL http://grinder.sourceforge.net/ 145

Åström KJ (1983) Theory and applications of adaptive control—a survey. automatica 19(5):471–486 19, 27

Åström KJ, Wittenmark B (2013) Adaptive control. Courier Corporation 27

Atlam HF, Walters RJ, Wills GB (2018) Fog computing and the internet of things: A review. big data and cognitive computing 2(2):10 278

Avram MG (2014) Advantages and challenges of adopting cloud computing from an enterprise perspective. Procedia Technology 12(0):529–534 160

Awad A, El-Hefnawy N, Abdel_kader H (2015) Enhanced particle swarm optimization for task scheduling in cloud computing environments. Procedia Computer Science 65:920–929 198

Bainomugisha E, Carreton AL, Cutsem Tv, Mostinckx S, Meuter Wd (2013) A survey on reactive programming. ACM Computing Surveys (CSUR) 45(4):1–34 243

Baker EL, Orton SN (2010) Practicing management and leadership: vision, strategy, operations, and tactics. Journal of Public Health Management and Practice 16(5):470–471 229

Baldini I, Castro P, Chang K, Cheng P, Fink S, Ishakian V, Mitchell N, Muthusamy V, Rabbah R, Slominski A, et al (2017) Serverless computing: Current trends and open problems. In: Research Advances in Cloud Computing, Springer, pp 1–20 34, 35, 167, 171, 219, 283

Banks J, Carson J, Nelson B (2000) DM Nicol, Discrete-Event System Simulation. Prentice Hall 108

Bareinboim E, Pearl J (2016) Causal inference and the data-fusion problem. Proceedings of the National Academy of Sciences 113(27):7345–7352 262

Barroso LA, Hölzle U (2007) The case for energy-proportional computing. Computer 40(12) 36, 37

Barroso LA, Clidaras J, Hölzle U (2013) The datacenter as a computer: An introduction to the design of warehouse-scale machines. Synthesis lectures on computer architecture 8(3):1–154 36, 37

Bartolini DB, Sironi F, Sciuto D, Santambrogio MD (2012) An infrastructure to instrument applications and measure performance in self-adaptive computing. In: Workshop on Self-Awareness in Reconfigurable Computing Systems (SRCS), Citeseer, p 44 101

Bass L, Weber I, Zhu L (2015) DevOps: A software architect's perspective. Addison-Wesley Professional 18, 220

Bechlioulis CP, Rovithakis GA (2008) Robust adaptive control of feedback linearizable mimo nonlinear systems with prescribed performance. IEEE Transactions on Automatic Control 53(9):2090–2099 27

Beck K (2007) Implementation patterns. Pearson Education 12

Beck K, Beedle M, Van Bennekum A, Cockburn A, Cunningham W, Fowler M, Grenning J, Highsmith J, Hunt A, Jeffries R, et al (2001) Manifesto for agile software development. agileallianceorg 219, 220, 275

Becker K (2002) Test-driven development: by example. The Addison-Wesley signature series. Addison-Wesley, 12

Becker S, Koziolek H, Reussner R (2007) Model-based performance prediction with the palladio component model. In: Proceedings of the 6th international workshop on Software and performance, pp 54–65 213

Becker S, Koziolek H, Reussner R (2009) The palladio component model for model-driven performance prediction. Journal of Systems and Software 82(1):3–22 109, 213

Behnezhad S, Dhulipala L, Esfandiari H, Łącki J, Mirrokni V, Schudy W (2021) Massively parallel computation via remote memory access. ACM Transactions on Parallel Computing 8(3):1–25 277

Bélisle CJ (1992) Convergence theorems for a class of simulated annealing algorithms on R^d. Journal of Applied Probability pp 885–895 112, 273

Bell AJ, Sejnowski TJ (1995) An information-maximization approach to blind separation and blind deconvolution. Neural computation 7(6):1129–1159 98, 102

Beloglazov A (2013) Energy-efficient management of virtual machines in data centers for cloud computing. PhD thesis, Department of Computing and Information Systems, University of Melbourne 36, 279, 284

Beloglazov A, Buyya R (2010a) Energy efficient allocation of virtual machines in cloud data centers. In: Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on, IEEE, pp 577–578 37, 283

Beloglazov A, Buyya R (2010b) Energy efficient resource management in virtualized cloud data centers. In: Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, IEEE Computer Society, pp 826–831 61, 97, 99

Beloglazov A, Buyya R (2012a) Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints. IEEE transactions on parallel and distributed systems 24(7):1366–1379 279, 283

Beloglazov A, Buyya R (2012b) Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurrency and Computation: Practice and Experience 24(13):1397–1420 30

Beloglazov A, Buyya R (2015) Openstack neat: a framework for dynamic and energy-efficient consolidation of virtual machines in openstack clouds. Concurrency and Computation: Practice and Experience 27(5):1310–1333 30

Beloglazov A, Buyya R, Lee YC, Zomaya A (2011) A taxonomy and survey of energy-efficient data centers and cloud computing systems. In: Advances in computers, vol 82, Elsevier, pp 47–111 12

Beloglazov A, Abawajy J, Buyya R (2012) Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Generation Computer Systems 28(5):755–768 37, 61, 97, 99

Bendtsen C (2012) PSO: Particle Swarm Optimization, {pso}, R Documentation. URL http://cran.r-project.org/web/packages/pso/ 112, 115

Berl A, Gelenbe E, Di Girolamo M, Giuliani G, De Meer H, Dang MQ, Pentikousis K (2010) Energy-efficient cloud computing. The computer journal 53(7):1045–1051 37, 283

Bernard HR, Shelley GA, Killworth P (1987) How much of a network does the gss and rsw dredge up? Social Networks 228

Berndt D, Clifford J (1994) Using dynamic time warping to find patterns in time series. In: KDD workshop, Seattle, WA, vol 10, pp 359–370 72

Bernroider EW, Ivanov M (2011) It project management control and the control objectives for it and related technology (cobit) framework. International Journal of Project Management 29(3):325–336 296

Bertoli M, Casale G, Serazzri G (2006) Java modelling tools: an open source suite for queueing network modelling andworkload analysis. In: Quantitative Evaluation of Systems, 2006. QEST 2006. Third International Conference on, IEEE, pp 119–120 109

Bertoli M, Casale G, Serazzi G (2009) JMT: performance engineering tools for system modeling. ACM SIGMETRICS Performance Evaluation Review 36(4):10–15 109

Bertoncini M, Pernici B, Salomie I, Wesner S (2011) Games: Green active management of energy in it service centres. Information Systems Evolution pp 238–252 26, 27, 30

Bezemer CP, Pouwelse J, Gregg B (2015) Understanding software performance regressions using differential flame graphs. In: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), IEEE, pp 535–539 61

Bhamare D, Samaka M, Erbad A, Jain R, Gupta L, Chan HA (2017) Optimal virtual network function placement in multi-cloud service function chaining architecture. Computer Communications 102:1–16 284

Bhattacharyya A, Bandyopadhyay S, Pal A, Bose T (2016) Constrained application protocol (coap) option for no server response. Tech. Rep. 7967, The Internet Engineering Task Force (IETF) - RFC Editor, URL https://www.rfc-editor.org/info/rfc7967 264

Bi J, Yuan H, Tie M, Tan W (2015) Sla-based optimisation of virtualised resource for multi-tier web applications in cloud data centres. Enterprise Information Systems 9(7):743–767 38

Bi J, Yuan H, Tan W, Zhou M, Fan Y, Zhang J, Li J (2017) Application-aware dynamic fine-grained resource provisioning in a virtualized cloud data center. IEEE Transactions on Automation Science and Engineering 14(2):1172–1184 158

Bianco P, Lewis GA, Merson P (2008) Service level agreements in service-oriented architecture environments. Integration of Software-Intensive Systems Initiative 3

Bigus J (1994) Applying neural networks to computer system performance tuning. In: Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on, IEEE, vol 4, pp 2442–2447 26, 29, 129

Bigus J, Hellerstein J, Jayram T, Squillante M (2000) Autotune: A generic agent for automated performance tuning. Practical Application of Intelligent Agents and Multi Agent Technology 26, 129

Bishop M, Snyder ME, Harvey O (2015) Cloud economics: Making the business case for cloud – an economic framework for decision making. KPMG, CIO Advisory URL https://assets.kpmg/content/dam/kpmg/pdf/2015/11/cloud-economics.pdf 275

Bittner K, Kong P, Naiburg E, West D (2017) The Nexus Framework for scaling Scrum: Continuously Delivering an integrated product with multiple Scrum teams. Addison-Wesley Professional 229

Blackstock MA (2004) Aspect weaving with c# and .net. See http://www cs ubc ca/michael/publications/AOPNET5 pdf 238

Bodık P, Griffith R, Sutton C, Fox A, Jordan M, Patterson D (2009) Statistical machine learning makes automatic control practical for internet datacenters. In: Proceedings of the 2009 conference on Hot topics in cloud computing, HotCloud, vol 9 26, 30

Bogatinovski J, Nedelkoski S, Acker A, Schmidt F, Wittkopp T, Becker S, Cardoso J, Kao O (2021) Artificial intelligence for it operations (aiops) workshop white paper. arXiv preprint arXiv:210106054 224

Bohr MT, Young IA (2017) Cmos scaling trends and beyond. IEEE Micro 37(6):20–29 277

Bonér J, Farley D, Kuhn R, Thompson M (2014) The reactive manifesto. We Are Reactive URL https://www.reactivemanifesto.org/ 242

Boniface M, Nasser B, Papay J, Phillips S, Servin A, Yang X, Zlatev Z, Gogouvitis S, Katsaros G, Konstanteli K, et al (2010) Platform-as-a-service architecture for real-time quality of service management in clouds. In: Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on, IEEE, pp 155–160 26, 30

Bonomi F, Milito R, Zhu J, Addepalli S (2012) Fog computing and its role in the internet of things. In: Proceedings of the first edition of the MCC workshop on Mobile cloud computing, pp 13–16 218, 278

Bormann C, Lemay S, Tschofenig H, Hartke K, Silverajan B, Raymor B (2018) Coap (constrained application protocol) over tcp, tls and websockets. Tech. Rep. 8323, The Internet Engineering Task Force (IETF) - RFC Editor, URL https://www.rfc-editor.org/info/rfc8323 264

Boucadair M, Reddy K, T Shallow J (2020) Application protocol (coap) hop-limit option. Tech. Rep. 8768, The Internet Engineering Task Force (IETF) - RFC Editor, URL https://www.rfc-editor.org/info/rfc8768 264

Box GE, Jenkins GM, Reinsel GC (2011a) Time series analysis: forecasting and control, vol 734. Wiley 72, 77

Box GE, Luceno A, del Carmen Paniagua-Quiñones M (2011b) Statistical control by monitoring and adjustment, vol 898. Wiley 71

Brebner PC (2016) Automatic performance modelling from application performance management (apm) data: an experience report. In: Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering, pp 55–61 12

Breslau L, Knightly EW, Shenker S, Stoica I, Zhang H (2000) Endpoint admission control: Architectural issues and performance. In: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, pp 57–69 65

Briggs B, Kassner E (2016) Enterprise Cloud Strategy: Enterprise Cloud epUB _1. Microsoft Press 218, 276

Broda T, Esau D, Hemmert M (2019) System and method for real-time visualization of website performance data. US Patent 10,177,999 61

Brodersen KH, Gallusser F, Koehler J, Remy N, Scott SL (2015) Inferring causal impact using bayesian structural time-series models. The Annals of Applied Statistics 9(1):247–274 89

Brooks P, Chittenden J (2012) Metrics for Service Management: Designing for ITIL®. Van Haren Publishing 58

Brooks Jr FP (1995) The mythical man-month: essays on software engineering. Pearson Education 228

Brown B, Chui M, Manyika J (2011) Are you ready for the era of 'big data'? McKinsey Global Institute 128

Brun Y, Marzo Serugendo GD, Gacek C, Giese H, Kienle H, Litoiu M, Müller H, Pezzè M, Shaw M (2009) Engineering self-adaptive systems through feedback loops. In: Software engineering for self-adaptive systems, Springer, pp 48–70 28

Brünink M, Schmitt A, Knauth T, Süßkraut M, Schiffel U, Creutz S, Fetzer C (2011) Aaron: An adaptable execution environment. In: 2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN), IEEE, pp 411–421 285

Buenstorf G, Fornahl D (2009) B2c—bubble to cluster: the dot-com boom, spin-off entrepreneurship, and regional agglomeration. Journal of evolutionary Economics 19(3):349–378 221

Busoniu L, Babuska R, De Schutter B (2008) A comprehensive survey of multiagent reinforcement learning. IEEE Trans Systems, Man, and Cybernetics, Part C 38(2):156–172 169

Buyya R, Murshed M (2002) GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. Concurrency and computation: practice and experience 14(13-15):1175–1220 109

Buyya R, Yeo C, Venugopal S (2008) Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In: High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on, IEEE, pp 5–13 127, 128

Buyya R, Ranjan R, Calheiros RN (2009a) Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and opportunities. In: High Performance Computing & Simulation, 2009. HPCS'09. International Conference on, IEEE, pp 1–11 109

Buyya R, Yeo C, Venugopal S, Broberg J, Brandic I (2009b) Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation computer systems 25(6):599–616 26, 128, 159

Buyya R, Garg SK, Calheiros RN (2011) Sla-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions. In: Cloud and Service Computing (CSC), 2011 International Conference on, IEEE, pp 1–10 25, 165

Calheiros RN, Ranjan R, Beloglazov A, De Rose CA, Buyya R (2011) CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience 41(1):23–50 109, 163

Cantelon M, Harter M, Holowaychuk T, Rajlich N (2014) Node. js in Action. Manning Greenwich 242

Carlos A, Coello A, Lamont G, Van V (2002) Evolutionary algorithms for solving multi-objective problems 154

Carlson M, Chapman M, Heneveld A, Hinkelman S, Johnston-Watt D, Karmarkar A, Kunze T, Malhotra A, Mischkinsky J, Otto A, et al (2012) Cloud application management for platforms. OASIS, http://cloudspecs org/camp/CAMP-v1 0 pdf, Tech Rep 17

Castro P, Ishakian V, Muthusamy V, Slominski A (2017) Serverless programming (function as a service). In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), IEEE, pp 2658–2659 34, 283

Cavaness C (2006) Quartz Job Scheduling Framework: Building Open Source Enterprise Applications. Pearson Education 250

Čerešňák R, Kvet M (2019) Comparison of query performance in relational a non-relation databases. Transportation Research Procedia 40:170–177 246

Chadha M, John J, Gerndt M (2020) Extending slurm for dynamic resource-aware adaptive batch scheduling. arXiv preprint arXiv:200908289 198

Chalmers TC, Smith Jr H, Blackburn B, Silverman B, Schroeder B, Reitman D, Ambroz A (1981) A method for assessing the quality of a randomized control trial. Controlled clinical trials 2(1):31–49 262

Chambers C (1992) The design and implementation of the self compiler, an optimizing compiler for object-oriented programming languages. PhD thesis, Stanford University 102

Chandra AK, Hirschberg DS, Wong CK (1976) Approximate algorithms for some generalized knapsack problems. Theoretical Computer Science 3(3):293–304 198

Chattopadhyay S, Nelson N, Au A, Morales N, Sanchez C, Pandita R, Sarma A (2020) A tale from the trenches: cognitive biases and software development. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, pp 654–665 226

Chen A, Datta S, Hu XS, Niemier MT, Rosing TŠ, Yang JJ (2019) A survey on architecture advances enabled by emerging beyond-cmos technologies. IEEE Design & Test 36(3):46–68 277

Chen Y, Rangarajan G, Feng J, Ding M (2004) Analyzing multiple nonlinear time series with extended granger causality. Physics Letters A 324(1):26–35 72

Chen Y, Gmach D, Hyser C, Wang Z, Bash C, Hoover C, Singhal S (2010) Integrated management of application performance, power and cooling in data centers. In: Network Operations and Management Symposium (NOMS), 2010 IEEE, IEEE, pp 615–622 26, 30, 33, 58

Cheng D, Chen Y, Zhou X, Gmach D, Milojicic D (2017) Adaptive scheduling of parallel jobs in spark streaming. In: IEEE INFOCOM 2017-IEEE Conference on Computer Communications, IEEE, pp 1–9 198, 263

Cherkasova L, Phaal P (2002) Session-based admission control: A mechanism for peak load management of commercial web sites. IEEE Transactions on computers 51(6):669–685 159

Cherniack M, Balakrishnan H, Balazinska M, Carney D, Cetintemel U, Xing Y, Zdonik SB (2003) Scalable distributed stream processing. In: CIDR, vol 3, pp 257–268 244

Chun B, Culler D, Roscoe T, Bavier A, Peterson L, Wawrzoniak M, Bowman M (2003) Planetlab: an overlay testbed for broad-coverage services. ACM SIGCOMM Computer Communication Review 33(3):3–12 284

Chung L, do Prado Leite JCS (2009) On non-functional requirements in software engineering. In: Conceptual modeling: Foundations and applications, Springer, pp 363–379 11

Chung L, Nixon BA, Yu E, Mylopoulos J (2012) Non-functional requirements in software engineering, vol 5. Springer Science & Business Media 11

Cichocki A, Amari Si, et al (2002) Adaptive blind signal and image processing. John Wiley Chichester 98, 102

Cingolani P (2006) Jswarm-pso is a particle swarm optimization package written in java. URL `http://jswarm-pso.sourceforge.net/` 205

Cirani S, Picone M, Veltri L (2015) mjcoap: An open-source lightweight java coap library for internet of things applications. In: Interoperability and Open-Source Solutions for the Internet of Things, Springer, pp 118–133 264

Clerc M (2010) Particle swarm optimization, vol 93. John Wiley & Sons 115

Clerc M, Kennedy J (2002) The particle swarm-explosion, stability, and convergence in a multidimensional complex space. Evolutionary Computation, IEEE Transactions on 6(1):58–73 115

Clerc M, et al (2011) Standard PSO 2011. Tech. rep., Technical Report [online] http://www. particleswarm. info/, Particle Swarm Central 112, 115, 273

Cleveland W, Devlin S (1988) Locally weighted regression: an approach to regression analysis by local fitting. Journal of the American Statistical Association 83(403):596–610 114, 151

Cleveland WS (1979) Robust locally weighted regression and smoothing scatterplots. Journal of the American Statistical Association 74(368):829–836 114

Cleveland WS, Grosse E, Shyu WM (1992) Local regression models. Statistical models in S pp 309–376 114

Cockburn A (2004) Crystal clear: A human-powered methodology for small teams: A human-powered methodology for small teams. Pearson Education 220

Cockburn A (2006) Agile software development: the cooperative game. Pearson Education 220, 287

Cohn M (2004) User stories applied: For agile software development. Addison-Wesley Professional 220

Cohn M (2010) Succeeding with agile: software development using Scrum. Pearson Education 228, 229

Collins E, Dias-Neto A, de Lucena Jr VF (2012) Strategies for agile software testing automation: An industrial experience. In: 2012 IEEE 36th Annual Computer Software and Applications Conference Workshops, IEEE, pp 440–445 12

Comon P, Jutten C (2010) Handbook of Blind Source Separation: Independent component analysis and applications. Academic press 86

Computing A, et al (2006) An architectural blueprint for autonomic computing. IBM White Paper 31(2006):1–6 28

Conboy K, Carroll N (2019) Implementing large-scale agile frameworks: challenges and recommendations. IEEE Software 36(2):44–50 229

Coplien JO, Bjørnvig G (2011) Lean architecture: for agile software development. John Wiley & Sons 218

Coward D, Yoshida Y (2003) Java servlet specification version 2.3. Sun Microsystems, Nov 135

Crispin L, Gregory J (2009) Agile testing: A practical guide for testers and agile teams. Pearson Education 12, 39

Croskerry P (2003) The importance of cognitive errors in diagnosis and strategies to minimize them. Academic medicine 78(8):775–780 226

Cui H, Li Y, Liu X, Ansari N, Liu Y (2017) Cloud service reliability modelling and optimal task scheduling. Iet Communications 11(2):161–167 23

Dang Y, Lin Q, Huang P (2019) Aiops: real-world challenges and research innovations. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), IEEE, pp 4–5 224

Danovaro E, Janes A, Succi G (2008) Jidoka in software development. In: Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications, pp 827–830 294

Danowitz A, Kelley K, Mao J, Stevenson JP, Horowitz M (2012) Cpu db: recording microprocessor history. Communications of the ACM 55(4):55–63 276

Davis J, Daniels R (2016) Effective DevOps: building a culture of collaboration, affinity, and tooling at scale. "O'Reilly Media, Inc." 226

Debboub S, Meslati D (2013) Quantitative and qualitative evaluation of aspectj, jboss aop and caesarj, using gang-of-four design patterns. International Journal of Software Engineering and Its Applications 7(6):157–174 237

Debski A, Szczepanik B, Malawski M, Spahr S, Muthig D (2017) A scalable, reactive architecture for cloud applications. IEEE Software 35(2):62–71 243

Delcev S, Draskovic D (2018) Modern javascript frameworks: A survey study. In: 2018 Zooming Innovation in Consumer Technologies Conference (ZINC), IEEE, pp 106–109 239

Deming WE (1950) To management. Lecture presented in Japan Retrieved August 31:2018 294

Demirci M (2015) A survey of machine learning applications for energy-efficient resource management in cloud computing environments. In: Machine Learning and Applications (ICMLA), 2015 IEEE 14th International Conference on, IEEE, pp 1185–1190 159

Di Pillo G, Grippo L (1989) Exact penalty functions in constrained optimization. SIAM Journal on control and optimization 27(6):1333–1360 202

Dikert K, Paasivaara M, Lassenius C (2016) Challenges and success factors for large-scale agile transformations: A systematic literature review. Journal of Systems and Software 119:87–108 13

Ding H, Trajcevski G, Scheuermann P, Wang X, Keogh E (2008) Querying and mining of time series data: experimental comparison of representations and distance measures. Proceedings of the VLDB Endowment 1(2):1542–1552 70, 72

Dingsøyr T, Moe NB (2014) Towards principles of large-scale agile development. In: International Conference on Agile Software Development, Springer, pp 1–8 13

DiPierro M (2009) Web2py Enterprise Web Framework. Wiley Publishing 239

Dmitry N, Manfred SS (2014) On micro-services architecture. International Journal of Open Information Technologies 2(9) 167

Dongarra J, Meuer H, Strohmaier E (1993) Top 500 supercomputers. RUM 33(93) 282

Dongarra JJ (1987) The linpack benchmark: An explanation. In: International Conference on Supercomputing, Springer, pp 456–474 282

Dongarra JJ, Luszczek P, Petitet A (2003) The linpack benchmark: past, present and future. Concurrency and Computation: practice and experience 15(9):803–820 282

Dragoni N, Giallorenzo S, Lafuente AL, Mazzara M, Montesi F, Mustafin R, Safina L (2017a) Microservices: yesterday, today, and tomorrow. In: Present and ulterior software engineering, Springer, pp 195–216 2, 23, 218, 219

Dragoni N, Lanese I, Larsen ST, Mazzara M, Mustafin R, Safina L (2017b) Microservices: How to make your application scale. In: International Andrei Ershov Memorial Conference on Perspectives of System Informatics, Springer, pp 95–104 218

Dudley JJ, Kristensson PO (2018) A review of user interface design for interactive machine learning. ACM Transactions on Interactive Intelligent Systems (TiiS) 8(2):1–37 224

Dunbar RI (1992) Neocortex size as a constraint on group size in primates. Journal of human evolution 22(6):469–493 228

Duolikun D, Enokido T, Takizawa M (2017) An energy-aware algorithm to migrate virtual machines in a server cluster. International Journal of Space-Based and Situated Computing 7(1):32–42 37

Dutreilh X, Moreau A, Malenfant J, Rivierre N, Truck I (2010) From data center resource allocation to control theory and back. In: Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, IEEE, pp 410–417 22, 34, 156

Duvall PM, Matyas S, Glover A (2007) Continuous integration: improving software quality and reducing risk. Pearson Education 39

Duy TVT, Sato Y, Inoguchi Y (2010) Performance evaluation of a green scheduling algorithm for energy savings in cloud computing. In: Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on, IEEE, pp 1–8 38, 157

Easterbrook JA (1959) The effect of emotion on cue utilization and the organization of behavior. Psychological review 66(3):183 227

ECMAScript ED (2015) Language specification (rc4). URL `https://www.ecma-international.org/wp-content/uploads/ECMA-262_6th_edition_june_2015.pdf` 242

Eichler M (2007) Granger causality and path diagrams for multivariate time series. Journal of Econometrics 137(2):334–353 72

Eivy A, Weinman J (2017) Be wary of the economics of" serverless" cloud computing. IEEE Cloud Computing 4(2):6–12 156, 276

Ellis S, Brown M (2017) Hacking growth: how today's fastest-growing companies drive breakout success. Currency 249, 295

Emeakaroha V, Brandic I, Maurer M, Dustdar S (2010) Low level metrics to high level slas-lom2his framework: Bridging the gap between monitored metrics and sla parameters in cloud environments. In: High Performance Computing and Simulation (HPCS), 2010 International Conference on, IEEE, pp 48–54 26, 27, 30

Emeakaroha VC, Brandic I, Maurer M, Breskovic I (2011) Sla-aware application deployment and resource allocation in clouds. In: Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual, IEEE, pp 298–303 101

EPA (2021) Sources of greenhouse gas emissions. Tech. rep., United States Environmental Protection Agency, URL `https://www.epa.gov/ghgemissions/sources-greenhouse-gas-emissions` 283

Esling P, Agon C (2012) Time-series data mining. ACM Computing Surveys (CSUR) 45(1):1–34 5

Etiemble D (2018) 45-year cpu evolution: one law and two equations. arXiv preprint arXiv:180300254 276

Etro F (2015) The economics of cloud computing. In: Cloud Technology: Concepts, Methodologies, Tools, and Applications, IGI Global, pp 2135–2148 276

Eugster PT, Felber PA, Guerraoui R, Kermarrec AM (2003) The many faces of publish/subscribe. ACM computing surveys (CSUR) 35(2):114–131 243

Evans E (2004) Domain-driven design: tackling complexity in the heart of software. Addison-Wesley Professional 145

Evans E (2012) Ddd sample application, the project is a joint effort by eric evans' company domain language and the swedish software consulting company citerus. URL http://dddsample.sourceforge.net/ 145

Fabijan A, Dmitriev P, McFarland C, Vermeer L, Holmström Olsson H, Bosch J (2018) Experimentation growth: Evolving trustworthy a/b testing capabilities in online software companies. Journal of Software: Evolution and Process 30(12):e2113 249

Fain Y, Moiseev A (2017) Angular 2 Development with TypeScript. Manning Publications Company 239

Fan X, Weber WD, Barroso LA (2007) Power provisioning for a warehouse-sized computer. In: ACM SIGARCH computer architecture news, ACM, vol 35, pp 13–23 37, 283, 285

Farokhi S, Jamshidi P, Brandic I, Elmroth E (2015) Self-adaptation challenges for cloud-based applications: A control theoretic perspective. In: 10th international workshop on feedback computing, vol 2015 28

Faust S (2020) Using google´s flutter framework for the development of a large-scale reference application. Master's thesis, Technical University Cologne 249

Fazio M, Celesti A, Ranjan R, Liu C, Chen L, Villari M (2016) Open issues in scheduling microservices in the cloud. IEEE Cloud Computing 3(5):81–88 284

Feathers M (2004) Working Effectively with Legacy Code: WORK EFFECT LEG CODE _p1. Prentice Hall Professional 12

Fehling C, Ewald T, Leymann F, Pauly M, Rütschlin J, Schumm D (2012) Capturing cloud computing knowledge and experience in patterns. In: 2012 IEEE Fifth international conference on cloud computing, IEEE, pp 726–733 219

Feng G, Garg S, Buyya R, Li W (2012) Revenue maximization using adaptive resource provisioning in cloud computing environments. In: Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing, IEEE Computer Society, pp 192–200 101

Feng Wc, Cameron K (2007) The green500 list: Encouraging sustainable supercomputing. Computer 40(12):50–55 283

Ferrer AJ, HernáNdez F, Tordsson J, Elmroth E, Ali-Eldin A, Zsigri C, Sirvent R, Guitart J, Badia RM, Djemame K, et al (2012) Optimis: A holistic approach to cloud service provisioning. Future Generation Computer Systems 28(1):66–77 158, 177

Festinger L (1962) A theory of cognitive dissonance, vol 2. Stanford university press 226

Fielding R, Reschke J (2013) Hypertext transfer protocol (http/1.1): Message syntax and routing. Tech. Rep. 7230, The Internet Engineering Task Force (IETF) - RFC Editor, URL https://www.rfc-editor.org/info/rfc7230 264

Fielding R, Gettys J, Mogul J, Frystyk H, Masinter L, Leach P, Berners-Lee T (1999) Hypertext transfer protocol – http/1.1. Tech. Rep. 2616, The Internet Engineering Task Force (IETF) - RFC Editor, URL https://www.rfc-editor.org/info/rfc2616 240

Fielding RT (2000) Architectural styles and the design of network-based software architectures, vol 7. University of California, Irvine Irvine 218

Figiela K, Gajek A, Zima A, Obrok B, Malawski M (2018) Performance evaluation of heterogeneous cloud functions. Concurrency and Computation: Practice and Experience 30(23):e4792 290

Fioccola G, Capello A, Cociglio M, Castaldelli L, Chen MG, Zheng L, Mirsky G, Mizrahi T (2018) Alternate-marking method for passive and hybrid performance monitoring. Tech. Rep. 8321, The Internet Engineering Task Force (IETF) - RFC Editor, URL https://www.rfc-editor.org/info/rfc8321 264

Fischer A, Botero JF, Beck MT, De Meer H, Hesselbach X (2013) Virtual network embedding: A survey. IEEE Communications Surveys & Tutorials 15(4):1888–1906 159

Fisher RA, et al (1937) The design of experiments. The design of experiments (2nd Ed) 262

Fleischmann M, Amirpur M, Benlian A, Hess T (2014) Cognitive biases in information systems research: A scientometric analysis. In: Twenty Second European Conference on Information Systems, Tel Aviv 226

Fodor I (2002) A survey of dimension reduction techniques. Center for Applied Scientific Computing, Lawrence Livermore National Laboratory 9:1–18 136

Forcier J, Bissex P, Chun WJ (2008) Python web development with Django. Addison-Wesley Professional 239

Ford N, Parsons R, Kua P (2017) Building evolutionary architectures: Support constant change. " O'Reilly Media, Inc." 218, 292

Forney A, Bareinboim E (2019) Counterfactual randomization: rescuing experimental studies from obscured confounding. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol 33, pp 2454–2461 263

Fowler M (2012) Patterns of Enterprise Application Architecture: Pattern Enterpr Applica Arch. Addison-Wesley 218

Fowler M (2018) Refactoring: improving the design of existing code. Addison-Wesley Professional 13, 34, 218, 292

Fowler M, Foemmel M (2006) Continuous integration 39

Fowler M, Highsmith J (2001) The agile manifesto. Software Development 9(8):28–35 1, 12, 219, 292

Fox A, Griffith R, Joseph A, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, et al (2009) Above the clouds: A berkeley view of cloud computing. Dept Electrical Eng and Comput Sciences, University of California, Berkeley, Rep UCB/EECS 28(13):2009 160

Fox GC, Ishakian V, Muthusamy V, Slominski A (2017a) Report from workshop and panel on the status of serverless computing and function-as-a-service (faas) in industry and research. In: First International Workshop on Serverless Computing (WoSC) 292

Fox GC, Ishakian V, Muthusamy V, Slominski A (2017b) Status of serverless computing and function-as-a-service (faas) in industry and research. arXiv preprint arXiv:170808028 128

Fradkov AL, Miroshnik IV, Nikiforov VO (1999) Nonlinear and adaptive control of complex systems, vol 491. Springer 32

Freeman A (2016) Pro Asp. net Core Mvc. Apress 239

Frei A, Grawehr P, Alonso G (2004) A dynamic aopengine for. net. Technical reports 445 238

Fritzsch J, Bogner J, Zimmermann A, Wagner S (2018) From monolith to microservices: A classification of refactoring approaches. In: International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment, Springer, pp 128–141 13, 234

Furda A, Fidge C, Zimmermann O, Kelly W, Barros A (2017) Migrating enterprise legacy source code to microservices: on multitenancy, statefulness, and data consistency. IEEE Software 35(3):63–72 290

Gamma E (1995) Design patterns: elements of reusable object-oriented software. Pearson Education India 65

Gamma E, Helm R, Johnson R, Vlissides J, Patterns D (1995) Elements of reusable object-oriented software, vol 99. Addison-Wesley Reading, Massachusetts 238

Gandhi A, Chen Y, Gmach D, Arlitt M, Marwah M (2011) Minimizing data center sla violations and power consumption via hybrid resource provisioning. In: 2011 International Green Computing Conference and Workshops, IEEE, pp 1–8 279

Gao Y, Rong H, Huang JZ (2005) Adaptive grid job scheduling with genetic algorithms. Future Generation Computer Systems 21(1):151–161 198

Garcia A, Sant'Anna C, Figueiredo E, Kulesza U, Lucena C, von Staa A (2006) Modularizing design patterns with aspects: a quantitative study. In: Transactions on Aspect-Oriented Software Development I, Springer, pp 36–74 238

García-Valls M, Cucinotta T, Lu C (2014) Challenges in real-time virtualization and predictable cloud computing. Journal of Systems Architecture 60(9):726–740 37, 156, 167

Gasparyan O (2008) Linear and nonlinear multivariable feedback control: a classical approach. John Wiley & Sons 32

Gasse M, Grasset D, Gaudron G, Oudeyer PY (2021) Causal reinforcement learning using observational and interventional data. arXiv preprint arXiv:210614421 263

Ge SS, Wang C (2004) Adaptive neural control of uncertain mimo nonlinear systems. IEEE Transactions on Neural Networks 15(3):674–692 27

Ge SS, Hong F, Lee TH (2004) Adaptive neural control of nonlinear time-delay systems with unknown virtual control coefficients. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 34(1):499–516 27

Gehlsen B, Page B (2001) A framework for distributed simulation optimization. In: Proceedings of the 33nd conference on Winter simulation, IEEE Computer Society, pp 508–514 108

Geiger P, Carata L, Schölkopf B (2016) Causal inference for data-driven debugging and decision making in cloud computing. arXiv preprint arXiv:160301581 71

Geng X, Liu TY, Qin T, Li H (2007) Feature selection for ranking. In: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, ACM, pp 407–414 72

Gey F, Van Landuyt D, Joosen W, Jonckers V (2015) Continuous evolution of multi-tenant saas applications: A customizable dynamic adaptation approach. In: 2015 IEEE/ACM 7th International Workshop on Principles of Engineering Service-Oriented and Cloud Systems, IEEE, pp 10–16 290

Gigerenzer G, Gaissmaier W (2011) Heuristic decision making. Annual review of psychology 62:451–482 227

Gigerenzer G, Todd PM (1999) Simple heuristics that make us smart. Oxford University Press, USA 227

Gill SS, Buyya R (2018) Secure: Self-protection approach in cloud resource management. IEEE Cloud Computing 5(1):60–72 30

Gill SS, Buyya R (2019) Resource provisioning based scheduling framework for execution of heterogeneous and clustered workloads in clouds: from fundamental to autonomic offering. Journal of Grid Computing 17(3):385–417 30

Gill SS, Tuli S, Toosi AN, Cuadrado F, Garraghan P, Bahsoon R, Lutfiyya H, Sakellariou R, Rana O, Dustdar S, et al (2020) Thermosim: Deep learning based framework for modeling and simulation of thermal-aware resource management for cloud computing environments. Journal of Systems and Software 166:110,596 28, 30

Gill SS, Xu M, Ottaviani C, Patros P, Bahsoon R, Shaghaghi A, Golec M, Stankovski V, Wu H, Abraham A, et al (2022) Ai for next generation computing: Emerging trends and future directions. Internet of Things 19:100,514 28

Glinz M (2007) On non-functional requirements. In: 15th IEEE International Requirements Engineering Conference (RE 2007), IEEE, pp 21–26 11

Göbel J, Joschko P, Koors A, Page B (2013) The discrete event simulation framework DESMO-J: Review, comparison to other frameworks and latest development. In: ECMS, pp 100–109 110, 162

Goldberg D, David Edward G, Goldberg D, Goldberg V (1989) Genetic Algorithms in Search, Optimization, and Machine Learning. Artificial Intelligence, Addison-Wesley Publishing Company 114, 205

Goldberg RP (1974) Survey of virtual machine research. Computer 7(6):34–45 218

Goldfarb BD, Pfarrer MD, Kirsch D (2005) Searching for ghosts: business survival, unmeasured entrepreneurial activity and private equity investment in the dot-com era. Robert H Smith School Research Paper No RHS pp 06–027 221

Gong Z, Gu X, Wilkes J (2010) Press: Predictive elastic resource scaling for cloud systems. In: 2010 International Conference on Network and Service Management, Ieee, pp 9–16 98, 156

González A (2019) Trends in processor architecture. In: Harnessing Performance Variability in Embedded and High-performance Many/Multi-core Platforms, Springer, pp 23–42 277

Gonzalez T, Sahni S (1976) Open shop scheduling to minimize finish time. Journal of the ACM (JACM) 23(4):665–679 198

Goudarzi H, Pedram M (2011) Multi-dimensional sla-based resource allocation for multi-tier cloud computing systems. In: Cloud Computing (CLOUD), 2011 IEEE International Conference on, IEEE, pp 324–331 26

Govindan S, Choi J, Urgaonkar B, Sivasubramaniam A, Baldini A (2009) Statistical profiling-based techniques for effective power provisioning in data centers. In: Proceedings of the 4th ACM European conference on Computer systems, pp 317–330 279

Granger CW (1969) Investigating causal relations by econometric models and cross-spectral methods. Econometrica: Journal of the Econometric Society pp 424–438 72

Grefenstette JJ (1986) Optimization of control parameters for genetic algorithms. Systems, Man and Cybernetics, IEEE Transactions on 16(1):122–128 114

Gregg B (2010) Visualizing system latency. Communications of the ACM 53(7):48–54 61

Gregg B (2019) BPF Performance Tools. Addison-Wesley Professional 57

Grinberg M (2018) Flask web development: developing web applications with python. " O'Reilly Media, Inc." 239

Grinshpan L (2012) Workload characterization and transaction profiling. Solving Enterprise Applications Performance Puzzles: Queuing Models to the Rescue pp 57–94 31, 38, 46, 54, 61, 68, 96, 97, 104

Grönroos M (2011) Book of vaadin. Lulu. com 239

Gruschke B, et al (1998) Integrated event management: Event correlation using dependency graphs. In: Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 98), pp 130–141 73

Günther S, Sunkle S (2009) Enabling feature-oriented programming in ruby. Univ., Fak. für Informatik 238

Gupta JN, Stafford Jr EF (2006) Flowshop scheduling research after five decades. European Journal of Operational Research 169(3):699–711 198

Guyon I, Elisseeff A (2003) An introduction to variable and feature selection. The Journal of Machine Learning Research 3:1157–1182 32, 69, 136

Guyon I, Aliferis C, Elisseeff A (2007) Causal feature selection. Computational methods of feature selection pp 63–82 72

Guzek M, Gniewek A, Bouvry P, Musial J, Blazewicz J (2015) Cloud brokering: Current practices and upcoming challenges. IEEE Cloud Computing 2(2):40–47 265

Hacigumus H, Iyer B, Mehrotra S (2002) Providing database as a service. In: Proceedings 18th International Conference on Data Engineering, IEEE, pp 29–38 245

Haensch W, Nowak EJ, Dennard RH, Solomon PM, Bryant A, Dokumaci OH, Kumar A, Wang X, Johnson JB, Fischetti MV (2006) Silicon cmos devices beyond scaling. IBM Journal of Research and Development 50(4.5):339–361 277

Hahn E (2016) Express in Action: Writing, building, and testing Node. js applications. Manning Publications, 239

Haines S (2006) Pro Java EE 5 performance management and optimization. Apress 28, 38, 54, 61, 68, 96, 104

Hall MA (1999) Correlation-based feature selection for machine learning. PhD thesis, The University of Waikato 72

Hanoch Y, Vitouch O (2004) When less is more: Information, emotional arousal and the ecological reframing of the yerkes-dodson law. Theory & Psychology 14(4):427–452 228, 287

Harchol-Balter M (2013) Performance Modeling and Design of Computer Systems: Queueing Theory in Action. Cambridge University Press 108, 110

Harris A, Haase K (2011) Sinatra: Up and Running: Ruby for the Web, Simply. " O'Reilly Media, Inc." 239

Harrop R, Machacek J (2005) Job scheduling with spring. Pro Spring pp 487–516 250

Hartke K (2015) Observing resources in the constrained application protocol (coap). Tech. Rep. 7641, The Internet Engineering Task Force (IETF) - RFC Editor, URL https://www.rfc-editor.org/info/rfc7641 264

Hartl M (2015) Ruby on rails tutorial: learn Web development with rails. Addison-Wesley Professional 239

Hasselmeyer P, Koller B, Schubert L, Wieder P (2006) Towards sla-supported resource management. In: International Conference on High Performance Computing and Communications, Springer, pp 743–752 225

Hathaway DH, Wilson RM, Reichmann EJ (2002) Group sunspot numbers: Sunspot cycle characteristics. Solar Physics 211(1-2):357–370 183

Hawilo H, Jammal M, Shami A (2019) Network function virtualization-aware orchestrator for service function chaining placement in the cloud. IEEE Journal on Selected Areas in Communications 37(3):643–655 284

He Y, Huang J, Duan Q, Xiong Z, Lv J, Liu Y (2014) A novel admission control model in cloud computing. arXiv preprint arXiv:14014716 159

Heger C, van Hoorn A, Mann M, Okanović D (2017) Application performance management: State of the art and challenges for the future. In: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, pp 429–432 12

Hellerstein J (2004) Challenges in control engineering of computing systems. In: American Control Conference, 2004. Proceedings of the 2004, IEEE, vol 3, pp 1970–1979 25, 29, 33

Hellerstein J, Parekh S, Diao Y, Tilbury D (2004) Feedback control of computing systems. Wiley-IEEE Press 25, 26, 32, 47, 261

Hendrickson S, Sturdevant S, Harter T, Venkataramani V, Arpaci-Dusseau AC, Arpaci-Dusseau RH (2016) Serverless computation with openlambda. Elastic 60:80 35, 291

Hennessy JL, Patterson DA (2011) Computer architecture: a quantitative approach. Elsevier 23, 292

Henney K (2010) 97 things every programmer should know: collective wisdom from the experts. O'Reilly Media 12

Herbst NR, Huber N, Kounev S, Amrehn E (2013) Self-adaptive workload classification and forecasting for proactive resource provisioning. In: Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering, pp 187–198 30

Hermes D (2015) Xamarin mobile application development: Cross-platform c# and xamarin. forms fundamentals. Apress 249

Herrnstein R (1970) On the law of effect. Journal of the experimental analysis of behavior 13(2):243 130

Highsmith JA, Highsmith J (2002) Agile software development ecosystems. Addison-Wesley Professional 220

Hintemann R, Hinterholzer S (2019) Energy consumption of data centers worldwide. In: The 6th International Conference on ICT for Sustainability (ICT4S). Lappeenranta 280

Hintze JL, Nelson RD (1998) Violin plots: a box plot-density trace synergism. The American Statistician 52(2):181–184 114

Hirzel M, Soulé R, Schneider S, Gedik B, Grimm R (2014) A catalog of stream processing optimizations. ACM Computing Surveys (CSUR) 46(4):1–34 244

Hoang HN, Le Van S, Maue HN, Bien CPN (2016) Admission control and scheduling algorithms based on aco and pso heuristic for optimizing cost in cloud computing. In: Recent Developments in Intelligent Information and Database Systems, Springer, pp 15–28 158

Hohpe G, Woolf B (2004) Enterprise integration patterns: Designing, building, and deploying messaging solutions. Addison-Wesley Professional 218, 238, 243

Holland PW (1986) Statistics and causal inference. Journal of the American statistical Association 81(396):945–960 72

Hoorn Av, Siegl S (2018) Application performance management: measuring and optimizing the digital customer experience. Troisdorf: SIGS DATACOM GmbH 17

Hoover D, Oshineye A (2009) Apprenticeship patterns: Guidance for the aspiring software craftsman. " O'Reilly Media, Inc." 220

Huang J (2005) etom and itil. Business Process Trends 296

Huebscher MC, McCann JA (2008) A survey of autonomic computing—degrees, models, and applications. ACM Computing Surveys (CSUR) 40(3):1–28 28

Humble J, Farley D (2011) Continuous delivery: reliable software releases through build, test, and deployment automation. Pearson Education 220, 292

Humble J, Kim G (2018) Accelerate: The science of lean software and devops: Building and scaling high performing technology organizations. IT Revolution 39, 218, 219, 220, 229, 275, 292

Hume D (1758) Essays and treatises on several subjects. A. Millar; and A. Kincaid and A. Donaldson, at Edinburgh 71

Hunter J, McIntosh N (1999) Knowledge-based event detection in complex time series data. Artificial Intelligence in Medicine pp 271–280 72

Hutchison S (2013) Shift left! ITEA Journal of Test & Evaluation: Official Publication of the International Test and Evaluation Association 34(2):133–137 292

Hwang K, Bai X, Shi Y, Li M, Chen WG, Wu Y (2016) Cloud performance modeling with benchmark evaluation of elastic scaling strategies. IEEE Transactions on Parallel and Distributed Systems 27(1):130–143 25, 165

Ibrahim S, Jin H, Lu L, He B, Wu S (2011) Adaptive disk i/o scheduling for mapreduce in virtualized environment. In: Parallel Processing (ICPP), 2011 International Conference on, IEEE, pp 335–344 101

Ihler A, Hutchins J, Smyth P (2006) Adaptive event detection with time-varying poisson processes. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, pp 207–216 72

Infield D, Freris L (2020) Renewable energy in power systems. John Wiley & Sons 283

Ishikawa K (1985) What is total quality control? The Japanese way. Prentice Hall 294

Ismaeel S, Karim R, Miri A (2018) Proactive dynamic virtual-machine consolidation for energy conservation in cloud data centres. Journal of Cloud Computing 7(1):1–28 283

Jaber A, Kocaoglu M, Shanmugam K, Bareinboim E (2020) Causal discovery from soft interventions with unknown targets: Characterization and learning. Advances in neural information processing systems 33:9551–9561 263

Jackson K (2012) OpenStack cloud computing cookbook. Packt Publishing Ltd 60

Jackson KL, Goessling S (2018) Architecting Cloud Computing Solutions: Build cloud strategies that align technology and economics while effectively managing risk. Packt Publishing Ltd 275

Jakob N (1993) Usability engineering. Fremont, California: Morgan 174

Jamshidi P, Pahl C, Chinenyeze S, Liu X (2015) Cloud migration patterns: a multi-cloud service architecture perspective. In: Service-Oriented Computing-ICSOC 2014 Workshops, Springer, pp 6–19 265

Janes A, Succi G (2014) Lean Software Development in Action. Springer 220

Jang SH, Kim TY, Kim JK, Lee JS (2012) The study of genetic algorithm-based task scheduling for cloud computing. International Journal of Control and Automation 5(4):157–162 198

Janssens N, Truyen E, Sanen F, Joosen W (2007) Adding dynamic reconfiguration support to jboss aop. In: Proceedings of the 1st workshop on Middleware-application interaction: in conjunction with Euro-Sys 2007, pp 1–8 238

Javidan M (1998) Core competence: what does it mean in practice? Long range planning 31(1):60–71 228

Jena R (2017) Energy efficient task scheduling in cloud environment. Energy Procedia 141:222–227 198

Jensen ED, Locke CD, Tokuda H (1985) A time-driven scheduling model for real-time operating systems. In: RTSS, vol 85, pp 112–122 164

Jerald J, Asokan P, Prabaharan G, Saravanan R (2005) Scheduling optimisation of flexible manufacturing systems using particle swarm optimisation algorithm. The International Journal of Advanced Manufacturing Technology 25(9):964–971 198

Jian C, Wang Y (2014) Batch task scheduling-oriented optimization modelling and simulation in cloud manufacturing. International Journal of Simulation Modelling 13(1):93–101 198

Jiang G, Chen H, Yoshihira K, Saxena A (2011) Ranking the importance of alerts for problem determination in large computer systems. Cluster Computing 14(3):213–227 73

Jin Lj, Machiraju V, Sahai A (2002) Analysis on service level agreement of web services. HP June 19:1–13 3

Johnson R, Hoeller J, Donald K, Sampaleanu C, Harrop R, Risberg T, Arendsen A, Davison D, Kopylenko D, Pollack M, et al (2004) The spring framework–reference documentation. interface 21:27, URL https://docs.spring.io/spring-framework/docs/3.2.19.BUILD-SNAPSHOT/spring-framework-reference/pdf/spring-framework-reference.pdf 238, 239

Jonas E, Schleier-Smith J, Sreekanti V, Tsai CC, Khandelwal A, Pu Q, Shankar V, Carreira J, Krauth K, Yadwadkar N, et al (2019) Cloud programming simplified: A berkeley view on serverless computing. arXiv preprint arXiv:190203383 283

Jones N (2018) How to stop data centres from gobbling up the world's electricity. Nature 561(7722):163–167 279

Jordehi AR, Jasni J (2015) Particle swarm optimisation for discrete optimisation problems: a review. Artificial Intelligence Review 43(2):243–258 198

Juditsky A, Hjalmarsson H, Benveniste A, Delyon B, Ljung L, Sjöberg J, Zhang Q (1995) Nonlinear black-box models in system identification: Mathematical foundations. Automatica 31(12):1725–1750 31

Kalbfleisch C, Cole R, Romascanu D (2005) Definition of managed objects for synthetic sources for performance monitoring algorithms. Tech. Rep. 4149, The Internet Engineering Task Force (IETF) - RFC Editor, URL https://www.rfc-editor.org/info/rfc4149 264

Kamiya G (2021) Data centres and data transmission networks: More efforts needed. URL https://www.iea.org/reports/data-centres-and-data-transmission-networks 279, 283

Kamiya G, Kvarnström O (2019) Data centres and energy–from global headlines to local headaches. Tech. rep., International Energy Agency, URL https://www.iea.org/commentaries/data-centres-and-energy-from-global-headlines-to-local-headaches 279

Kandasamy N, Abdelwahed S, Hayes J (2004) Self-optimization in computer systems via on-line control: Application to power management. In: Autonomic Computing, 2004. Proceedings. International Conference on, IEEE, pp 54–61 26, 29, 33

Kaner C, Falk J, Nguyen HQ (1999) Testing computer software. John Wiley & Sons 12

Kantz H, Schreiber T (2003) Nonlinear time series analysis, vol 7. Cambridge university press 72

Karystinos GN, Pados DA (2000) On overfitting, generalization, and randomly expanded training sets. Neural Networks, IEEE Transactions on 11(5):1050–1057 194

Kaur S, Verma A (2012) An efficient approach to genetic algorithm for task scheduling in cloud computing environment. International Journal of Information Technology and Computer Science (IJITCS) 4(10):74–79 198

Keller A, Kar G (2000) Dynamic dependencies in application service management. IBM Research Report RC 21770 (97933) 29, 96

Keller A, Kar G (2001) Determining service dependencies in distributed systems. In: Communications, 2001. ICC 2001. IEEE International Conference on, IEEE, vol 7, pp 2084–2088 73

Keller A, Ludwig H (2003) The wsla framework: Specifying and monitoring service level agreements for web services. Journal of Network and Systems Management 11(1):57–81 17

Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of ICNN'95-International Conference on Neural Networks, IEEE, vol 4, pp 1942–1948 114, 205

Keogh E (2002) Exact indexing of dynamic time warping. In: Proceedings of the 28th international conference on Very Large Data Bases, VLDB Endowment, pp 406–417 72, 75

Keogh E, Chakrabarti K, Pazzani M, Mehrotra S (2001a) Dimensionality reduction for fast similarity search in large time series databases. Knowledge and information Systems 3(3):263–286 72

Keogh E, Chakrabarti K, Pazzani M, Mehrotra S (2001b) Locally adaptive dimensionality reduction for indexing large time series databases. In: ACM SIGMOD Record, ACM, vol 30, pp 151–162 72

Keogh EJ, Pazzani MJ (2000) Scaling up dynamic time warping for datamining applications. In: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, pp 285–289 72

Kephart JO, Chess DM (2003) The vision of autonomic computing. Computer 36(1):41–50 28

Keranen A (2019) "too many requests" response code for the constrained application protocol. Tech. Rep. 8516, The Internet Engineering Task Force (IETF) - RFC Editor, URL https://www.rfc-editor.org/info/rfc8516 264

Kertesz A (2014) Characterizing cloud federation approaches. In: Cloud Computing, Springer, pp 277–296 265

Khan A, Sim H, Vazhkudai SS, Butt AR, Kim Y (2021) An analysis of system balance and architectural trends based on top500 supercomputers. In: The International Conference on High Performance Computing in Asia-Pacific Region, pp 11–22 279

Khuri S, Bäck T, Heitkötter J (1994) The zero/one multiple knapsack problem and genetic algorithms. In: Proceedings of the 1994 ACM symposium on Applied computing, pp 188–193 198

Kıcıman E (2005) Using statistical monitoring to detect failures in internet services. PhD thesis, stanford university 73

Kiczales G, Lamping J, Mendhekar A, Maeda C, Lopes C, Loingtier J, Irwin J (1997) Aspect-oriented programming. ECOOP'97—Object-Oriented Programming pp 220–242 65, 135, 237

Kim G, Humble J, Debois P, Willis J (2016) The DevOps Handbook:: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. IT Revolution 18, 220, 292

Kim KH, Beloglazov A, Buyya R (2009) Power-aware provisioning of cloud resources for real-time services. In: Proceedings of the 7th International Workshop on Middleware for Grids, Clouds and e-Science, ACM, p 1 37, 283

Kim KH, Beloglazov A, Buyya R (2011) Power-aware provisioning of virtual machines for real-time cloud services. Concurrency and Computation: Practice and Experience 23(13):1491–1505 37

Kiran M, Murphy P, Monga I, Dugan J, Baveja SS (2015) Lambda architecture for cost-effective batch and speed big data processing. In: Big Data (Big Data), 2015 IEEE International Conference on, IEEE, pp 2785–2792 35, 171

Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. science 220(4598):671–680 205

Knaster R, Leffingwell D (2017) SAFe 4.0 distilled: applying the Scaled Agile Framework for Lean software and systems engineering. Addison-Wesley Professional 13

Kneller M (2010) Executive briefing: the benefits of itil®. Stationery Office, London 296

Kniberg H, Ivarsson A (2012) Scaling agile @ spotify. online, UCVOF URL http://www.agileleanhouse.com/lib/lib/People/HenrikKniberg/SpotifyScaling.pdf 229

Kocaoglu M, Shanmugam K, Bareinboim E (2017) Experimental design for learning causal graphs with latent variables. Advances in Neural Information Processing Systems 30 263

Kocaoglu M, Jaber A, Shanmugam K, Bareinboim E (2019) Characterization and learning of causal graphs with latent variables from soft interventions. Advances in Neural Information Processing Systems 32 263

Koetsier J (2016) Evaluation of javascript frameworks for the development of a web-based user interface for vampires. Informatica—Universiteit van Amsterdam 239

Kolb S, Röck C (2016) Unified cloud application management. In: 2016 IEEE World Congress on Services (SERVICES), IEEE, pp 1–8 17

Konshin K (2018) Next.js Quick Start Guide: Server-side rendering done right. Packt Publishing Ltd 239

Koomey J, Brill K, Turner P, Stanley J, Taylor B (2007) A simple model for determining true total cost of ownership for data centers. Uptime Institute White Paper, Version 2:2007 279

Koomey J, et al (2011) Growth in data center electricity use 2005 to 2010. A report by Analytical Press, completed at the request of The New York Times 9(2011):161 6, 279

Koomey JG (2008) Worldwide electricity used in data centers. Environmental research letters 3(3):034,008 6, 279

Korchak R, Rodman R (2001) ebusiness adoption among us small manufacturers and the role of manufacturing extension. Economic Development Review 17(3):20 222

Kowall J, Cappelli W (2012) Magic quadrant for application performance monitoring. Gartner Research ID:G00232180 26, 129

Krancher O, Luther P, Jost M (2018) Key affordances of platform-as-a-service: self-organization and continuous feedback. Journal of Management Information Systems 35(3):776–812 290

Kratzke N (2014) A lightweight virtualization cluster reference architecture derived from open source paas platforms. Open Journal of Mobile Computing and Cloud Computing 1(2):17–30 290

Kusic D, Kephart J, Hanson J, Kandasamy N, Jiang G (2009) Power and performance management of virtualized computing environments via lookahead control. Cluster computing 12(1):1–15 26, 30

Kwak YH, Anbari FT (2006) Benefits, obstacles, and future of six sigma approach. Technovation 26(5-6):708–715 296

Kyriakidis A, Maniatis K (2016) The Majesty of Vue. js. Packt Publishing Ltd 239

Kyriazis D (2013) Cloud computing service level agreements–exploitation of research results. European Commission Directorate General Communications Networks Content and Technology Unit, Tech Rep 5:29 58

Laddad R (2009) Aspectj in action: enterprise AOP with spring applications. Manning Publications Co. 135

Laganà D, Mastroianni C, Meo M, Renga D (2018) Reducing the operational cost of cloud data centers through renewable energy. Algorithms 11(10):145 283

Larman C (1998) Applying UML and patterns, vol 2. Prentice Hall Englewood Cliffs, NJ 238

Larman C, Vodde B (2016) Large-scale scrum: More with LeSS. Addison-Wesley Professional 13, 220, 229

Latecki LJ, Megalooikonomou V, Wang Q, Lakaemper R, Ratanamahatana CA, Keogh E (2005) Elastic partial matching of time series. In: Knowledge Discovery in Databases: PKDD 2005, Springer, pp 577–584 72

Lattimore T, Szepesvári C (2020) Bandit algorithms. Cambridge University Press 263

Lazowska ED, Zahorjan J, Graham GS, Sevcik KC (1984) Quantitative system performance: computer system analysis using queueing network models. Prentice-Hall, Inc. 108, 110

Lechler T, Claassen S, Goebel J, Wueppen P, Meyer R, Joschko P, Broder F, Unkrig M, Mentz C, Knaak N, Kiesel G, Janz T (2014) DESMO-J: A framework for discrete-event modeling and simulation. URL http://desmoj.sourceforge.net/ 108, 162

Lee M, Chen KT, Liao CY, Gu SS, Siang GY, Chou YC, Chen HY, Le J, Hong RC, Wang ZY, et al (2018) Extremely steep switch of negative-capacitance nanosheet gaa-fets and finfets. In: 2018 IEEE International Electron Devices Meeting (IEDM), IEEE, pp 31–8 276

Lee S, Bareinboim E (2018) Structural causal bandits: where to intervene? Advances in Neural Information Processing Systems 31 263

Lee S, Bareinboim E (2019) Structural causal bandits with non-manipulable variables. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol 33, pp 4164–4172 263

Lee S, Bareinboim E (2020) Characterizing optimal mixed policies: Where to intervene and what to observe. Advances in neural information processing systems 33:8565–8576 263

Lee YC, Zomaya AY (2012) Energy efficient utilization of resources in cloud computing systems. The Journal of Supercomputing 60(2):268–280 37

Leffingwell D (2018) SAFe 4.5 Reference Guide: Scaled Agile Framework for Lean Enterprises. Addison-Wesley Professional 13, 220, 229

Lehrig S, Eikerling H, Becker S (2015) Scalability, elasticity, and efficiency in cloud computing: A systematic literature review of definitions and metrics. In: Proceedings of the 11th international ACM SIGSOFT conference on quality of software architectures, pp 83–92 22

Lei K, Ma Y, Tan Z (2014) Performance comparison and evaluation of web development technologies in php, python, and node. js. In: 2014 IEEE 17th international conference on computational science and engineering, IEEE, pp 661–668 242

Leitner P, Wittern E, Spillner J, Hummer W (2019) A mixed-method empirical study of function-as-a-service software development in industrial practice. Journal of Systems and Software 149:340–359 291

Lenski D (2020) Top500 supercomputers performance data collection. Dan Lenski URL https://github.com/karlrupp/microprocessor-trend-data 280

Leontiou N, Dechouniotis D, Denazis S (2010) Adaptive admission control of distributed cloud services. In: Network and Service Management (CNSM), 2010 International Conference on, IEEE, pp 318–321 159

Levin A, Garion S, Kolodner EK, Lorenz DH, Barabash K, Kugler M, McShane N (2019) Aiops for a cloud object storage service. In: 2019 IEEE International Congress on Big Data (BigDataCongress), IEEE, pp 165–169 18

Lewis J, Fowler M (2014) Microservices. MartinFowlercom URL https://martinfowler.com/articles/microservices.html 2, 23, 34, 218, 219

Li EY, Chen HG, Cheung W (2000) Total quality management in software development process. The Journal of Quality Assurance Institute 14(1):4–6 294

Li J, Qiu M, Niu JW, Chen Y, Ming Z (2010) Adaptive resource allocation for preemptable jobs in cloud systems. In: 2010 10th International Conference on Intelligent Systems Design and Applications, IEEE, pp 31–36 198, 263

Li Q, Hao Q, Xiao L, Li Z (2009) Adaptive management of virtualized resources in cloud computing using feedback control. In: 2009 First International Conference on Information Science and Engineering, IEEE, pp 99–102 27, 30

Li Z, Zhang H, O'Brien L, Jiang S, Zhou Y, Kihl M, Ranjan R (2016) Spot pricing in the cloud ecosystem: A comparative investigation. Journal of Systems and Software 114:1–19 276

Lim HC, Babu S, Chase JS, Parekh SS (2009) Automated control in cloud computing: challenges and opportunities. In: Proceedings of the 1st workshop on Automated control for datacenters and clouds, pp 13–18 28, 30

Lin M, Wierman A, Andrew LL, Thereska E (2012) Dynamic right-sizing for power-proportional data centers. IEEE/ACM Transactions on Networking 21(5):1378–1391 30

Lin W, Liang C, Wang JZ, Buyya R (2014) Bandwidth-aware divisible task scheduling for cloud computing. Software: Practice and Experience 44(2):163–174 198

Liu B, Wang L, Jin YH (2007) An effective pso-based memetic algorithm for flow shop scheduling. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 37(1):18–27 198

Liu C, Cheng J, Wang Y, Gao S (2016) Time performance optimization and resource conflicts resolution for multiple project management. IEICE Transations on Information and Systems 99(3):650–660 230

Liu L, Wang H, Liu X, Jin X, He WB, Wang QB, Chen Y (2009) Greencloud: a new architecture for green data center. In: Proceedings of the 6th international conference industry session on Autonomic computing and communications industry session, pp 29–38 283

Liu N, Li Z, Xu J, Xu Z, Lin S, Qiu Q, Tang J, Wang Y (2017) A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning. In: 2017 IEEE 37th international conference on distributed computing systems (ICDCS), IEEE, pp 372–382 158

Liu S, Quan G, Ren S (2010) On-line scheduling of real-time services for cloud computing. In: Services (SERVICES-1), 2010 6th World Congress on, IEEE, pp 459–464 38, 157

Liu X, Heo J, Sha L, Zhu X (2006) Adaptive control of multi-tiered web applications using queueing predictor. In: 2006 IEEE/IFIP Network Operations and Management Symposium NOMS 2006, IEEE, pp 106–114 27, 29

Llorente IM (2017) The limits to cloud price reduction. IEEE Cloud Computing 4(3):8–13 276

Lomet D (2018) Cost/performance in modern data stores: How data caching systems succeed. In: Proceedings of the 14th International Workshop on Data Management on New Hardware, pp 1–10 277

Lorenczik S, Kim S, Wanner B, Bermudez Menendez JM, Remme U, Hasegawa T, Keppler JH, Mir L, Sousa G, Berthelemy M, et al (2020) Projected costs of generating electricity-2020 edition. Tech. rep., Organisation for Economic Co-Operation and Development 280, 283

Lorido-Botran T, Miguel-Alonso J, Lozano JA (2014) A review of auto-scaling techniques for elastic applications in cloud environments. Journal of grid computing 12(4):559–592 159

Lu C, Stankovic JA, Abdelzaher TF, Tao G, Son SH, Marley M (2000) Performance specifications and metrics for adaptive real-time systems. In: Proceedings 21st IEEE Real-Time Systems Symposium, IEEE, pp 13–23 27, 29

Lu Y, Abdelzaher T, Lu C, Tao G (2002) An adaptive control framework for qos guarantees and its application to differentiated caching. In: Quality of Service, 2002. Tenth IEEE International Workshop on, IEEE, pp 23–32 25, 27, 29, 263

Lu Y, Abdelzaher T, Lu C, Sha L, Liu X (2003) Feedback control with queueing-theoretic prediction for relative delay guarantees in web servers. In: Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings. The 9th IEEE, IEEE, pp 208–217 25, 29, 33

Lwakatare LE, Kuvaja P, Oivo M (2016) Relationship of devops to agile, lean and continuous deployment. In: International conference on product-focused software process improvement, Springer, pp 399–415 228

Lynn T, Rosati P, Lejeune A, Emeakaroha V (2017) A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms. In: 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, pp 162–169 284, 291

Lynn T, Mooney JG, Rosati P, Fox G (2020) Measuring the business value of cloud computing. Springer Nature 276

Madni SHH, Abd Latiff MS, Coulibaly Y, et al (2016) Resource scheduling for infrastructure as a service (iaas) in cloud computing: Challenges and opportunities. Journal of Network and Computer Applications 68:173–200 291

Madsen M, Lhoták O, Tip F (2017) A model for reasoning about javascript promises. Proceedings of the ACM on Programming Languages 1(OOPSLA):1–24 242

Malawski M, Juve G, Deelman E, Nabrzyski J (2015) Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. Future Generation Computer Systems 48:1–18 158, 291

Malawski M, Figiela K, Gajek A, Zima A (2017) Benchmarking heterogeneous cloud functions. In: European Conference on Parallel Processing, Springer, pp 415–426 290

Malmodin J, Lundén D (2018) The energy and carbon footprint of the global ict and e&m sectors 2010–2015. Sustainability 10(9):3027 283

Mangalaraj G, Singh A, Taneja A (2014) It governance frameworks and cobit-a literature review. In: Twentieth Americas Conference on Information Systems, Savannah, Georgia, USA 296

Manipatruni S, Nikonov DE, Young IA (2018) Beyond cmos computing with spin and polarization. Nature Physics 14(4):338–343 277

Manner J, Endreß M, Heckel T, Wirtz G (2018) Cold start influencing factors in function as a service. In: 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), IEEE, pp 181–188 290

Manvi SS, Shyam GK (2014) Resource management for infrastructure as a service (iaas) in cloud computing: A survey. Journal of network and computer applications 41:424–440 28

Mao M, Humphrey M (2011) Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In: SC'11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, pp 1–12 30

Mao M, Humphrey M (2012) A performance study on the vm startup time in the cloud. In: Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, IEEE, pp 423–430 34, 155

Mao M, Li J, Humphrey M (2010) Cloud auto-scaling with deadline and budget constraints. In: Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on, IEEE, pp 41–48 34, 156

Martello S (1990) Knapsack problems: algorithms and computer implementations. Wiley-Interscience series in discrete mathematics and optimization 198, 204

Martin RC (2000a) Design principles and design patterns. Object Mentor 1(34):597 238

Martin RC (2000b) More C++ gems. Cambridge University Press 238

Martin RC (2002) Agile software development: principles, patterns, and practices. Prentice Hall 2, 292

Martin RC (2009) Clean code: a handbook of agile software craftsmanship. Pearson Education 12, 220

Martin RC, Grenning J, Brown S (2018) Clean architecture: a craftsman's guide to software structure and design. s 31, Prentice Hall 238

Martins P, Abbasi M, Sá F (2019) A study over nosql performance. In: World Conference on Information Systems and Technologies, Springer, pp 603–611 246

Masaaki I (1986) Kaizen: The key to Japan's competitive success. McGraw-Hill/Irwin 294

Masanet E, Shehabi A, Lei N, Smith S, Koomey J (2020) Recalibrating global data center energy-use estimates. Science 367(6481):984–986 283

Masood A, Hashmi A (2019) Aiops: Predictive analytics & machine learning in operations. In: Cognitive Computing Recipes, Springer, pp 359–382 18

Matchett C, Doheny R, Gonzalez K (2017) Magic quadrant for it service management tools 266, 295

Matusiak M (2009) Strategies for aspect oriented programming in python 238

McBreen P, Hendrickson M (2002) Software craftsmanship: The new imperative. Addison-Wesley Professional 220

McGill R, Tukey JW, Larsen WA (1978) Variations of box plots. The American Statistician 32(1):12–16 114

McGrath G, Brenner PR (2017) Serverless computing: Design, implementation, and performance. In: 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), IEEE, pp 405–410 283

Mehdi NA, Mamat A, Amer A, Abdul-Mehdi ZT (2011) Minimum completion time for power-aware scheduling in cloud computing. In: 2011 Developments in E-systems Engineering, IEEE, pp 484–489 23

Mendoza MA, Amistadi HR (2018) Machine learning for anomaly detection on vm and host performance metrics. Tech. rep., MITRE CORP BEDFORD MA BEDFORD 22

Messina F, Pappalardo G, Santoro C, Rosaci D, Sarné GM (2014) An agent based negotiation protocol for cloud service level agreements. In: WETICE Conference (WETICE), 2014 IEEE 23rd International, IEEE, pp 161–166 158

Messina F, Pappalardo G, Santoro C, Rosaci D, Sarné GM (2016) A multi-agent protocol for service level agreement negotiation in cloud federations. International Journal of Grid and Utility Computing 7(2):101–112 158

Michael S, Michael B, Thomas S (2019) It service management frameworks compared–simplifying service portfolio management. In: 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), IEEE, pp 421–427 296

Michalewicz Z (1996) Genetic algorithms+ data structures= evolution programs. Springer 114, 116, 205

Miller GA (1956) The magical number seven, plus or minus two: Some limits on our capacity for processing information. Psychological review 63(2):81 228

Miller RB (1968) Response time in man-computer conversational transactions. In: Proceedings of the December 9-11, 1968, fall joint computer conference, part I, ACM, pp 267–277 174

Minella M (2011) Pro Spring Batch. Apress 250

Mitchell M (1998) An introduction to genetic algorithms. MIT press 114, 205

Mocuta A, Weckx P, Demuynck S, Radisic D, Oniki Y, Ryckaert J (2018) Enabling cmos scaling towards 3nm and beyond. In: 2018 IEEE Symposium on VLSI Technology, IEEE, pp 147–148 276

Moen R, Norman C (2006) Evolution of the pdca cycle 294

Mohammadi M, Bazhirov T (2018) Comparative benchmarking of cloud computing vendors with high performance linpack. In: Proceedings of the 2nd International Conference on High Performance Compilation, Computing and Communications, pp 1–5 280, 282

Mohana R, Thangaraj P (2013) Machine learning approaches in improving service level agreement-based admission control for a software-as-a-service provider in cloud. Journal of Computer Science 9:1283–1294 158

Mohanani R, Salman I, Turhan B, Rodríguez P, Ralph P (2018) Cognitive biases in software engineering: a systematic mapping study. IEEE Transactions on Software Engineering 46(12):1318–1339 226

Mohanty SK, Premsankar G, Di Francesco M, et al (2018) An evaluation of open source serverless computing frameworks. In: CloudCom, pp 115–120 290, 291

Monteiro MP, Fernandes JM (2004) Pitfalls of aspectj implementations of some of the gang-of-four design patterns. In: Aspect-Oriented Software Development Workshop (DSO), 2, Universidad de Extremadura, Málaga, Spain 238

Moon Y, Yu H, Gil JM, Lim J (2017) A slave ants based ant colony optimization algorithm for task scheduling in cloud computing environments. Human-centric Computing and Information Sciences 7(1):1–10 198

Moore D, Budd R, Wright W (2008) Professional Python Frameworks: Web 2.0 Programming with Django and Turbogears. John Wiley & Sons 239

Moore GE, et al (1965) Cramming more components onto integrated circuits 23, 276

Mora M, Marx-Gomez J, Wang F, Diaz O (2021) Agile it service management frameworks and standards: A review. Advances in Software Engineering, Education, and e-Learning pp 921–936 296

Mörchen F (2006) Time series knowlegde mining. University of Marburg 5

Moreno IS, Yang R, Xu J, Wo T (2013) Improved energy-efficiency in cloud datacenters with interference-aware virtual machine placement. In: Autonomous Decentralized Systems (ISADS), 2013 IEEE Eleventh International Symposium on, IEEE, pp 1–8 37

Mormul M, Stach C (2020) A context model for holistic monitoring and management of complex it environments. In: 2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), IEEE, pp 1–6 18

Muggleton S (1999) Inductive logic programming: issues, results and the challenge of learning language in logic. Artificial Intelligence 114(1):283–296 130, 272

Muggleton S, De Raedt L (1994) Inductive logic programming: Theory and methods. The Journal of Logic Programming 19:629–679 130, 272

Mullen KM, Ardia D, Gil DL, Windover D, Cline J (2011) DEoptim: An R package for global optimization by differential evolution. Journal of Statistical Software 40 112, 116

Muppala S, Chen G, Zhou X (2014) Multi-tier service differentiation by coordinated learning-based resource provisioning and admission control. Journal of Parallel and Distributed Computing 74(5):2351–2364 158, 240

Myers GJ, Badgett T, Thomas TM, Sandler C (2004) The art of software testing, vol 2. Wiley Online Library 12

Mytton D (2020) Hiding greenhouse gas emissions in the cloud. Nature Climate Change 10(8):701–701 283

Mégnin C (2003) jannealer: Java simulated annealing package. URL `http://jannealer.sourceforge.net/` 205

Nachira F (2002) Towards a network of digital business ecosystems fostering the local development. Tech. rep., European Commission 222

Nachira F, Dini P, Nicolai A (2007a) A network of digital business ecosystems for europe: roots, processes and perspectives. Tech. rep., European Commission, Bruxelles, Introductory Paper 222

Nachira F, Nicolai A, Dini P, Le Louarn M, León LR (2007b) Digital business ecosystems. European Commission 222

Napper J, Bientinesi P (2009) Can cloud computing reach the top500? In: Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop, pp 17–20 280

Narkhede N, Shapira G, Palino T (2017) Kafka: the definitive guide: real-time data and stream processing at scale. " O'Reilly Media, Inc." 244

Nedelcu C (2013) Nginx HTTP Server. Packt Publishing 239

Nedjah N, Alba E, de Macedo Mourelle L (2006) Parallel Evolutionary Computations. Springer 125

Nelder JA, Mead R (1965) A simplex method for function minimization. The Computer Journal 7(4):308–313 112, 273

Netcraft (2021) Web server survey. URL https://news.netcraft.com/archives/2021/10/15/october-2021-web-server-survey.html 239

Neto P (2011) Demystifying cloud computing. In: Proceeding of doctoral symposium on informatics engineering, Citeseer, vol 24, pp 16–21 276

Neugebauer R, McAuley D (2001) Energy is just another resource: Energy accounting and energy pricing in the Nemesis OS. In: Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on, IEEE, pp 67–72 97

Nguyen HV, Choi Y (2010) Proactive detection of ddos attacks utilizing k-nn classifier in an anti-ddos framework. International Journal of Electrical, Computer, and Systems Engineering 4(4):247–252 249

Noor A, Jha DN, Mitra K, Jayaraman PP, Souza A, Ranjan R, Dustdar S (2019) A framework for monitoring microservice-oriented cloud applications in heterogeneous virtualization environments. In: 2019 IEEE 12th international conference on cloud computing (CLOUD), IEEE, pp 156–163 17

Norton WB (2016) Cloud interconnections 219

Ohno T (1988) Toyota production system: beyond large-scale production. crc Press 294

Omara FA, Arafa MM (2009) Genetic algorithms for task scheduling problem. In: Foundations of Computational Intelligence Volume 3, Springer, pp 479–507 198

Opara-Martins J, Sahandi R, Tian F (2016) Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective. Journal of Cloud Computing 5(1):1–18 291

Osman IH, Potts C (1989) Simulated annealing for permutation flow-shop scheduling. Omega 17(6):551–557 198

Ostermann S, Plankensteiner K, Prodan R, Fahringer T (2011) GroudSim: An event-based simulation framework for computational grids and clouds. In: Euro-Par 2010 Parallel Processing Workshops, Springer, pp 305–313 109

Padala P, Shin KG, Zhu X, Uysal M, Wang Z, Singhal S, Merchant A, Salem K (2007a) Adaptive control of virtualized resources in utility computing environments. In: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, pp 289–302 29

Padala P, Zhu X, Wang Z, Singhal S, Shin KG, et al (2007b) Performance evaluation of virtualization technologies for server consolidation. HP Labs Tec Report 137 233

Page B, Kreutzer W, Gehlsen B (2005) The Java simulation handbook: simulating discrete event systems with UML and Java. Shaker Verlag 108, 110, 162

Palade A, Kazmi A, Clarke S (2019) An evaluation of open source serverless computing frameworks support at the edge. In: 2019 IEEE World Congress on Services (SERVICES), IEEE, vol 2642, pp 206–211 291

Paraiso F, Haderer N, Merle P, Rouvoy R, Seinturier L (2012) A federated multi-cloud paas infrastructure. In: 2012 IEEE Fifth International Conference on Cloud Computing, IEEE, pp 392–399 265

Pardo C, Pino FJ, Garcia F (2016) Towards an integrated management system (ims), harmonizing the iso/iec 27001 and iso/iec 20000-2 standards. International Journal of Software Engineering and Its Applications 10(9):217–230 296

Parekh S, Gandhi N, Hellerstein J, Tilbury D, Jayram T, Bigus J (2002) Using control theory to achieve service level objectives in performance management. Real-Time Systems 23(1):127–141 25, 27, 29, 58

Park BJ, Choi HR, Kim HS (2003) A hybrid genetic algorithm for the job shop scheduling problems. Computers & industrial engineering 45(4):597–613 198

Park K, Pai VS (2006) Comon: a mostly-scalable monitoring system for planetlab. ACM SIGOPS Operating Systems Review 40(1):65–74 284

Park L, Baek J, Woon-Ki Hong J (2001) Management of service level agreements for multimedia internet service using a utility model. Communications Magazine, IEEE 39(5):100–106 26, 29

Paschos GS, Iosifidis G, Tao M, Towsley D, Caire G (2018) The role of caching in future communication systems and networks. IEEE Journal on Selected Areas in Communications 36(6):1111–1125 278

Patel P, Ranabahu A, Sheth A (2009) Service level agreement in cloud computing. In: Cloud Workshops at OOPSLA 3, 26

Paulsson V, Emeakaroha VC, Morrison J, Lynn T (2020) Cloud service brokerage: Exploring characteristics and benefits of b2b cloud marketplaces. In: Measuring the Business Value of Cloud Computing, Palgrave Macmillan, Springer Nature, pp 57–71 265

Pautasso C, Zimmermann O, Amundsen M, Lewis J, Josuttis N (2017a) Microservices in practice, part 1: Reality check and service design. IEEE software 34(01):91–98 219

Pautasso C, Zimmermann O, Amundsen M, Lewis J, Josuttis N (2017b) Microservices in practice, part 2: Service integration and sustainability. IEEE Software 34(02):97–104 219

Pavlidis T, Horowitz SL (1974) Segmentation of plane curves. Computers, IEEE Transactions on 100(8):860–870 72

Pearl J (2000) Causality: models, reasoning and inference, vol 29. Cambridge Univ Press 32, 68, 69, 71, 72, 74, 77, 263, 272

Pearl J (2010) The foundations of causal inference. Sociological Methodology 40(1):75–149 86, 263

Pearl J, Mackenzie D (2018) The book of why: the new science of cause and effect. Basic Books 28, 71, 85, 87, 263

Pearl J, et al (2009) Causal inference in statistics: An overview. Statistics surveys 3:96–146 28, 68, 71

Pedersen MEH (2010a) Good parameters for particle swarm optimization. Hvass Lab, Copenhagen, Denmark, Tech Rep HL1001 114

Pedersen MEH (2010b) Tuning & simplifying heuristical optimization. PhD thesis, University of Southampton 114

Peng Z, Cui D, Zuo J, Li Q, Xu B, Lin W (2015) Random task scheduling scheme based on reinforcement learning in cloud computing. Cluster computing 18(4):1595–1607 195

Pérez PA, Antonio S, Sala A (2004) Multivariable control systems: an engineering approach. Springer Verlag 32, 68

Perez Q, Le Borgne A, Urtado C, Vauttier S (2019) An empirical study about software architecture configuration practices with the java spring framework. In: SEKE: Software Engineering and Knowledge Engineering, vol 2019, pp 465–468 239

Perrey R, Lycett M (2003) Service-oriented architecture. In: 2003 Symposium on Applications and the Internet Workshops, 2003. Proceedings., IEEE, pp 116–119 2

Perros HG, Elsayed KM (1996) Call admission control schemes: A review. IEEE Communications Magazine 34(11):82–91 65

Pesce M (2021) Cloud computing's coming energy crisis: The cloud's electricity needs are growing unsustainably. IEEE Spectrum 58(07):22–22 279

Peters J, Janzing D, Schölkopf B (2017) Elements of causal inference: foundations and learning algorithms. The MIT Press 71

Peterson L, Muir S, Roscoe T, Klingaman A (2006) Planetlab architecture: An overview. PlanetLab Consortium May 1(15):1–4 284

Phan LT, Zhang Z, Zheng Q, Loo BT, Lee I (2011) An empirical analysis of scheduling techniques for real-time cloud-based data processing. In: Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on, IEEE, pp 1–8 38

Pisinger D (1995) Algorithms for knapsack problems. PhD thesis, Department of Computer Science, University of Copenhagen 198, 204

Plant RT (2000) eCommerce: formulation of strategy. Prentice Hall Professional 221

Plummer DC, Bittman TJ, Austin T, Cearley DW, Smith DM (2008a) Cloud computing: Defining and describing an emerging phenomenon. Gartner, June 17:1–9 222

Plummer DC, Cearley DW, Smith DM (2008b) Cloud computing confusion leads to opportunity. Gartner Report 222

Poggi N, Moreno T, Berral JL, Gavalda R, Torres J (2009) Self-adaptive utility-based web session management. Computer Networks 53(10):1712–1721 240

Poli R, Kennedy J, Blackwell T (2007) Particle swarm optimization. Swarm intelligence 1(1):33–57 114, 205

Polo J, Castillo C, Carrera D, Becerra Y, Whalley I, Steinder M, Torres J, Ayguadé E (2011) Resource-aware adaptive scheduling for mapreduce clusters. In: Middleware 2011, Springer, pp 187–207 101

Polycarpou MM (1996) Stable adaptive neural control scheme for nonlinear systems. IEEE Transactions on Automatic control 41(3):447–451 27

Popovici A, Alonso G, Gross T (2003) Just-in-time aspects: efficient dynamic weaving for java. In: Proceedings of the 2nd international conference on Aspect-oriented software development, pp 100–109 237

Poppendieck M, Poppendieck T (2003) Lean Software Development: An Agile Toolkit: An Agile Toolkit. Addison-Wesley 2, 12, 219, 220, 231, 269, 275, 287, 289, 292

Pourmajidi W, Steinbacher J, Erwin T, Miranskyy A (2018) On challenges of cloud monitoring. arXiv preprint arXiv:180605914 17

Powley W, Martin P, Ogeer N, Tian W (2005) Autonomic buffer pool configuration in postgresql. In: Systems, Man and Cybernetics, 2005 IEEE International Conference on, IEEE, vol 1, pp 53–58 26, 29

Prabhakaran S, Neumann M, Rinke S, Wolf F, Gupta A, Kale LV (2015) A batch system with efficient adaptive scheduling for malleable and evolving applications. In: 2015 IEEE international parallel and distributed processing symposium, IEEE, pp 429–438 198

Prasad P, Rich C (2018) Market guide for aiops platforms. Retrieved March 12:2020 17, 18, 224, 294

Prusty N (2015) Learning ECMAScript 6. Packt Publishing 242

Przybyłek A (2018) An empirical study on the impact of aspectj on software evolvability. Empirical Software Engineering 23(4) 238

Pultorak D, Henry C (2008) Microsoft Operations Framework 4.0-A Pocket Guide. Van Haren 296

Putnam LH (1978) A general empirical solution to the macro software sizing and estimating problem. IEEE transactions on Software Engineering SE-4(4):345–361 230

Pyle D (1999) Data preparation for data mining, vol 1. Morgan Kaufmann 75, 104

Pyzdek T, Keller P (2014) Six sigma handbook. McGraw-Hill Education 296

Qian L, Luo Z, Du Y, Guo L (2009) Cloud computing: An overview. In: IEEE International Conference on Cloud Computing, Springer, pp 626–631 275

R Core Team (2014) Fitting Linear Models, lm stats, R Documentation. URL http://stat.ethz.ch/R-manual/R-patched/library/stats/html/lm.html 102

Raghavan B, Vishwanath K, Ramabhadran S, Yocum K, Snoeren AC (2007) Cloud control with distributed rate limiting. ACM SIGCOMM Computer Communication Review 37(4):337–348 28, 29

Raisinghani MS, Ette H, Pierce R, Cannon G, Daripaly P (2005) Six sigma: concepts, tools, and applications. Industrial management & Data systems 296

Ramezani F, Lu J, Hussain FK (2014) Task-based system load balancing in cloud computing using particle swarm optimization. International journal of parallel programming 42(5):739–754 198

Ran Y, Yang J, Zhang S, Xi H (2015) Dynamic iaas computing resource provisioning strategy with qos constraint. IEEE Transactions on Services Computing 10(2):190–202 291

Ranaldo N, Zimeo E (2016) Capacity-driven utility model for service level agreement negotiation of cloud services. Future Generation Computer Systems 55:186–199 158

Rasmussen N (2011) Determining total cost of ownership for data center and network room infrastructure. Relatório técnico, Schneider Electric, Paris 8 279, 284

Raymond E (1999) The cathedral and the bazaar. Knowledge, Technology & Policy 12(3):23–49 12

Razavieh A, Zeitzoff P, Nowak EJ (2019) Challenges and limitations of cmos scaling for finfet and beyond architectures. IEEE Transactions on Nanotechnology 18:999–1004 277

Reinsch CH (1967) Smoothing by spline functions. Numerische Mathematik 10(3):177–183 75

Reis E (2011) The lean startup. New York: Crown Business 219, 220, 275, 295

Reussner R, Becker S, Burger E, Happe J, Hauck M, Koziolek A, Koziolek H, Krogmann K, Kuperberg M (2011) The palladio component model. Tech. rep., KIT, Fakultät für Informatik 213

Reussner RH, Becker S, Happe J, Heinrich R, Koziolek A (2016) Modeling and simulating software architectures: The Palladio approach. MIT Press 213

Reynolds J (2000) ecommerce: a critical review. International Journal of Retail & Distribution Management 221

Ripley BD (2012) LOWEES Scatter Plot Smoothing, R Statistical Data Analysis, R Documentation. URL http://stat.ethz.ch/R-manual/R-patched/library/stats/html/lowess.html 151

Roberts DB (1999) Practical analysis for refactoring. PhD thesis, University of Illinois at Urbana-Champaign 13, 234

Rocher G, Brown JS (2009) The definitive guide to grails. Apress 239

Rodríguez P, Mikkonen K, Kuvaja P, Oivo M, Garbajosa J (2013) Building lean thinking in a telecom software development organization: strengths and challenges. In: Proceedings of the 2013 international conference on software and system process, pp 98–107 228

Rosati P, Lynn T (2020) Measuring the business value of infrastructure migration to the cloud. In: Measuring the Business Value of Cloud Computing, Palgrave Macmillan, Springer Nature, pp 19–37 232, 275

Rosenblum M, Garfinkel T (2005) Virtual machine monitors: Current technology and future trends. Computer 38(5):39–47 218

Rubin DB (1990) Comment: Neyman (1923) and causal inference in experiments and observational studies. Statistical Science 5(4):472–480 262

Ruby S, Copeland DB, Thomas D (2020) Agile Web Development with Rails 6. Pragmatic bookshelf 239

Ruiz R, Maroto C, Alcaraz J (2006) Two new robust genetic algorithms for the flowshop scheduling problem. Omega 34(5):461–476 198

Rupp K (2020) Years of microprocessor trend data. Karl Rupp URL https://github.com/karlrupp/microprocessor-trend-data 276, 277

Russell S, Norvig P (2002) Artificial intelligence: a modern approach. Prentice Hall, Englewood Cliffs, New Jersey, USA 223

Ruszczyński AP (2006) Nonlinear optimization, vol 13. Princeton University Press 102

Saeys Y, Abeel T, Van de Peer Y (2008) Robust feature selection using ensemble feature selection techniques. Machine Learning and Knowledge Discovery in Databases pp 313–325 136

Saks E (2019) Javascript frameworks: Angular vs react vs vue. Master's thesis, Haaga-Helia University of Applied Sciences, Helsinki, Finland 239

Salehi MA, Javadi B, Buyya R (2012) Preemption-aware admission control in a virtualized grid federation. In: Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on, IEEE, pp 854–861 158

Salvaneschi G, Margara A, Tamburrelli G (2015) Reactive programming: A walkthrough. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, IEEE, vol 2, pp 953–954 243

Sastry V (2020) Cloud Computing Economics For Information Technology industry. Idea Publishing 276

Satman MH (2013) RCaller: A library for calling R from Java. URL https://code.google.com/p/rcaller/ 109

Satman MH (2014) RCaller: A software library for calling R from Java. British Journal of Mathematics & Computer Science 4:2188–2196 109

Satrom B (2018) Choosing the right javascript framework for your next web application. Prog/Kendo UI p 34 239

Savoia A (2011) Pretotype it. Make sure you are building the right it before you build it right 249

Scheffler K, Young S (2002) Automatic learning of dialogue strategy using dialogue simulation and reinforcement learning. In: Proceedings of the second international conference on Human Language Technology Research, Morgan Kaufmann Publishers Inc., pp 12–19 169

Schroeder RG, Linderman K, Liedtke C, Choo AS (2008) Six sigma: Definition and underlying theory. Journal of operations Management 26(4):536–554 296

Sellami M, Yangui S, Mohamed M, Tata S (2013) Paas-independent provisioning and management of applications in the cloud. In: 2013 IEEE Sixth International Conference on Cloud Computing, IEEE, pp 693–700 60

Seppala C, Harris T, Bacon D (2002) Time series methods for dynamic analysis of multiple controlled variables. Journal of Process Control 12(2):257–276 72

Serrano D, Bouchenak S, Kouki Y, Ledoux T, Lejeune J, Sopena J, Arantes L, Sens P (2013) Towards qos-oriented sla guarantees for online cloud services. In: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, IEEE, pp 50–57 28

Serrano D, Bouchenak S, Kouki Y, de Oliveira Jr FA, Ledoux T, Lejeune J, Sopena J, Arantes L, Sens P (2016) Sla guarantees for cloud services. Future Generation Computer Systems 54:233–246 30

Shahin M, Babar MA, Zhu L (2017) Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. IEEE Access 5:3909–3943 12

Shahrad M, Balkind J, Wentzlaff D (2019) Architectural implications of function-as-a-service computing. In: Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, pp 1063–1075 129

Shahrad M, Fonseca R, Goiri Í, Chaudhry G, Batum P, Cooke J, Laureano E, Tresness C, Russinovich M, Bianchini R (2020) Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In: 2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20), pp 205–218 283, 290

Shalf J (2020) The future of computing beyond moore's law. Philosophical Transactions of the Royal Society A 378(2166):20190,061 277

Shao J, Wei H, Wang Q, Mei H (2010) A runtime model based monitoring approach for cloud. In: 2010 IEEE 3rd International Conference on Cloud Computing, IEEE, pp 313–320 238

Sharifian S, Motamedi SA, Akbari MK (2008) A content-based load balancing algorithm with admission control for cluster web servers. Future Generation Computer Systems 24(8):775–787 158

Sharma R, Sood M, Sharma D (2011) Modeling cloud saas with soa and mda. In: International Conference on Advances in Computing and Communications, Springer, pp 511–518 2, 219

Sharma S, Hsu CH, Feng Wc (2006) Making a case for a green500 list. In: Proceedings 20th IEEE International Parallel & Distributed Processing Symposium, IEEE, pp 8–pp 283

Shehabi A, Smith S, Sartor D, Brown R, Herrlin M, Koomey J, Masanet E, Horner N, Azevedo I, Lintner W (2016) United states data center energy usage report. Tech. rep., Lawrence Berkeley National Laboratory, Berkeley, United States 283

Shehabi A, Smith SJ, Masanet E, Koomey J (2018) Data center growth in the united states: decoupling the demand for services from electricity use. Environmental Research Letters 13(12):124,030 279

Shi L, Zhang Z, Robertazzi T (2016) Energy-aware scheduling of embarrassingly parallel jobs and resource allocation in cloud. IEEE Transactions on Parallel and Distributed Systems 28(6):1607–1620 292

Shukur H, Zeebaree SR, Ahmed AJ, Zebari RR, Ahmed O, Tahir BSA, Sadeeq MA (2020) A state of art survey for concurrent computation and clustering of parallel computing for distributed systems. Journal of Applied Science and Technology Trends 1(4):148–154 277

Siddiqui A, Romascanu D, Golovinsky E (2003) Real-time application quality of service monitoring (raqmon) framework. Tech. Rep. 4710, The Internet Engineering Task Force (IETF) - RFC Editor, URL https://www.rfc-editor.org/info/rfc4710 264

Sikora T (2012) Allmon, a generic system collecting and storing metrics used for performance and availability monitoring. URL http://code.google.com/p/allmon/ 133

Sikora TD, Magoulas GD (2012) Neural adaptive control in application service management environment. In: Conference Proceedings EANN 2012 - CCIS: Volume number 311., London, UK, Springer, pp 223–233 9, 258, 271

Sikora TD, Magoulas GD (2013a) Finding relevant dimensions in application service management control: A features selection approach. In: Science and Information Conference (SAI), 2013, IEEE, pp 387–395 32, 272

Sikora TD, Magoulas GD (2013b) Neural adaptive control in application service management environment. Evolving Systems 4(4) pp 267–287 9, 26, 27, 30, 31, 33, 38, 46, 47, 74, 96, 97, 108, 127, 159, 166, 169, 232, 259, 272

Sikora TD, Magoulas GD (2014a) Finding relevant dimensions in application service management control. In: Intelligent Systems for Science and Information, Springer, pp 335–353 10, 32, 68, 73, 96, 259, 261, 273

Sikora TD, Magoulas GD (2014b) Search-guided activity signals extraction in application service management control. In: Computational Intelligence (UKCI), 2014 14th UK Workshop on, IEEE, pp 1–8 112, 176, 259, 263, 274

Sikora TD, Magoulas GD (2015) Evolutionary approaches to signal decomposition in an application service management system. Soft Computing pp 1–22 37, 177, 263

Sikora TD, Magoulas GD (2021) Neural adaptive admission control framework: Sla-driven action termination for real-time application service management. Enterprise Information Systems pp 1–41 30, 163, 191, 274

Sim KM (2013) Complex and concurrent negotiations for multiple interrelated e-markets. IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics 43(1):230–245 156

Singh H, Tyagi S, Kumar P, Gill SS, Buyya R (2021a) Metaheuristics for scheduling of heterogeneous tasks in cloud computing environments: Analysis, performance evaluation, and future directions. Simulation Modelling Practice and Theory 111:102,353 30

Singh J, Singh P, Gill SS (2021b) Fog computing: A taxonomy, systematic review, current trends and research challenges. Journal of Parallel and Distributed Computing 157:56–85 218, 230, 278

Singh S, Chana I (2015) Qrsf: Qos-aware resource scheduling framework in cloud computing. The Journal of Supercomputing 71(1):241–292 158

Singh S, Chana I (2016) A survey on resource scheduling in cloud computing: Issues and challenges. Journal of grid computing 14(2):217–264 127

Singh S, Chana I, Buyya R (2017) Star: Sla-aware autonomic management of cloud resources. IEEE Transactions on Cloud Computing 8(4):1040–1053 30

Sirbi K, Kulkarni PJ (2010) Modularization of enterprise application security through spring aop. International Journal of Computer Science & Communication 1(2):227–231 238

Sironi F, Bartolini DB, Campanoni S, Cancare F, Hoffmann H, Sciuto D, Santambrogio MD (2012) Metronome: operating system level performance management via self-adaptive computing. In: Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE, IEEE, pp 856–865 101

Sjöberg J, Zhang Q, Ljung L, Benveniste A, Delyon B, Glorennec PY, Hjalmarsson H, Juditsky A (1995) Nonlinear black-box modeling in system identification: a unified overview. Automatica 31(12):1691–1724 31

Skinner B (1938) The behavior of organisms: An experimental analysis. Appleton-Century 130

Skinner B (1963) Operant behavior. American Psychologist 18(8):503 130

Smith AE, Coit DW (1997) Penalty functions. Handbook of evolutionary computation 97(1):C5 202

Smith J, Nair R (2005) Virtual machines: versatile platforms for systems and processes. Elsevier 218

Soares S, Laureano E, Borba P (2002) Implementing distribution and persistence aspects with aspectj. ACM Sigplan Notices 37(11):174–190 238

Soltani N, Soleimani B, Barekatain B (2017) Heuristic algorithms for task scheduling in cloud computing: A survey. International Journal of Computer Network & Information Security 9(8) 195

Spirtes P, Glymour C, Scheines R, Meek C, Fienberg S, Slate E (1999) Prediction and experimental design with graphical causal models. Computation, causation, and discovery pp 65–94 71

Splawa-Neyman J, Dabrowska DM, Speed T (1990) On the application of probability theory to agricultural experiments. essay on principles. section 9. Statistical Science pp 465–472 262

Srikantaiah S, Kansal A, Zhao F (2008) Energy aware consolidation for cloud computing. Tech. rep., USENIX: The Advanced Computing Systems Association 283

Sriraghavendra M, Chawla P, Wu H, Gill SS, Buyya R (2022) Dosp: A deadline-aware dynamic service placement algorithm for workflow-oriented iot applications in fog-cloud computing environments. In: Energy Conservation Solutions for Fog-Edge Computing Paradigms, Springer, pp 21–47 30

Staff C (2016) React: Facebook's functional turn on writing javascript. Communications of the ACM 59(12):56–62 239

Stallings W (2014) Operating Systems: Internals and Design Principles| Edition: 8. Pearson 163

Stantchev V, Schröpfer C (2009) Negotiating and enforcing qos and slas in grid and cloud computing. Advances in Grid and Pervasive Computing pp 25–35 26

Stellman A, Greene J (2014) Learning agile: Understanding scrum, XP, lean, and kanban. " O'Reilly Media, Inc." 220

Stephan E (2005) Ip performance metrics (ippm) metrics registry. Tech. Rep. 4148, The Internet Engineering Task Force (IETF) - RFC Editor, URL https://www.rfc-editor.org/info/rfc4148 264

Stephan E, Palet J (2004) „remote network monitoring (rmon) protocol identifiers for ipv6 and multi protocol label switching (mpls)". Tech. Rep. 3919, The Internet Engineering Task Force (IETF) - RFC Editor, URL https://www.rfc-editor.org/info/rfc3919 264

Stewart GA, Lampl W, et al (2017) How to review 4 million lines of atlas code. Journal of Physics: Conference Series - IOP Publishing 898(7):072,013 276

Storn R, Price K (1997) Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization 11(4):341–359 116

Strohmaier E, Dongarra J, Simon H, Meuer M (2020) 56th edition of the top500 supercomputer list. Tech. rep., Top500.org, URL https://www.top500.org/lists/top500/2020/11/ 278

Sturm R, Pollard C, Craig J (2017) Application performance management (APM) in the digital enterprise: managing applications for cloud, mobile, iot and eBusiness. Morgan Kaufmann 12

Sun D, Chang G, Li F, Wang C, Wang X (2011) Optimizing multi-dimensional qos cloud resource scheduling by immune clonal with preference. Acta Electronica Sinica 8:018 26

Suroto S (2017) A review of defense against slow http attack. JOIV: International Journal on Informatics Visualization 1(4):127–134 249

Sussna J (2015) Designing delivery: Rethinking IT in the digital service economy. " O'Reilly Media, Inc." 18, 220

Sutton R (1984) Temporal credit assignment in reinforcement learning. PhD thesis, Department of Computer Science and Information Science, University of Massachusetts Amherst 130, 262

Sutton RS (1992) Introduction: The challenge of reinforcement learning. In: Reinforcement Learning, Springer, pp 1–3 168

Sutton RS, Barto AG (1998) Reinforcement learning: An introduction. MIT press Cambridge 130, 168, 169, 262

Sydor MJ (2010) APM Best Practices: Realizing Application Performance Management. Apress 12, 38, 46, 54, 61, 68, 96, 97

Taillard E (1993) Benchmarks for basic scheduling problems. european journal of operational research 64(2):278–285 198

Takagi H (2001) Interactive evolutionary computation: Fusion of the capabilities of ec optimization and human evaluation. Proceedings of the IEEE 89(9):1275–1296 225

Tarkoma S (2012) Publish/subscribe systems: design and principles. John Wiley & Sons 243

Taur Y (2002) Cmos design near the limit of scaling. IBM Journal of Research and Development 46(2.3):213–222 276

Teigen KH (1994) Yerkes-dodson: A law for all seasons. Theory & Psychology 4(4):525–547 227

Tenorio F, Tenorio MF (2013) Package gaoptim. URL http://cran.r-project.org/web/packages/gaoptim/ 112, 116, 273

Thanasias V, Lee C, Hanif M, Kim E, Helal S (2016) Vm capacity-aware scheduling within budget constraints in iaas clouds. PloS one 11(8):e0160,456 291

Theilmann W, Happe J, Kotsokalis C, Edmonds A, Kearney K, Lambea J (2010) A reference architecture for multi-level sla management. Journal of Internet Engineering 4(1):289–298 17

Thomas D, Fowler C, Hunt A (2004) Programming Ruby. Pragmatic 238

Thompson NC, Ge S, Sherry YM (2021) Building the algorithm commons: Who discovered the algorithms that underpin computing in the modern enterprise? Global Strategy Journal 11(1):17–33 287, 288

Thorndike E, Bruce D (1911) Animal intelligence: Experimental studies. Transaction Pub 130

Tilkov S, Vinoski S (2010) Node. js: Using javascript to build high-performance network programs. IEEE Internet Computing 14(6):80–83 242

Todd PM, Gigerenzer G (2000) Précis of" simple heuristics that make us smart". Behavioral and brain sciences 23(5):727–741 227

Tran D, Tran N, Nguyen G, Nguyen BM (2017) A proactive cloud scaling model based on fuzzy time series and sla awareness. Procedia Computer Science 108:365–374 25, 165

Troisi O, Maione G, Grimaldi M, Loia F (2020) Growth hacking: Insights on data-driven decision-making from three firms. Industrial Marketing Management 90:538–557 249, 295

Trümper J, Döllner J, Telea A (2013) Multiscale visual comparison of execution traces. In: 2013 21st International Conference on Program Comprehension (ICPC), IEEE, pp 53–62 61

Truyen E, Van Landuyt D, Reniers V, Rafique A, Lagaisse B, Joosen W (2016) Towards a container-based architecture for multi-tenant saas applications. In: Proceedings of the 15th international workshop on adaptive and reflective middleware, pp 1–6 290

Tuli S, Gill SS, Garraghan P, Buyya R, Casale G, Jennings N (2021) Start: Straggler prediction and mitigation for cloud computing environments using encoder lstm networks. IEEE Transactions on Services Computing 30, 33

Tuli S, Gill SS, Xu M, Garraghan P, Bahsoon R, Dustdar S, Sakellariou R, Rana O, Buyya R, Casale G, et al (2022) Hunter: Ai based holistic resource management for sustainable cloud computing. Journal of Systems and Software 184:111,124 28, 30

Turnquist GL (2010) Spring Python 1.1. Packt Publishing Ltd 238

Urdaneta G, Pierre G, Van Steen M (2009) Wikipedia workload analysis for decentralized hosting. Computer Networks 53(11):1830–1845 183

Urgaonkar B, Shenoy P (2004) Sharc: Managing cpu and network bandwidth in shared clusters. IEEE Transactions on Parallel and Distributed Systems 15(1):2–17 158

Van Bon J, Clifford D (2008) Implementing ISO/IEC 20000 certification: The roadmap. Van Haren 296

Van Bon J, Van Selm L (2008) ISO/IEC 20000-an introduction. Van Haren 296

Van Der-Wees A, Daniele C, Jesus L, Edwards M, Schifano N, Maddalena S (2014) Cloud service level agreement standardisation guidelines. C-SIG on Service Level Agreements pp 1–41 16, 58

Van Eyk E, Iosup A, Seif S, Thömmes M (2017) The spec cloud group's research vision on faas and serverless architectures. In: Proceedings of the 2nd International Workshop on Serverless Computing, pp 1–4 129

Van Eyk E, Toader L, Talluri S, Versluis L, Uță A, Iosup A (2018) Serverless is more: From paas to present cloud computing. IEEE Internet Computing 22(5):8–17 129, 156, 284, 291

VanHoose D (2011) Ecommerce economics. Taylor & Francis 221

Vaquero L, Rodero-Merino L, Caceres J, Lindner M (2008) A break in the clouds: towards a cloud definition. ACM SIGCOMM Computer Communication Review 39(1):50–55 26

Vaquero LM, Rodero-Merino L, Buyya R (2011) Dynamically scaling applications in the cloud. ACM SIGCOMM Computer Communication Review 41(1):45–52 28

Verma A, Dasgupta G, Nayak TK, De P, Kothari R (2009) Server workload analysis for power minimization using consolidation. In: Proceedings of the 2009 conference on USENIX Annual technical conference, pp 28–28 279

Vogels W (2008) Beyond server consolidation. Queue 6(1):20–26 233

Waldbusser S (2006) Remote network monitoring management information base version 2. Tech. Rep. 4502, The Internet Engineering Task Force (IETF) - RFC Editor, URL https://www.rfc-editor.org/info/rfc4502 264

Walraven S, Truyen E, Joosen W (2014) Comparing paas offerings in light of saas development. Computing 96(8):669–724 290

Wang F, Yang CQ, Du YF, Chen J, Yi HZ, Xu WX (2011) Optimizing linpack benchmark on gpu-accelerated petascale supercomputer. Journal of Computer Science and Technology 26(5):854 282

Wang X, Chen M (2008) Cluster-level feedback power control for performance optimization. In: 2008 IEEE 14th International Symposium on High Performance Computer Architecture, IEEE, pp 101–110 30

Wang Y (1998) Smoothing spline models with correlated random errors. Journal of the American Statistical Association 93(441):341–348 75

Wang Z, Chen Y, Gmach D, Singhal S, Watson B, Rivera W, Zhu X, Hyser C (2009) Appraise: Application-level performance management in virtualized server environments. Network and Service Management, IEEE Transactions on 6(4):240–254 26

Watada J, Roy A, Kadikar R, Pham H, Xu B (2019) Emerging trends, techniques and open issues of containerization: a review. IEEE Access 7:152,443–152,472 218, 219

Watkins C (1989) Learning from delayed rewards. PhD thesis, King's College, Cambridge 130

Wei HL, Billings SA (2007) Feature subset selection and ranking for data dimensionality reduction. Pattern Analysis and Machine Intelligence, IEEE Transactions on 29(1):162–166 72

Welsh M, Culler D (2002) Overload management as a fundamental service design primitive. In: Proceedings of the 10th workshop on ACM SIGOPS European workshop, ACM, pp 63–69 26, 29

Welsh M, Culler D (2003) Adaptive overload control for busy internet servers. In: Proceedings of the 4th USENIX Conference on Internet Technologies and Systems, vol 2 26, 29, 263

Wen W, Zhang S (2014) Research and implementation of aop technology in. net framework. In: 2014 Enterprise Systems Conference, IEEE, pp 97–101 238

Weng C, Liu Q, Yu L, Li M (2011) Dynamic adaptive scheduling for virtual machines. In: Proceedings of the 20th international symposium on High performance distributed computing, ACM, pp 239–250 101

Wickham H (2009) ggplot2: elegant graphics for data analysis. Springer 114

Wilder B (2012) Cloud architecture patterns: using microsoft azure. " O'Reilly Media, Inc." 243

Wilhelmstötter F (2012) Jenetics: Java genetic algorithm library. URL http://jenetics.io 205

Wilkinson G, Rogers C (1973) Symbolic description of factorial models for analysis of variance. Applied Statistics pp 392–399 102

Wilkinson P (2006) Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers, 2/E. Pearson Education India 292

Williamson DP, Hall LA, Hoogeveen JA, Hurkens CA, Lenstra JK, Sevast'janov SV, Shmoys DB (1997) Short shop schedules. Operations Research 45(2):288–294 198

Windmill E (2020) Flutter in action. Simon and Schuster 249

Wirth N (1971) Program development by stepwise refinement. In: Pioneers and Their Contributions to Software Engineering, Springer, pp 545–569 291

Wirth N (1995) A plea for lean software. Computer 28(2):64–68 226, 287

Wohlgethan E (2018) Supportingweb development decisions by comparing three major javascript frameworks: Angular, react and vue. js. PhD thesis, Hochschule für Angewandte Wissenschaften Hamburg 239

Wojciechowski J, Sakowicz B, Dura K, Napieralski A (2004) Mvc model, struts framework and file upload issues in web applications based on j2ee platform. In: Proceedings of the International Conference Modern Problems of Radio Engineering, Telecommunications and Computer Science, 2004., IEEE, pp 342–345 239

Wright S (1921) Correlation and causation. Journal of agricultural research 20(7):557–585 72

Wright S (1934) The method of path coefficients. The Annals of Mathematical Statistics 5(3):161–215 72

Wright S, Nocedal J (1999) Numerical optimization, vol 2. Springer New York 107

Wu J, Shen YL, Reinhardt K, Szu H, Dong B (2013) A nanotechnology enhancement to moore's law. Applied Computational Intelligence and Soft Computing 2013 277

Wu K (2014a) A tunable workflow scheduling algorithm based on particle swarm optimization for cloud computing. Master's thesis, San Jose State University 198

Wu L, Garg SK, Buyya R (2012) Sla-based admission control for a software-as-a-service provider in cloud computing environments. Journal of Computer and System Sciences 78(5):1280–1299 23, 28, 30

Wu L, Tordsson J, Elmroth E, Kao O (2021) Causal inference techniques for microservice performance diagnosis: Evaluation and guiding recommendations. In: ACSOS 2021-2nd IEEE International Conference on Autonomic Computing and Self-Organizing Systems 71

Wu Q (2014b) Making facebook's software infrastructure more energy efficient with autoscale. Facebook Eng Blog URL https://engineering.fb.com/2014/08/08/production-engineering/making-facebook-s-software-infrastructure-more-energy-efficient-with-autoscale/ 283

Wulf J, Winkler TJ, Brenner W (2015) Measuring it service management capability: Scale development and empirical validation. In: 12th International Conference on Wirtschaftsinformatik, Universität Osnabrück, Osnabrück, Germany 295

Xhafa F, Barolli L, Durresi A (2007) Batch mode scheduling in grid systems. International Journal of Web and Grid Services 3(1):19–37 198

Xia Y (2015) Cloud control systems. IEEE/CAA Journal of Automatica Sinica 2(2):134–142 28, 30

Xiong P, Wang Z, Jung G, Pu C (2010) Study on performance management and application behavior in virtualized environment. In: Network Operations and Management Symposium (NOMS), 2010 IEEE, IEEE, pp 841–844 26, 30

Xiong P, Chi Y, Zhu S, Tatemura J, Pu C, HacigümüŞ H (2011) Activesla: a profit-oriented admission control framework for database-as-a-service providers. In: Proceedings of the 2nd ACM Symposium on Cloud Computing, ACM, p 15 158, 177

Xu M, Buyya R (2020) Managing renewable energy and carbon footprint in multi-cloud computing environments. Journal of Parallel and Distributed Computing 135:191–202 283

Xue J, Jarvis S (2018) Mining association rules for admission control and service differentiation in e-commerce applications. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 8(3):e1241 30

Yan Q, Yu FR, Gong Q, Li J (2015) Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges. IEEE communications surveys & tutorials 18(1):602–622 249

Yavatkar R, Pendarakis D, Guerin R, et al (2000) A framework for policy-based admission control. Tech. Rep. 2753, The Internet Engineering Task Force (IETF) - RFC Editor, URL https://www.rfc-editor.org/info/rfc2753 65

Yeap G, Lin S, Chen Y, Shang H, Wang P, Lin H, Peng Y, Sheu J, Wang M, Chen X, et al (2019) 5nm cmos production technology platform featuring full-fledged euv, and high mobility channel finfets with densest 0.021 μm 2 sram cells for mobile soc and high performance computing applications. In: 2019 IEEE International Electron Devices Meeting (IEDM), IEEE, pp 36–7 276

Yerkes RM, Dodson JD, et al (1908) The relation of strength of stimulus to rapidity of habit-formation. Punishment: Issues and experiments pp 27–41 227

Yi S, Hao Z, Qin Z, Li Q (2015) Fog computing: Platform and applications. In: 2015 Third IEEE workshop on hot topics in web systems and technologies (HotWeb), IEEE, pp 73–78 278

Yoo S, Kim S (2013) Sla-aware adaptive provisioning method for hybrid workload application on cloud computing platform. In: Proceedings of the International MultiConference of Engineers and Computer Scientists, vol 1 101

Yoro KO, Daramola MO (2020) Co2 emission sources, greenhouse gases, and the global warming effect. In: Advances in carbon capture, Elsevier, pp 3–28 283

Yu KM, Chen CK (2008) An adaptive scheduling algorithm for scheduling tasks in computational grid. In: 2008 Seventh International Conference on Grid and Cooperative Computing, IEEE, pp 185–189 198

Yu M, Yi Y, Rexford J, Chiang M (2008) Rethinking virtual network embedding: substrate support for path splitting and migration. ACM SIGCOMM Computer Communication Review 38(2):17–29 158

Yuan H, Bi J, Tan W, Li BH (2016) Cawsac: Cost-aware workload scheduling and admission control for distributed cloud data centers. IEEE Transactions on Automation Science and Engineering 13(2):976–985 158

Yussupov V, Breitenbücher U, Leymann F, Müller C (2019) Facing the unplanned migration of serverless applications: A study on portability problems, solutions, and dead ends. In: Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing, pp 273–283 291

Zakas NC (2016) Understanding ECMAScript 6: the definitive guide for JavaScript developers. No Starch Press 242

Zalewski A, Borowa K, Ratkowski A (2017) On cognitive biases in architecture decision making. In: European Conference on Software Architecture, Springer, pp 123–137 226

Zambrano A, Alvarez A, Fabry J, Gordillo S (2009) Aspect coordination for web applications in java/aspectj and ruby/aquarium. In: Proceedings, 28th International Conference of Chilean Computer Society, Citeseer 238

Zavala E, Franch X, Marco J, Berger C (2021) Adaptive monitoring for autonomous vehicles using the hafloop architecture. Enterprise Information Systems 15(2):270–298 260

Zhan ZH, Liu XF, Gong YJ, Zhang J, Chung HSH, Li Y (2015) Cloud computing resource scheduling and a survey of its evolutionary approaches. ACM Computing Surveys (CSUR) 47(4):1–33 127

Zhang A, Santos P, Beyer D, Tang H (2002a) Optimal server resource allocation using an open queueing network model of response time. HP laboratories Technical Report, HPL2002301 25, 29

Zhang J, Kumor D, Bareinboim E (2020) Causal imitation learning with unobserved confounders. Advances in neural information processing systems 33:12,263–12,274 263

Zhang Q, Cheng L, Boutaba R (2010) Cloud computing: state-of-the-art and research challenges. Journal of internet services and applications 1(1):7–18 156, 233

Zhang Q, Zhu Q, Boutaba R (2011) Dynamic resource allocation for spot markets in cloud computing environments. In: 2011 Fourth IEEE International Conference on Utility and Cloud Computing, IEEE, pp 178–185 28

Zhang Q, Zhani MF, Zhang S, Zhu Q, Boutaba R, Hellerstein JL (2012) Dynamic energy-aware capacity provisioning for cloud computing environments. In: Proceedings of the 9th international conference on Autonomic computing, ACM, pp 145–154 26

Zhang R, Lu C, Abdelzaher T, Stankovic J (2002b) Controlware: A middleware architecture for feedback control of software performance. In: Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on, IEEE, pp 301–310 25, 29

Zhang WJ, Lin Y (2010) On the principle of design of resilient systems–application to enterprise information systems. Enterprise Information Systems 4(2):99–110 6, 34, 156

Zhang X, Lu J, Qin X (2013) Bfepm: Best fit energy prediction modeling based on cpu utilization. In: 2013 IEEE Eighth International Conference on Networking, Architecture and Storage, IEEE, pp 41–49 285

Zheng K, Wang X, Li L, Wang X (2014) Joint power optimization of data center network and servers with correlation analysis. In: IEEE INFOCOM 2014-IEEE conference on computer communications, IEEE, pp 2598–2606 279

Zheng W, Sakellariou R (2013) Budget-deadline constrained workflow planning for admission control. Journal of grid computing 11(4):633–651 159

Zhu X, Yang LT, Chen H, Wang J, Yin S, Liu X (2014) Real-time tasks oriented energy-aware scheduling in virtualized clouds. IEEE Transactions on Cloud Computing 2(2):168–180 38, 157

Zhu Z, Zhang G, Li M, Liu X (2016) Evolutionary multi-objective workflow scheduling in cloud. IEEE Transactions on parallel and distributed Systems 27(5):1344–1357 171

Ziegler W (2017) A framework for managing quality of service in cloud computing through service level agreements. PhD thesis, University of Göttingen, Germany 17

Zimmerer P (2018) Strategy for continuous testing in idevops. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, pp 532–533 292

Zimmermann O (2017) Microservices tenets. Computer Science-Research and Development 32(3-4):301–310 23, 218

Zou Y, Zhu Y, Li Y, Wu FX, Wang J (2021) Parallel computing for genome sequence processing. Briefings in Bioinformatics 277