Adaptive resource provisioning in cloud computing environments

Adaptive Resource Provisioning in Cloud Computing Environments

Spyridon Chouliaras

A thesis submitted for the degree of
Doctor of Philosophy

Department of Computer Science & Information Systems
Birkbeck, University of London
2022

Declaration

I declare that this thesis has been composed solely by myself and that the work contained herein is my own except where explicitly states otherwise by reference or acknowledgement.

Spyridon Chouliaras

Abstract

Cloud computing emerged as a technology that offers scalable access to computing resources in conjunction with low maintenance costs. In this domain, cloud users utilize virtualized resources to benefit from elastic computing and efficient pricing strategies. Although, cloud users have access to large amount of resources, it is yet a challenging task to efficiently manage the resources in cloud computing environments. In that context, cloud providers offer auto-scaling services that need to be configured by the users according to application requirements. Still, tuning scaling parameters is not trivial, since it is mainly based on static scaling rules that may lead to unreasonable costs and quality of service violations. This thesis introduces a reliable adaptive resource provisioning framework for database applications in cloud computing environments. The framework is organized around three main services to enable a) anomaly detection for reliable decision making, b) resource provisioning for effective and efficient workload execution and c) constrained optimization to identify the optimal configurations that maximize application performance based on user requirements. The services included in this thesis utilize a variety of artificial intelligence techniques including Artificial Neural Networks for supervised learning, K-means for unsupervised learning and Genetic Algorithms for constrained optimization. The proposed techniques are based on monitored metrics collected from the running systems deployed in the cloud. Finally, this thesis presents an extended experimental analysis using industry standard applications and state-of-the-art benchmarks to demonstrate the robustness and effectiveness of the proposed framework.

Acknowledgements

I would like to thank the following people who accompanied me on my PhD journey.

Foremost, I would like to express my deepest appreciation and gratitude to my principal supervisor Dr Stelios Sotiriadis for taking me under his wing and giving me the opportunity to study a PhD at Birkbeck, University of London. His continuous support, inspiration, enthusiasm and willingness to explore creative ideas made this research possible.

I would like to thank my second supervisor Dr Andrea Calì for his guidance and support.

I would like to thank my colleagues at Birkbeck Knowledge Lab for their friendship and the inspirational discussions we have had the past years.

I would like to thank my parents, Nikolaos and Georgia, and my sister Eirini for their support during this journey. Their encouragement in many instances has been invaluable.

Finally, I would like to thank my fiancée Nikolina for her endless support and the many sacrifices she made during my studies.

Publications

The research presented in this thesis was partly published prior to thesis submission.

1. *(Under Review)* Chouliaras, S., & Sotiriadis, S. (2023). Towards Constrained Optimization of Cloud Applications: A Hybrid Approach. Future Generation Computer Systems.
2. *(Under Review)* Chouliaras, S., & Sotiriadis, S. (2023). An adaptive autoscaling framework for cloud resource provisioning. Future Generation Computer Systems.
3. Chouliaras, S., & Sotiriadis, S. (2022). Auto-scaling containerized cloud applications: A workload-driven approach. Simulation Modelling Practice and Theory, 102654.
4. Chouliaras, S., & Sotiriadis, S. (2021). Inferring Anomalies from Cloud Metrics Using Recurrent Neural Networks. In International Conference on Network-Based Information Systems (pp. 154-164). Springer, Cham.
5. Chouliaras, S., & Sotiriadis, S. (2021). Detecting performance degradation in cloud systems using LSTM autoencoders. In International Conference on Advanced Information Networking and Applications (pp. 472-481). Springer, Cham.
6. Chouliaras, S., & Sotiriadis, S. (2019). Real-time anomaly detection of NoSQL systems based on resource usage monitoring. IEEE Transactions on Industrial Informatics, 16(9), 6042-6049.

Contents

Declaration **2**

Abstract **3**

Acknowledgements **4**

Publications **5**

Contents **6**

List of Figures **10**

List of Tables **13**

List of Abbreviations **14**

1 Introduction **15**

1.1 Cloud Computing 15

1.2 Brief Discussion of the Problem Area 17

1.3 Research Questions and Objectives 19

1.4 Major Contributions 20

1.5 Thesis Organization 22

2 Literature Review **24**

2.1 Database Systems 24

2.1.1 SQL databases 25

2.1.2 NoSQL databases 26

2.2 Machine Learning 29

2.2.1 Supervised learning 29

2.2.2 Unsupervised learning 30

2.2.3 Artificial neural networks 31

2.3 Evolutionary Computation 36

2.3.1 Simulation of evolution 36

2.3.2 Genetic algorithms 37

2.4 Anomaly Detection . . . 39
2.4.1 Data types . . . 39
2.4.2 Anomaly types . . . 40
2.4.3 Anomaly detection techniques . . . 41
2.4.4 Discussion of anomaly detection approaches . . . 43
2.5 Auto-scaling . . . 45
2.5.1 Auto-scaling process . . . 45
2.5.2 Auto-scaling techniques . . . 46
2.5.3 Auto-scaling containerized applications . . . 50
2.6 Resource Provisioning and Optimization . . . 54
2.7 Summary . . . 56

3 Modeling the PACE Framework . . . 57
3.1 The PACE Topology . . . 57
3.2 Architecture of PACE . . . 60
3.3 ADM2 Service for Anomaly Detection . . . 62
3.3.1 LSTM Autoencoder for anomaly detection . . . 62
3.3.2 The Algorithmic structure of ADM2 service . . . 64
3.4 ADA-RP Service for Resource Provisioning . . . 65
3.4.1 ADA-RP service . . . 65
3.4.2 The elasticity process of ADA-RP service . . . 66
3.4.3 The algorithmic structure of ADA-RP service . . . 70
3.4.4 Learning phase . . . 74
3.5 SONA Service for System Optimization . . . 78
3.5.1 SONA service . . . 78
3.5.2 The optimization process of SONA service . . . 79
3.5.3 The algorithmic structure of SONA service . . . 88
3.6 Summary . . . 90

4 Deployment Environment and Benchmarks . . . 92
4.1 Cloud Environment . . . 92
4.2 System Deployment . . . 95
4.2.1 MySQL . . . 95

4.2.2 MongoDB . . . 96
4.2.3 HBase . . . 97
4.2.4 Redis . . . 99
4.3 Benchmarks . . . 100
4.3.1 TPC-C . . . 100
4.3.2 TPCx-IoT . . . 102
4.3.3 Yahoo! Cloud Serving Benchmark . . . 103
4.4 Summary . . . 104

5 PACE Datasets . . . 105
5.1 Dataset Creation Process . . . 105
5.2 ADM2 Dataset . . . 106
5.3 ADA-RP Dataset . . . 108
5.3.1 SQL data collection . . . 108
5.3.2 NoSQL data collection . . . 110
5.4 SONA Dataset . . . 111
5.5 Summary . . . 113

6 Performance Evaluation . . . 115
6.1 Experimental Design . . . 115
6.2 ADM2 Experimental Evaluation . . . 118
6.2.1 Detecting performance degradation of MongoDB . . . 118
6.2.2 Detecting performance degradation of HBase . . . 122
6.3 ADA-RP Experimental Evaluation . . . 125
6.3.1 Reactive auto-scaling mechanism evaluation . . . 125
6.3.2 Proactive auto-scaling mechanism evaluation . . . 128
6.4 SONA Experimental Evaluation . . . 141
6.4.1 SONA performance analysis . . . 141
6.4.2 SONA experimental results . . . 148
6.5 Summary . . . 154

7 Conclusions and Future Directions . . . 155
7.1 Major Contributions of the Thesis . . . 155
7.2 Limitations and Future Directions . . . 156

7.3 Concluding Remarks 158

Bibliography **159**

List of Figures

1.1 Major Contributions. 21

2.1 Visualization of CAP theorem. 28
2.2 Artificial Neural Network architecture with two inputs x_1,x_2, one output y_1 and two hidden layers. 32
2.3 Single-neuron computation given two inputs x_1 and x_2. 33
2.4 RNNs Architecture. 34
2.5 Auto-scaling MAPE (Monitoring, Analysis, Planning, Execution) loop. 46
2.6 Horizontal and vertical scaling in cloud computing. 51

3.1 PACE support operations in a cloud environment. 58
3.2 PACE framework main services for anomaly detection, resource provisioning and system optimization. 59
3.3 Architecture of PACE framework. 60
3.4 Univariate time series data representation based on LSTM Autoencoder. 63
3.5 ADA-RP proactive auto-scaling stages. 69
3.6 SONA operation flow diagram. 82

4.1 System organization for Virtual Machines 93
4.2 System organization for containers 94
4.3 Column-oriented and row-oriented storage. 98
4.4 TPC-C execution cycle 101
4.5 TPCx-IoT execution phases 103
4.6 YCSB Client architecture 104

5.1 Dataset creation process. 106

6.1 The experimental map of PACE services. 117
6.2 Throughput of MongoDB on YCSB workload. 119
6.3 The distribution of train MAE loss on YCSB normal runs 120
6.4 Throughput of MongoDB on YCSB workload with Stress. 120
6.5 The distribution of test MAE loss on YCSB abnormal runs 121

6.6 Throughput of HBase on TPCx-IoT workload. 122
6.7 The distribution of train MAE loss on TPCx-IoT normal runs . . 123
6.8 Throughput of HBase on TPCx-IoT workload with network delay. 123
6.9 The distribution of test MAE loss on TPCx-IoT abnormal runs . . 124
6.10 Memory usage percentage and memory allocation of Redis container over YCSB workload execution. 127
6.11 Operations per second in Redis over YCSB workload execution of 5 million read and update records. 128
6.12 K-means algorithm for clustering time series data into High, Medium and Low demand clusters over a 60 seconds period of time. The average sequence for each cluster is shown in red color. 130
6.13 CPU usage percentage, CPU usage predictions and virtual CPU allocation based on adaptive auto-scaling strategy for single tenant architecture. 131
6.14 QPS of MySQL container and total estimated costs based on adaptive auto-scaling strategy for single-tenant architecture. . . 132
6.15 CPU usage percentage, CPU usage predictions and virtual CPU allocation based on adaptive auto-scaling strategy for multi-tenant architecture. 134
6.16 QPS of MySQL container and total estimated cost based on adaptive auto-scaling strategy for multi-tenant architecture. . . . 135
6.17 K-means algorithm for clustering time series data into High, Medium and Low demand clusters over a 15 seconds period of time. The average sequence for each cluster is shown in red color. 137
6.18 CPU usage percentage training set (blue line) and CPU usage predictions (green dotted line). 138
6.19 CPU usage prediction (green dotted line), CPU usage after scaling actions (blue line) and vCPU cores allocation (red line) using ADA-RP service 139
6.20 Operations per second in $Redis-main$ in multi-tenant environment. Red shaded area indicates system overload period of time and green shaded area shows system recovery. 140
6.21 Cost-performance ratio and TpmC performance 142

6.22 CPU idle percentage over Load and Run stage. 144
6.23 Artificial Neural Network TpmC predictions over the training and test set vs the Experimental TpmC. 147
6.24 Evolution of generations for TpmC optimization. 148
6.25 Scale down for improving efficiency. 149
6.26 Application tuning for TpmC optimization. 151
6.27 Scale up for improving TpmC performance. 152
6.28 System configuration and application tuning for TpmC optimization. 153

List of Tables

3.1 Optimal CNN configurations 77

5.1 Dataset configuration based on the Cartesian product of sets $\mathbb{C}$ and $\mathbb{W}$. 109
5.2 Dataset configuration based on the Cartesian product of sets $\mathbb{Y}$ and $\mathbb{V}$. 110
5.3 SONA Dataset. 113

6.1 System configurations 142
6.2 Average system utilization and system idle. 145

List of Abbreviations

ACID	Atomicity, Consistency, Isolation, and Durability
ANN	Artificial Neural Networks
CAP	Consistency, Atomicity, Partition
CNN	Convolutional Neural Networks
CPU	Central Processing Unit
CSV	Comma Separated Values
GA	Genetic Algorithms
IaaS	Infrastructure as a Service
JSON	JavaScript Object Notation
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
NoSQL	Not only SQL
OLTP	Online Transaction Processing
PaaS	Platform as a Service
QoS	Quality of Service
RNN	Recurrent Neural Networks
SLA	Service Level Agreement
SQL	Structured Query Language
SaaS	Software as a Service
VM	Virtual Machine
XML	Extensible Markup Language
YCSB	Yahoo! Cloud Serving Benchmark

1 Introduction

This chapter provides background work on cloud computing and introduces key terms and concepts that are discussed in the thesis. It further presents the motivation, research questions and objectives as well as the methodological approach taken to produce this research work. Finally, the thesis organization and structure is presented.

1.1 Cloud Computing

Today, advancements in networking and other technological domains are mainly responsible for the acceptance of the idea that information processing can be done more efficiently in distant data centers rather than local computing systems (Marinescu, 2022). Therefore, cloud computing has emerged as a successful computing paradigm which aims to provide a fertile environment for deploying application and services. Fundamentally, it offers an infrastructure where various services and applications are available to users through the public Internet (Sotiriadis, 2021).

Various cloud providers such as Amazon, Google, IBM and Microsoft have made computational services available to clients to form a public cloud. However, there are various types of clouds depending on the service availability and accessibility levels with the most common being public, private, virtual private, community and hybrid cloud.

- The public cloud is owned by an organization that provides services to the general public or a large industry group using the public Internet.
- The private cloud provides services to a single organization. It can be managed by the organization or a third party and could be hosted inside or outside the premises of the organization (Marinescu, 2022).
- The virtual private cloud is a derivative of the private cloud deployment model but it further provides an isolated segment of resources on top of public infrastructure (Buyya, Broberg, and Goscinski, 2010).
- The community cloud provides an infrastructure that is shared by several organizations and supports a specific community that has shared concerns (Marinescu, 2022).
- The hybrid cloud is a composition of two or more clouds (private, community, or public) that are bounded together in order to compose a single, optimal cloud environment.

Clouds offer three predominant service models namely Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). In principle, cloud service models consist of different levels of abstraction at which virtualized resources are being offered and managed by the cloud provider (Kächele et al., 2013). IaaS model offers the lowest level of abstraction since it provides virtualized resources on hardware level. In such model, cloud users can directly use infrastructure components such as Central Processing Unit (CPU), memory, disk and networks in the form of Virtual Machine (VM) instances. On the other hand, PaaS and SaaS offer higher levels of abstraction in terms of specific solution stacks and application software suites (Bruneo, 2013). In particular, PaaS service model provides an environment for deploying application and services while SaaS service model offers the highest level of abstraction where software is provided to the end-users as a service. In a SaaS model, the end-users execute their applications without having any control on the host environment (Stavrinides and Karatza, 2019).

Cloud scalability and elasticity are key aspects of cloud computing systems that support rapid and dynamic resource provisioning to everyday users. Cloud scalability refers to the ability of the system to accommodate larger loads, while elasticity refers to the ability of the system to scale with loads dynamically (Sotiriadis et al., 2016). Additionally, cloud providers offer flexible pricing models including on-demand plans that enable cloud users to pay just for the services they need and for the time they use them (Grossman, 2009). As a result, users can adjust cloud resources at any moment according to their needs without paying any penalty (Mireslami et al., 2021). Cloud providers also offer discount rates on reserved resources in specific availability zones (Amazon, 2022b). Ergo, cloud users benefit from long term-contracts to launch a number of low-cost reserved instances that ensure high performance under various workloads.

1.2 Brief Discussion of the Problem Area

Cloud computing technology is on the rise, as it provides an easy to scale environment for Internet users in terms of computational resources. Although, cloud computing offers various types of delivery models with numerous benefits, it also presents challenges that need to be considered. With a dramatic increase in complexity of cloud infrastructure, cloud users face difficulties to configure cloud resources based on application requirements. In particular, the over-provisioning problem can occur where cloud users reserve more resources than the actual workload demand (Chaisiri, Lee, and Niyato, 2011), leading to unnecessary costs. Similarly, under-provisioning problem can occur when cloud users underestimate workload tasks and allocate inadequate amount of resources that often lead to Quality of Service (QoS) violations.

Moreover, cloud users are being challenged to choose between different pricing models based on application requirements. For instance, on-demand pricing models introduce additional flexibility to the system that can increase or decrease in size to support workload changes. Although, this solution looks more appealing to cloud users, it comes with a higher cost compared to long-term contracts that include reserved resources. However, cloud users could benefit from long-term contracts as long as they can produce accurate estimations of future workload demand. In that context, inaccurate estimations can introduce over-utilized or under-utilized resources that lead to application performance degradation and cost-inefficiency.

Furthermore, cloud systems do not currently support customized solutions for optimizing application performance based on user requirements. This is due to the fact that, it is a challenge to address both adaptive resource allocation and application tuning over various workload executions. In addition, users usually introduce various requirements that need to be considered during the optimization process such as cost, performance and resource efficiency. Thus, the problem is formulated as a challenging multi-objective task where a framework needs to provide the optimal amount of system resources based on application demands, tune the application configurations as well as satisfy user requirements. Since cloud providers struggle to deliver a combination of features that satisfy all the aforementioned requirements, users often attempt to optimize application performance offline to the best of their knowledge. However, this approach could increase rapidly the infrastructure expenditures while often becomes infeasible due to time constraints.

In addition, data is becoming massive in terms of volume, variety and velocity, while different techniques have been implemented in order to deal with this new phenomenon. A NoSQL (meaning 'not only SQL') has come to describe a large class of databases sufficient to deal with massive amounts of data, supporting also non-schema structure and real-time analysis (Gudivada, D. Rao, and Raghavan, 2014). Such systems can store data from different sources without a specific structure, a characteristic that makes them powerful for diverse applications such as e-commerce, navigation systems, Web services and the Internet of Things (IoT). However, the robustness and the reliability of cloud systems remains a key problem for information technology companies as cloud systems could suffer from hardware, software and network failures. Therefore, efficient ways on detecting abnormalities are more important than ever, since anomalies in the system can cause performance degradation, unreasonable energy consumption as well as false scaling estimation from automated algorithms. The latter, indicates the importance of finding techniques in order to automatically detect abnormal system behavior of cloud systems.

1.3 Research Questions and Objectives

The aim of this PhD thesis is to develop an adaptive framework to provision cloud resources based on user and application requirements. To achieve this, the following research questions have to be addressed.

1. Is it possible to develop a framework that automatically allocates virtualized resources that ensure cloud application performance?
2. Is it possible to identify and propose the configurations that improve application performance while they also satisfy user requirements?
3. Is it possible to automatically detect system abnormalities to ensure that scaling-decision mechanisms are not prone to errors due to abnormal system behavior?

To answer the research questions, the following objectives should be met:

O1 A literature review of techniques and methods for reliable and adaptive resource provisioning in cloud environments with the aim of identifying relevant critical issues.

O2 The development of a novel framework for adaptive resource provisioning in cloud computing environments. This includes the implementation of three main services for anomaly detection, resource provisioning and optimization of cloud applications.

O3 The identification of methods that improve the overall system performance. This includes efficient deployment strategies to increase the effectiveness of the proposed framework.

O4 The design and deployment of real-world applications to experimentally evaluate the proposed framework. This includes the identification of application performance metrics with the aim of selecting state-of-the-art benchmarks to evaluate the proposed method.

1.4 Major Contributions

Cloud provisioning enables resource anticipation in a proactive manner. As a result, cloud users have the ability to plan resource allocation based on future workload demand as well as to benefit from cost-efficient strategies. However, provisioning systems often face the under-provisioning and over-provisioning problems that lead to unreasonable costs and performance SLA violations. Additionally, such systems are prone to incorrect decisions under abnormal system behavior (e.g., network delays).

This research study presents an auto-scaling framework for adaptive resource provisioning in cloud environments. The contributions of this work is fourfold:

C1 An anomaly detection service to automatically detect aberrant patterns in cloud computing environments.

C2 An auto-scaling service that enables adaptive resource provisioning based on workload demands.

C3 A constrained optimization service to identify system configurations that maximize application performance and satisfy user requirements.

C4 An experimental setting that utilizes efficient strategies, state-of-the-art benchmarks and novel datasets to evaluate framework functionalities.

To further demonstrate the contributions of this thesis, the following scenario is being presented. A database application is being deployed in the cloud while a number of resources utilized to execute different types of workloads. In such a scenario, the user has to monitor the running application to ensure that application performance (e.g., throughput) remains above normal thresholds and is not disrupted due to abnormal system behavior. Additionally, the user needs to manually configure application resources in order to meet workload requirements without over-provisioning or under-provisioning any of the resources. Lastly, the application resources need to be optimized based on user requirements to ensure feasible and efficient solutions.

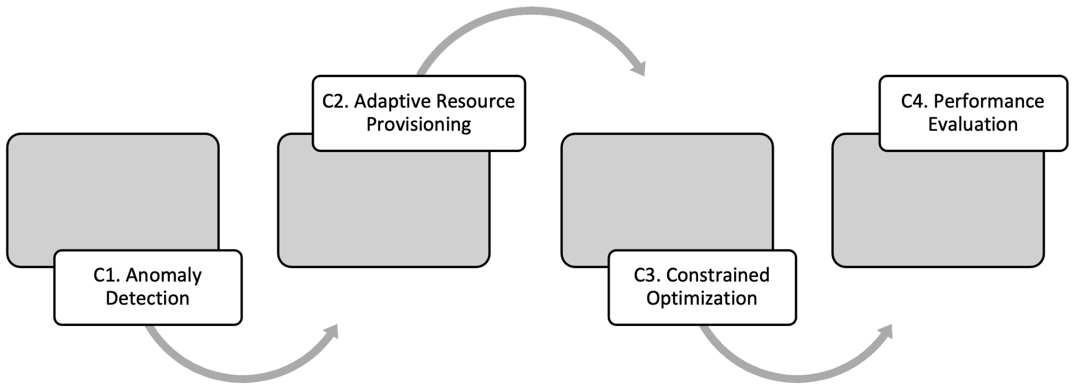

C1. Anomaly Detection
C2. Adaptive Resource Provisioning
C3. Constrained Optimization
C4. Performance Evaluation

Figure 1.1: Major Contributions.

Figure 1.1 illustrates the major contributions of this thesis to address the aforementioned challenges in cloud computing environments. Firstly, it proposes an anomaly detection service for detecting abnormal application behavior using application monitoring metrics. Then, it ensures that cloud applications scale dynamically to meet workload demands without over-estimating or under-estimating the amount of allocated resources. Additionally, it enables application optimization to identify system configurations that maximize application performance based on user requirements including cost, cost-performance ratio, baseline performance and idle resources. Finally, introduces an experimental setting to evaluate the effectiveness of the proposed framework.

1.5 Thesis Organization

This thesis is organised as follows:

Chapter 2 presents techniques and algorithms used for anomaly detection, resource provisioning and optimization of cloud applications. Specifically, it presents a survey of database systems widely used to store and manage large volumes of data in the cloud. Furthermore, it discusses a family of algorithms and techniques used to support the proposed methodology. Lastly, a state-of-the-art literature review of existing works is presented focusing on the problem of anomaly detection, auto-scaling and resource optimization in cloud environments.

Chapter 3 presents the model of Performance-Aware Cloud Elasticity (PACE) framework and describes its main services that support anomaly detection, resource provisioning and optimization of cloud applications. In more detail, Anomaly Detection based on Metric Monitoring (ADM2) service is presented to automatically detect performance degradation in cloud systems using LSTM Autoencoders. Additionally, Adaptive Resource Provisioning (ADA-RP) service is introduced to enable both reactive and proactive auto-scaling of cloud applications using K-means and CNN algorithms. System Optimization using Neuro-genetic Algorithm (SONA) service is also presented to support constrained performance optimization of cloud applications using a hybrid approach of Genetic Algorithms and Artificial Neural Networks. This chapter also presents the algorithmic structure of each service along with their pseudocode.

Chapter 4 presents the experimental platform of the PACE framework that includes the deployment environment, systems and benchmarks. The discussion presents the cloud computing environment, cloud computing service models and the virtualization techniques used throughout this research. Additionally, it discusses the database application systems, including MySQL, MongoDB, HBase and Redis, as well as state-of-the-art benchmarks used to emulate different cloud application scenarios, such as TPC-C, TPCx-IoT and YCSB.

Chapter 5 presents the dataset creation process to form the datasets used to support the proposed methodology. It includes the monitoring, data storage and data transformation phases. The monitoring phase supports the collection process of various metrics that describe the behavior of a running cloud system. These metrics are being stored and transformed in order to create the datasets used by ADM2, ADA-RP and SONA services, respectively.

Chapter 6 presents the experimental design and evaluation of the PACE framework and its services namely a) the ADM2 service for anomaly detection, b) the ADA-RP service for adaptive resource provisioning and c) the SONA service for constrained optimization. It also presents the experimental map of PACE services to illustrate the experiments conducted in the cloud to evaluate PACE models, algorithms and methods. As a result, chapter 6 demonstrates the ability of PACE framework to detect performance degradation in cloud systems, to auto-scale cloud applications using reactive and proactive techniques as well as to optimize system configurations in order to maximize application performance based on user requirements.

Finally, chapter 7 summarizes the research findings and presents the contribution of this research. It concludes with study's limitations and proposes future research directions.

2 Literature Review

Chapter 1 presented an introduction to cloud computing technology and its key characteristics. It also provided a brief discussion of the problem area and demonstrated the aims, the research questions and the objectives of this thesis. This chapter explores these ideas in greater depth, presenting a survey of systems and techniques used for cloud anomaly detection, resource provisioning and optimization that realize the key aspects of this thesis.

Section 2.1 provides a discussion around database systems and their key characteristics. Understanding the attributes of each database system motivates architecture design choices throughout the rest of this thesis. Section 2.2 and 2.3 provide a background on machine learning and evolutionary computation. Section 2.4 explains the different aspects of an anomaly detection problem including the types of anomalies and detection techniques.

Section 2.5 and 2.6 discuss auto-scaling, resource provisioning and optimization techniques in cloud computing environments. Finally, section 2.7 presents the summary of this chapter.

2.1 Database Systems

A database is a collection of integrated records that is represented of some physical or conceptual object (Berg, Seymour, Goel, et al., 2013). The end-user interacts with the database via a software called database management system (DBMS). These interactions include defining a database (i.e., specify data structures), constructing the database (i.e., storing the data), manipulating the database (i.e., querying and retrieve data) and sharing databases among users and applications (Elmasri et al., 2000). In that context, database management systems are categorized based on the database models they support. A database model is a collection of conceptual tools used to model real-world entities and their relationships (Silberschatz, Korth, and Sudarshan, 1996). It consists of three main components: a collection of data structure types, a collection of operators and a collection of general integrity rules (Codd, 1980). Although various database models have been proposed this research focuses on the relational and non-relational database models.

2.1.1 SQL databases

The relational database model was first introduced in 1970 from IBM research laboratory (Codd, 2002). In this model, a database uses a collection of tables to represent both data and their relationships. In that context, a structured query language (SQL) has been accepted as the standard query language for relational database management systems (Lu, Chan, and Wei, 1993). Relational databases considered the most popular database management systems with a variety of available implementations including IBM DB2, Oracle, Microsoft SQL server, MySQL and PostgreSQL.

A database application running into a relational database mainly executes a number of transactions, that is, a collection of database operations including read, insert, delete and update (Elmasri et al., 2000). Various database applications execute a large number of concurrent transactions without imposing excessive delays known as Online Transaction Processing (OLTP) applications (Sumathi and Esakkirajan, 2007). In that context, DBMS should ensure four desirable properties of atomicity, consistency, isolation and durability known as ACID (Haerder and Reuter, 1983).

- Atomicity requires that a transaction is indivisible and must be processed in its entirety or not at all. This property ensures that there are no incomplete transactions in case of a system failure or any unexpected situation.

- Consistency requires that each successful transaction preserves the consistency of the database after its completion.
- Isolation requires that transaction events must be hidden from other transactions executing concurrently.
- Durability requires that after the completion of a transaction, the system guarantees the changes applied to the database.

Relational databases consist of a schema that defines the structure of a database as a set of table definitions and derivation rules (Halpin and Morgan, 2010). Although, a database schema is useful for managing database objects in logical groups, various applications cannot conform to a rigid relationship schema and more flexible solutions are required.

2.1.2 NoSQL databases

Nowadays, data are becoming massive in terms of volume, variety and velocity. As a result, non-relational databases have been proposed to deal with this new phenomenon. A NoSQL (meaning 'not only SQL') has come to describe a large class of databases sufficient to deal with massive amounts of data, supporting also non-schema structure and distributed computing (Gudivada, D. Rao, and Raghavan, 2014). Such systems can store data from different sources without a specific structure, a characteristic that makes them powerful for different applications such as e-commerce, navigation systems, web services and IoT.

NoSQL systems are mainly categorized into four major types including document-oriented, key-value, column-oriented and graph databases. Each type consists of different characteristics as follows:

1. Document-oriented databases: These databases are designed to store, retrieve and manage document data using various formats such as

JavaScript Object Notation (JSON) and Extensible Markup Language (XML). Popular document-oriented databases include MongoDB[1], Amazon DocumentDB[2] and Apache CouchDB [3].

2. Key-value: These databases are based on a simple data model where every data element is stored as a key value pair. In these systems, the key is used to uniquely identify and access the value that can be a record, a document or an object. Popular key-value databases include Redis[4], Riak[5] and Voldermort[6].

3. Column-oriented databases: These databases partition a table by column with values belonging to the same column stored contiguously, compressed and densely packed (Abadi, Boncz, and Harizopoulos, 2009). Examples of this type of databases are Apache HBase[7] and Apache Cassandra[8].

4. Graph-based databases: In these databases, data are represented by graphs and data manipulation is expressed by graph-oriented operations (Angles and Gutierrez, 2008). Examples of graph-based databases include Neo4j[9] and Nebula[10].

The three main requirements in a distributed system are consistency, availability and partition tolerance. Consistency means that each node will have the same version of a replicated data item. Availability means that the system will remain operational and each request will eventually receive a response. Partition tolerance refers to system endurance if a communication fails between two nodes within a system.

[1]https://www.mongodb.com/
[2]https://aws.amazon.com/documentdb/
[3]https://couchdb.apache.org/
[4]https://redis.io/
[5]https://riak.com/
[6]https://www.project-voldemort.com/voldemort/
[7]https://hbase.apache.org/
[8]https://cassandra.apache.org/_/index.html
[9]https://neo4j.com/
[10]https://nebula-graph.io/

NoSQL applications often require a continuous system availability. As a result, data are being replicated to multiple machines to ensure that are still available if a node fails to perform. However, data distribution may slow down write performance if consistency is required due to the fact that every copy of the replicated data needs to be updated (Elmasri et al., 2000). In that context, the CAP (Availability, Consistency, Partition tolerance) theorem introduced the idea that there is a general trade-off between the three desirable properties. (Brewer, 2000). As a result, the CAP theorem states that it is not possible to guarantee availability, consistency and partition tolerance at the same time. Figure 2.1 shows the CAP theorem. Although, SQL applications guarantee consistency through the ACID properties, many NoSQL applications adopt a more relaxed form of consistency known as eventual consistency in order to guarantee availability and partition tolerance (Burckhardt, 2014).

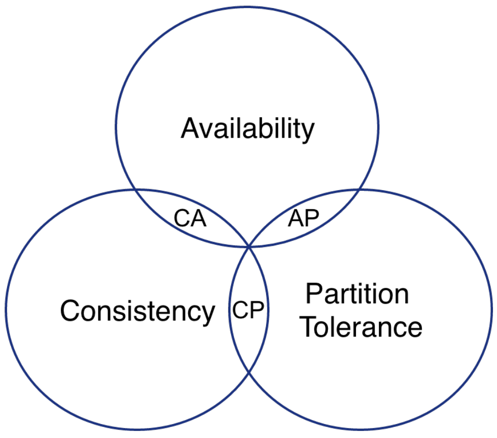


Figure 2.1: Visualization of CAP theorem.

SQL and NoSQL systems are being widely used to serve different types of applications. In this thesis, both SQL and NoSQL systems are being used to demonstrate the effectiveness of the proposed methodology. Ergo, database systems will be deployed in the cloud while state-of-the-art workloads tailored for each type of system will be used to emulate various realistic application scenarios.

2.2 Machine Learning

This thesis uses a variety of machine learning algorithms to enable anomaly detection, resource provisioning and optimization in cloud environments. Machine learning refers to a set of tools and methods that can learn from data. The learning process has been defined in the past as follows: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" (Mitchell, 1997). This type of learning can be applied to solve a variety of tasks including natural language processing, computer vision, anomaly detection and software engineering.

Machine learning algorithms can be categorized as supervised or unsupervised based on the experience they allowed to have during the learning process (Goodfellow, Bengio, and Courville, 2016). In most cases, machine learning algorithms experience an entire dataset that consists of a number of observations. These are also called training examples and used to train the algorithm given a supervised or unsupervised learning approach.

2.2.1 Supervised learning

Supervised learning considered to be a method that learns with an external 'teacher' or a supervisor (Negnevitsky, 2005). In this setting, supervised learning algorithms experience a training dataset to learn a mapping from inputs x (features) to outputs y (labels). Ergo, the training set contains a collection of observations as follows:

$$D = \{(x_i, y_i)\}_{i=1}^{n} \tag{2.1}$$

where each observation contains a feature vector x_i that is associated with a label y_i for $i = 1, ..., n$. In that context, two main supervised tasks are being defined namely as classification and regression.

- Classification: In this task, the classification algorithm performs a mapping between an input vector x and an output variable y that is a qualitative variable from a finite set of categories such that $y_i \in \{1,...,K\}$. As a result, the learning algorithm produces a mapping function $f : \mathbb{R}^n \rightarrow \{1,...K\}$. Other variants of classification tasks have been seen in the literature where f function outputs a probability distribution over classes (Goodfellow, Bengio, and Courville, 2016). Some examples of classification tasks include image recognition and sentiment analysis.

- Regression: This type of task is similar to classification except that the output is a quantitative variable also called numeric variable. In regression, the learning algorithm performs a mapping between an input vector x and a numerical output value y. Ergo, the algorithm produces a mapping function $f : \mathbb{R}^n \rightarrow R$. An example of a regression task is the prediction of future performance metric values such as database throughput.

2.2.2 Unsupervised learning

The second main type of machine learning is the unsupervised learning method. In this type of learning there are inputs variables but without a supervising output variable (James et al., 2013). As a result, algorithms experience a dataset of numerous training examples in order to discover useful properties and relationships in the data. For that reason, this type of learning is also called knowledge discovery (Murphy, 2012). The training set contains a collection of observations as follows:

$$D = \{x_i\}_{i=1}^{n} \tag{2.2}$$

where x_i is the input vector and n is the number of observations. Unsupervised learning tasks include clustering and dimensionality reduction.

- Clustering: In this type of task, the clustering algorithm learns to partition the observations into subgroups or clusters that share similar characteristics. As a result, observations that belong to the same cluster are similar to each other while observations that belong to different clusters are not similar (James et al., 2013). In this domain, there are clustering algorithms that partition the data into a predefined number of groups (e.g., K-means) and clustering algorithms that do not need to know this information in advance (e.g., hierarchical clustering). Clustering used in many fields including biology and astronomy.

- Dimensionality reduction: A dataset may consist of numerous feature variables that increase the processing time during the learning process and the space required for storage. In that context, dimensionality reduction is useful to project the data into lower dimensional subspace that explains most of the variability in the data (Murphy, 2012). The most common approach is the Principal Component Analysis that projects the data into a lower dimension by minimizing the mean squared distance between the data points and their projections (Bishop and Nasrabadi, 2006). Real-world applications include image compression and data noise reduction.

There is a third type of machine learning algorithms, namely, reinforcement learning. In this type of learning, algorithms do not experience a training dataset with numerous observations. On the contrary, reinforcement learning focused on goal-directed learning from interaction with its environment (Sutton and Barto, 2018). As a result, reinforcement learning consists of closed-loop problems that explore which actions to take on a given environment in order to maximize a reward signal. More of this type of learning will be discussed later in this chapter in section 2.5 where different auto-scaling techniques will be reviewed.

2.2.3 Artificial neural networks

Artificial neural networks (ANN) is a special type of machine learning algorithms widely used to solve a variety of problems such as pattern recognition, forecasting and optimization (Jain, J. Mao, and Mohiuddin, 1996). ANN contain connected computational units called neurons organised together in layers. The idea behind this type of models is to simulate the human brain where neurons are being activated to perform functions faster and more accurate (Negnevitsky, 2005). Figure 2.2 shows an example of Artificial Neural Network architecture with two inputs x_1, x_2, one output y_1 and two hidden layers.

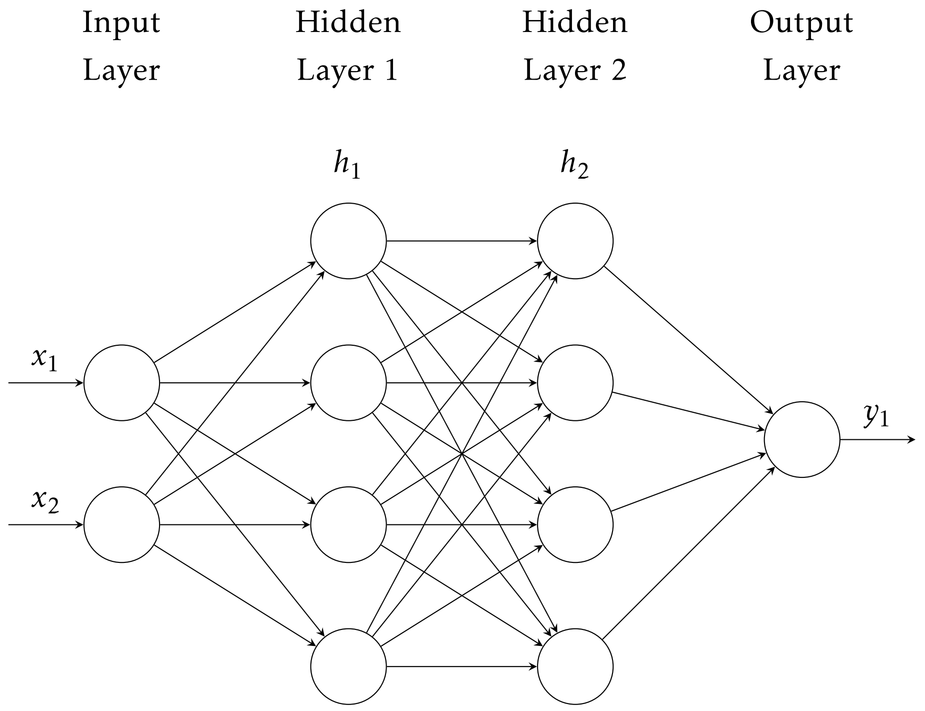


Figure 2.2: Artificial Neural Network architecture with two inputs x_1, x_2, one output y_1 and two hidden layers.

In more detail, an ANN consists of an input layer, a set of hidden layers and an output layer. An ANN architecture also determines the number of artificial neurons in each layer. Figure 2.2 illustrates an ANN architecture with one input layer of two neurons, two hidden layers of four neurons in each layer and one output layer of one neuron. Each neuron in hidden and output layers are interconnected via adaptive weights that are calibrated during the training process (Bre, Gimenez, and Fachinotti, 2018). Thus, a neuron computes the weighted sum of the input neurons and then compares the result with a threshold value, θ (McCulloch and Pitts, 1943). Then, an activation function is being used to calculate the output of the neuron. For example, sign activation function outputs -1 if the weighted sum is less than θ threshold and

+1 if the weighted sum is greater or equal than θ threshold. Figure 2.3 illustrates the computation process of a single neuron also known as perceptron (Rosenblatt, 1958) with two inputs x_1 and x_2.

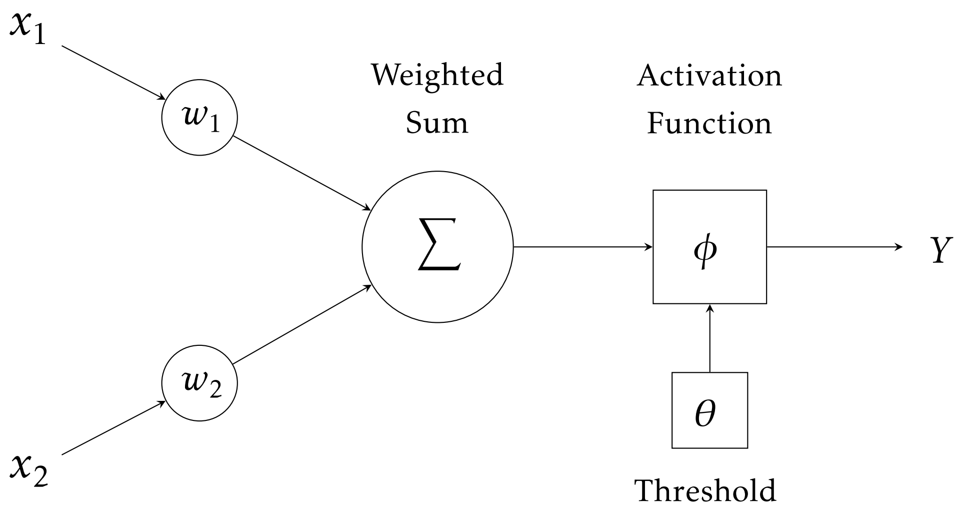


Figure 2.3: Single-neuron computation given two inputs x_1 and x_2.

It has to be mentioned that activation functions in neural networks have a significant effect on the training dynamics and task performance (Ramachandran, Zoph, and Le, 2017). The most common activation functions are the logistic sigmoid, the tangent sigmoid and the Rectified Linear Unit (Relu).

Recurrent Neural Networks (RNN) proposed as powerful neural networks models for sequential data. This type of models have been used to remember patterns, maintaining context and process complex signals for long time periods (Williams and Zipser, 1989) which makes them ideal for real-time resource usage analytics. Figure 2.4 demonstrates the architecture of RNN. In RNN the information flows from the input layer denoted as $x(t)$ in time step t as well as the hidden layer denoted as $h(t-1)$ from the previous time step $t-1$. The latter allows the network to have a memory of past events (Raschka, 2015). W_{xh} is the weight matrix between the input $x(t)$ and the hidden layer h, W_{hh} is the weight matrix associated with the information from the previous time step, that is, the recurrent edge, and W_{hy} is the weight matrix between the hidden layer h and the output layer y.

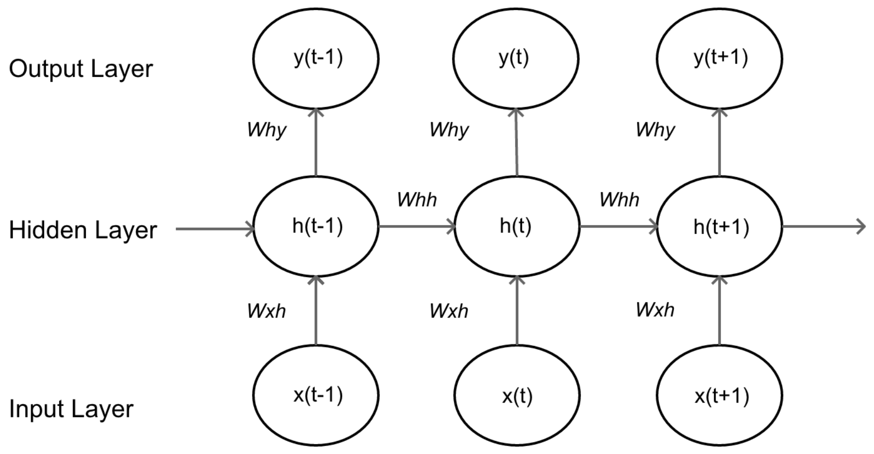


Figure 2.4: RNNs Architecture.

Long Short-Term Memory (LSTM) model is a type of a RNN that promises a substantial improvement in sequential data with temporal sequences and long-range dependencies (Hochreiter and Schmidhuber, 1997). An LSTM model contains special units called memory cells that are organised inside the recurrent hidden layer. Each memory cell contains an input gate, an output gate and a forget gate. The forget gate has been subsequently added in order to enable processing continuous input streams that are not segmented into subsequences (Sak, Senior, and Beaufays, 2014). The forget gate gives the ability to LSTM cell to learn to reset itself at appropriate times and prevent the network to break down in situations where time series grow arbitrary through time (e.g., continuous input streams) (Gers, Schmidhuber, and Cummins, 2000).

An LSTM receives an input sequence $x = \langle x_1, x_2, x_3, ..., x_T \rangle$ and maps it to an output sequence $y = \langle y_1, y_2, y_3, ..., y_T \rangle$ by calculating the following equations iteratively from t=1 to T:

$$i_t = \sigma(W_{ix}x_t + W_{im}m_{t-1} + W_{ic}c_{t-1} + b_i) \tag{2.3}$$

$$f_t = \sigma(W_{fx}x_t + W_{fm}m_{t-1} + W_{fc}c_{t-1} + b_f) \tag{2.4}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g(W_{cx}x_t + W_{cm}m_{t-1} + b_c) \tag{2.5}$$

$$o_t = \sigma(W_{ox}x_t + W_{om}m_{t-1} + W_{oc}c_t + b_o) \tag{2.6}$$

$$m_t = o_t \odot h(c_t) \tag{2.7}$$

$$y_t = \phi(W_{ym}m_t + b_y) \tag{2.8}$$

where the i is the input gate that uses the logistic sigmoid function denoted as σ in order to control the information that flows into the cell, f is the forget gate that uses the logistic sigmoid function to decide what information to keep outside the cell state, o is the output gate that uses the logistic sigmoid function to control the information that flows out of the cell, c is the cell activation vectors, m is the output activation vector where $\odot$ is the element-wise product of the vectors, g is the cell input activation function, h is the cell output activation function which is usually the $tanh$ function, ϕ is the output activation function of the output sequence y which is usually the $softmax$ function, W terms denote weight matrices and b terms denote bias vectors.

Another type of artificial neural networks namely as Convolutional Neural Networks (CNN) has been widely used for different tasks including image recognition (Hijazi, R. Kumar, Rowen, et al., 2015) and time series forecasting (Chouliaras and Sotiriadis, 2022), (Barino et al., 2020). CNN design inspired from the visual cortex, that is brain's visual mechanism. CNN contains convolution layers that are responsible for extracting different patterns in the data. Ergo, CNN uses the extracted information to predict future events based on a given task. CNN architecture promises a much simpler and easier to train alternative to recurrent neural networks while achieving at least as good or better performance in time series forecasting tasks (Borovykh, Bohte, and Oosterlee, 2017).

This thesis focuses on both supervised and unsupervised learning tasks. The proposed methodology uses a variety of algorithms including K-means, ANN, LSTM and CNN. For example, CNN have been used as a supervised learning approach to predict the future CPU utilization values based on historical time series data. Furthermore, ANN used during the resource optimization process to support an ANN-based objective function. Additionally, unsupervised learning algorithms also used for triggering scaling decisions. In more detail, K-means clustering algorithm used to partition time series data into different groups based on CPU utilization levels. Lastly, unsupervised learning methods used for detecting abnormalities in cloud systems.

2.3 Evolutionary Computation

Evolutionary computation represents a powerful method to simulate natural evolution. It consists a series of optimization algorithms usually referred to as evolutionary algorithms. These algorithms make use of genetic-inspired operations in order to evolve a set of candidate solutions (Dumitrescu et al., 2000). Ergo, during the optimization process the algorithms iteratively improve a set of solutions until an optimal or a feasible solution is found (Negnevitsky, 2005). In this thesis, evolutionary computation algorithms used to enable constrained resource optimization of cloud database systems. In more detail, PACE framework uses evolutionary computation algorithms to explore the optimum configurations that maximize application performance based on user requirements.

2.3.1 Simulation of evolution

Biological evolution entails changes in the characteristics of biological populations over time, leading to differences among them (Strickberger, 2000). Most organisms evolve through natural selection and reproduction. The former refers to the process of selecting members of the population to reproduce while the latter ensures recombination among the genes of their offspring (Holland, 1992b). As a result, the evolutionary process improves population's ability to survive and reproduce in a specific environment, an ability known as evolutionary fitness (Negnevitsky, 2005).

Let us take as an example a population of rabbits (Michalewicz, 1996b). Some rabbits in the population are faster and smarter while some other rabbits are slower and dumber. The faster and smarter rabbits have a higher chance to avoid foxes thus, surviving. Nevertheless, some slower and dumber rabbits are going to survive but with lower chances. Consequently, the surviving population starts breeding resulting to some fast rabbits breed with fast rabbits, fast rabbits breed with slow rabbits and slow rabbits breed with slow rabbits. The new population will be faster and smarter (on average) than the original population since a greater number of faster and smarter parent rabbits survived the foxes. It has to be mentioned that some rabbits experience genetic mutation resulting to changes of their genetic material.

Evolutionary computation algorithms designed to follow the principles of natural selection and the "survival of the fittest" (Michalewicz, 1996b). These stochastic algorithms model some natural phenomena including genetic inheritance and Darwinian strife for survival (Michalewicz, 1996a). Although, evolutionary computation refers to a family of algorithms, this work focuses on Genetic Algorithms inspired by natural evolution to solve both constrained and non-constrained optimization problems.

2.3.2 Genetic algorithms

Genetic Algorithms (GA) are a class of stochastic search algorithms that simulate the process of natural evolution (Negnevitsky, 2005). This type of algorithms explore a much greater range of candidate solutions to a problem than conventional programs do (Holland, 1992b). GA involve individuals and populations. Individuals are being encoded to represent a single solution while the population consists a set of individuals involved in the search process (Sivanandam and Deepa, 2008). Each individual is represented by a chromosome that is subdivided into genes. Ergo, a chromosome is a sequence of genes that represent a single factor of the solution.

GA apply the following steps during the optimization process (Negnevitsky, 2005):

1. Represent the problem variable as a chromosome that contains a fixed number of genes, a size of population N, a crossover probability p_c and a mutation probability p_m.
2. Define the objective function, that is the fitness function, to evaluate each chromosome in the population.
3. Randomly generate a population of size N.
4. Evaluate the fitness of each individual chromosome in the population using the objective function.
5. Select a pair of chromosomes with a probability related to their fitness.
6. Generate a pair of offsprings by applying crossover and mutation operators.
7. Include the generated offsprings to the new population.
8. Repeat step 5 until the size of the new population equals to N.
9. Replace the old population with the new population.
10. Repeat the process starting from step 4 until a desired criterion is reached (e.g., number of generations).

GA take advantage of modern parallel processing systems and provide solutions in areas ranging from the design of integrated circuits to the design of aircraft turbines (Holland, 1992a). In this work, GA used to solve a constrained optimization problem. The chromosome defined as a sequence of genes to represent resource and application configuration parameters. During the optimization process an ANN-based objective function used to evaluate the fitness of each chromosome while numerous constraints applied to explore solutions that satisfy user requirements.

2.4 Anomaly Detection

Anomaly detection refers to the process of identifying data points, events or patterns that deviate from the normal data distribution. These nonconforming patterns inherited different names based on application domain including anomalies, outliers, surprises, or peculiarities (Chandola, Banerjee, and V. Kumar, 2009). Anomaly detection has a wide variety of applications, including network intrusion detection for cyber security and fraud detection in finance.

2.4.1 Data types

Data types refer to the nature of input and output data during the anomaly detection process. The following two types define the problem characteristics and the anomaly detection techniques to be used in different application domains.

Input data

Input data are usually a collection of attributes that contain information to be analyzed by the detection system. These attributes can be of the same type or a mixture of different types such as binary, categorical or continuous (Chandola, Banerjee, and V. Kumar, 2009). Furthermore, an input might consist of multiple data attributes (multivariate) or a single attribute (univariate). For example, a monitoring system collects resource usage metrics as a sequence of data points indexed in time order. Then, a detection system analyzes the time series to detect abnormal behavior of the running system. In that context, anomaly detection for multivariate data refers to a system that takes decision based on multiple metrics (e.g., CPU, memory and disk usage). On the contrary, anomaly detection for univariate data refers to a system that detects abnormalities based on a single metric (e.g., database throughput).

Output data

Given different application scenarios, anomaly detection systems report anomalies in two types of outputs. These are as follows:

- Scores: Anomaly detection scoring techniques used to compute an anomaly score for each test data instance. In this type of systems, analysts have the ability to order all data instances based on the anomaly score and define the most important anomalies in the system.
- Labels: In this type, the anomaly detection system assigns a label to each data instance. For example, an anomaly detection system may use a machine learning algorithm that learns from historical data in order to classify future instances as normal or abnormal.

Score-based techniques allow the administrator to use thresholds based on application requirements in order to effectively detect abnormal instances. On the other hand, label-based techniques promote an automated solution where no predefined thresholds required, however, availability of labeled data is usually a major issue (Chandola, Banerjee, and V. Kumar, 2009).

2.4.2 Anomaly types

The type of anomaly defines the nature of the problem and the techniques needed to solve it. There are three main types of anomalies as follows:

- Point Anomalies: In this type of anomalies a single data instance flagged as an anomaly with respect to the rest of data. Real-world examples include credit card fraud detection and system intrusion detection.
- Contextual Anomalies: In this type, a data instance is flagged as anomalous in a specific context. This type of anomaly also referred as conditional anomaly in the literature (Song et al., 2007). Contextual anomalies can be found in surveillance video and in big sensor data systems.

- Collective Anomalies: This type refers to a collection of data instances that have been classified as anomalies. Such anomalies suggest a combination of abnormal events that may be identified based on various data sources (Y. Zheng, H. Zhang, and Y. Yu, 2015).

Of the three aforementioned anomaly types, point anomalies are the simplest type of anomalies and the easiest to detect. This is due to the fact that can be treated independently during detection and no temporal relationships need to be considered (Lin et al., 2020). On the other hand, contextual and collective anomalies are more challenging since additional information is needed during the detection process. It has to be mentioned that, a point anomaly or a collective anomaly can be also a contextual anomaly (Chandola, Banerjee, and V. Kumar, 2009). For example, time series data points may be flagged as anomalies based on the value of surrounding data points.

2.4.3 Anomaly detection techniques

Anomaly detection approaches can be categorized in three main classes: supervised, unsupervised and semi-supervised.

Supervised anomaly detection

Supervised anomaly detection includes a training dataset that contains a number of observations. Each observation consists of attributes mapped to a specific label variable (e.g., normal or abnormal) that adds informative context to the data. In that context, the most common use case is to build a machine learning model that learns to classify normal and abnormal events. However, there are two main issues that arise using supervised anomaly detection methods: imbalanced data and data labeling (Chandola, Banerjee, and V. Kumar, 2009). Imbalanced data refers to datasets where the classification categories are not approximately equally represented (Chawla, 2009). In general, observations of abnormal instances appear less frequently in a dataset compared to normal instances that may negatively impact the performance of the model. Additionally, data labeling is not a trivial task since it requires a high degree of consistency, accuracy and precision.

Unsupervised anomaly detection

In this anomaly detection technique, the objective is to assign a score to each data instance that reflects the degree to which the instance is an anomaly (Tan, Steinbach, and V. Kumar, 2016). This type of techniques appear to be powerful especially in the setting where labels are not available or the abnormal class appears less frequently in the dataset. However, this type of anomaly detection make two basic assumptions. The first assumption is that the number of normal instances appear more frequently in the dataset compared to the abnormal instances. Secondly, unsupervised anomaly detection methods assume that the anomalies are qualitatively different from the normal instances (Eskin et al., 2002). However, in some application areas unsupervised methods fail to achieve the required detection rates (e.g., network intrusion detection) (Görnitz et al., 2013). The main reason is that intrusion instances may appear in the same number as normal instances, thus can cause them all to be labeled as normal since they are not distinct from one another (Tan, Steinbach, and V. Kumar, 2016).

Semi-supervised anomaly detection

Semi-supervised anomaly detection techniques assume that labels exist only for the normal instances, while there is no information about the abnormal instances. In that context, semi-supervised methods use the information from a small pool of labeled normal instances to find an abnormal label for a set of given instances (Tan, Steinbach, and V. Kumar, 2016). In semi-supervised anomaly detection methods the occurrence of many abnormal test instances does not effect the detection rates. However, in many cases finding a representative set of normal instances is not a trivial task (Tan, Steinbach, and V. Kumar, 2016). Furthermore, semi-supervised anomaly detection methods are vulnerable to noisy instances that have been mistakenly labeled (Pang et al., 2021). In such cases unsupervised anomaly detection techniques can be utilized.

2.4.4 Discussion of anomaly detection approaches

This subsection presents a discussion around different methods that are based on resource usage and application monitoring for detecting abnormal events in the cloud.

Dhingra, Lakshmi, and Nandy (2012) underlined the importance of resource usage cloud monitoring for both cloud-user and service provider. The cloud-user's interest is to arrive at an appropriate Service Level Agreement (SLA) while the cloud-provider's interest is to ensure user's satisfiability. In order to meet the aforementioned requirements, they propose a distributed monitoring framework, which enables application monitoring and ensures SLA based on application's Quality of Service (QoS) requirements. Three Web Servers hosted on three VMs on a single host and httperf (Mosberger and Jin, 1998) benchmarking tool was used to measure web server performance by generating specific HTTP workloads. The basic architecture of their virtual environment consist four components VM Agent, Dom0 Agent, Metrics Collector and Customer Interface Module. The VM agent collects the metrics for each VM while The Dom0 Agent is specific to Xen hypervisor and collects the per-VM effort. Both Agents communicate with the Metrics Collector which in turn communicates with the Customer Interface Module in order to meet customer's monitoring requirements.

Lately, different approaches have been used to detect abnormalities in cloud systems based on resource usage data. Such techniques consist a powerful tool to tackle denial-of-service (DoS) attacks that occupy large amounts of computing resources from unauthorised users. S. Zhao, Chen, and W. Zheng (2009) proposed a defending schema that uses virtual machine monitor (VMM) to detect DoS attacks effectively based on a resource availability threshold. Due to the fact that DoS attacks occupy available resources without crashing the operating system, the VMM collects resource usage metrics (e.g., CPU usage) to acquire information on lower level. Consequently, they detect an abnormal behavior as the resources of a virtual machine decrease to a predefined threshold. Additionally, if the defending system assures an attack, it will migrate the operating system alongside the tagged applications, on-the-fly, in a new isolated environment while the initial VM will be interrupted or even destroyed.

Bertero et al. (2017) introduced the logging as a key source of information on a system state. Their work presented efficient algorithms for log mining with minimum human intervention. They applied a word embedding technique based on Google's word2vec algorithm that requires little intervention and promises strong predictive performance. Natural language processing techniques give a linguistic approach to anomaly detection that promises automation and further performance improvements. However, log-based anomaly detection techniques face their own challenges in modern cloud environments. As virtual machines scale horizontally in huge sizes, the complexity of those systems is rising and auditing becomes a challenging task as the volume of logs increases accordingly.

Malhotra et al. (2016) proposed an LSTM based Encoder-Decoder scheme for Anomaly detection as an alternative of standard approaches. In their findings, the LSTM Autoencoder learns to reconstruct normal time series behavior. Then, their model used the reconstruction error to effectively detect anomalies from predictable, unpredictable, periodic, aperiodic and quasi-periodic time series. Nguyen et al. (2020) proposed an LSTM Autoencoder network-based method combined with a one-class support vector machine algorithm used for anomaly detection. The LSTM Autoencoder network trained on normal representations and used to calculate a prediction error vector. Then, the one-class Support Vector Machine algorithm used to separate the abnormal observations from normal samples based on the predicted error vector.

Longari et al. (2020) proposed CANolo, an intrusion detection system based on LSTM Autoencoder to identify anomalies in controller area networks. Their framework automatically trained on controller area networks streams to build a model based on the legitimate data sequences. Then, the reconstructed error used to detect anomalies between legitimate sequences and simulated attacks. Trinh et al. (2019) proposed an LSTM Autoencoder to reconstruct mobile traffic data samples and an LSTM traffic predictor to predict the traffic of future time instants. In both cases, they analyzed the reconstructed and the predicted error to assess if the mobile traffic presents anomalies or not.

Although, all the aforementioned techniques proposed alternative solutions in order to detect abnormalities in cloud systems, they are mainly focused on log mining and data storage techniques. On the contrary, this thesis focuses on LSTM Autoencoders to reconstruct normal sequence representations of application metric data streamed from MongoDB and HBase systems. Then, unseen observations are being classified as normal or abnormal on-the-fly based on the reconstruction error produced by the LSTM Autoencoder. It has to be mentioned that anomaly detection techniques improve system awareness and enhance the ability of system administrators to automatically detect application performance degradation. Furthermore, aberrant patterns may negatively effect the proposed auto-scaling decision mechanism that is vulnerable to abnormal system behavior. Thus, anomaly detection techniques improve the auto-scaling process and the overall effectiveness of PACE framework.

2.5 Auto-scaling

An auto-scaling system automatically adjusts allocated resources to a running application to incorporate workload demands. As a result, the system is able to run more effectively and more efficiently with minimum human intervention.

2.5.1 Auto-scaling process

Auto-scaling process follows the MAPE loop for autonomous systems that consists of Monitoring, Analysis, Planning and Execution phases (Maurer et al., 2011). Figure 2.5 illustrates the four phases of MAPE loop given a managed resource (e.g., cloud application).

The monitoring phase enables the collection of different measurements about the running application (managed resource) including application resources (e.g., allocated CPU cores, allocated memory) and application usage metrics (e.g., CPU usage, memory usage and throughput). Next, auto-scaling

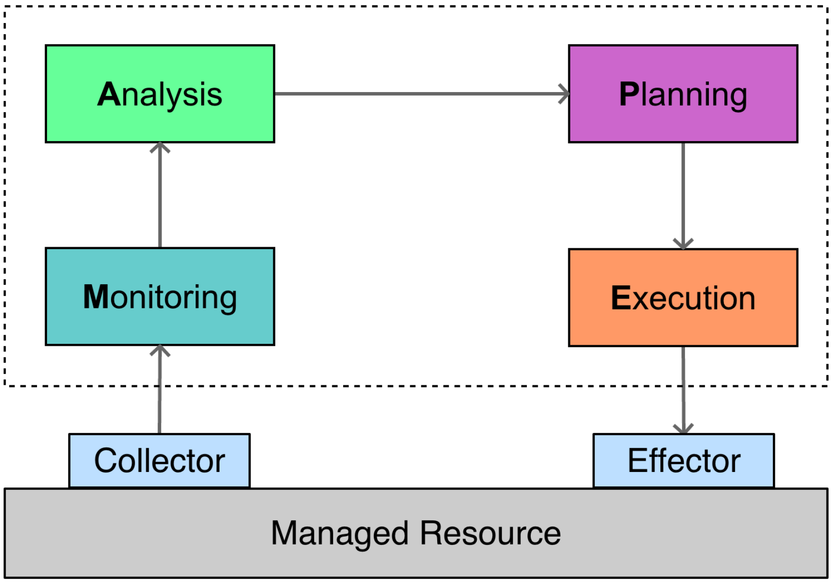


Figure 2.5: Auto-scaling MAPE (Monitoring, Analysis, Planning, Execution) loop.

systems enable the analysis phase where the system processes the monitored metrics. In that context, reactive auto-scalers trigger scaling decisions based on the current system status. However, more complex strategies known as proactive auto-scaling, predict future demands to adjust system resources with enough anticipation (Lorido-Botran, Miguel-Alonso, and Lozano, 2014). Once the future demand is known, the auto-scaler enables the planning phase to adjust system resources based on application and user requirements. Finally, the execution phase triggers all the scaling actions through the effector.

2.5.2 Auto-scaling techniques

Recent works suggested different auto-scaling techniques to facilitate resource provisioning in the cloud. Lorido-Botran, Miguel-Alonso, and Lozano (2014) suggested that most reviewed works can be fit in one or more of the following categories: a) threshold-based rules, b) reinforcement learning, c) queuing theory, d) control theory and e) time series analysis.

Threshold-based rules

Threshold-based rules or policies are being widely used by cloud providers such as Amazon EC2 (Amazon, 2022a) and Google Cloud Platform Compute Engine (Google, 2022c). Threshold-based auto-scaling rules enable cloud users to define an upper and a lower threshold based on one or more performance metrics such as CPU utilization and average response time. As a result, if the performance metric value is over the upper threshold or under the lower threshold, then a scaling decision will be triggered.

This technique has been widely used in the literature. R. Han et al. (2012) proposed a lightweight threshold-based approach to enable cost-effective elasticity for cloud applications. The proposed framework triggers scaling decisions based on predefined utilization thresholds (e.g., CPU usage and memory usage) to satisfy response time requirements. F. Zhang et al. (2019) proposed a lightweight container auto-scaler for elastic auto-scaling. Their framework monitors containers resource usage and accordingly scales in or out the containers based on a threshold-based strategy. Gulisano et al. (2012) introduced an elastic and scalable data streaming system where provisioning and decommissioning are triggered based on an upper and lower CPU utilization threshold. Threshold-based techniques promise lightweight and faster solutions, however setting the corresponding thresholds is not a trivial task since it requires a deep understanding of the running application. Furthermore, the effectiveness of a threshold-based approach is questionable under bursty workload behavior where metrics fluctuate strongly.

Reinforcement learning

Reinforcement learning (RL) involves a mapping between situations and actions so as to maximize a numerical reward signal (Sutton and Barto, 2018). In this domain, RL uses a trial-and-error approach to automate the scaling task by learning the most suitable scaling action for each particular state. RL techniques are suitable for automatic scaling decisions as they do not require a priori knowledge of the application performance model. As a result, an agent, that is the auto-scaler, reinforces scaling decisions that lead to high rewards.

The reward system is based on application performance improvement such as increase application throughput or reduce response time.

RL techniques have been extensively used by several authors to implement auto-scalers. Schuler, Jamil, and Kühl (2021) proposed an RL-based auto-scaling technique for dynamic resource provisioning in serverless environments. Barrett, Howley, and Duggan (2013) used a reinforcement learning algorithm known as Q-learning to determine optimal scaling policies. Arabnejad et al. (2017) proposed a self-adaptive fuzzy logic controller that combines two reinforcement learning approaches: (i) Fuzzy SARSA learning and (ii) Fuzzy Q-learning. J. Rao et al. (2009) recommended an RL-based agent to automate the VM configuration process that includes VM's memory size, scheduler credit and virtual CPU number respectively. Their agent automates VM reconfiguration process by generating policies learned from iterations with the environment. Horovitz and Arian (2018) presented Q-Threshold algorithm as an innovative auto-scaling approach based on RL. Q-Threshold learns the best utilization thresholds to enable dynamic auto-scaling according to application behavior. Although, RL-based auto-scaling techniques have been widely used to automatically adjust application resources, they present several problems including a long period of training time and poor performance before an acceptable solution is found.

Control theory

Control theory has been applied to automate the scaling tasks of cloud applications. The main objective of a controller is to adjust system resources in order to maintain the controlled variable (e.g., memory usage) closed to a desired level. Control theory has been widely used in the literature. H. C. Lim, Babu, and Chase (2010) proposed an elastic controller to adjust the number of VMs based on average response time and CPU utilization. Gandhi et al. (2016) used control theory for auto-scaling cloud-deployed data processing clusters. They employed a simple reactive controller to ensure execution time SLA compliance for Hadoop jobs. Ali-Eldin, Tordsson, and Elmroth (2012) introduced two adaptive hybrid controllers that use both reactive and proactive control for scaling cloud resources based on present and future demand respectively. Padala et al. (2009) presented AutoControl, a resource control system that automatically adapts to dynamic workload changes to satisfy service-level objectives. The AutoControl is able to adjust CPU and disk I/O usage to mitigate system bottlenecks. However, their approach does not control application adaptive parameters which can impact both the application benefit and execution time.

Queuing theory

Queuing theory enables auto-scaling by modeling applications and systems based on specific performance metrics such as the queue length or the average waiting time for requests. This technique has been widely used as an auto-scaling method in the literature.

Jiang et al. (2013) proposed a novel cloud resource auto-scaling scheme for web application providers. They predicted the number of requests by exploiting machine learning techniques and then they utilized queuing theory and multi objective optimization to discover the optimal number of VMs. G. Huang et al. (2016) used queuing theory for auto-scaling virtual machines in web applications. In their work, a queuing model has been used to predict the arrival time of each customer in order to calculate the minimum amount of resources needed to meet application needs.

Ali-Eldin, Kihl, et al. (2012) enabled queuing theory to construct an autonomous elasticity controller that changes the number of virtual machines allocated to a service based on load changes and predictions of future load. Feng et al. (2012) used queuing theory to formalize the resource allocation problem in cloud computing environments. They proposed optimal solutions for the problem considering various QoS parameters such as pricing mechanisms and available resources. The main limitation of queuing theory models is that they need to be recomputed when changes applied to application environment.

Time series analysis

Time series analysis covers a wide range of methods mainly focused on learning historical sequences in order to make future estimations about various metrics such as CPU and memory usage (Chouliaras and Sotiriadis, 2019). In this domain, time series analysis consists the main enabler of proactive auto-scaling techniques, where scaling decisions are being planned based on the predicted values.

Roy, Dubey, and Gokhale (2011) developed a model-predictive algorithm for workload forecasting that is used for resource auto-scaling. The proposed algorithm used to allocate or deallocate resources that satisfy application QoS with low operational costs. Marie-Magdelaine and Ahmed (2020) used a proactive auto-scaling algorithm based on Long Short-Term Memory (LSTM) to improve latency for cloud-native applications. Gong, Gu, and Wilkes (2010) presented a novel predictive resource scaling system for cloud systems. They used an auto-regression method for resource usage forecasting and performed elastic VM resource scaling based on the prediction results. Nikravesh, Ajila, and Lung (2015) considered the number of user requests per time unit as the performance metric and utilized Support Vector Machines and Artificial Neural Networks as time series prediction techniques. Golshani and Ashtiani (2021) used convolutional neural networks to predict future workload demand of cloud services. Furthermore, they proposed a mechanism for scalability decisions based on the predicted workload demand and user requirements. However, their multi-criteria decision making method consists of various threshold-based parameters that need to be set according to application and SLA requirements.

2.5.3 Auto-scaling containerized applications

Data centers use two main types of virtualization technologies in order to decouple applications from hardware namely as hardware level virtualization and operating system level virtualization (Sharma et al., 2016). Hardware level virtualization includes a hypervisor that virtualize resources across multiple VMs. However, applications deployed on a VM require the installation of an operating system that increase the demand for resources on the server. On the contrary, operating system level virtualization involves containers that make available protected portions of the operation system (Merkel et al., 2014). As a result, containers continue to gain traction as lightweight solution where resource utilization is more efficient and the deployment is faster.

In that context, numerous orchestration platforms have been proposed for automating deployment, scaling, and management of containerized applications including Amazon Elastic Container Service (Amazon ECS) (Amazon, 2022c) and Kubernetes (Kubernetes, 2022). One of the main aspects of these platforms is the dynamic resource configuration in terms of vertical and horizontal elasticity. Vertical scaling refers to the resizing of existing containers (scale up or down), while horizontal scaling consists of decreasing or increasing the number of containers in a cloud environment (scale in or out). Figure 2.6 illustrates vertical and horizontal scaling in cloud computing.

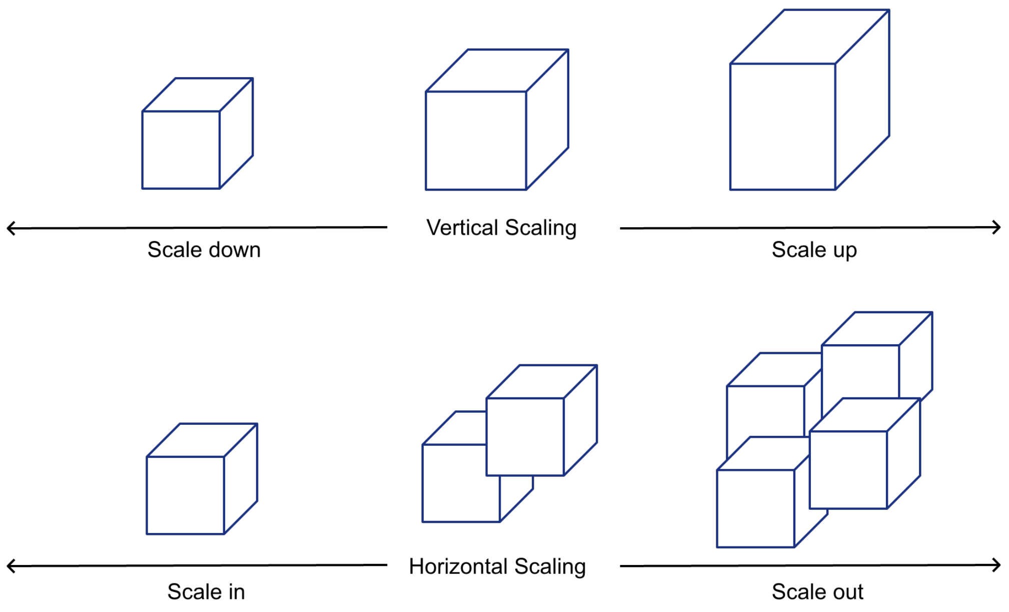


Figure 2.6: Horizontal and vertical scaling in cloud computing.

A variety of auto-scaling techniques and methods have been proposed to enable horizontal and vertical scaling in cloud environments. These are discussed in the next subsections.

Horizontal auto-scaling

Horizontal scaling of container-based applications has been extensively supported by cloud providers including AWS and Google Cloud. For instance, Kubernetes Horizontal Pod Autoscaler (HPA) used by Google Kubernetes Engine (GKE) to increase or decrease the number of Pods in response to workload's CPU and memory demand (HPA, 2022). Many researchers also proposed different techniques to support horizontal scaling of containerized applications. Imdoukh, Ahmad, and Alfailakawi (2020) proposed a proactive machine learning-based approach to perform auto-scaling of Docker containers in response to dynamic workload changes. In their work, LSTM model used to predict future HTTP workload to decide the number of containers needed to handle requests ahead of time. Klinaku, Frank, and Becker (2018) presented CAUS, a controller that consists of two auto-scaling mechanisms that manage the number of replicated containers. Kan (2016) introduced DoCLoud, an elastic cloud platform for web applications based on docker. DoCloud uses a hybrid elasticity controller that dynamically increases or decreases the number of containers to maintain higher resource utilization. Bello et al. (2021) proposed a predictive horizontal auto-scaling mechanism that scales containerized mobile core network entities based on their CPU utilization. Although, most of the works focus on horizontal elasticity, fewer studies address the problem of vertical elasticity of container-based applications

Vertical auto-scaling

Vertical elasticity considered to be a better option than horizontal elasticity when enough resources are available to the server. One of the main reasons is that vertical elasticity eliminates booting delays of adding new instances to the server, as opposed to horizontal elasticity. Furthermore, horizontal elasticity is restricted to applications that can be replicated or decomposed while vertical elasticity is applicable to any cloud application (Al-Dhuraibi et al., 2017). Additionally, vertical elasticity does not require supplementary tools (e.g., load balancers) or replicated instances that may add extra overhead and additional costs to the running system.

Vertical auto-scaling has enabled cloud customers to automatically adjust the size of containers based on workload requirements. As for example, Amazon Elastic Kubernetes Service (Amazon EKS) and GKE use the Kubernetes Vertical Pod Autoscaler (VPA) to automatically analyze and set CPU and memory reservations in the Pods (VPA, 2022). Additionally, numerous works proposed different mechanisms to scale containerized applications vertically. Paraiso et al. (2016) proposed a model-driven approach to address vertical elasticity of Docker containers. In their work, they designed a graphical model-driven tool called Docker Designer to design, reason and deploy containers. Furthermore, they used Model-Driven Engineering techniques for managing the Docker containers. Their tool allows the synchronization between the designed and the deployed containers respectively. However, their approach suggests that users need to manually adjust container's resources to enable vertical elasticity.

Nicodemus, Boeres, and Rebello (2020) presented VEMoC, a tool that implements vertical memory elasticity in containers. The management iteration of VEMoC tool consists seven phases: 1) calculate memory demand of inactive containers, 2) classify running containers by recent memory consumption, 3) passive memory limit reduction, 4) active memory limit reduction, 5) increase container memory limits, 6) pause or suspend containers and 7) start or resume inactive containers. Their solution manages to increase the utilization in the server without affecting the performance of each container. However, their tool does not yet integrate CPU throttling with memory elasticity to reduce the number of preemption events. Furthermore, VEMoC does not consider user requirements (e.g., budget constraints) in cases where the containers need to scale up in order meet workload demands.

Baresi et al. (2016) presented an auto-scaling technique based on a discrete-time feedback controller for containerized cloud applications. Their auto-scaling technique vertically scales the resources of cloud-based Web applications by increasing the number of allocated cores in the container. As a result, they managed to improve the performance of web applications by keeping the response time under a desired threshold. However, their approach only considers the amount of CPU cores required per tier, while their controller requests a specific amount of RAM per core. As a result, their approach is limited to Web applications where memory saturation is not an issue.

Al-Dhuraibi et al. (2017) proposed ElasticDocker for powering vertical elasticity of Docker containers autonomously. Their system scales up and down both CPU and memory container's resources according to application workload demand. ElasticDocker consists of two main components namely a monitoring system and an elasticity controller. The monitoring system is responsible for collecting metrics and limits from Docker containers including CPU usage and the number of allocated vCPU cores. The elasticity controller adjusts memory, CPU time and the number of vCPU cores based on workload requirements. Furthermore, their system supports live container migration when the host machine has inadequate resources. However, the proposed controller does not support predictive approaches to estimate future workload demand. Thus, ElasticDocker does not have the ability to anticipate workload requirements and rapidly adjust the resources accordingly.

Although, the techniques discussed in this section propose different auto-scaling techniques, they do not provide an end-to-end solution that includes automatic scaling decisions based on the predicted workload demand. Furthermore, the aforementioned techniques do not include a mechanism to recognize abnormal conditions in a cloud environment. As a result, the proposed auto-scaling mechanisms are prone to incorrect decisions under unusual system behavior.

2.6 Resource Provisioning and Optimization

This section discusses relevant methods for resource provisioning and optimization of cloud applications. There has been a lot of research in the area of provisioning virtualized resources. Abrahao et al. (2006) presented a dynamic capacity management framework in a multi-service environment based on an optimization model so as to maximize provider's business objective. L. Zhao, Sakr, and Liu (2013) introduced a framework that facilitates adaptive and dynamic provisioning for consumer-centric service level agreements management of cloud-hosted databases. Singh, Chana, and Buyya (2017) proposed an SLA-aware autonomic resource management technique that focuses on reducing SLA violation rate and satisfy QoS requirements. Comellas, Presa, and Fernández (2010) presented an elastic web hosting provider that makes use of cloud computing infrastructures for scaling the web applications deployed on it. However, monitoring the SLA compliance in (Abrahao et al., 2006; L. Zhao, Sakr, and Liu, 2013; Singh, Chana, and Buyya, 2017; Comellas, Presa, and Fernández, 2010) may require centralized services.

Previous works have introduced control theory techniques to facilitate virtualized resource management. Padala et al. (2009) introduced AutoControl, a resource control system that automatically allocates the right amount of CPU and disk I/O in order to adapt to dynamic workload changes. However, their approach does not control application adaptive parameters which has a significant impact to the application benefit. Zhu and Agrawal (2012) used a multi-input-multi-output feedback control model for dynamic resource provisioning to ensure the optimal application benefit within a time constraint. In addition, their model changes resource allocation based on budgets. However, the dynamic resource provisioning algorithm introduces a runtime overhead to the main application.

In addition, many approaches introduced resource provisioning with budget constraints. J. Yu and Buyya (2006) proposed a new type of GA to solve a budget constraint based scheduling that minimizes the execution time while meeting a specified budget. Mohan et al. (2016) proposed a big data workflow scheduler that supports scheduling in heterogeneous cloud computing environments under a provided budget constraint. Faragardi et al. (2019) introduced a resource provisioning mechanism and a workflow scheduling algorithm for minimizing the makespan of a given workflow subject to a budget constraint. Sakellariou et al. (2007) implemented an algorithm to schedule directed acyclic graphs onto heterogeneous machines to optimize the overall time of the workflow application under budget constraints. PACE framework not only supports resource provisioning based on budget constrained, but also enables the user to further penalize the recommended system based on idle CPU resources, cost-performance ratio and baseline performance.

Islam et al. (2012) proposed Neural Networks and Linear Regression models as a prediction-based resource measurement and provisioning strategy. In their work, both prediction techniques, used a dataset obtained by using TPC-W e-commerce application benchmark for forecasting resource utilization in the cloud. However, their prediction framework does not follow an adaptive and optimal resource provisioning strategy since business-level SLAs such as performance and monetary cost are not integrated into the prediction models. Biswas et al. (2014) proposed a proactive approach based on machine learning algorithms for automatic resource provisioning. Both Support Vector Machines and Linear Regression models used past workload data to predict the characteristics of future requests and determine the number of the allocated resources. However, their approach increases the execution time and adds extra overhead to the system in contrast with PACE framework that uses a cloning strategy. In addition, their framework does not support online training, thus, the system cannot adapt to application changes, as opposed to this work.

2.7 Summary

This chapter presented a literature review of systems and techniques used throughout this thesis. The discussion encompassed the characteristics of SQL and NoSQL databases and introduced machine learning and evolutionary computation. This chapter also presented anomaly detection process to identify patterns that deviate from the normal data distribution. Furthermore, the chapter discussed auto-scaling methods to automatically adjust resources of a running system in order to satisfy application requirements. Lastly, the chapter discussed different resource provisioning and optimization techniques found in the literature. The next chapter introduces the PACE framework and its three main services that enable adaptive resource provisioning in cloud environments.

3 Modeling the PACE Framework

This chapter presents PACE, a framework for adaptive resource provisioning in cloud environments. The chapter is organized as follows. Section 3.1 and 3.2 explain the topology of the proposed framework and give an overview of its architecture components. Section 3.3 demonstrates the anomaly detection service of PACE framework. Section 3.4 and 3.5 present the resource provisioning and optimization services respectively. Finally, section 3.6 gives the summary of this chapter.

3.1 The PACE Topology

The PACE framework enables an adaptive technique for reliable resource provisioning in cloud environments. This research focuses on database systems that are being deployed in the cloud while state-of-the-art workloads are being executed to emulate different real-world application scenarios. Then, the target application is being monitored for collecting and analyzing resource usage and application metrics. These metrics are being used as an input for learning algorithms to support the proposed data-driven approach. Figure 3.1 illustrates PACE framework support operations and its interactions in a cloud environment. The key entities of the PACE are the users (represented by workloads), the cloud application, the support operations (monitoring, analysis, planning and execution), and the low-level infrastructure (datacenter and hosts).

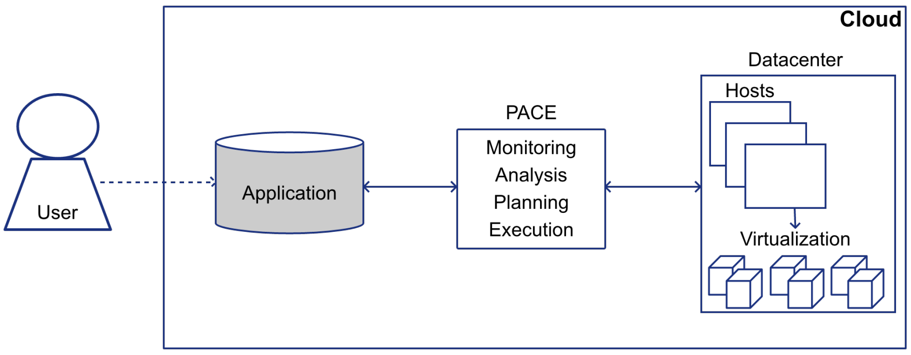


Figure 3.1: PACE support operations in a cloud environment.

The datacenter is the core of the cloud system where multiple servers and communication gear are co-located. A host represents the physical machine of the datacenter and its computational units including CPU, memory, storage and bandwidth that is available for virtualization (Sotiriadis, 2021). Initially, the cloud database application is being deployed in a virtualized environment with a particular amount of allocated resources. Then, PACE monitors the database application for collecting and analyzing resource usage and application metrics. Consequently, PACE enables planning and execution phases to dynamically adjust virtualized resources based on workload demand and user requirements.

PACE support actions, i.e., monitoring, analysis, planning and execution, are the stronghold of PACE services to support reliable and adaptive resource provisioning in the cloud. The PACE framework consists of three main services as follows.

- Anomaly detection based on metric monitoring (ADM2) service.
- Adaptive resource provisioning (ADA-RP) service.
- System optimization using neuro-genetic algorithm (SONA) service.

ADM2 service automatically detects system abnormalities based on real-time resource usage and application metric monitoring. Time series are being monitored, collected and analyzed on-the-fly using state-of-the-art monitoring tools and advanced machine learning techniques. As a result, ADM2 service alarms the administrator each time the system generates aberrant patterns. Furthermore, ADM2 ensures that PACE framework is not prone to incorrect scaling decisions due to abnormal system behavior.

ADA-RP service automatically provisions system resources based on monitored metrics. Ergo, ADA-RP avoids over-utilized and under-utilized resources that may lead to unnecessary costs and QoS violations. It enables both a) reactive auto-scaling using threshold-based rules to avoid application failures during intensive workload tasks and b) proactive auto-scaling to generate elastic scaling policies that incorporate future workload demands.

SONA service enables constrained resource optimization based on user requirements. SONA collects historical monitoring data to identify the optimal configurations that maximize application performance in the long term. The optimization process is being constrained based on user requirements, including the cloud infrastructure expenditures, the cost-performance ratio, the baseline performance and the average percentage of idle CPU resources.

Figure 3.2 illustrates the aforementioned service capabilities: a) ADM2 for anomaly detection, b) ADA-RP for resource provisioning and c) SONA for sys-

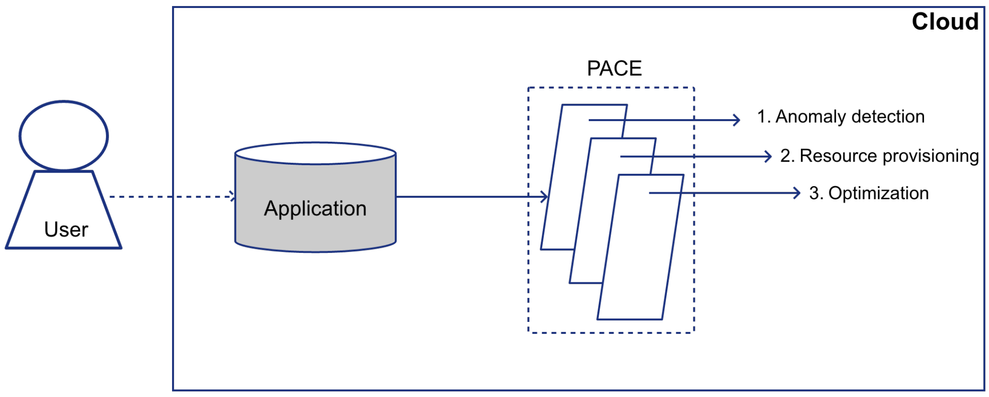


Figure 3.2: PACE framework main services for anomaly detection, resource provisioning and system optimization.

tem optimization. As mentioned earlier, each service enables different functionalities to support PACE framework. These are as follows:

a) The detection of aberrant patterns to alarm application user and to provide information regarding system status.

b) The automated resource provisioning than enables dynamic resource allocation based on workload demand.

c) The constrained system optimization that explores the optimal features that maximize application performance based on user requirements. The latter consists a fine-grained resource provisioning solution.

The methodology, architecture and algorithmic structure of each service is being discussed in the next sections.

3.2 Architecture of PACE

This section discusses the architecture components of PACE framework. Figure 3.3 illustrates the flow of events described as follows.

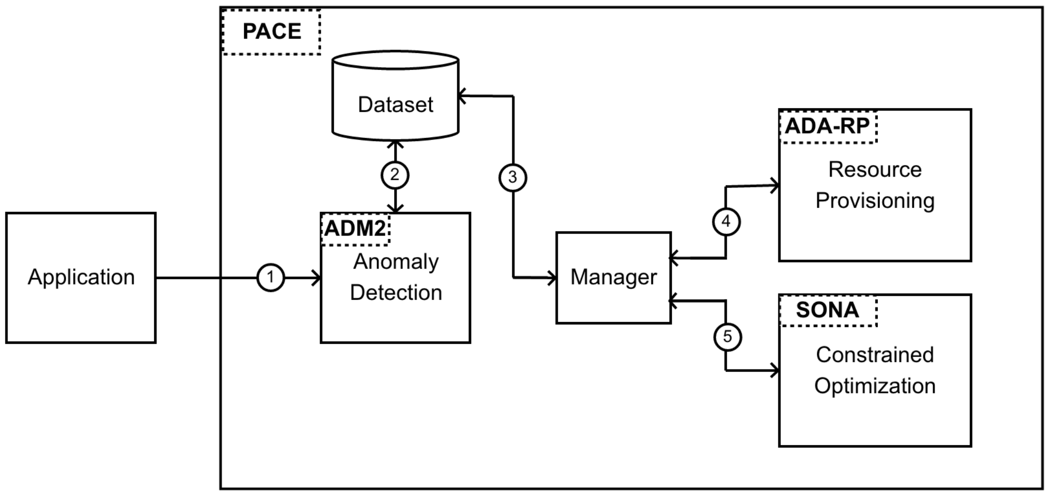


Figure 3.3: Architecture of PACE framework.

1. PACE monitors the target application to extract a variety of metrics including CPU resource usage, memory usage, disk usage and application throughput among others. Consequently, ADM2 service analyzes application metrics to automatically detect aberrant application behavior.

2. ADM2 service ensures that PACE collects data that have been extracted under normal application behavior. The latter suggests that ADA-RP and SONA services are not prone to incorrect decisions due to abnormal system behavior.

3. PACE manager extracts the required datasets (e.g., SONA dataset) to support the proposed data-driven techniques of ADA-RP and SONA services.

4. The manager injects the datasets and the configuration settings to ADA-RP service to enable the auto-scaling mechanism for resource provisioning. Ergo, the manager adjusts system resources based on ADA-RP scaling plan.

5. The manager injects the datasets and configuration settings to SONA service that enables constrained system optimization based on user requirements. The manager incorporates SONA recommendations to consider long-term application requirements.

It has to be mentioned that PACE manager incorporates the information from both ADA-RP and SONA services. First, ADA-RP service is used to enable both reactive and proactive auto-scaling in order to adjust application resources based on workload demands. ADA-RP uses time series analysis based on CPU usage and memory usage metrics to respond to workload fluctuations. In that context the user has the ability to configure the frequency of scaling actions. For example, a lower frequency suggests short-term resource adjustments so as to adapt to workload fluctuations. Consequently, ADA-RP service communicates with the resource manager to manage system resources accordingly.

At the same time, aggregated time series data and contextual application information is being collected into the database. Time series data that describe a single workload execution (e.g., 300 records for 300 seconds of workload execution) are being aggregated to a single record in the database. Additionally, each record is enriched with information about the running system during the workload execution including the number of vCPU cores, the size of memory, the size of disk and the system cost among others. SONA uses the aforementioned records to support the constrained optimization process. Finally, PACE manager decides to configure system resources based on SONA recommendations that enable long-term improvements and fine-grained resource management over numerous workload executions.

3.3 ADM2 Service for Anomaly Detection

This section introduces ADM2 service for anomaly detection in cloud computing environments. In particular ADM2 identifies abnormal patterns to alert system administrators. This also ensures that only normal patterns are being collected and used for further analysis. In more detail, ADM2 service uses LSTM Autoencoders to reconstruct normal sequence representations of application metric data streamed from cloud systems. Then, unseen observations are being classified as normal or abnormal based on the reconstruction error produced by the LSTM Autoencoder. The LSTM model introduced as an algorithm that follows RNN architecture but promises a substantial improvement in sequential data with long-range dependencies. As a result, the LSTM Autoencoder model architecture used to support the proposed methodology for detecting abnormal patterns in time series.

3.3.1 LSTM Autoencoder for anomaly detection

Autoencoder is a fundamental paradigm of unsupervised learning with the ability to reconstruct input features with the least possible amount of distortion (Baldi, 2012). The Autoencoder consists of an encoder and a decoder. The encoder extracts the hidden features from the input data and encodes the information into a learned representation vector. Then, the decoder receives the encoded vector as an input and reconstructs the original sequence with a reconstruction error.

LSTM Autoencoder combines the ability of the Autoencoder to extract the most representative information with the advantage of the LSTM to process sequential data with long-range dependencies. ADM2 uses normal data to train the LSTM Autoencoder that learns to reconstruct normal univariate time series data while on the contrary produces high reconstruction error on abnormal time series. Thus, the reconstruction error used as a score to classify future data points as normal or abnormal. Figure 3.4 shows the architecture of the LSTM Autoencoder. ADM2 uses one LSTM layer for the encoder and one LSTM layer for the decoder. The encoder takes the input sequence $\langle x_1, x_2, ..., x_n \rangle$ and encodes the information into an encoded feature vector. Then, the encoded feature vector is being used as an input for the decoder that tries to reconstruct the original sequence into $\langle \widehat{x_1}, \widehat{x_2}, ..., \widehat{x_n} \rangle$.

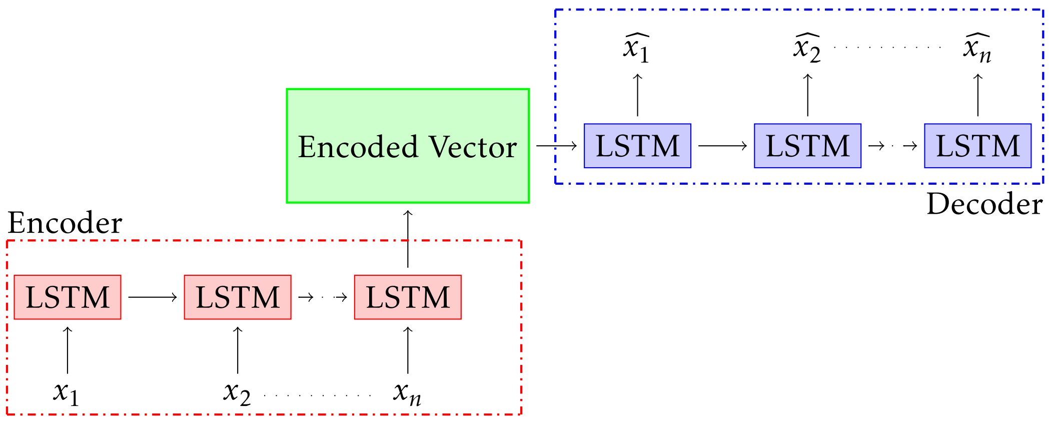


Figure 3.4: Univariate time series data representation based on LSTM Autoencoder.

The LSTM Autoencoder is trained to minimize the Mean Absolute Error (MAE) that measures the average of the absolute differences between the observed and the reconstructed observation as follows

$$MAE = \frac{\sum_{i=1}^{n} |x_i - \widehat{x_i}|}{n} \tag{3.1}$$

where x_i and $\widehat{x_i}$ are the actual and the reconstructed observations respectively. Then, the reconstruction error is used as a score to detect future outliers. Since the LSTM Autoencoder trained on normal sequences, it produces low reconstruction error on future normal series while on the contrary generates high reconstruction error on abnormal sequences. Thus, new observations are being classified as normal if the reconstruction error is lower than user's threshold or abnormal if it is not.

3.3.2 The Algorithmic structure of ADM2 service

Algorithm 1 illustrates the algorithmic structure of ADM2 service. The input values include the time series of normal workload executions Tn, the time series of monitored data Tm (e.g., CPU usage), the training parameters Pt to train the LSTM Autoencoder and the reconstruction error threshold Et to label a series as normal or abnormal.

Algorithm 1 ADM2

```
    Input
      $Tn$ Time series of normal workload executions
      $Tm$ Time series of monitored data
      $Pt$ Training parameters
      $Et$ Reconstruction error threshold
 1: procedure ANOMALY DETECTION($Tn, Tm, Pt, Et$)
 2:     $LSTM\_AE\_train(Tn, Pt)$
 3:     $Rs = LSTM\_AE\_reconstructed(Tm)$                ▷ Reconstructed series
 4:     for $series$ in $Rs$ do
 5:         if $error(series) > Et$ then
 6:             $System\_Alert()$
 7:         else
 8:             $System\_not\_Alert()$
 9:         end if
10:     end for
11: end procedure
```

Then, the anomaly detection procedure is being initialized to automatically detect aberrant patterns in cloud environments. As mentioned earlier, the LSTM Autoencoder is being trained on normal times series data, that is workload executions without abnormal interference such as network delay, CPU overload or disk failure. Consequently, the LSTM Autoencoder reconstructs new monitored data that labeled as normal or abnormal based on their reconstruction error and the threshold parameter Et. For example, if the reconstruction error of a data sequence is greater than the reconstruction error threshold, then the record is being labeled as abnormal and the system administrator is being alarmed. On the other hand, if the reconstructing error is less than the reconstruction error threshold, the record is being labeled as normal and stored into the database.

3.4 ADA-RP Service for Resource Provisioning

This section presents the ADA-RP service to support resource provisioning in cloud environments. ADA-RP enables both reactive auto-scaling based on threshold-based rules and proactive auto-scaling based on time series analysis. This section discusses i) the ADA-RP service, ii) the elasticity process of ADA-RP service, iii) the algorithmic structure of ADA-RP service and iv) the learning phase to support proactive auto-scaling based on time series analysis.

3.4.1 ADA-RP service

ADA-RP service consists a hybrid auto-scaler that supports both reactive and proactive scaling techniques. ADA-RP uses threshold-based rules to support reactive auto-scaling while it also introduces a machine learning-based proactive approach to adjust system resources based on future workload demand. In more detail, threshold-based rules have been used to auto-scale containerized applications based on memory usage percentage metric. As a result, if the memory usage percentage exceeds a predefined upper threshold called UT, then additional memory is being allocated to the container based on a scaling parameter called Sv. On the other hand, if the memory usage percentage is smaller than a predefined lower threshold called LT, the ADA-RP service reduces the amount of allocated memory based on Sv parameter.

Although, various resource usage metrics can be handled by simplified threshold-based rules, additional techniques required to ensure availability of virtualized resources. In this thesis, a hybrid machine learning approach proposed to provision cloud resources based on future workload demand. In more detail, historical data used to train K-means clustering algorithm to partition time series into High, Medium and Low demand clusters based on CPU utilization levels. Then, the average sequence of each cluster is measured and used as a representative sequence for High, Medium and Low demand groups respectively.

Consequently, the target application is being deployed and monitored to collect resource usage metrics including the CPU usage percentage. The latter is being used as an input for the CNN model to predict future workload demand. The predicted sequence is being segmented into smaller sequences based on a time window length parameter Wl. Then, the distance between each segment and the three representative sequences of each cluster is calculated. As a result, each segment inherits the label from the closest representative sequence. Finally, ADA-RP creates a scaling plan based on the label of each segment and a budget limit parameter Bl to automatically adjust the allocated resources.

3.4.2 The elasticity process of ADA-RP service

The elasticity process of the ADA-RP service enables both reactive and proactive auto-scaling techniques to ensure QoS requirements of cloud applications. The reactive approach supports on-demand resource allocation based on monitored data collected from the running system. In that context, threshold-based rules have been used to trigger scaling actions based on application memory usage percentage. Ergo, the proposed framework considers the following three parameters to initiate reactive auto-scaling: i) upper threshold parameter UT, ii) lower threshold parameter LT and iii) the scaling parameter Sv. ADA-RP service analyze the monitored data on-the-fly and increases the allocated memory resources if the memory usage percentage exceeds the upper threshold parameter UT. On the other hand, ADA-RP decreases the allocated memory resources if the memory usage percentage is below the lower threshold parameter LT. In this context, the scaling parameter Sv defines the amount of memory (in GB) that ADA-RP allocates or deallocates on each scaling action. For instance, if the lower threshold parameter has been set to 25% ($LT = 0.25$) and the scaling parameter is set to 2 ($Sv = 2$), ADA-RP service will subtract 2 GB of memory from the running application each time the memory usage percentage drops below 25%. The elasticity process of the reactive technique is as follows:

(i) The user configures the input parameters including the upper threshold UT, lower threshold LT and the scale parameter Sv.

(ii) ADA-RP service initiates the monitoring engine to retrieve application and resource usage metrics based on a given scrape interval (e.g., 5 seconds).

(iii) The application is being deployed on cloud while a workload is being executed to demonstrate a real-world application paradigm.

(iv) Finally, ADA-RP analyzes the monitored data on-the-fly and applies the threshold-based rules to trigger scaling decisions.

Although, threshold-based rules are efficient for handling memory resource requirements, alternative techniques are required to ensure availability of CPU resources. This is due to the fact that CPU usage percentage metric fluctuates strongly during the run phase of a workload (e.g., YCSB). As a result, a reactive approach would introduce QoS violations since it triggers scale decisions when the application has already reached a saturation level (Iqbal, Erradi, and Mahmood, 2018).

Having said that, a proactive auto-scaling approach is introduced to incorporate future workload demand and meet application requirements. As mentioned earlier, ADA-RP uses K-means algorithm to partition historical time series into High, Medium and Low demand clusters based on the CPU utilization metric. Consequently, the Dynamic Time Warping Barycenter Averaging (DBA) (Petitjean, Ketterlin, and Gançarski, 2011) has been used as a global averaging method to compute a) the average sequence during the K-means clustering algorithm and b) for averaging the sequences of each K-means cluster. The three average sequences form the representative sequences for High, Medium and Low demand clusters respectively.

In the prediction phase, CNN model is being used to predict future workload demand based on the prediction step parameter Ps, that is, the prediction step number. Then, the predicted sequence is being segmented into smaller sequences based on a time window length parameter Wl that defines the time, measured in seconds, before ADA-RP makes a new scaling decision. The dynamic time warping (DTW) (Berndt and Clifford, 1994) algorithm has been used to calculate the distance between each segment and the representative sequences for High, Medium and Low demand clusters. As a result, each segment inherits the label from the closest representative sequence. Finally, a deployment manager adjusts application resources according to the label of each segment, the scaling parameter Sv and the budget limit parameter Bl. The latter enables the user to set a cost limit before ADA-RP triggers a scaling action. The elasticity process of the proactive approach is as follows:

(i) The user configures the input parameters including the prediction step parameter Ps, the time window length parameter Wl, the scale parameter Sv and the budget limit parameter Bl.

(ii) ADA-RP service initiates the monitoring engine to retrieve application and resource usage metrics.

(iii) Historical data used to train K-means algorithm that clusters time series into High, Medium and Low demand clusters.

(iv) The DBA method is used to compute the average sequence of each cluster that form the representative sequences for High, Medium and Low demand clusters respectively.

(v) The application is being deployed on cloud and the workload is being executed to stress the running system.

(vi) CNN model is trained on CPU usage percentage metric and produces future predictions based on the prediction step parameter Ps.

(vii) The predicted sequence is being segmented into smaller sequences based on the time window length parameter Wl.

(viii) Each segment inherits the label from the closest representative sequence based on the DTW algorithm.

(ix) ADA-RP creates a scaling plan based on the labels of each segment and the cost limit constraints.

(x) Finally, PACE manager allocates resources to the running application.

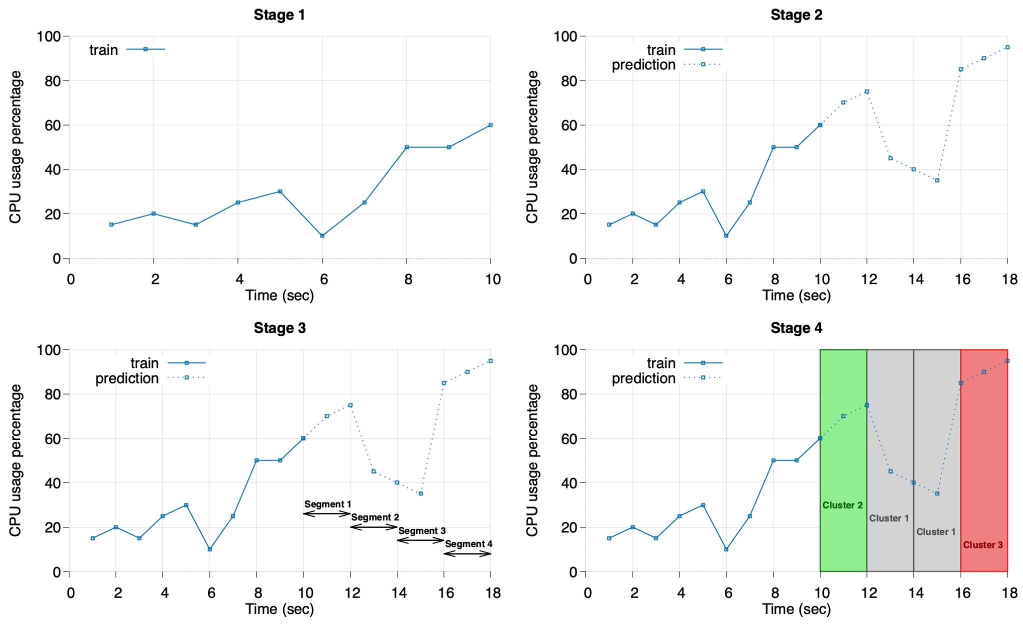


Figure 3.5: ADA-RP proactive auto-scaling stages.

Figure 3.5 illustrates ADA-RP proactive auto-scaling stages. In Stage 1, the main application is being monitored based on the CPU usage percentage metric. In Stage 2, the CPU usage percentage is being used as a training input for the CNN model that generates future predictions based on the prediction step parameter Ps (e.g., $Ps = 8$). In Stage 3, the predicted sequence is being segmented into smaller sequences based on a time window length parameter Wl (e.g., $Wl = 2$). Finally, in Stage 4, each segment is being labeled as High, Medium or Low demand based on the representative sequences extracted from each K-means cluster. Given that the predicted workload has been segmented and labeled as High, Medium or Low demand, ADA-RP service creates a scaling plan that automatically adjusts container's resources based on application demand.

To demonstrate the above interactions, an example is introduced as follows. The CNN model predicts the CPU utilization for the next 60 seconds ($Ps = 60$) and the time window length parameter Wl is set to 15 seconds. As a result, ADA-RP service will segment the prediction sequence into 4 subsequences each consisting of 15 time steps (Ps/Wl). Then, the DTW distance between each segment and the three representative sequences extracted from the High, Medium and Low demand clusters is calculated. Consequently, each sequence will be labeled as High, Medium or Low demand and the values will be stored into an array i.e., $S_r = [High, Medium, Medium, Low]$. Then, ADA-RP will produce a scaling plan, called Sp for the next 60 seconds, such as $S_p = [Vc + Sv, Vc, Vc, Vc - Sv]$ where Vc represents the initial amount of vCPU cores and Sv represents the scaling parameter that specifies the amount of vCPU cores that will be added or subtracted from the running container. This example suggests that the application should scale up the initial resources ($Vc + Sv$) for the first segment $[High]$, return to the initial amount of vCPU cores (Vc, Vc) for the next two segments $[Medium, Medium]$ and scale down resources ($Vc - Sv$) for the last segment $[Low]$.

3.4.3 The algorithmic structure of ADA-RP service

The ADA-RP service enables both reactive and proactive methods for autoscaling cloud applications. Algorithm 2 demonstrates the algorithmic structure of ADA-RP reactive auto-scaling approach.

The input values include the time series of monitored data Tm, the upper threshold UT, the lower threshold LT and the scaling parameter Sv. The ADA-RP reactive auto-scaling procedure is being initialized to automatically adjust

Algorithm 2 ADA-RP reactive auto-scaling

```
    Input
      $Tm$ Time series of monitored data
      $UT$ Upper threshold
      $LT$ Lower threshold
      $Sv$ Scaling parameter
 1: procedure REACTIVE AUTO-SCALING($UT, LT, Sv$)
 2:     while $Monitoring == true$ do
 3:         if $Tm > UT$ then
 4:             $Scale\_Up(Sv)$
 5:         else if $Tm < LT$ then
 6:             $Scale\_Down(Sv)$
 7:         else
 8:             $No\_Scaling\_Action()$
 9:         end if
10:     end while
11: end procedure
```

cloud resources based on the proposed threshold based approach. Ergo, if a data point of the monitoring data exceeds the upper threshold parameter UT, ADA-RP increases the allocate resources based on the Sv scaling parameter. On the contrary, if a data point is lower than the lower threshold parameter LT, ADA-RP decreases the amount of allocated resources based on the Sv parameter. Lastly, if a data point is neither greater than the UT parameter or smaller than the LT parameter, no scaling action is being triggered.

Algorithm 3 illustrates the input and output data as well as the step-wise procedure of ADA-RP proactive approach. As previously discussed, ADA-RP uses a time series dataset for workload characterization namely Tw, a time series dataset for workload prediction namely Tp and the input parameters including the prediction step number Ps, the initial amount of vCPU cores Vc, the scaling parameter Sv, the time window length Wl and the budget limit Bl.

Algorithm 3 ADA-RP proactive auto-scaling

```
    Input
      $Tw$ Time series for workload characterization
      $Tp$ Time series for workload prediction
      $Ps$ Prediction step number
      $Wl$ Time window length
      $Vc$ Initial vCPU cores
      $Bl$ Budget limit
      $Sv$ Scaling parameter
    Output
      $Dp$ Deployment Plan
 1: procedure PROACTIVE AUTO-SCALING($Tw, Tp, Ps, Wl, Vc, Bl, Sv$)
 2:     Initialize an empty list $Sp$
 3:     function CNN($Tp, Ps$)
 4:         $CNN\_train()$
 5:         $CNN\_predict()$
 6:         return $Predicted\_Workload$
 7:     end function
 8:     function K-MEANS($Tw$)
 9:         $K - means\_train()$
10:         $DTW\_Barycenter\_Averaging()$
11:         return $R$                                    ▷ Representative Sequences
12:     end function
13:     function SEGMENT($Predicted\_Workload, Wl$)
14:         return $S$                                    ▷ Segmented Sequences
15:     end function
16:     function DTW($R, S$)
17:         return $Ls$                                   ▷ Labeled Sequences
18:     end function
19:     function SCALE_PLAN($Ls, Vc, Sv$)
20:         for $eachlabel$ in $Ls$ do
```

```
21:             if $label == High$ then
22:                 $Sp.append(Vc + Sv)$
23:             else if $label == Low$ then
24:                 $Sp.append(Vc - Sv)$
25:             else
26:                 $Sp.append(Vc)$
27:             end if
28:         end for
29:         return $Sp$
30:     end function
31:     function DEPLOYMENT_MANAGER($Sp, Bl$)
32:         $Deploy(Sp, Bl)$                        ▷ Budget-aware Deployment
33:         return $Dp$
34:     end function
35: end procedure
```

Firstly, the ADA-RP initializes an empty list Sp that will store the future scaling plan. Then, the CNN model is trained on time series data collected by monitoring the target application. During the training process the prediction step number Ps is being considered to estimate the future workload demand, that is the $Predicted\_Workload$ sequence. Additionally, the K-means model is trained on time series to enable workload characterization by clustering each sequence into High, Medium and Low demand clusters. Consequently, three cluster representative sequences are being created by using the DBA averaging method. Given $k = 3$ number of clusters, let R be a set of three representative sequences as follows:

$$R = \{H, M, U\} \tag{3.2}$$

where $H = \langle h_1, h_2, ..., h_l \rangle$ is the representative sequence for High demand cluster, $M = \langle m_1, m_2, ..., m_l \rangle$ is the representative sequence for Medium demand cluster and $U = \langle u_1, u_2, ..., u_l \rangle$ is the representative sequence for Low demand cluster respectively, while l is the length of each sequence. It has to be mentioned that each representative sequence has the same length with the time window length where $Wl = l$ seconds.

The $Predicted\_Workload$ sequence is being segmented into Ps/Wl segmented sequences based on the time window length parameter Wl that defines the length of each segment, therefore the frequency of decision making. Let the predicted sequence consisting of $Ps = p$ time steps and the time window length parameter set to $Wl = l$ seconds, ADA-RP will produce $s = Ps/Wl$ sequence segments as follows:

$$S = \{S_1, S_2, ..., S_s\} \tag{3.3}$$

where S_s is the s-th segmented sequence of length l. Then, the distance between each sequence of the segmented sequences S and the representative sequences R produced by the K-means model is being calculated. Ergo, each segment sequence inherits the label from the closest representative sequence. As a result a list of labels is constructed as follows:

$$Ls = (y_1, y_2, ..., y_s) \mid y \in A \tag{3.4}$$

where the label y_i is an element belonging to a finite set of classes $\mathbb{A} = \{High, Medium, Low\}$ indicate the state of the running application in terms of CPU utilization level. Then, a scaling plan is being created based on the labeled list Ls, the initial vCPU cores Vc and the scale parameter Sv. Finally, the deployment manager will initiate the scaling decisions based on user's cost limit requirements.

3.4.4 Learning phase

In this thesis, a hybrid learning approach has been proposed to enable adaptive resource provisioning in cloud environments. A dataset has been generated under different system and workload configurations and used to train the K-means clustering algorithm. In the past, the K-means algorithm has been widely used across a wide range of research fields for clustering time series data (Niennattrakul and Ratanamahatana, 2007), (Guo, Jia, and N. Zhang, 2008), (Dong, Qian, and L. Huang, 2017).

K-means clusters time series samples into k predefined number of groups by trying to minimize the Within Group Sum of Squares (WGSS), also called inertia, a measure for cluster coherence. Equation 3.5 represents the equation for the intraclass inertia (Petitjean, Ketterlin, and Gançarski, 2011). Let E^T be the space of all sequences of length T. Given $S = \{S_1, S_2, ..., S_N\}$ as the set of all sequences the average sequence $C^{(T)}$ must fulfill

$$\forall X \in E^T, \sum_{n=1}^{N} DTW^2(C^{(T)}, S_n) \leq \sum_{n=1}^{N} DTW^2(X, S_n) \tag{3.5}$$

The K-means randomly assigns each sample to a cluster and computes the average cluster sequence. Then, each sample is being re-assigned to the closest average cluster sequence and the average sequences are being re-calculated. The same process is being repeated until a decrease of the objective function is lower than a tolerance value.

In this domain, various strategies have been proposed for averaging a set of sequences. Local averaging techniques such as the NonLinear Alignment and Averaging Filters (NLAAF) (Gupta et al., 1996) and the Prioritized Shape Averaging (PSA) (Niennattrakul and Ratanamahatana, 2009) have been proposed to compute the average sequence over a set of sequences. Such techniques compute the mean of N sequences by taking the pairwise averaging. However, pairwise averaging is quite sensitive to order, thus repeating the averaging process may significantly alter the quality of the result. For that reason, Petitjean, Ketterlin, and Gançarski (2011) proposed the DBA, a global averaging method, that iteratively refines the initial average sequence in order to minimize its squared DTW distance to averaged sequences. For each refinement DBA works in two steps: i) computes the DTW distance between each sequence and the temporary average sequence and ii) updates each coordinate of the average sequence based on the contribution of one or more coordinates of each sequence. In step 1, the DTW distance between the average sequence and an individual sequence involves the searching sequence C at iteration i for instances on sequence T as:

$$C = \langle C_1, C_2, C_3, ..., C_l \rangle \tag{3.6}$$

$$T = \langle T_1, T_2, T_3, ..., T_l \rangle \tag{3.7}$$

where C is the average sequence, T is a sequence associated to it and l is the length of sequences. It has to be mentioned, that DTW can align sequences of different lengths, however in this work the average sequence and the segment sequences are of the same length.

As a result, the sequences C and T are used to form a l-by-l square matrix where each point (i, j) corresponds to an alignment between elements C_i and T_j. The cumulative distance for each point in the matrix is being calculated using dynamic programming to evaluate the following recurrence:

$$\gamma(i,j) = \delta(i,j) + min[\gamma(i-1,j), \gamma(i-1,j-1), \gamma(i,j-1)] \tag{3.8}$$

that is the sum of the distance between two elements and the minimum of the cumulative distances of the adjacent cells. In DBA, the goal is to minimize the sum of squared DTW distances between the average sequence and the set of sequences by refining the average sequence iteratively. Thus, in step 2, DBA updates each of the the average coordinates as the barycenter of coordinates associated to it during the first step. Given a set of sequences $S = \{S_1, ..., S_N\}$ to be averaged, DBA starts from the initial average sequence $C = \langle C_1, ..., C_l \rangle$ and iteratively refines the average sequence to $C' = \langle C_{1'}, ..., C_{l'} \rangle$, that is the updated average sequence at iteration $i+1$, where l is the length of sequences and N is the number of sequences.

As previously mentioned, ADA-RP uses a pro-active approach to reduce cloud costs and system inefficiencies based on CNN future predictions. The CNN model has been widely used in the past for predicting time series (Astillo et al., 2022), (Koprinska, Wu, and Z. Wang, 2018). In this thesis, the CNN learns from the historical time series in order to estimate future workload demand. Time series consist of a single dimension, that is the time dimension, thus can be defined as one dimensional vector. As a result, CNN uses one dimensional convolutions to process the input time series. During the learning phase, several hyperparameters considered to fine tune the CNN model. Table 3.1 shows the best values of the hyperparameters that obtained after conducting various experiments.

Table 3.1: Optimal CNN configurations

Parameter	**Value**
Convolutional layers	2
Convolutional layer filters	64
Pooling layers	2
Fully connected layer	1
Fully connected layer neurons	300
Optimizer	Adam
Activation function	Relu
Loss function	RMSE
Learning rate	0.0001
Number of epochs	100
Batch size	64

The CNN model contains an input layer, two convolutional layers, two pooling layers, a fully connected layer and an output layer. The convolutional layers consist of 64 filters and a kernel size of two. The max pooling operation is performed after each convolutional layer to downsample the input representation by taking the maximum value over a spatial window of size two. Then, flattening is performed to convert the convolved and pooled features into one dimensional array that is passed to a fully connected layer. Lastly, an output layer generates a predicted sequence of equal size with the input sequence. Rectified Linear Unit (Relu) (Nair and Hinton, 2010) is applied after each convolution operation and Adam optimizer used for first-order gradient-based optimization based on adaptive estimates of lower-order moments (Kingma and Ba, 2014). The learning rate set to 0.0001 and the model is trained for 100 epochs with a batch size of 64. To evaluate the effectiveness of the proposed model, we use the Root Mean Squared Error (RMSE). The latter can be represented mathematically as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3.9}$$

where y_i and $\hat{y}_i$ are the observed and the predicted values, and n is the total number of observations. The train and the test RMSE values are being reported in the performance evaluation section.

3.5 SONA Service for System Optimization

This section introduces SONA, a service for optimizing cloud application performance based on user requirements. SONA uses ANN and GA algorithms as a hybrid approach for enabling constraint performance optimization. This section discusses i) SONA service, ii) the optimization process of SONA service and iii) the algorithmic structure of SONA service.

3.5.1 SONA service

SONA service supports constrained performance optimization of cloud-based systems. The service uses data records that describe application behavior across different workload tasks. The records are being organized in a form of a dataset that contains attributes to describe resource configurations (e.g., CPU cores, memory, hard disk), type of configurations (e.g., disk type), application configurations (e.g., InnoDB buffer pool size), workload parameters (e.g., number of warehouses), performance metrics (e.g., transactions per minute), resource utilization metrics (e.g., average percentage of idle CPU resources), cloud costs, cost-performance ratio, and cloud infrastructure information (e.g., cloud region).

Consequently, SONA uses GA to identify the optimal configurations that produce the highest application performance. However, running all possible combinations in a virtualized environment, each time the optimization method needs to evaluate the proposed configurations, is not always feasible due to time and cost constraints. Thus, ANN have been proposed to model the high-dimensional data and extract non-linear relationships among variables. Then, the ANN model used to evaluate the fitness of each individual in the optimization process. Furthermore, the proposed method allows users to introduce their requirements in an interactive manner. In more detail, SONA uses penalty techniques to constrain the optimization process based on four requirements including (a) the cloud costs, (b) the cost-performance ratio, (c) the baseline performance as a reference for comparison with the main application and (d) the average percentage of idle CPU resources to ensure sufficient utilization levels.

3.5.2 The optimization process of SONA service

SONA service proposed to support constrained optimization of cloud systems with zero overhead to the main application. The system includes (a) a source database application that is monitored on-the-fly, (b) a clone database application that is used to run test workload executions, (c) the coordinator for constrained performance optimization and decision making and (d) the source manager for triggering scaling actions. Thus, SONA service provides the following functionalities:

- A monitoring engine that collects valuable information of the running system. The latter, consists of a database system that executes state-of-the-art workloads to emulate a real-world case scenario.
- A hybrid approach that uses ANN and GA for constrained performance optimization.
- A lightweight architecture using Linux containers to support fast application deployment with zero overhead to the main system.
- An engine that provides recommendations to the system administrator based on user and application requirements.

The objective of the optimization process is to select the optimal system parameters that maximize the performance metric and satisfy user requirements. SONA uses real-coded GA where each chromosome consists a sequence of genes that represent the optimization variables. However, executing a workload each time the GA needs to evaluate an individual of a population is not always feasible due to time and resource constraints. For that reason, a novel dataset has been created and used as an input for the ANN to model the underlined relationships and form an ANN-based objective function for the GA. Instead of generating simulated application executions, the proposed dataset consists of real experimental observations that have been pre-processed and used for the neuro-genetic approach. This approach enables SONA to execute faster and more efficient.

Furthermore, SONA introduces a containerized environment, consisting of a source and a clone system that allows fast application deployment. The clone system is being deployed as a replica of the source system to avoid system overhead. Figure 3.6 illustrates the flow of events described as follows.

1. The source system provides a variety of configurations to the source manager. These include the system status that characterizes the baseline performance of the source system and the system policies that define the constraints based on user requirements.
2. The source manager injects the configuration file in the Coordinator in order to be used for the constrained optimization process.
3. The Coordinator access the records to insert, update and collect data for modeling and optimization tasks.
4. The Coordinator applies pre-processing and feature selection techniques to the records to create an input dataset. This latter is used for the training and validation process of the ANN model. Finally, the Coordinator saves the trained ANN model.
5. The Coordinator initiates the constrained optimization process based on the hybrid neuro-genetic approach. The GA uses an ANN-based fitness function in the optimization process to select the best parameters that produce the highest application performance in terms of executed transactions. Furthermore, penalty techniques applied to constrain the optimization problem based on different system policies such as the cost-performance ratio.

6. The Coordinator injects the recommended configurations to the clone system (e.g., number of CPU cores, Memory Size, InnoDB buffer pool size, number of warehouses etc.). The clone system resources are being adjusted to create the desired experimental environment. The Coordinator configures the database and the selected workload (e.g., MySQL and TPC-C) in the clone system to execute the experimental scenario.
7. The clone system returns the experimental records (e.g., resource size, resource type, TpmC) and the Coordinator updates the records accordingly. In addition, the Coordinator retrains the ANN model with the updated data to produce more accurate predictions. Thus, the fitness function of the GA is dynamic, and it is being improved online over time, when new runs are executed. It is expected that as more data is generated from new runs, SONA recommendations will be improved over time. At this stage the Coordinator identifies the optimal configuration for the particular workload and the selected user policies.
8. The Coordinator injects the recommended configurations to the source manager, based on a decision making process, that is the acceptable difference between the experimental and the predicted TpmC value. The allowable residual should not exceed 10% of the experimental value.
9. The source manager communicates the recommended configurations to PACE manager that decides to scale and tune the source system accordingly.

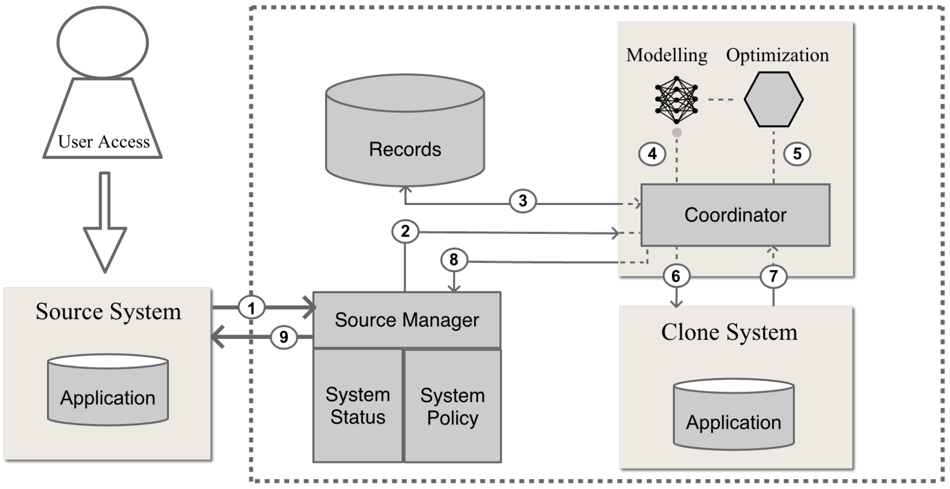


Figure 3.6: SONA operation flow diagram.

It has to be mentioned that using a clone container as an experimental setting, does not add overhead to the source system. In addition, the proposed method is flexible and supports dynamic online configuration, thus, the user is able to control the System policies in real-time. As a result, the feature space of the GA is updated based on user policies in order to explore new possible performance improvements. This dynamic setup is supported by the Linux Containers (LXC) that support on-the-fly resource allocation using the LXC control groups (cgroups) [1]. The cgroups resource management allows the clone system to adjust resources and set the size of a container according to different experimental scenarios.

Data pre-processing

The SONA service supports constraint optimization of cloud applications based on monitored data. During the monitoring process, SONA extracts and collects information about the running system including application metrics, resource usage metrics and the baseline performance of the source system. Next, SONA completes a pre-processing stage to create the input dataset for training the ANN in a supervised manner. SONA supports a variance

[1]`https://www.kernel.org/doc/Documentation/cgroup-v1/`

threshold-based technique for selecting features that enhance ANN model predictive power. As a result, ANN uses the number of CPU cores, the memory size, the disk size, the disk Type, the InnoDB buffer pool size and the number of Warehouses to predict the executed transactions per minute (TpmC).

The selected features form the input dataset in which data normalization applied for effective training of the ANN (Sola and Sevilla, 1997). SONA uses the min-max normalization approach to measure the difference between the field value and the minimum value, and to scale this difference by the range (D. T. Larose and C. D. Larose, 2014). Equation 3.10 shows the normalization formula:

$$\tilde{x_i} = \frac{x_i - x_{min}}{x_{max} - x_{min}}. \tag{3.10}$$

where $\tilde{x_i}$ is the normalized value, x_i is the original value, and x_{min} , x_{max} are the minimum and the maximum values of the attribute x. Additional features used to support the constraint optimization process including the Idle CPU, the Cost and the cost-performance ratio. Next, the hybrid neuro-genetic approach based on ANN and GA algorithms is being explained.

ANN based objective function

ANN is an algorithm inspired by the biological neurons and it is widely used in many research fields to solve complex problems. In this work, an ANN model is trained to relate input configurations with an application performance metric (e.g., transactions per minute), with the aim to accurately predicting the performance of future configurations. The ANN model uses the back propagation algorithm (Rumelhart, Hinton, and Williams, 1985) to find the optimal weights that minimize the loss function, that is the mean squared error. The number of neurons for the input layer is determined by the number of features selected to train the model while the output layer consists of one neuron to produce predictions of the performance metric. SONA uses 2 hidden layers with 50 neurons in each layer to learn the hidden representations. The Rectified Linear Unit function (ReLU) (Maas, Hannun, and Ng, 2013) used as the activation function with HE weight initialization strategy

(He et al., 2015). Furthermore, SONA uses the adaptive moment estimation (Adam) as a computationally efficient algorithm for first-order gradient-based optimization of stochastic objective functions (Kingma and Ba, 2014).

In the training phase, the ANN fits the training data to find the optimal weights that produce the lowest mean squared error. However, creating a highly flexible model that follows the errors too closely may lead to overfitting (James et al., 2013). Thus, SONA enables an early stopping technique to interrupt the training process before convergence if the validation score is not improving over a particular number of epochs as shown in (Caruana, Lawrence, and Giles, 2001). As a result, the ANN model forms an ANN-based objective function used for the optimization process of the GA.

Neuro-genetic optimization

GA are stochastic search algorithms that mimic biological evolution to locate optimal solutions in complex landscapes. The objective is to select the optimal configurations under certain constraints, in order to obtain the highest number of executed transactions in the database. GA introduced to optimize that objective using a fitness function based on the ANN model. As a result, the ANN model predictions used to evaluate the fitness of each individual during the optimization process.

Real-coded genetic algorithms

GA's efficiency strongly depends from the technique used to encode variables (Deep et al., 2009). In this thesis, real-coded GA used since natural representations are more efficient and produce better results (Shen, L. Wang, and Q. Li, 2007), (Damousis, Bakirtzis, and Dokopoulos, 2003). GA represent each individual as a chromosome of a fixed length. Ergo, each chromosome contains a sequence of genes that represent the optimization variables while each gene ranges between its upper and lower boundaries based on the user requirements. Genes are related to data records including CPU cores, Memory, Disk, Disk type, InnoDB buffer pool size and number of warehouses. The latter remains stable during the optimization process as a non-tunable component.

Selection

The selection stage in GA ensures that a fixed size of the chromosome population is used to improve the average fitness and ensure the convergence of the algorithm (Negnevitsky, 2005). In this research, SONA uses the roulette wheel selection (Goldberg, 2006) where each parent is being selected for mating with a probability p_i based on their fitness score. The probability of selection follows equation 3.11:

$$p_i = \frac{w_i}{\Sigma_{i=1}^{N} w_i} \tag{3.11}$$

where w_i is the fitness of the individual i and N is the number of the population.

Genetic operators

GA use crossover and mutation as two basic genetic operators. Crossover is applied with a probability p_c and selects a random crossover point where the two parents break. Then, two parents exchange the breaking parts to create two new offsprings that contain information of both parents. If a pair of chromosomes does not crossover, with a probability $1 - p_c$, each offspring is described as the exact copy of each parent. Mutation genetic operator introduces a randomness in the evolution process as an attempt to avoid a trap on a local optimum. The mutation operation randomly changes the value of a gene, between the lower and upper boundaries, with a mutation probability of p_m.

Fitness function based on ANN

In this work, a chromosome consists a sequence of genes that represent system, application and workload configurations. Each chromosome is being evaluated based on the ability to produce high number of executable transactions. One way to evaluate a chromosome is to execute a workload in a database system and obtain the experimental throughput. However, this is not always feasible due to time and resource constraints. Thus, the ANN model is used to calculate the fitness, that is the throughput of each chromosome. Then, selection, crossover and mutation used to create new generations. This hybrid neuro-genetic approach is repeated for n number of generations until the population convergence is reached.

Penalty Techniques

Penalty techniques enables users to introduce constraints during the optimization process. SONA introduces the following constraint criteria:

- C1 The cloud infrastructure expenditures, to penalize costs that exceed user budget.
- C2 The cost-performance ratio, to penalize inefficient configurations.
- C3 The baseline performance, to penalize solutions with a lower throughput in relation to the main application.
- C4 The CPU idle, to penalize solutions with high underutilized CPU resources.

In this work, the additive form is used as follows:

$$eval(x) = f(x) + p(x) \tag{3.12}$$

where x represents the chromosome, $f(x)$ represents the fitness of the chromosome x and $p(x)$ the penalty term. The penalty term is given as follows:

$$p(x) = \sum_{j=1}^{m} R_j min[0, g_j(x)] \tag{3.13}$$

where m is the number of constraints, R_j is a positive penalty coefficient for the j-th constraint and g_j is the function of the constraint. The minimization term ensures that SONA penalizes only infeasible solutions.

To avoid negative fitness values SONA also require that:

$$|p(x)|_{max} \leq |f(x)|_{min} \tag{3.14}$$

where $|p(x)|_{max}$ is the maximum value of the penalty term and $|f(x)|_{min}$ is the minimum fitness value among the current population (Gen and Cheng, 1996).

Constraints C1, C2, C3 and C4 are represented by $g_1(x)$,$g_2(x)$, $g_3(x)$ and $g_4(x)$ functions respectively. Cloud infrastructure expenditures constraint (C1) is given as follows:

$$g_1(x) = B - c(x) \tag{3.15}$$

where B is a user Budget and $c(x)$ is a cost function using the platform policy. In has to be mentioned that during the experiments presented later in this thesis, Google Cloud Pricing calculator (Google, 2022e) is used to estimate the cost of the CPU cores, Memory, Disk Size and Disk type expressed as genes of the chromosome x. The more the expected cost exceeds users Budget, the higher the penalty term.

Cost-performance ratio constraint (C2) is given as follows:

$$g_2(x) = -\frac{c(x)}{f(x)} \tag{3.16}$$

where $c(x)$ is the cost function as in equation 3.15 and $f(x)$ is the ANN-based estimation of the TpmC score given chromosome x. The lower the cost-performance ratio, the higher the efficiency, thus, the lower the penalty term.

The baseline performance constraint (C3) is given as follows:

$$g_3(x) = f(x) - P \tag{3.17}$$

where $f(x)$ is the ANN-based estimation of the throughput score and P is the baseline performance, that is the performance produced by the source system without using the SONA framework. The smaller the estimated performance in relation to the baseline performance, the higher the penalty term.

CPU idle constraint (C4) is given as follows:

$$g_4(x) = T - h(x) \tag{3.18}$$

where T is user threshold for idle CPU resources and $h(x)$ is an ANN-based estimation of the idle CPU resources given system and application configurations, expressed as genes of the chromosome x. SONA uses an ANN model that accepts as input the CPU cores, Memory, Disk Size, Disk type, InnoDB buffer pool size and Warehouses to estimate the idle CPU resources of each individual in a population.

Clone system

SONA uses a clone system, as an efficient mechanism to validate SONA recommendations with zero overhead to the main system. As mentioned earlier, the GA outputs the fittest population, that is, the configuration parameters that produce the highest number of executed transactions based on user constraints. Then, SONA deploys a clone system as an experimental environment used to validate the predicted performance of the recommended configurations. As a result, the clone system replicates the main application and validates SONA recommendations without adding overhead to the running application. The cloning strategy uses Linux containers to enable fast application deployment and replication. Thus, the clone system deploys the replicated application, executes the desired workload and returns the experimental throughput score of the genuine execution. At this stage the Coordinator collects the genuine execution configurations and updates the records. Consequently, the ANN algorithm is retrained to learn from new patterns and increase the overall system effectiveness. The Coordinator forwards the recommended system configurations to the source manager that in turn injects them to the source system based on an allowable residual as shown in Line 8 of figure 3.6.

3.5.3 The algorithmic structure of SONA service

Algorithm 4 demonstrates the algorithmic structure of SONA service. The $Workload$ function initializes the database system (e.g., MySQL) for loading and executing the desired workload (e.g., TPC-C) based on different configurations. These include the workload configurations parameters (e.g., number of Warehouses) and the database configurations parameters (e.g., MySQL InnoDB buffer pool size).

Algorithm 4 SONA service

```
procedure SONA
    function WORKLOAD(workloadConf,DatabaseConf)
        StartDatabase()
        LoadWorkload()
        StartWorkload()
        return RawDataset
    end function
    function PREPROCESSING(RawDataset)
        FeatureSelection()
        DataNormalization()
        return InputDataset
    end function
    function ANN(InputDataset)
        trainingANN()
        save ANNModel
    end function
    function GA(SystemPolicy)
        $InitPop\ P$;
        $evaluate\ P$;
        while $isNotTerminated()$ do
            $P' \leftarrow selectparents\ P(t)$;
            $reproduction\ P'(t)$;
            $mutate\ P'(t)$;
            $evaluate\ P'(t)$;
            $P(t+1) \leftarrow nextGeneration\ P(t), P'(t)$;
            $t \leftarrow t+1$;
        end while
        return P, eThr                          ▷ Population, Expected Throughput
    end function
    function CloneRun(P)
        GenuineWorkloadRun()
        return aThr                              ▷ ExperimentalThroughput
    end function
    function SourceManager($P$)
        if $PopulationValid() == True$ then
        UpdateSystem()
        end if
    end function
end procedure
```

During the workload execution phase, SONA monitors the running system in order to collect raw data. The latter, is being pre-processed to create an input dataset used for the proposed neuro-genetic approach. Ergo, the ANN model is being trained on the input dataset to predict the number of executable transactions based on different system and application configurations. As a result, the neuro-genetic optimization uses a combination of GA and ANN-based fitness function to select the optimal configuration that maximize the number of executed transactions based on user's requirements. The selected configurations are being executed in a clone environment to avoid system overhead. Finally, the source manager validates the experimental results in order to adjust and tune system resource accordingly.

3.6 Summary

This chapter presented PACE framework to effectively provision and optimize cloud applications. The chapter introduced the topology of PACE framework and the algorithmic solution of its services. In particular, ADM2 service proposed to support anomaly detection in the cloud. As a result, ADM2 service ensures that the PACE framework is not prone to incorrect decisions due to unusual system behavior.

Furthermore, ADA-RP service introduced to enable both reactive and proactive auto-scaling techniques for cloud resource provisioning. ADA-RP service supports efficient resource allocation to avoid over-utilized and under-utilized resources during workload executions. Finally, this chapter proposed SONA service to identify system and application configurations that maximize application performance. The optimization process included a variety of constraints to satisfy user requirements and to ensure QoS requirements. The next chapter discusses PACE deployment environments and benchmarks.

4 Deployment Environment and Benchmarks

This chapter provides a discussion around the deployment environment and benchmarks of PACE framework. This thesis enables both public and private clouds in order to demonstrate the effectiveness of the proposed methodology under different use case scenarios. In that context, a variety of database systems used to emulate real-world case applications using state-of-the-art benchmarks and workloads.

Section 4.1 presents the cloud environments used to deploy PACE framework and its services. Section 4.2 presents a variety of systems used throughout this research including SQL and NoSQL databases. Section 4.3 provides an overview of the benchmarks used to evaluate the proposed framework and to demonstrate its effectiveness. Finally, section 4.4 presents the summary of the chapter.

4.1 Cloud Environment

Cloud computing includes various types of clouds depending on the service availability and accessibility level. This research enables both public and private clouds to demonstrate the effectiveness of SONA framework in different cloud environments. In that context, the computing resources of Birkbeck College have been used as a private cloud while GCP used as public cloud platform offered by Google (Google, 2022a). Google Cloud infrastructure services are available in different geographical locations across the world including North America, South America, Europe, Asia and Australia (Google, 2022d). As a result, Google Cloud offers a variety of service models to the end-user via the cloud including IaaS, PaaS and SaaS. However, this research focuses on IaaS service model that offers on demand resources such as networking, storage and computing.

GCP Compute Engine used to create and run VMs in Google's infrastructure (Google, 2022b). A VM consists of an isolated virtual environment that uses a subset of the physical resources of a computer system (Marinescu, 2022). As shown in Figure 4.1, a hypervisor allows several VMs to share hardware resources. The latter enables resource distribution over multiple machines improving both flexibility and efficiency. As a result, a VM runs its own operating system, binaries and libraries while appears to be running on the

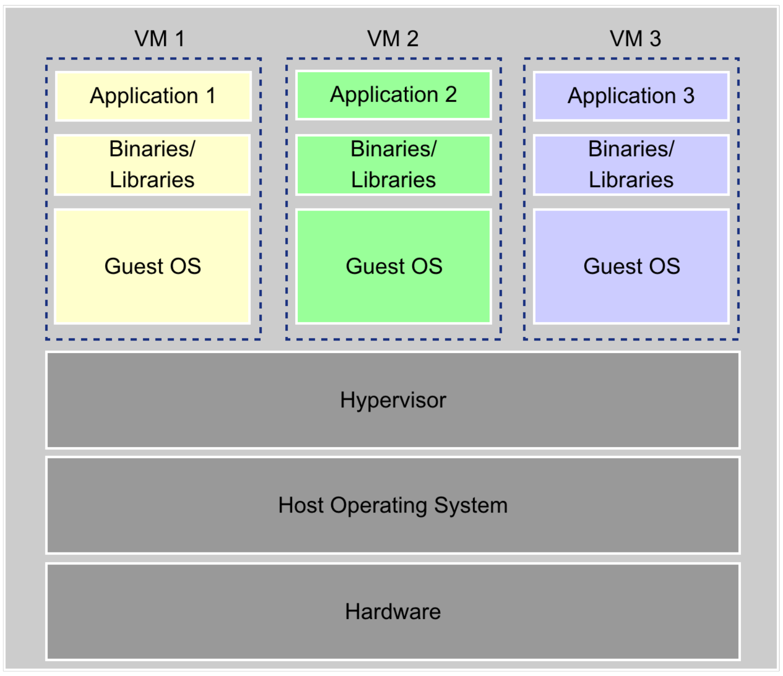


Figure 4.1: System organization for Virtual Machines

bare hardware. In this research, different types of VMs have been deployed in the cloud to host a variety of database applications that support the experimental analysis and evaluation of ADM2 service. Consequently, VMs have been configured accordingly in terms of operating system, CPU, memory, network and storage.

Additionally, PACE framework supports dynamic resource provisioning and system optimization using ADA-RP and SONA services respectively. To support the proposed methodology, containers have been used as an alternative virtualization technique also known as containerization. Containers support virtualization on the operating system level allowing multiple containers to run on the same host operating system kernel. As a result, containers have a smaller memory footprint and a shorter start-up time than VMs (Marinescu, 2022) making it feasible to deploy hundreds of containers on a physical host (Bernstein, 2014). Furthermore, containers offer an isolated environment to deploy applications and services but without the overhead introduced by virtual machines especially to I/O operations (Felter et al., 2015).

Figure 4.2 presents the system organization for containers. As shown in Figure 4.2 each container shares the host operating system while it runs its own application, binaries and libraries. As a result, containers support a light weight and cost effective solution. In this thesis, Linux containers used to al-

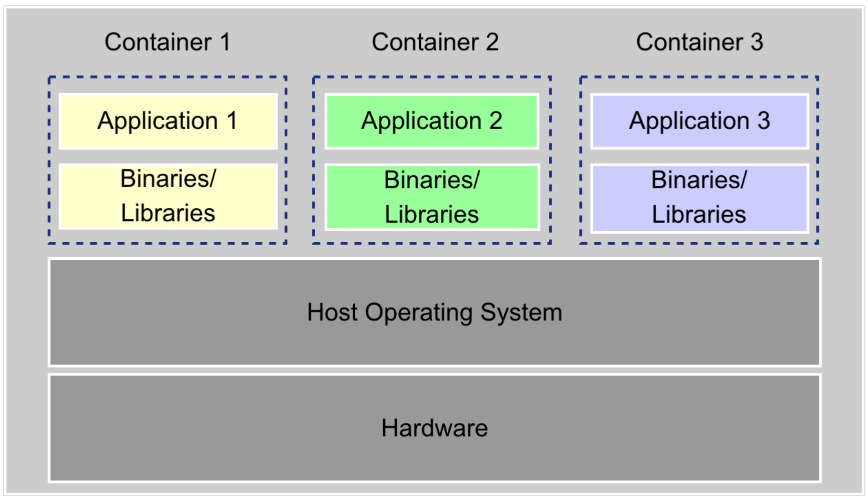


Figure 4.2: System organization for containers

low fast application deployment in different cloud environments while Linux containers cgroups used to control containers resources according to PACE framework decision making.

4.2 System Deployment

This research focuses on both SQL and NoSQL database systems deployed in the cloud. Thus, this section discusses the characteristics and the functionality of each system respectively. Additionally, it provides a discussion around database applications to a variety of domains.

4.2.1 MySQL

MySQL is an open-source relational database management system where a database consists of a structured collection of data. In that context, MySQL server is being used to access and process data stored in the database (MySQL, 2022b). Considering that computers have increased their capability of storing and computing large volumes of data, database management systems have a vital role in managing data and enabling integration with other platforms.

It has to be mentioned that, MySQL databases are relational meaning that data stored in tables of columns and rows that represent their relationships. As a result, MySQL database server is fast and capable to scale up CPU, memory and I/O capacity according to available resources. Furthermore, MySQL sever is open-source meaning that anyone can use and modify the software according to application requirements (MySQL, 2022b). Similar to other relational database management systems, SQL query language used to access and manipulate the data stored in MySQL server. For example data definition statements can be used to create a table in MySQL as follows:

```
CREATE TABLE users(
        user_id INT NOT NULL,
        forename VARCHAR(30),
        surname VARCHAR(30),
    email VARCHAR(50));
```

Furthermore, SQL data manipulation statements enable the user to manipulate the data stored in the database. These statements include the INSERT statement to insert new rows into an existing table, DELETE statement to remove rows from a table and UPDATE statement to modify rows in a table.

MySQL is used in different application domains such as aerospace, education, financial services, government, healthcare, retail, manufacturing and technology among others (MySQL, 2022a). In this work, MySQL server has been deployed as a containerized application hosted on Google Cloud Platform. Then, a large number of concurrent transactions have been executed to stress the running system and create different real-world application scenarios. The aforementioned database system monitored by PACE framework in order to analyze the data and automatically adjust system resources according to workload demand. Furthermore, PACE collects historical data of workload executions in MySQL to explore the optimum configurations that maximize application performance based on user requirements.

4.2.2 MongoDB

MongoDB is a NoSQL database management system designed to store JSON-like documents with dynamic schemas. Thus, a record in MongoDB consists of a document which is composed of field and value pairs. One of the main advantages of MongoDB is that the values may also include embedded documents and arrays that reduce the need for expensive joins (MongoDB, 2022b). Here is an example of a record in MongoDB database:

```
{
    name: "irene",
    gender: "female",
    interests: ["music", "painting"]
}
```

MongoDB uses collections to group documents inside a database analogous to tables in relational database management systems. Furthermore, MongoDB uses mongosh, an interactive Javascript environment that enables the command-line for interacting directly with the database. A variety of commands are available in mongosh including creating a new database and perform CRUD operations to create, read, update, and delete documents. The following example creates a new database and inserts a single document into a collection:

```
use databaseExample
use db.collectionExample.insertOne( { x: 1 } );
```

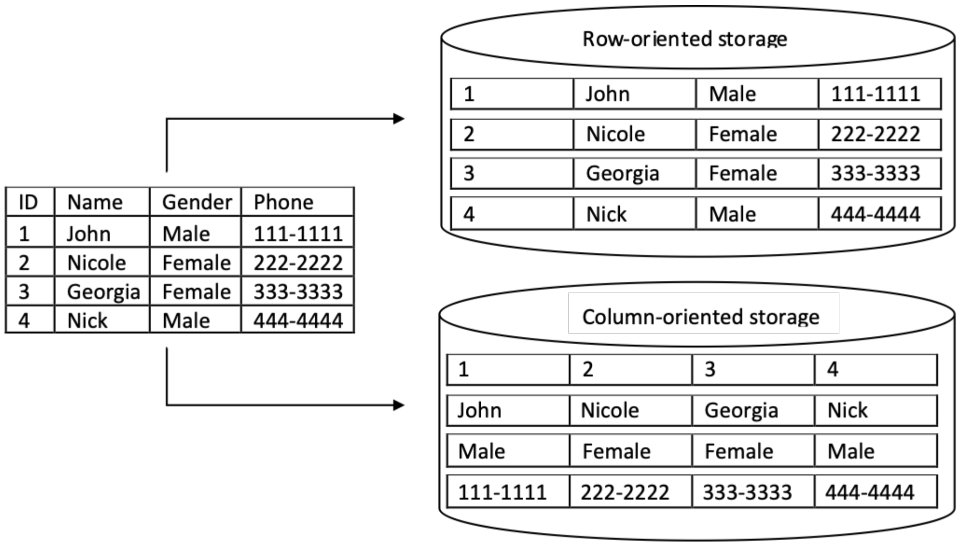
MongoDB enables database replication to distribute replicas of the data across multiple machines. Database replication topology known as a replica set, provides redundancy that enables the system to continue operate in the event of a server outage (Banker et al., 2016). Additionally, replica sets increase the performance of read-intensive application since users can read operations from different machines. Furthermore, MongoDB embedded data model reduces I/O activity while it provides low query execution times when dealing with data-intensive applications that need to support thousands of users simultaneously (Győrödi et al., 2015).

MongoDB has been widely used in different industries such as healthcare, energy utilities, education, government, telecommunications, hospitality and financial services (MongoDB, 2022a). Reported benefits include faster build time, reduced costs, increased performance and accurate data-driven decisions, to name a few. In this work, MongoDB deployed in VMs hosted in Google Cloud Platform in order to emulate different application scenarios. Then, PACE framework monitored the running application and analyzed system metrics to automatically detect abnormal system behavior on-the-fly.

4.2.3 HBase

HBase is an open-source non-relational distributed database capable of storing very large tables consist of billions of rows and millions of columns. HBase modeled after Google's Bigtable (Chang et al., 2008) to provide similar capabilities on top of Hadoop and HDFS (HBase, 2022). HBase stores data in a column-oriented format meaning that tables are being stored by column instead of rows. Ergo, rows are composed of columns that grouped into column families in order to build semantics boundaries between the data (George, 2011). Figure 4.3 displays column-oriented and row-oriented storage layouts respectively.

Furthermore, HBase contains a jruby-based (JIRB) shell to interact with the database (HBase, 2022). General commands include status to provide the status of the database, version to provide the version of HBase and whoami to provide information about the user. Data definition commands include create to create a table, list to list all the tables and drop to delete a table in HBase. Finally, data manipulation commands include put to add data in a table, get to retrieve a row from a table and delete to delete a cell value from a table.



ID | Name | Gender | Phone
1 | John | Male | 111-1111
2 | Nicole | Female | 222-2222
3 | Georgia | Female | 333-3333
4 | Nick | Male | 444-4444
Row-oriented storage
1 | John | Male | 111-1111
2 | Nicole | Female | 222-2222
3 | Georgia | Female | 333-3333
4 | Nick | Male | 444-4444
Column-oriented storage
1 | 2 | 3 | 4
John | Nicole | Georgia | Nick
Male | Female | Female | Male
111-1111 | 222-2222 | 333-3333 | 444-4444

Figure 4.3: Column-oriented and row-oriented storage.

Although, traditional databases offer improved real-time access to data, HBase improves key-based access to a specific cell of data or a sequential range of cells (George, 2011). Similar to other NoSQL systems, HBase provides automatic failover between servers due to its distributed nature. Additionally, HBase follows strict consistency meaning that changes in the data appear to take effect instantaneously (George, 2011). As a result, HBase has near-optimum performance to read operations and excellent performance to write operations respectively. Finally, HBase is highly scalable meaning that it can effectively operate on a cluster of commodity machines to accommodate large volume of data (Vora, 2011).

HBase has been used as a scalable storage solution in various applications including IoT (Pramukantoro, Kartikasari, and Siregar, 2019), e-commerce (L. Li and J. Zhang, 2021), bioinformatics (Taylor, 2010) and geospatial technology (D. Han and Stroulia, 2013). In this work, an HBase cluster deployed on top of Hadoop at Birkbeck college servers. Consequently, PACE framework monitored the running application and analyzed the collected data to automatically detect aberrant throughput patterns.

4.2.4 Redis

Redis is an open-source non-relational in-memory data structure server used as a database, cache, streaming engine and message broker (Redis, 2022). Redis provides a variety of data structures including strings, sets, hashes, lists and streams, a combination that allows Redis to be flexible and fast at the same time (Macedo and Oliveira, 2011). Furthermore, Redis implements replication to support high availability and automatic failover. Redis has been used to solve a variety of problems including reduce access complexity, improve data communication, store complex data as well as implement count activities in computationally efficient ways (Redis, 2022).

Redis command line interface (CLI) consists of a terminal program that used to interact with Redis server. Key operations using Redis CLI include SET to set a key that holds a value, GET to return the value held by the key, INCR, to increment the key value by one and DECR to decrement the key value by one. Here is an example that sets a key namely key that holds the value of one and then increments the value by one:

```
redis> SET key 1
"OK"
redis> INCR key
(integer) 2
```

To get the value of the key, GET operation can be used as follows:

```
redis> GET key
"2"
```

Redis has been used by numerous reputable companies including Twitter, GitHub, Snapchat, Craigslist and StackOverflow (Redis, 2022). In this work, Redis deployed as a containerized application hosted on Google Cloud Platform while state-of-the-art benchmarks have been used to emulate different realistic application scenarios. Then, PACE monitors the running application to collect system and applications metrics. These are used as an input to a hybrid learning system that automatically triggers scaling decisions in order to adjust system resources to workload requirements.

4.3 Benchmarks

This research enables both SQL and NoSQL system deployment in the cloud. In that context, state-of-the-art benchmarks have been used to emulate different application scenarios and evaluate the proposed framework under realistic workloads. This section discusses the main characteristics of the three benchmarks used in this work namely TPC-C, TPCx-IoT and Yahoo! Cloud Serving Benchmark.

4.3.1 TPC-C

TPC Benchmark C (TPC-C) (TPC, 2022a) is an On Line Transaction Processing (OLTP) workload approved by TPC council in 1992. The TPC-C benchmark consists a mixture of concurrent transactions executed by terminal operators in order to simulate different OLTP application environments. In more detail, the TPC-C benchmark emulates activities (transactions) of a wholesale supplier in an order-entry environment. These include entering orders, delivering orders, checking the status of orders, recording payments and monitoring the level of stock at the warehouses (TPC, 2022a). In that context, the

TPC-C designed to scale as additional warehouses are created. It has to be mentioned that the workload provides skewed access within individual data types to allow better usage of the main memory database buffer (Leutenegger and Dias, 1993). The performance metric of TPC-C workload measures the number of transactions that are fully processed during a minute period of time. Figure 4.4 illustrates the execution cycle of TPC-C workload available on TPC website (TPC, 2022a).

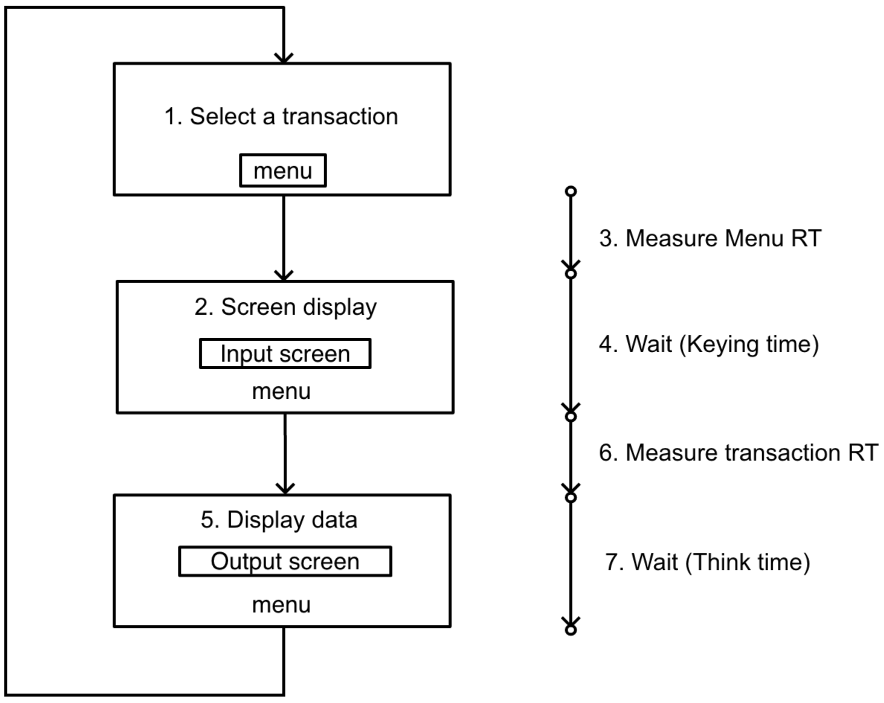


Figure 4.4: TPC-C execution cycle

During the execution cycle the following steps are executed: 1) the user selects a transaction type from the menu, 2) the system waits for the Input/Output screen to be displayed, 3) the response time (RT) of the menu is being measured, 4) the user enters the required number of input fields based on a minimum Keying time, 5) the system waits for the required number of output fields to be displayed, 6) the response time of the transaction is being measured and 7) the system waits for the defined minimum think time while the input/output screen remains displayed. Finally, at the end of the Think time (as shown in Step 7) the user loops back to select a transaction type from the menu (as shown in step 1).

TPC-C benchmark has been the industry standard to measure the performance of OLTP systems (Poess and Nambiar, 2008). Additionally, TPC-C has been used in numerous research studies to evaluate system performance (Avula and Zou, 2020; Kamsky, 2019; Cheong, C. S. Lim, and Cho, 2012). In this work, the TPC-C benchmark executed against MySQL database to emulate a realistic OLTP application environment. Then, PACE framework monitors the running application to automatically trigger scaling decisions that adapt to workload demand. Furthermore, PACE uses historical data of TPC-C executions in MySQL in order to find the optimum configurations that maximize application performance based on workload and user requirements.

4.3.2 TPCx-IoT

The Internet of things (IoT) refers to an interconnected network of smart physical objects that generate large amounts of data in operation (Siow, Tiropanis, and Hall, 2018). IoT topology consists of three tiers namely edge devices, gateway systems and data center. In that context, the TPC Express Benchmark IoT (TPCx-IoT) introduced by TPC to evaluate different software and hardware solutions for IoT gateway systems (TPC, 2022b). The TPCx-IoT workload enables data ingestion and concurrent queries to simulate realistic tasks that typically appear on IoT gateway systems.

In more detail, the workload used to model sensor data generated by power substations. The dataset consists data from 200 different types of sensors. As a result, the workload injects data into the running system while on the the background it executes analytic queries to retrieve readings of a randomly selected sensor. Figure 4.5 illustrates TPCx-IoT benchmark execution phases that are available on TPC website (TPC, 2022b).

As shown in Figure 4.5 the benchmark consists of two runs namely Run 1 and Run 2. Each run executes the workload two times in order to warmup the system and measure system performance respectively. The TPCx-IoT workload provides three metrics to measure system throughput, system price-

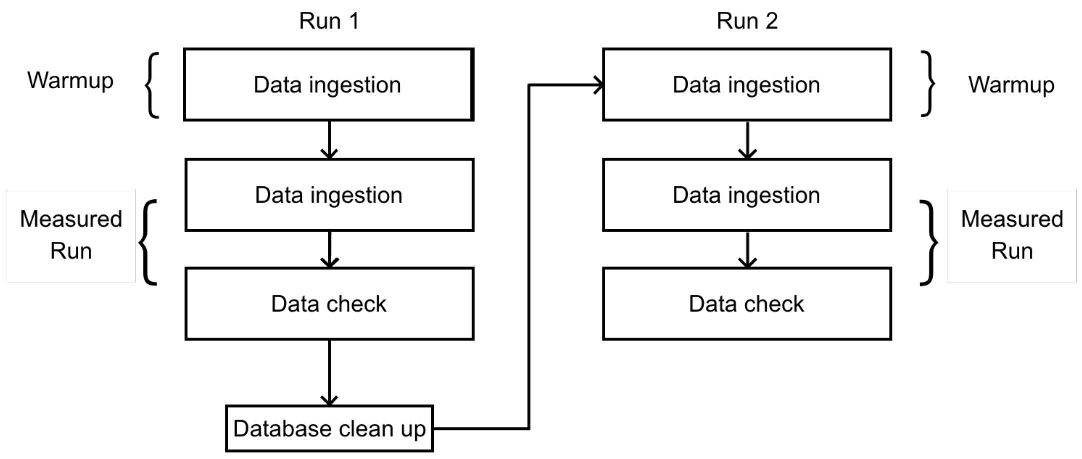


Figure 4.5: TPCx-IoT execution phases

performance and system availability. In this research, TPCx-IoT benchmark executed against HBase to emulate a realistic application environment. Then, PACE framework monitored the running application to automatically detect abnormal throughput patterns.

4.3.3 Yahoo! Cloud Serving Benchmark

Yahoo! Cloud Serving Benchmark (YCSB) (Cooper et al., 2010) proposed as a tool for facilitating performance comparisons of cloud serving systems. In that context, YCSB designed to evaluate both the performance and the scalability of cloud serving systems. The workload defines a dataset to be loaded into the database and then executes four main operations against the dataset to measure system performance. These include insert operation to insert a new record, update operation to update a record, read operation to read a record and scan operation to scan record.

The tool consists of a client namely YCSB Client designed to generate and execute the YCSB benchmark against the running systems. Figure 4.6 illustrates the architecture of the YCSB Client. In more detail, the YCSB client is responsible for generating both the load data and the operations that compose the executable workload. The client takes two main groups of properties namely workload and runtime properties that enable benchmark execution.

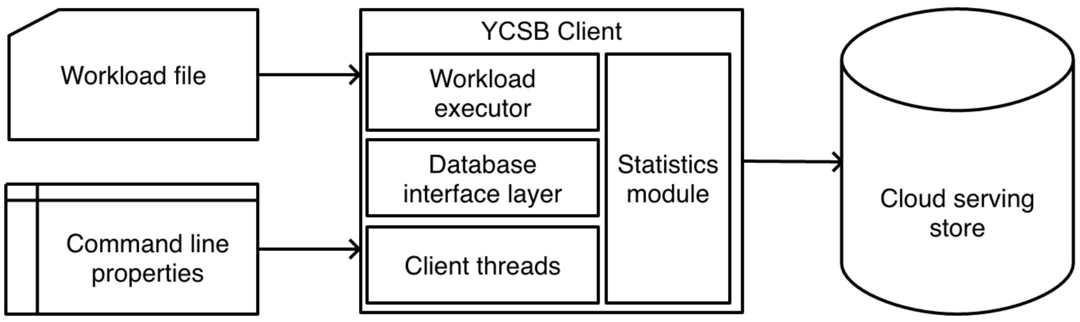


Figure 4.6: YCSB Client architecture

Workload properties define the workload characteristics to generate the load data including the record size, record type and the distribution to use. Runtime properties define the experimental characteristics such as the type of the database and the workload file to be used.

Then, the YCSB Client enables the workload executor, the database interface layer and the client threads to execute the desired workload. The workload executor executes the load and the run phase of the workload while the database interface layer translates requests (e.g., write()) from the client threads into calls against the database (e.g., MongoDB) (Cooper et al., 2010). Finally, the YCSB client statistic module reports the aggregated average latency and throughput metrics. In this work, the YCSB benchmark has been executed against NoSQL database systems deployed in the cloud to support the performance evaluation of the proposed framework.

4.4 Summary

To conclude, this chapter presented the deployment environment and the benchmarks used to evaluate the effectiveness of PACE framework. The design and implementation of the proposed solution is based on cloud-based storage systems that execute state-of-the-art benchmarks to emulate realistic application scenarios. The next chapter discusses the datasets of PACE framework.

5 PACE Datasets

This chapter presents the datasets used to support the proposed methodology. Section 5.1 discusses the dataset creation process that includes system monitoring, data storage and data transformation phases. Section 5.2 presents the dataset used to support the anomaly detection technique of ADM2 service. Section 5.3 introduces the datasets used by the ADA-RP service in order to enable the proposed auto-scaling mechanism. Section 5.4 introduces the datasets used during the constraint optimization process of SONA service. Finally, section 5.5 presents the summary of the chapter.

5.1 Dataset Creation Process

PACE framework enables anomaly-aware resource provisioning and optimization of cloud applications based on data-driven techniques. As a result, PACE uses Prometheus [1] engine to monitor the target system based on a given scrape interval (e.g., 5 seconds). The latter suggests that PACE monitors the running system to collect data records every 5 seconds. These records include CPU usage percentage, memory usage percentage, disk reads, disk writes, database throughput, time spent doing I/Os, disk type, disk size in GB, memory size in GB and number of vCPU cores among others.

[1] `https://prometheus.io/`

After the monitoring phase, PACE stores the monitored application and system metrics in a storage service (e.g., Google Cloud Storage [2]). Finally, the collected data are being transformed in order to create the datasets used by PACE services. Transformations include data aggregation, feature selection and feature preprocessing described in the following sections. Figure 5.1 illustrates the monitoring, collection and transformation phases of PACE framework. PACE uses Prometheus engine to monitor the database application in order to collect a variety of metrics. These are being transformed according to the needs of each service.

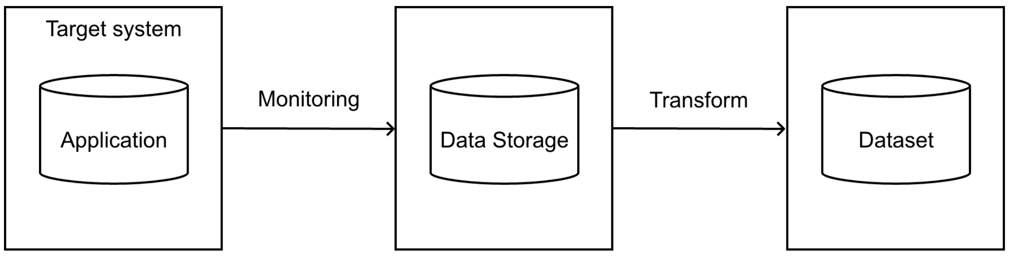


Figure 5.1: Dataset creation process.

The subsequent sections discuss the datasets used by ADM2, ADA-RP and SONA services respectively.

5.2 ADM2 Dataset

ADM2 service enables real-time anomaly detection of cloud application based on throughput performance metric. In more detail, ADM2 uses Prometheus engine to monitor and collect resource usage and application metrics based on a given time interval. Then, it analyzes the data to automatically

[2]`https://cloud.google.com/storage`

detect performance degradation of the running system. ADM2 uses historical throughput time series data in order to learn the behavior of the running system. During the training process, the LSTM Autoencoder learns to reconstruct normal historical time series. Ergo, LSTM Autoencoder records lower reconstruction error on normal YCSB workload executions, whereas it expected to produce high reconstruction error on future abnormal events. As a result, LSTM Autoencoder classifies future events as normal or abnormal based on the reconstruction error produced by the trained model and an error threshold parameter.

To support the proposed methodology, two experimental datasets emerged i) YCSB benchmark executed in MongoDB and ii) TPCx-IoT benchmark executed in HBase. For each scenario, a univariate time series dataset has been created with a single column of database throughput (operations per second) and a timestep for each value as an implicit variable. It has to be mentioned, that the historical time series consists of normal workload executions in order to create repeatable workload patterns to train LSTM Autoencoder.

In more detail, YCSB benchmark executed in MongoDB while PACE monitored the running system to collect data to create ADM2-YCSB dataset. The first 10 runs include 5 load and 5 run stages that have been used for the training phase of LSTM Autoencoder. Then, the 11th and 12th runs used to test the proposed algorithm that successfully detects abnormal throughput patterns. As a result, ADM2-YCSB dataset consists a single column of MongoDB throughput and a timestep for each value. The dataset has 2663 number of records, one for each second, stored in a Comma Separated Values (CSV) file format.

Additionally, TPCx-IoT executed in HBase while PACE framework monitored the running system to create ADM2-TPCx-IoT dataset. The latter consists of 12 TPCx-IoT runs in HBase cluster. The first 8 runs compose the training set while the last 4 runs compose the test set. In total, ADM2-TPCx-IoT dataset has 2758 number of records where each record row has a timestamp value. Similar with ADM2-YCSB dataset, the ADM2-TPCx-IoT dataset is stored in a CSV file format.

5.3 ADA-RP Dataset

ADA-RP service supports both reactive and proactive vertical auto-scaling of cloud applications. The reactive approach uses a threshold-based rule to automatically adjust system resources on-the-fly without processing historical data events. On the contrary, the proactive approach is based on a hybrid machine learning-based technique that combines CNN and K-means algorithms. These are being used to forecast and cluster future events so as to create a scaling plan that avoids over-utilized and under-utilized resources.

In that context, the CNN model is trained on resource usage metrics extracted from the running application, whereas K-means algorithm is trained on previous workload executions that include different resource usage and application configurations. The next two subsections present the datasets used throughout this study to support the proposed auto-scaling technique. These include a) data collection for SQL systems and b) data collection for NoSQL systems

5.3.1 SQL data collection

As mentioned earlier, ADA-RP service is based on a data-driven approach to automatically scale cloud resources and ensure application performance. To support the proposed method, this study uses TPC-C benchmark as a real world workload and Prometheus monitoring engine for data collection. As a result, two datasets have been generated: i) a dataset for workload characterization using the clustering method and ii) a dataset for workload prediction.

For the first dataset, the TPC-C workload executed in a Linux MySQL container deployed on GCP with different configurations. In more detail, the Cartesian product of two sets called $\mathbb{C}$ and $\mathbb{W}$ is defined as the set of all ordered pairs (c, w) where c is an element of $\mathbb{C}$ and w is an element of $\mathbb{W}$. The notation is given as follows:

$$\mathbb{C} \times \mathbb{W} = \{(c, w) \mid c \in \mathbb{C} \text{ and } w \in \mathbb{W}\} \tag{5.1}$$

Set $\mathbb{C} = \{2,4,6,8,10,12,14,16\}$ consists a 8-elements set where each element defines the vCPU cores allocated to a Linux container. Set $\mathbb{W} = \{1,5,10,15,20\}$ consists a 5-elements set where each element defines the number of Warehouses used in the TPC-C configurations. Table 5.1 shows the dataset configuration based on the Cartesian product of sets $\mathbb{C}$ and $\mathbb{W}$.

Table 5.1: Dataset configuration based on the Cartesian product of sets $\mathbb{C}$ and $\mathbb{W}$.

	vCPU cores							
W.	2	4	6	8	10	12	14	16
1	(1,2)	(1,4)	(1,6)	(1,8)	(1,10)	(1,12)	(1,14)	(1,16)
5	(5,2)	(5,4)	(5,6)	(5,8)	(5,10)	(5,12)	(5,14)	(5,16)
10	(10,2)	(10,4)	(10,6)	(10,8)	(10,10)	(10,12)	(10,14)	(10,16)
15	(15,2)	(15,4)	(15,6)	(15,8)	(15,10)	(15,12)	(15,14)	(15,16)
20	(20,2)	(20,4)	(20,6)	(20,8)	(20,10)	(20,12)	(20,14)	(20,16)

In total, 40 configurations have been used to generate unique CPU patterns while Prometheus monitoring engine collected the CPU utilization metrics. It has to be mentioned, that each run consists an average of 10 runs for ensuring reproducible data. Thus 40×10 experiments executed, for 300 seconds each, whereas the results averaged to obtain 40 runs, one for each configuration. In this work, ADA-RP makes scaling decisions for MySQL application every minute, thus each run of 300 seconds is being segmented into five 60 seconds segments. As a result, a univariate time series dataset of 200 sequences formed and used as an input to train the K-means clustering algorithm. The latter, categorizes each unique sequence into High, Medium or Low demand clusters based on the CPU utilization levels.

To support the workload prediction task, MySQL Linux containers deployed and monitored in GCP. Then, TPC-C workload executed consecutively to demonstrate real-world experimental scenarios. As a result, a historical univariate time series dataset has been formed for each scenario with a single column of CPU utilization and a timestep for each value as an implicit variable. The historical sequence has been pre-processed and used as an input for the CNN model to predict future events.

5.3.2 NoSQL data collection

ADA-RP service achieves on-demand scalability based on data collected directly from the running applications. To support the proposed proactive approach, two datasets have been generated namely: i) a dataset for time series clustering and ii) a dataset for time series forecasting. In time series clustering phase, various YCSB workloads executed in a Linux Redis container deployed on GCP under different system configurations. In more detail, the Cartesian product of two sets called $\mathbb{Y}$ and $\mathbb{V}$ is defined as the set all ordered pairs (y, v) where y is an element of $\mathbb{Y}$ and v is an element of $\mathbb{V}$. Thus, we give the following notation:

$$\mathbb{Y} \times \mathbb{V} = \{(y, v) \mid y \in \mathbb{Y} \text{ and } v \in \mathbb{V}\} \tag{5.2}$$

Set $\mathbb{Y} = \{1, 2, 4, 6, 8\}$ consists a 5-elements set where each element defines the vCPU cores allocated to a Linux container. Set $\mathbb{V} = \{A, B, C, D\}$ consists a 4-elements set where each element defines the type of YCSB workload. For instance, (A,1) pair includes YCSB workload A execution in Redis container with 1 vCPU core. Table 5.2 shows the dataset configuration based on the Cartesian product of sets $\mathbb{Y}$ and $\mathbb{V}$.

Table 5.2: Dataset configuration based on the Cartesian product of sets $\mathbb{Y}$ and $\mathbb{V}$.

	vCPU cores				
Workload	1	2	4	6	8
A—Update heavy	$(A, 1)$	$(A, 2)$	$(A, 4)$	$(A, 6)$	$(A, 8)$
B—Read heavy	$(B, 1)$	$(B, 2)$	$(B, 4)$	$(B, 6)$	$(B, 8)$
C—Read only	$(C, 1)$	$(C, 2)$	$(C, 4)$	$(C, 6)$	$(C, 8)$
D—Read latest	$(D, 1)$	$(D, 2)$	$(D, 4)$	$(D, 6)$	$(D, 8)$

In total, 20 configurations have been used to generate different CPU patterns while Prometheus monitoring engine collected CPU utilization metrics.

It has to be mentioned that each run consists an average of 10 runs for ensuring reproducibility. Thus, 20×10 experiments executed for 300 seconds each and the results averaged to obtain 20 runs, one for each configuration. In this work, ADA-RP service makes scaling decisions for Redis application every 15 seconds, thus each run of 300 seconds is being segmented into 20 segments of 15 seconds each. As a result, a univariate time series dataset of 400 sequences is being formed. The latter, used as an input to train K-means algorithm to partition each sequence into High, Medium and Low demand clusters. Then, the average sequence of each cluster is calculated and used as a group representative.

To support the time series forecasting phase, Redis containers deployed and monitored in GCP. Consequently, the YCSB workload executed in Redis to introduce a realistic application scenario. As a result, a univariate time series dataset has been formed with a single column of CPU utilization and a timestep for each value as an implicit variable. The dataset has been preprocessed and used as an input for the CNN model to predict future events. Then, each event will be labeled as High, Medium or Low demand based on the representative sequences extracted from K-means clusters.

5.4 SONA Dataset

SONA service supports cloud application optimization and tuning based on a neuro-genetic data-driven approach. To support the proposed methodology, a novel dataset presented consisting of (a) 350 observations of experimental TPC-C runs in MySQL server and (b) 16 features that represent various components. These are the system resources (Cores, Memory, Disk), type of resources (Disk type), the application parameters (InnoDB buffer pool size), the workload configurations (warehouses and connections), the workload performance metrics (TpmC), the idle CPU resources, the monthly cloud costs , the cost performance ratio, the system type and the cloud information (platform, region and zone). The runs represent a total of around 30 hours of TPC-C workload executions in Google Cloud Platform (GCP) where each observation is a unique execution of the TPC-C benchmark. The dataset features are as follows:

- **Cores**: The number of CPU cores of the running system.
- **Memory**: The memory size of the running system in GB.
- **Disk**: The disk size of the running system in GB.
- **Disk.T**: The disk type of the running system (magnetic disk coded as 0 and solid-state drive coded as 1).
- **IDB**: The InnoDB buffer pool size of MySQL server measured in MB.
- **Warehouse**: The number of warehouses of the TPC-C workload.
- **Con**: The number of connections of the TPC-C workload.
- **TpmC**: The number of transactions per minute, provided by the TPC-C workload.
- **IdleL**: The average percentage of idle CPU resources during TPC-C load phase.
- **IdleR**: The average percentage of idle CPU resources during TPC-C run phase.
- **Cost**: The monthly cost of the system in USD ($).
- **C-P**: The cost-performance ratio ($/TpmC).
- **System**: The type of the running system in terms of CPU cores and memory size.
- **Platform**: The cloud computing platform.
- **Region**: The cloud geographical region.
- **Zone**: The cloud geographical zone.

Table 5.3 presents a sample of SONA dataset. For example, the first observation consists of 1 CPU Core, 3.75 GB of Memory, 10 GB of magnetic disk (type 0) and 128MB of MySQL InnoDB buffer pool size. The TPC-C executed in 1 warehouse, 10 connections and executes 5,334 transactions per minute. The average idle CPU resources in the load and the run stage are 7.7% and 2.2% respectively. The monthly cost of the system is calculated to 24.67$ according to the Google Cloud Pricing calculator (Google, 2022e). The cost-performance ratio ($/TpmC) is 0.0046 and System A operates in GCP in Iowa Region, us-central1-a zone. It has to be mentioned that the proposed dataset is being preprocessed to support the proposed constrained optimization technique. The preprocessing phase includes feature selection based on a variance threshold. As a result, a threshold has been used to discard features with low variance such as the number of connections, the System, the Platform, the Region and the Zone. The dataset is available online at IEEE Dataport (Chouliaras and Sotiriadis, 2021).

Table 5.3: SONA Dataset.

Cores	Mem.	Disk	Disk.T	IDB	Wareh.	Con.	TpmC	IdleL	IdleR	Cost	C-P	System	Platform	Region	Zone
1	3.75	10	0	128	1	10	5,334	7.7	2.2	24.67	0.0046	A	GCP	Iowa	us-c1-a
2	7.5	500	1	1024	5	10	8,645	42.9	2.7	133.54	0.0154	B	GCP	Iowa	us-c1-a
16	60	500	0	256	1	10	22,821	92.6	63.9	408.36	0.0179	F	GCP	Iowa	us-c1-a
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
4	15	10	0	128	1	10	14,100	71.1	14.8	97.49	0.0069	C	GCP	Iowa	us-c1-a

5.5 Summary

This chapter presented the datasets used to support the research of this thesis. It presented ADM2 dataset used for detecting anomaly detection based on YCSB and TPCx-IoT historical workload executions in NoSQL databases. Furthermore, this chapter introduced the datasets to support the ADA-RP service for clustering and forecasting time series events. The proposed auto-scaling mechanism is based on historical time series data to enable both reactive and proactive auto-scaling of cloud applications.

This chapter also presented SONA dataset that includes information about historical TPC-C runs on MySQL server. The dataset used as an input for the SONA service to explore the underlying relationship between configurations and performance metrics using ANN. As a result, SONA uses a neuro-genetic optimization method to select the optimal configurations that maximize application performance in terms of executable transactions. The datasets are available online at `https://github.com/SpyrosChouliaras/PACE-Datasets.git`.

6 Performance Evaluation

This chapter presents the performance evaluation of PACE models, algorithms and methods. Section 6.1 describes the experimental design of PACE framework and presents the experimental map of PACE services. Section 6.2, 6.3 and 6.4 present the experimental evaluation of PACE services, namely ADM2, ADA-RP and SONA respectively. Finally, the summary of the chapter presented in section 6.5.

6.1 Experimental Design

This section discusses the experimental design of PACE framework to demonstrate the effectiveness of its services, namely ADM2, ADA-RP and SONA. These are as follows:

(a) The ADM2 experimental evaluation demonstrates the ability of PACE framework to detect aberrant patterns based on application and resource usage monitoring. Real-world database systems have been deployed in the cloud including MongoDB and HBase. Additionally, state-of-the-art benchmarks have been used to emulate realistic application scenarios such as TPCx-IoT and YCSB respectively. Each system has been deployed in the cloud with different resource and architecture specifications. In particular, MongoDB has been deployed in GCP in a standalone host while on the contrary, HBase has been deployed in a cluster mode. Both systems have been monitored and analyzed to detect abnormal throughput patterns that lead to application performance degradation. The experimental design includes different abnormal scenarios including network delays and CPU intensive tasks.

(b) The ADA-RP benchmark and performance analysis show the effectiveness of PACE framework to automatically adjust cloud resources based on monitored data. The experimental evaluation shows that ADA-RP analyzes memory and CPU usage percentage metrics in order to trigger scaling actions. The ADA-RP service uses both reactive and proactive auto-scaling techniques, thus different experimental scenarios are presented to evaluate the proposed methods. Both SQL and NoSQL databases have been deployed in the cloud including MySQL and Redis respectively. To that extend TPC-C and YCSB workloads have been used to stress the deployed systems. It has to be mentioned that ADM2 service has been evaluated to both single-tenant and multi-tenant cloud environments.

(c) The SONA service evaluation and experimental analysis demonstrate the validity of the proposed methodology to optimize application performance based on user requirements. The experimental design consists of three scenarios for resource optimization, database application tuning and resource and application optimization. Resource optimization enables the user to automatically adjust system resources based on application and user requirements. Additionally, application tuning optimizes database configurations to maximize application performance including the executed transactions per minute. The third experiment presents a combination of resource and application optimization. To support the proposed experimental design, MySQL and TPC-C benchmark deployed in GCP as a real-world cloud application to be monitored, tuned and optimized.

Figure 6.1 demonstrates the experimental map of ADM2, ADA-RP and

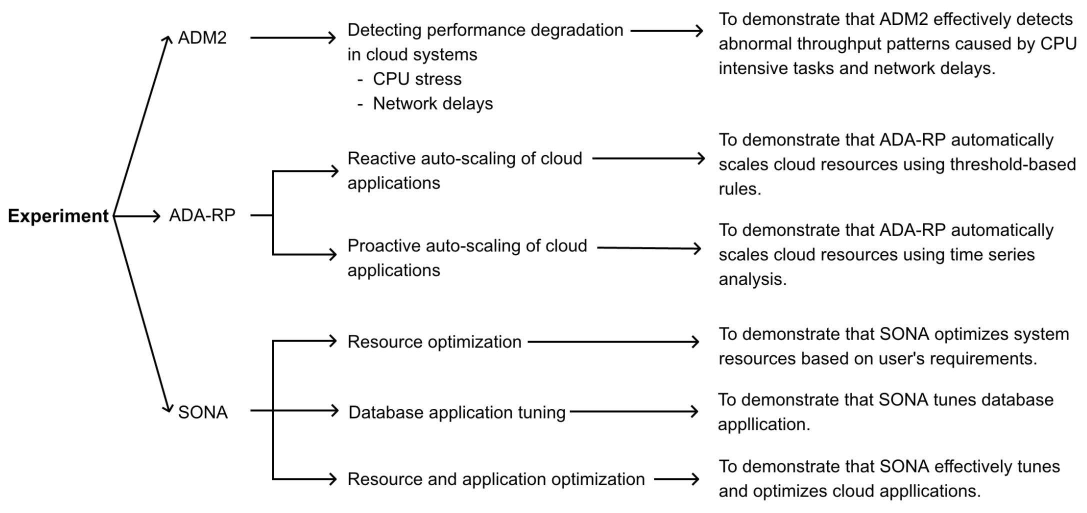

Experiment
ADM2
Detecting performance degradation in cloud systems
- CPU stress
- Network delays
To demonstrate that ADM2 effectively detects abnormal throughput patterns caused by CPU intensive tasks and network delays.
ADA-RP
Reactive auto-scaling of cloud applications
To demonstrate that ADA-RP automatically scales cloud resources using threshold-based rules.
Proactive auto-scaling of cloud applications
To demonstrate that ADA-RP automatically scales cloud resources using time series analysis.
SONA
Resource optimization
To demonstrate that SONA optimizes system resources based on user's requirements.
Database application tuning
To demonstrate that SONA tunes database appllication.
Resource and application optimization
To demonstrate that SONA effectively tunes and optimizes cloud appllications.

Figure 6.1: The experimental map of PACE services.

SONA services respectively. Different experimental scenarios conducted to demonstrate the effectiveness of ADM2 service. In particular, a) CPU stress intensive tasks and b) network delays have been injected to the running system that lead to abnormal throughput patterns. The experimental results show that ADM2 automatically detects system abnormal behavior and alerts the system administrator.

ADA-RP service enables both reactive and proactive auto-scaling techniques. This study presents experiments to demonstrate that ADA-RP automatically scales cloud resources using a) threshold-based rules and b) time series analysis. SONA service uses a neuro-genetic technique to support constraint resource optimization and application tuning. As shown in Figure 6.1, the experimental design consists of three main scenarios to demonstrate the effectiveness of SONA to a) optimize resources, b) tune database application and c) enable both resource and application optimization.

The following sections present experimental results and discussions for the aforementioned scenarios.

6.2 ADM2 Experimental Evaluation

In this section, experimental scenarios are being discussed and visualized to demonstrate the effectiveness of ADM2 service. The performance of the proposed method is being verified based on YCSB and TPCx-IoT real world workload benchmarks that execute in MongoDB and HBase system respectively. It includes the experimental setup and the benchmark analysis of two scenarios: (a) LSTM Autoencoder for anomaly detection of MongoDB system and (b) LSTM Autoencoder for anomaly detection of HBase system.

The experimental setup and the benchmark analysis of ADM2 service consists of two infrastructure settings to support MongoDB and HBase cloud deployment. The first environment consists of a single Virtual Machine (VM) that is running a Linux operating system with 2 CPU cores, 8 GB RAM and 512 GB Hard Disk Drive (HDD). The VM hosts MongoDB as a real-world NoSQL application system to be monitored by Prometheus. While YCSB workload executes a mix of 50/50 reads and writes, Prometheus collects and stores applications metrics with equally spaced points in time. The second environment includes a cluster deployment with 2 nodes each configured with 2 CPU cores, 8 GB RAM and 256 GB HDD. Each node consists of an HBase server that runs on top of Hadoop Distributed File System (HDFS). The TPCx-IoT workload used as representative of activities typical in IoT gateway systems. Similarly, Prometheus collects and stores application metrics in real-time. It has to be mentioned that both infrastructures have been deployed on GCP public cloud vendor.

6.2.1 Detecting performance degradation of MongoDB

This subsection evaluates the ability of ADM2 service to detect abnormal throughput patterns in MongoDB system based on application throughput. Figure 6.2 illustrates application throughput under YCSB workload execution. In total, 5 load and 5 run stages have been executed alternately to create 10 representative runs. The first run, executed between 12:37 and 12:42, illus-

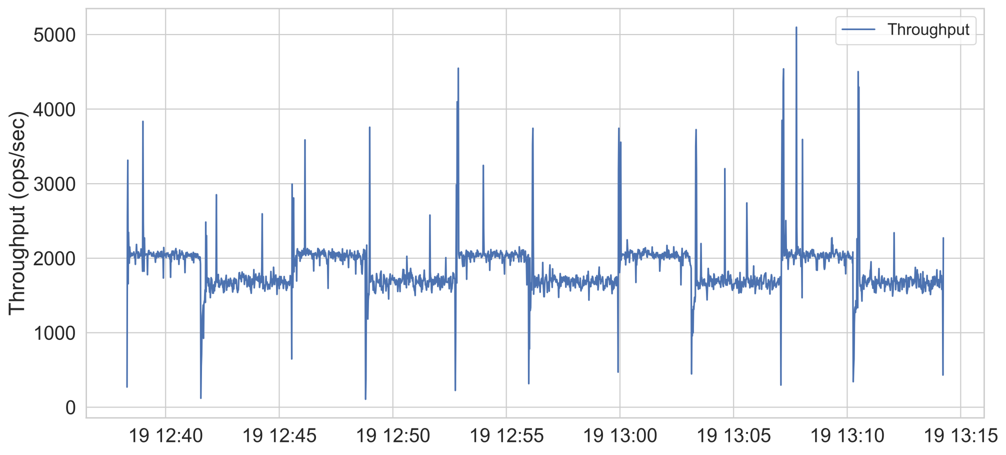


Figure 6.2: Throughput of MongoDB on YCSB workload.

trates the load phase of YCSB workload where 200,000 records loaded in the database. The second run, executed between 12:42 and 12:46, shows the run stage where YCSB uses a mix of 50% read and 50% write operations. Figure 6.2 shows that executing YCSB workload with the same configurations generate a repeatable application throughput pattern. The throughput sequence consists of 2157 data points that have been normalized by removing the mean and scaling to the standard deviation. Then, the sequence is reconstructed into 2152 subsequences based on a 5 time-step sliding time-window parameter $s_w = 5$ that defines the length of each subsequence. Each subsequence consists a training sample for the LSTM Autoencoder that learns to reconstruct normal throughput representations by minimizing the MAE.

During the training process the LSTM Autoencoder learns to reconstruct the training sequence samples while the reconstruction error has been used as a score to detect future throughput abnormalities. Figure 6.3 illustrates the histogram of the training MAE with the bin size set to 50. As shown in Figure 6.3, the x axis display MAE values of the training set while y axis show the frequency that is the number of cases in each bin. In this experiment, the error threshold set to 1.19 ($Et = 1.19$) meaning that future sequences that produce a reconstruction error higher than Et, will be classified as abnormal. The threshold parameter is calculated by multiplying the median value of the train MAE loss by r, that is a parameter that adjust the error threshold value

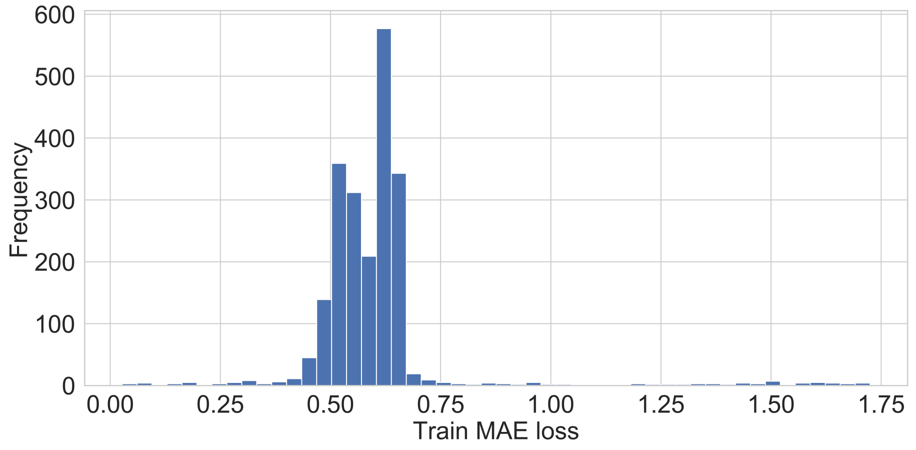


Figure 6.3: The distribution of train MAE loss on YCSB normal runs

based on application requirements. In this experiment the *r* value has been set to 2.

To demonstrate the effectiveness of the proposed method, Stress Linux package is being used to introduce additional intensive tasks that negatively impact application performance. Figure 6.4 shows a continuity of Figure 6.2 with two additional wavelets that represent the load and the run stage of YCSB workload. While YCSB executed the load stage under normal conditions (11th wavelet), in the run stage (12th wavelet) Stress package is being executed in parallel to introduce additional CPU intensive tasks to the running system.

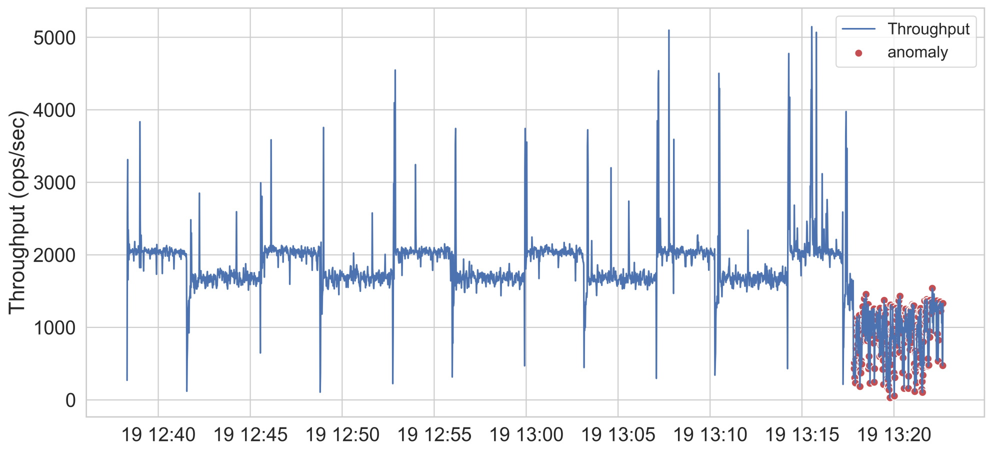


Figure 6.4: Throughput of MongoDB on YCSB workload with Stress.

The latter, impacts the application throughput as it creates abnormal patterns shown in Figure 6.4. In more detail, the application throughput decreased significantly between 13:17 and 13:23 where CPU intensive tasks execute in parallel with YCSB. Figure 6.4 shows that ADM2 effectively detects aberrant behavior illustrated with red points.

As mentioned earlier, ADM2 service is capable of detecting anomalies based on the reconstruction error produced by the LSTM Autoencoder algorithm and a predefined error threshold parameter. Since the LSTM Autoencoder trained to reconstruct normal throughput patterns, it produces high reconstruction error on throughput values that deviate from the normal data distribution. As a result, ADM2 classifies throughput observations as normal or abnormal based on the threshold parameter Et. Figure 6.5 illustrates the histogram of the test MAE and the error threshold value. As shown in Figure 6.5, if the reconstructed error of a given observation surpasses the application error threshold, then the observation is tagged as abnormal. It has to be mentioned that the error threshold parameter consists a tuning parameter based on application functionality. If the application is resilient in throughput fluctuations, a high threshold value is suggested to detect only extreme outliers. On the other hand, if the application is sensitive on throughput changes, a lower threshold will be less forgiving and more values will be classified as abnormal.

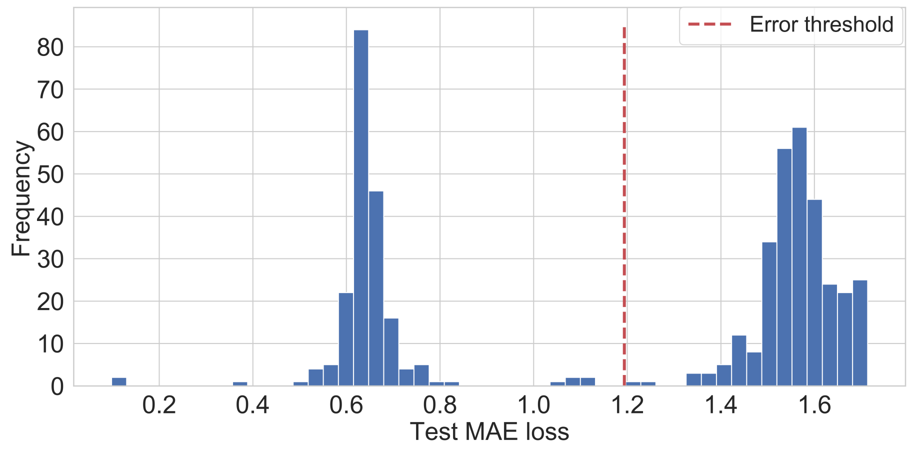


Figure 6.5: The distribution of test MAE loss on YCSB abnormal runs

6.2.2 Detecting performance degradation of HBase

This subsection demonstrates the ability of the ADM2 service to detect performance degradation of NoSQL systems. To emulate a real-world scenario, an HBase cluster with 2 nodes deployed in GCP while TPCx-IoT benchmark used to model sensor data generated by power substations. Figure 6.6 shows the throughput of the HBase cluster under TPCx-IoT workload execution.

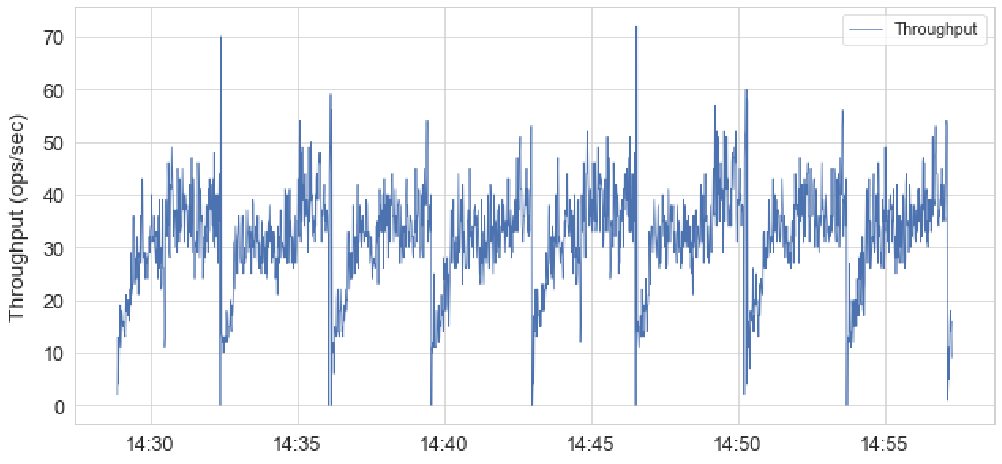


Figure 6.6: Throughput of HBase on TPCx-IoT workload.

In total the TPCx-IoT workload generated 8 wavelets that used as a representative normal input sequence for the LSTM Autoencoder. The input sequence consists of 1697 data points that have been normalized. Then, the sequence is reconstructed into 1692 subsequences based on a 5 time-step sliding time-window parameter $s_w = 5$. Ergo, the LSTM Autoencoder learns to reconstruct the input sequence with the lowest possible error rate. Figure 6.7 shows the histogram of the train MAE loss with the bin size set to 50. As shown in Figure 6.7, the x axis display the MAE loss values after the training process of normal TPCx-IoT runs while the y axis display the frequency. In this experiment the error threshold set to 1.04 ($Et = 1.04$) with the r parameter set to 1.5 ($r = 1.5$) respectively.

Figure 6.8 is a continuity of Figure 6.6 that shows the execution of the TPCx-IoT workload while at the same time network delays introduced in the system. In particular, two network delays of 200ms have been injected into the

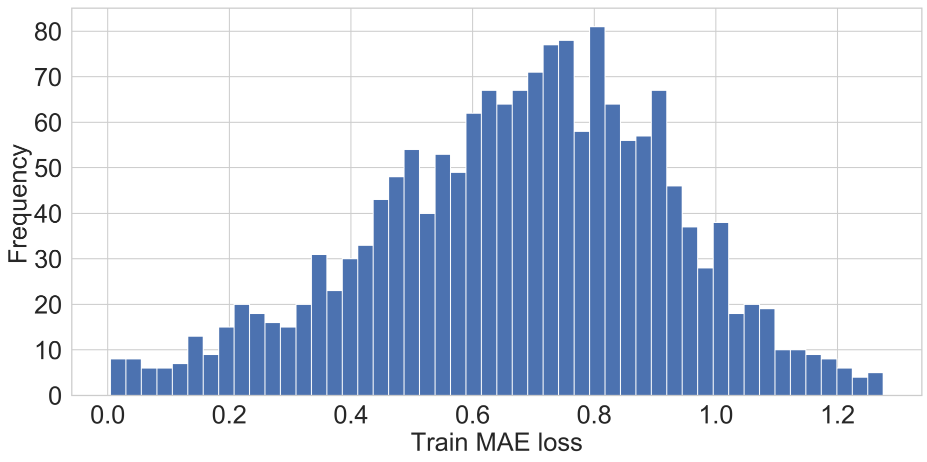


Figure 6.7: The distribution of train MAE loss on TPCx-IoT normal runs

network system. The first delay injected between 14:58 and 15:00 while the second delay injected between 15:07 and 15:09. As a result, the application throughput plummets to zero during that period of time. Figure 6.8 illustrates the ability of ADM2 service to effectively detect throughput abnormal values marked with red points.

As discussed earlier, ADM2 classifies throughput observations as normal or abnormal based on LSTM Autoencoder reconstruction error and a predefined error threshold parameter Et. Thus, observations with a reconstruction

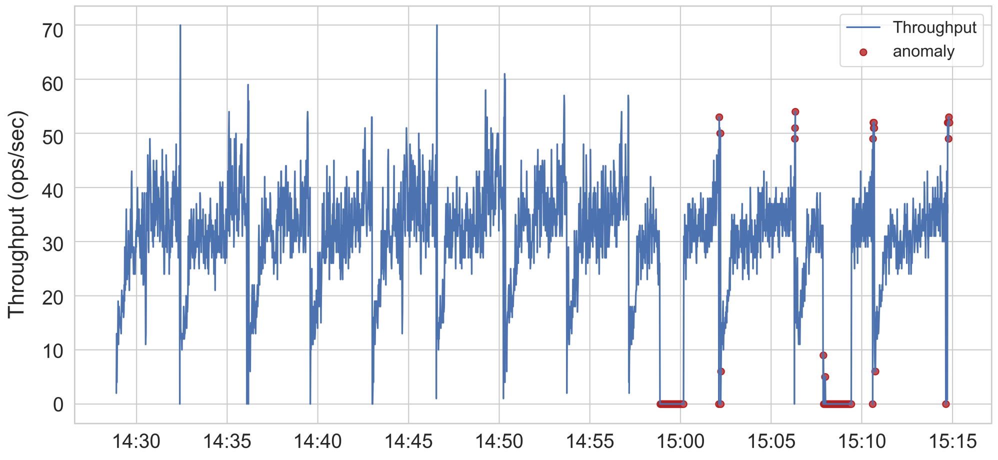


Figure 6.8: Throughput of HBase on TPCx-IoT workload with network delay.

error higher than the error threshold are tagged as abnormal. On the other hand, observations with a reconstruction error lower than the error threshold parameter are classified as normal. It has to be mentioned that, the error threshold parameter has been tuned according to application functionality in order to detect values that significantly vary from the normal distribution. Figure 6.9 presents the distribution of the test MAE loss produced by the LSTM Autoencoder. As shown in Figure 6.9 the reconstruction error threshold set to 1.02 in order to classify future throughput observations as normal or abnormal based on their test MAE loss value.

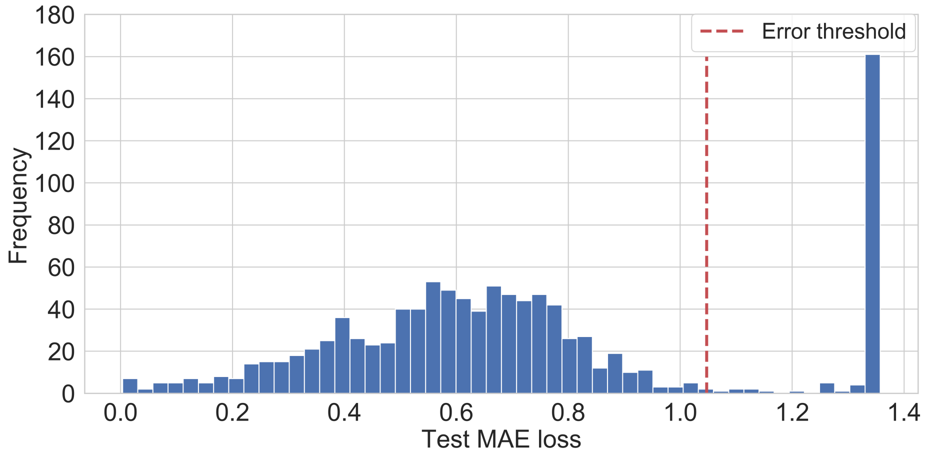


Figure 6.9: The distribution of test MAE loss on TPCx-IoT abnormal runs

This section presented the experimental setup and evaluation of the ADM2 service. It showed that ADM2 effectively detects throughput abnormalities using LSTM Autoencoder algorithm. In particular, ADM2 uses a predefined error threshold parameter and LSTM Autoencoder reconstruction error to enable automatic anomaly detection in cloud applications. The experimental design included different infrastructure settings and database systems. Furthermore, various types of anomalies presented including CPU intensive workload tasks and network delays to emulate realistic scenarios. The next section includes the experimental evaluation of ADA-RP service, that enables adaptive vertical elasticity of containerized applications.

6.3 ADA-RP Experimental Evaluation

This section presents the performance evaluation of ADA-RP service. ADA-RP enables a combination of reactive and proactive auto-scaling techniques to automatically allocate resources based on application and user requirements. It uses a threshold-based rule technique to trigger scaling decisions based on monitored resource usage metrics such as memory usage. Additionally, it enables proactive auto-scaling using a hybrid machine learning technique that includes time series forecasting and clustering. The performance evaluation of the proposed method is based on real-world database systems including SQL and NoSQL as well as state-of-the-art workloads and benchmarks such as YCSB and TPC-C. It includes the experimental evaluation and benchmark analysis of a) reactive auto-scaling and b) proactive auto-scaling.

6.3.1 Reactive auto-scaling mechanism evaluation

ADA-RP service uses a reactive auto-scaling technique to automatically adjust virtual resources that ensure QoS requirements. The proposed technique is based on application monitored data and a threshold based rule. Firstly, ADA-RP monitors the running system to collect various resource usage metrics including CPU and memory usage percentage as well as application metrics such as throughput. Then, it uses threshold-based scaling rules to increase or decrease the allocated resources on-the-fly based on metric value. In particular, ADA-RP increases the amount of allocated resources when a metric (e.g., memory usage percentage) exceeds a predefined upper threshold parameter. Similarly, ADA-RP decreases the allocates resources if the selected metrics is lower than a lower threshold parameter. The following subsections include the experimental setup and benchmark analysis of ADA-RP reactive mechanism.

Experimental setup and benchmark analysis

To evaluate and demonstrate ADA-RP effectiveness, a medium sized Virtual Machine has been deployed in GCP with 16 vCPU cores, 64 GB memory and 350GB HDD. Then, Redis NoSQL database has been deployed in a docker container. At first, 2 vCPU cores (0 and 1) and 32 GB memory are being allocated to the container. Additionally, YCSB Redis workload-a is being used as an application example for a session store recording recent actions in a user session (Cooper et al., 2010). YCSB record and operation counts set to 5,000,000 with 50% read and 50% update operations respectively. In this experiment, the threshold-based rule set to 80% and the scaling factor to 2, meaning that each time the memory usage percentage exceeds 80%, then ADA-RP service automatically allocates two additional GB of memory to the container.

Reactive auto-scaling of Redis containers

ADA-RP enables vertical elasticity to automatically scale Redis containerized applications. As a result, it scales up or down Redis containers in order to ensure high application performance under intensive workload tasks. As mentioned earlier a YCSB Redis workload-a is being used to inject 5,000,000 records with 50% read and 50% update operations. The latter, stress the running system that needs to load YCSB records without system failures. Consequently, ADA-RP monitors the running system to collect various resource usage metrics. In this experiment, the memory usage percentage is being used to trigger scaling decisions based on a threshold-based rule. Memory usage metric is vital to ensure system sustainability and to avoid system failures when the application reaches a saturation level.

Figure 6.10 illustrates the memory usage percentage on the left axis and the allocated memory (in GB) on the rights axis while YCSB workload-a executes in Redis. As shown in Figure 6.10, at the beginning of the loading phase, memory usage starts to increase until it reaches 80% threshold at 151 second. In that moment, ADA-RP automatically allocates two additional GB of memory to Redis container. As a result, the memory usage drops significantly to 40% while Redis has access to 4 GB of memory. Figure 6.10 shows a continuation of the same process where each time memory usage exceeds 80% threshold, ADA-RP allocates two additional GB of memory to the container. In total,

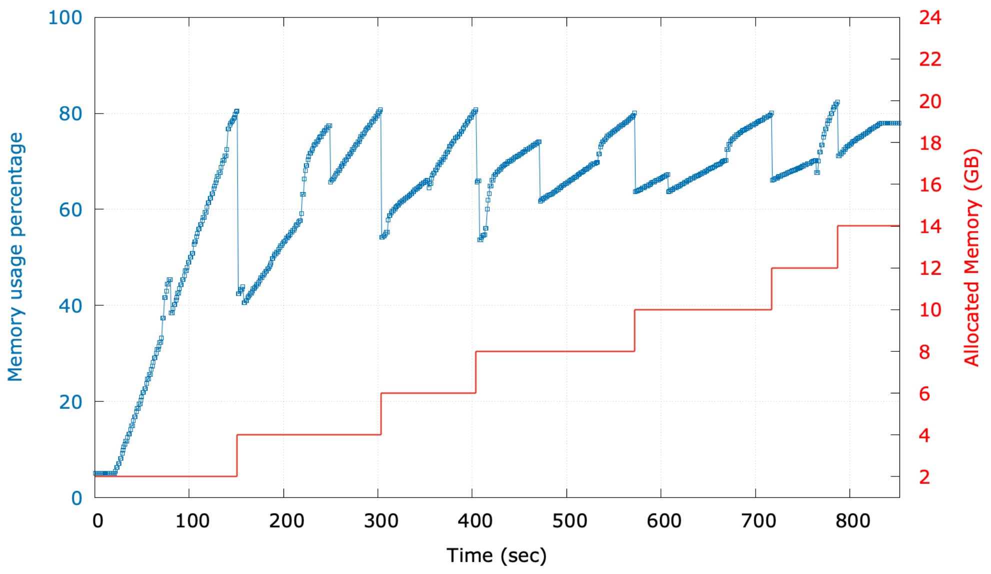


Figure 6.10: Memory usage percentage and memory allocation of Redis container over YCSB workload execution.

ADA-RP have enabled six scaling decisions to allocate 14 GB of memory to Redis which successfully loaded 5,000,000 records.

As mentioned earlier, ADA-RP monitors and collects application metrics including the executed operations per second. As a result, ADA-RP ensures that application performance remains on the desired level during YCSB workload execution. Figure 6.11 illustrates the operations per second executed in Redis at the same time interval as Figure 6.10. As shown in Figure 6.11 operations per second fluctuated between 8,428 and 13,477 during the load phase. Figure 6.11 illustrates that application performance has not been affected during scaling decisions while the executed operations remained above 8,000 throughout the whole experiment.

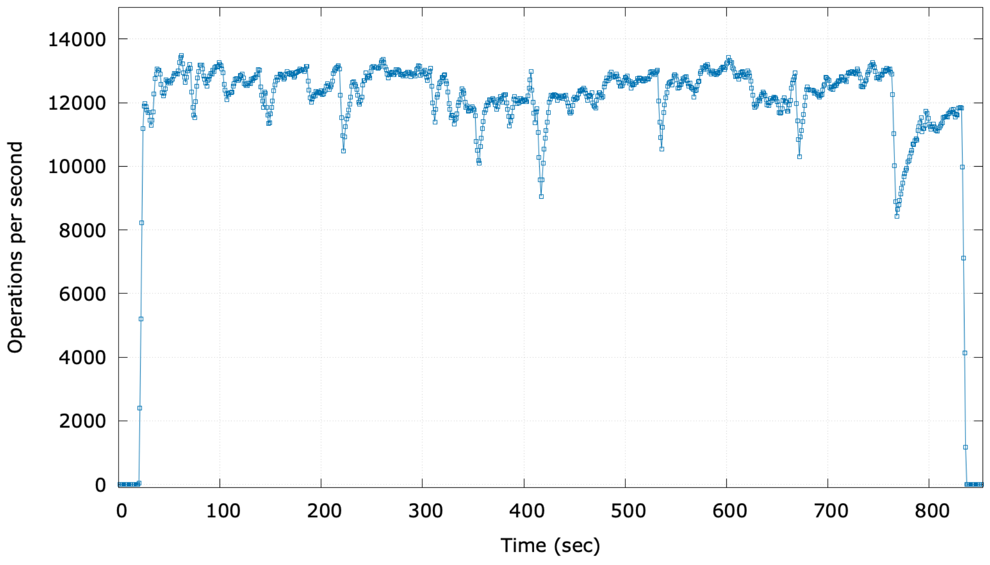


Figure 6.11: Operations per second in Redis over YCSB workload execution of 5 million read and update records.

In this experiment, ADA-RP service automatically scales Redis containerized application based on a threshold-based scaling rule and memory usage metric. ADA-RP effectively uses a reactive approach to prevent Redis application from reaching saturation levels that may lead to system failures and QoS violations. The next section presents the performance evaluation of ADA-RP proactive auto-scaling technique.

6.3.2 Proactive auto-scaling mechanism evaluation

Various experiments conducted in order to demonstrate the effectiveness of ADA-RP service to auto-scale containerized cloud applications using a hybrid proactive mechanism. These include a) auto-scaling SQL databases and b) auto-scaling NoSQL databases in cloud environments.

Auto-scaling SQL databases in cloud environments

ADA-RP service is capable of auto-scaling containerized applications deployed in either single-tenant or multi-tenant cloud environments. ADA-RP uses a hybrid machine learning approach for time series clustering and forecasting to enable automatic resource allocation. During the clustering phase, the K-means algorithm is being trained on historical data in order to cluster time series into High, Medium and Low demand clusters. To support the proposed methodology, a historical dataset for workload characterization has been generated based on different MySQL and TPC-C configurations (see Chapter 5). The latter, used to train the K-means algorithm with a predefined value of $k = 3$, that is the number of clusters.

As shown in Figure 6.12, the K-means algorithm clusters time series data into High, Medium and Low demand clusters based on their CPU utilization levels. Each time series consists of 60 timesteps over a 60 seconds period of time. The DBA averaging method has been used during the K-means learning algorithm that produced an intraclass inertia of 0.197. Figure 6.12 shows the average cluster sequences in red color that used as representative sequences for High, Medium and Low demand clusters respectively. In particular, the representative sequence for High demand cluster fluctuates between 81% and 98% suggesting over-utilized CPU resources. On the other hand, the representative sequence for the Medium demand cluster consists neither under-utilized nor over-utilized resources with values between 47% and 66% respectively. Finally, the representative sequence for the Low demand cluster takes values between 26% and 38% suggesting that the running system is under-utilized. ADA-RP enables an auto-scaling method for cloud resource provisioning. In this domain, K-means automatically characterize future workload demand based on CPU utilization levels without user intervention (e.g., threshold-based scaling rules).

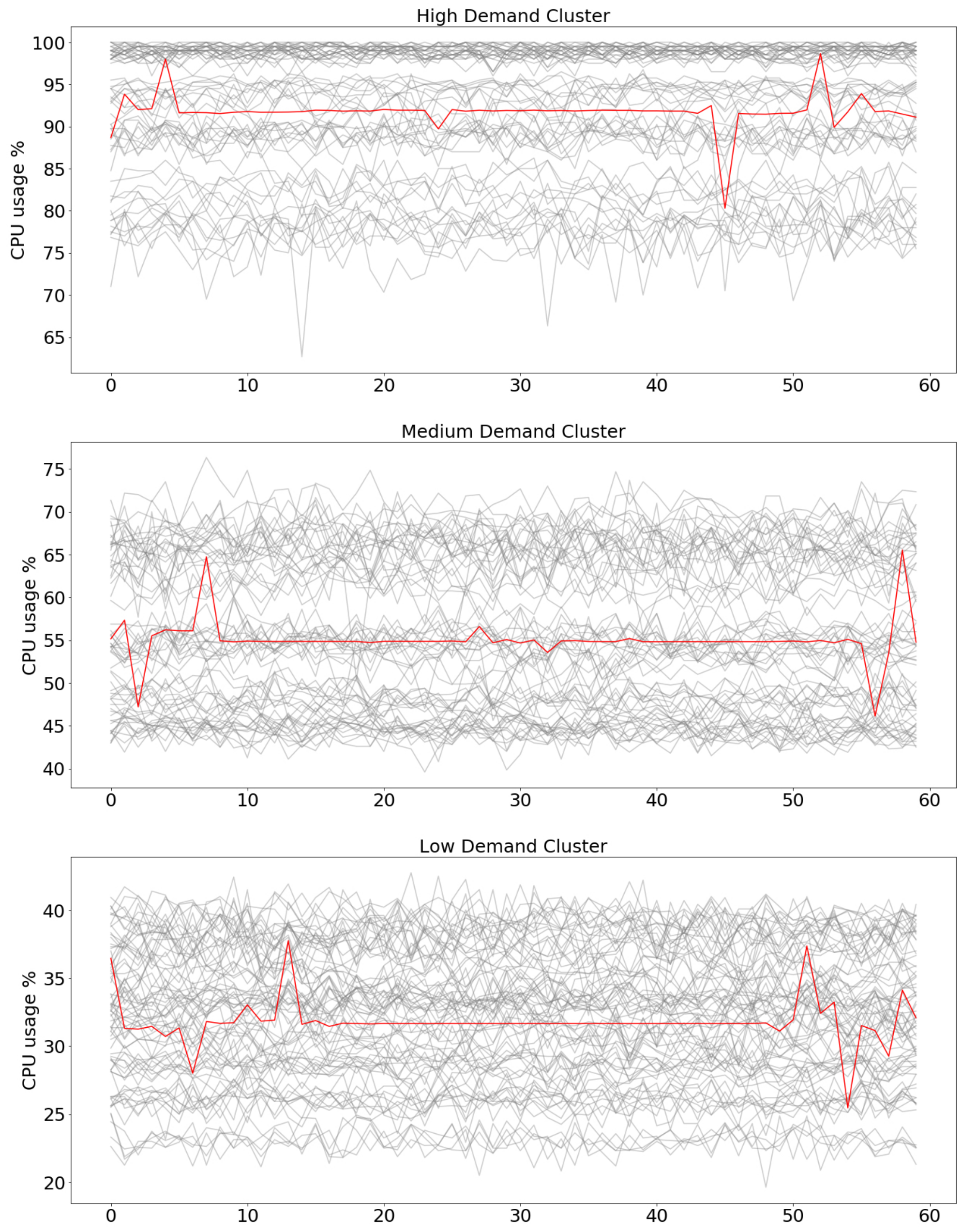


Figure 6.12: K-means algorithm for clustering time series data into High, Medium and Low demand clusters over a 60 seconds period of time. The average sequence for each cluster is shown in red color.

The representative average sequences for High, Medium and Low demand clusters are being used to support two main application deployment scenarios including MySQL deployment in a a) single-tenant environment and b) multi-tenant environment.

To demonstrate the effectiveness of ADA-RP service in a single-tenant cloud environment, an experimental infrastructure is being set up consisting of a VM with 16 vCPU cores and 64GB of Memory deployed in GCP. A MySQL Linux container is deployed using Linux cgroups to limit and isolate CPU resources based on workload demands. Figure 6.13 shows the CPU usage of MySQL container and the vCPU cores allocation during TPC-C workload execution. As shown in Figure 6.13 the MySQL container has no limitation to CPU resources while accessing 16 vCPU cores (VM total) between the first six TPC-C runs. This results to under-utilized CPU resources of MySQL container since the CPU usage percentage fluctuates between 26% and 32% respectively. To adapt to workload demands and reduce under-utilized resources, ADA-RP uses the historical data as an input to train the CNN model and produces CPU usage predictions of 300 seconds (green dotted line). Figure 6.13 shows 6 (out of 10) training wavelets used to extract the predicted wavelet while the CNN model achieves an RMSE score of 0.0162 in the training data and 0.0181 in the testing data respectively. Consequently, the predicted sequence is being

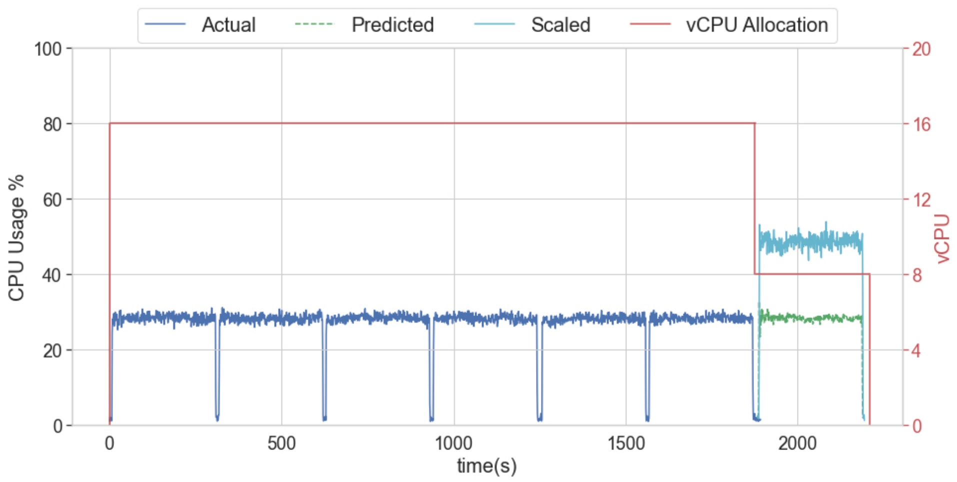


Figure 6.13: CPU usage percentage, CPU usage predictions and virtual CPU allocation based on adaptive auto-scaling strategy for single tenant architecture.

segmented into 60 seconds intervals ($Wl = 60$) and used as an input for the K-means algorithm that clusters each segment into a High, Medium or Low demand cluster. As a result, ADA-RP creates a scaling future plan for the next 300 seconds that contains all scaling decisions for each 60 second segment. As shown in Figure 6.13, ADA-RP reduces the allocated vCPU cores to avoid under-utilized resources in the next TPC-C workload run. In more detail, each segment has been characterised as Low demand, thus, ADA-RP scales down in advance the MySQL container for the next 300 seconds (segments 1-5) from 16 to 8 vCPU cores respectively. The latter results to an increase in CPU usage percentage of MySQL container (light blue line) that fluctuates between 46% and 52% respectively (Medium demand).

Furthermore, ADA-RP extracts application performance metrics and calculates cloud costs to ensure high performance without unreasonable costs. Figure 6.14 shows the QPS and the cloud costs of MySQL container at the same time interval as Figure 6.13. In the first 6 TPC-C runs, the MySQL container has access to 16 vCPU cores with a cost of 411.35$ per month while the executed QPS fluctuated between 21,000 and 24,500 respectively. However, in the last TPC-C run, ADA-RP reduces vCPU cores from 16 to 8 that resulted to a significant cloud cost drop to 215.67$ per month. The reduce of vCPU cores

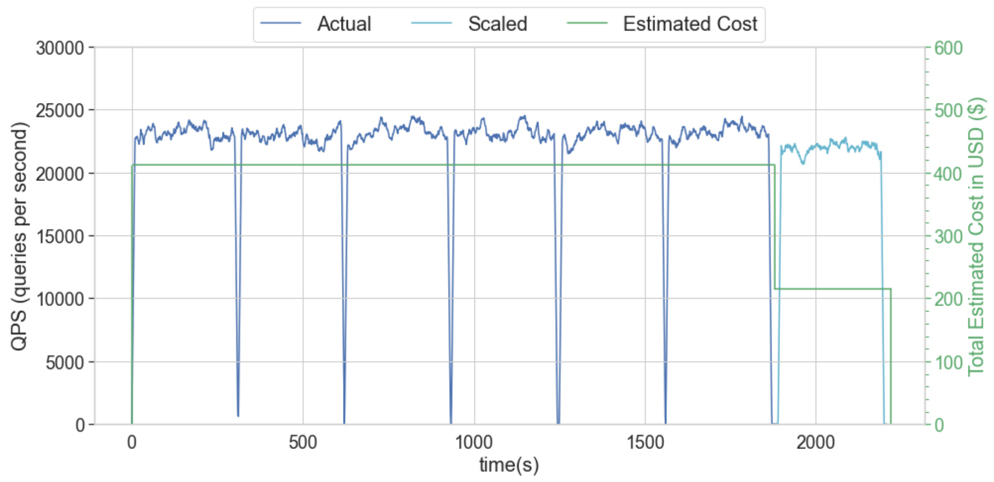


Figure 6.14: QPS of MySQL container and total estimated costs based on adaptive auto-scaling strategy for single-tenant architecture.

is based on the scale factor parameter $Sv = 8$ set by the user. Thus, ADA-RP reduces cloud costs by 48% with a decrease of 6.9% in executed QPS that fluctuated between 20,600 and 22,800. It has to be mentioned that an acceptable reduction threshold is set to 10%, meaning that if the QPS drops below this threshold for 60 seconds, ADA-RP returns to the initial resource configuration. As a result, the user has the ability to adjust the trade-off between cost and performance. This experiment demonstrated the ability of ADA-RP service to reduce under-utilized resources in a single-tenant cloud environment, by reducing the executed QPS. ADA-RP improves the efficiency of the containerized cloud application by decreasing both the CPU idle resources and the cloud costs within service level agreement.

Additionally, this study presents the ability of ADA-RP service to take automatic scaling decisions in a multi-tenant cloud environment. In such environments, tenants may share resources of a single instance to achieve economies. As a result, the system is often over-utilized and the containerized application may experience performance degradation. To demonstrate such a scenario, a VM with 16 vCPU cores and 64GB of Memory deployed in GCP. Furthermore, three MySQL Linux containers deployed in the VM where cgroups have been used to limit the resources of each container to 8 vCPU cores. It has to be mentioned that all three containers have been isolated to use the same number of vCPU cores (1-8) while (9-16) vCPU cores remain unallocated. Figure 6.15 illustrates the CPU usage and the vCPU cores allocation of $container1$ while the TPC-C workload is executed to create repeatable workload patterns. In more detail, each TPC-C run consists of 300 seconds execution time for $container1$ and 180 seconds execution time for $container2$ and $container3$. As shown in Figure 6.15, the CPU usage of $container1$ fluctuates between 54% and 62% for the first and last 60 seconds of each TPC-C run. In that period of time, $container2$ and $container3$ remain idle. Then, to demonstrate a multi-tenant environment where tenants require resources concurrently, for each TPC-C workload execution in $container1$, the TPC-C workload executed inside $container2$ and $container3$ between 61-240 seconds of each 300 second execution. The latter results to a sharp increase in the CPU usage percentage of $container1$ that fluctuates between 93% and 97% respectively.

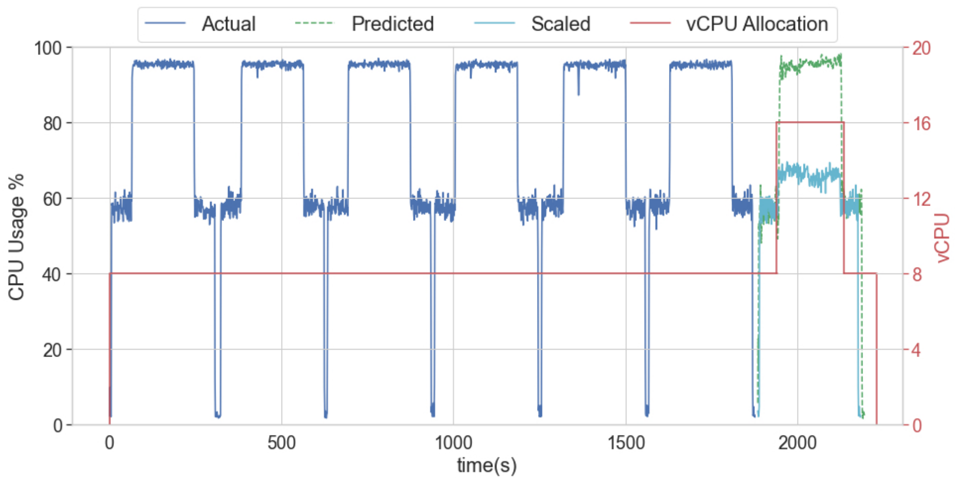


Figure 6.15: CPU usage percentage, CPU usage predictions and virtual CPU allocation based on adaptive auto-scaling strategy for multi-tenant architecture.

Figure 6.15 demonstrates the ability of ADA-RP to adapt to workload demand and avoid over-utilized resources. As shown in figure 6.15, the CNN model generates future CPU usage predictions of 300 seconds (green dotted line) while achieving an RMSE score of 0.0352 in the training data and 0.0627 in the testing data. Figure 6.15 shows 6 (out of 10) training wavelets used to extract the predicted wavelet. The latter has been segmented into five 60 seconds intervals ($Wl = 60$) that used as an input for the K-means clustering algorithm. Ergo, ADA-RP takes decisions about container vCPU cores allocation every minute. Then, ADA-RP creates a scaling plan that includes scaling decisions for each segment to adjust the allocated resources in the container based on workload demand and system availability.

As shown in Figure 6.15, in the 7th TPC-C execution, ADA-RP does not scale $container1$ between 1886-1945 and 2126-2185 seconds (segments 1 and 5) since the K-means clusters both segments as Medium demand. However, between 1946 and 2125 seconds (segments 2, 3 and 4), the K-means identifies the over-utilized CPU resources and clusters all three segments as High demand. Therefore, ADA-RP increases the allocated vCPU cores from 8 to 16 between 1946-2125 seconds (segments 2, 3 and 4) based on a scale factor parameter $Sv = 8$ set by the user. As a result, the state of $container1$ in that time period changed from High to Medium demand as the CPU usage decreased significantly and fluctuated between 58% and 66% respectively.

Additionally, Figure 6.16 shows the QPS and the cloud costs of *container*1 at the same time interval with Figure 6.15. On the one hand, the executed QPS in *container*1 fluctuates between 22,800 and 26,200 while *container*2 and *container*3 remain idle. On the other hand, the executed QPS in *container*1 sharply declined between 9,800 and 12,200, since *container*2 and *container*3 executed the TPC-C workload in parallel. However, the decision of ADA-RP service to increase the allocated CPU resources from 8 to 16 vCPU cores resulted to application performance recovery. As shown in Figure 6.16 the executed QPS in *container*1 significantly increased and fluctuated between 20,400 and 23,500 (light blue line) during the parallel workload execution of the last run.

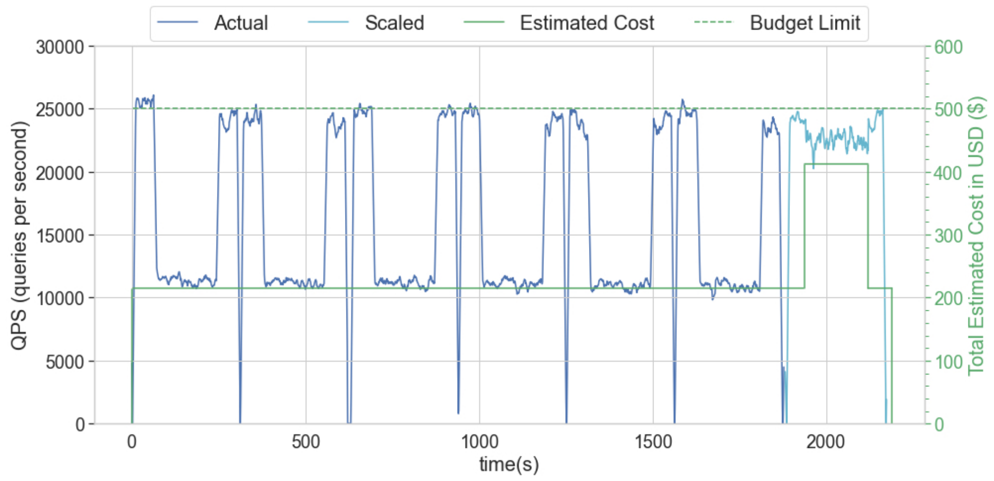


Figure 6.16: QPS of MySQL container and total estimated cost based on adaptive auto-scaling strategy for multi-tenant architecture.

Figure 6.16 shows that during the last TPC-C run, ADA-RP service manages to scale up the CPU resources of MySQL *container*1 in order to avoid over-utilized resources and ensure high application performance. However, as illustrated in Figure 6.16, additional CPU resources resulted to an increase in cloud costs from 215.67$ to 411.35$ respectively. To comply with user requirements, ADA-RP considers a budget limit parameter before decision making. In this experiment, the ADA-RP calculates the costs of the recommended configurations based on Google Cloud Pricing calculator (Google, 2022e) and scales up *container*1 without exceeding the budget limit which is set by the user to 500$ per month.

Auto-scaling NoSQL databases in cloud environments

This experiment demonstrates the effectiveness of ADA-RP service following a proactive auto-scaling technique. As mentioned earlier, ADA-RP uses K-means to cluster time series into High, Medium and Low demand categories. A historical dataset for workload characterization has been generated based on different system and workload configurations. The latter, used to train the K-means algorithm with a predefined value of $k = 3$, that is, the number of clusters. As shown in Figure 6.17, the K-means algorithm clusters time series data into High, Medium and Low demand clusters based on their CPU utilization levels. Each time series consists of 15 timesteps over a 15 seconds period of time.

The DBA averaging method has been used for the K-means learning algorithm that produced an intraclass inertia of 0.053. Figure 6.17 shows the average cluster sequences in red color that used as representative sequences for High, Medium and Low demand clusters respectively. The representative sequence for High demand cluster fluctuates between 88% and 98% suggesting over-utilized CPU resources. On the other hand, the representative sequence for the Medium demand cluster consists neither under-utilized nor over-utilized resources with values between 45% and 53% respectively. Finally, the representative sequence for the Low demand cluster takes values between 18% and 31% suggesting that the running system is under-utilized. In that context, ADA-RP service uses K-means to automatically characterize future workload demand based on average CPU utilization levels without user intervention.

To further support the experimental evaluation, a medium sized VM has been deployed in GCP with 16 vCPU cores, 64 GB memory and 350GB HDD. Then, Redis has been deployed to a docker container as the main application

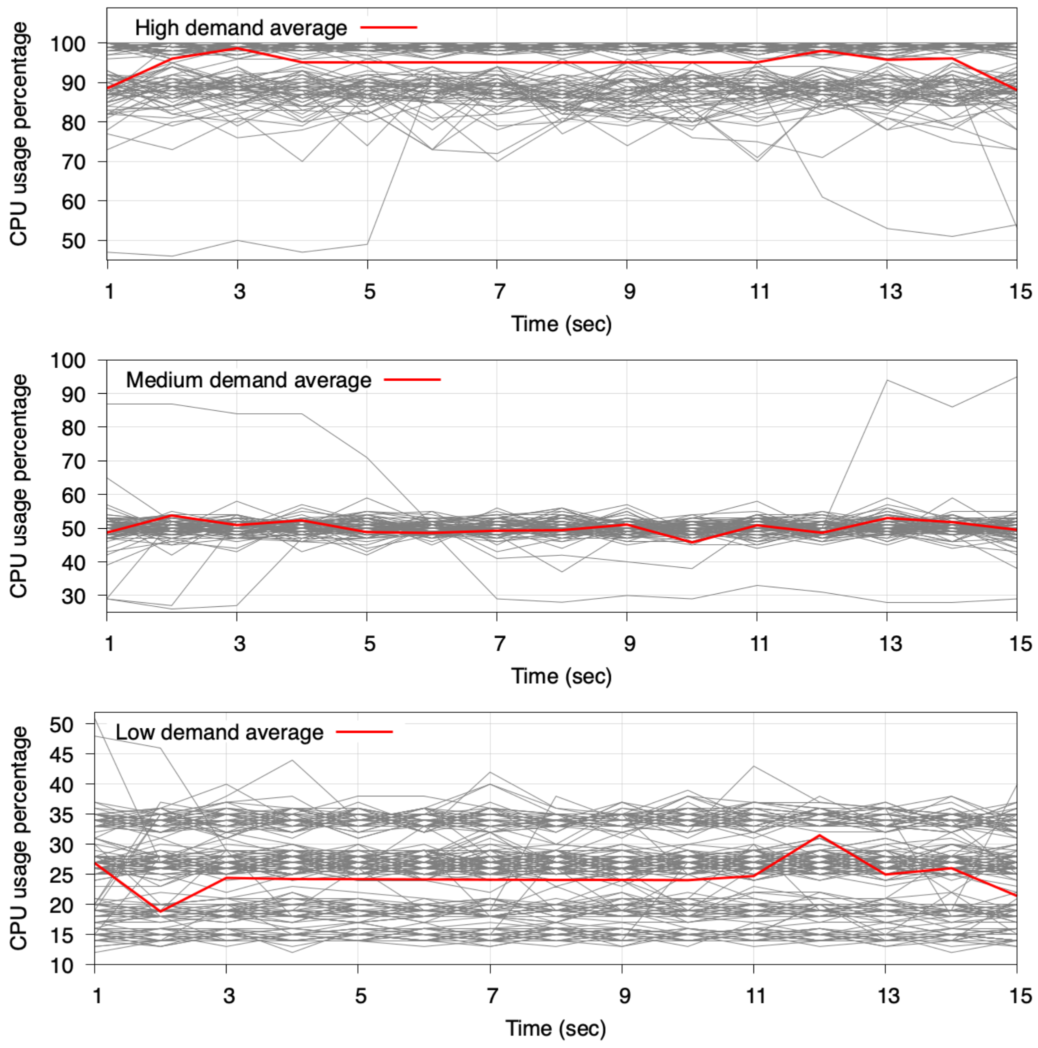


Figure 6.17: K-means algorithm for clustering time series data into High, Medium and Low demand clusters over a 15 seconds period of time. The average sequence for each cluster is shown in red color.

named as $Redis - main$ with access to two vCPU cores (0 and 1) and 16 GB of memory. Additionally, to further stress the main application, a multi-tenant cloud environment introduced where applications execute workloads in parallel and request the same amount of resources at the same period of time. Ergo, three additional Redis containers deployed named as $Redis - 2$, $Redis - 3$ and $Redis - 4$. Then, YCSB workload-c used as a real-world application example for user profile cache, where profiles are constructed elsewhere as discussed in (Cooper et al., 2010). The record and operation counts set to 3,000,000 with 100% read operations and executed to $Redis-main$. Similarly, YCSB Redis workload-c with 500,000 records executed to $Redis-2$, $Redis-3$ and $Redis-4$ respectively.

In order to demonstrate an over-utilized multi-tenant environment, YCSB workload executed to $Redis-main$ and after 15 seconds YCSB workload executed in parallel to $Redis-2$, $Redis-3$ and $Redis-4$. Figure 6.18 illustrates the CPU usage percentage of $Redis-main$ during 9 YCSB workload executions. The CPU usage percentage fluctuates between 40% and 60% while $Redis-2$, $Redis-3$ and $Redis-4$ remain idle. On the contrary, once YCSB workload executed to $Redis-2$, $Redis-3$ and $Redis-4$, the CPU usage percentage reaches a saturation level of 100%. This pattern is being repeated throughout the whole experiment.

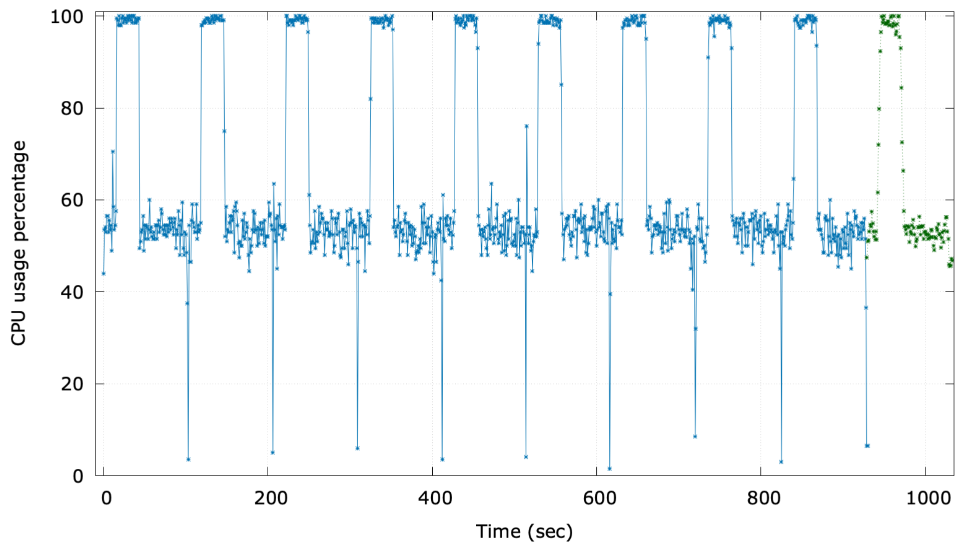


Figure 6.18: CPU usage percentage training set (blue line) and CPU usage predictions (green dotted line).

To support a proactive method, ADA-RP service uses the historical data as an input to train the CNN model that produces CPU usage predictions (green dotted line). Figure 6.18 shows 9 (out of 10) training wavelets used to extract the predicted wavelet while CNN model achieved an RMSE score of 0.0554 in the training data and 0.0651 in the testing data respectively.

Furthermore, Figure 6.19 illustrates the CNN CPU usage prediction (green dotted line), the allocated vCPU cores (red line) and the scaled CPU usage percentage (blue line) during ADA-RP scaling decisions. On the bottom, the green and red bars illustrate ADA-RP scaling decisions. In this experiment, the decision time window has been set to 15 seconds and the scaling parameter Sv to 4. As a result, ADA-RP service segments the predicted sequence and makes scaling decisions every 15 seconds.

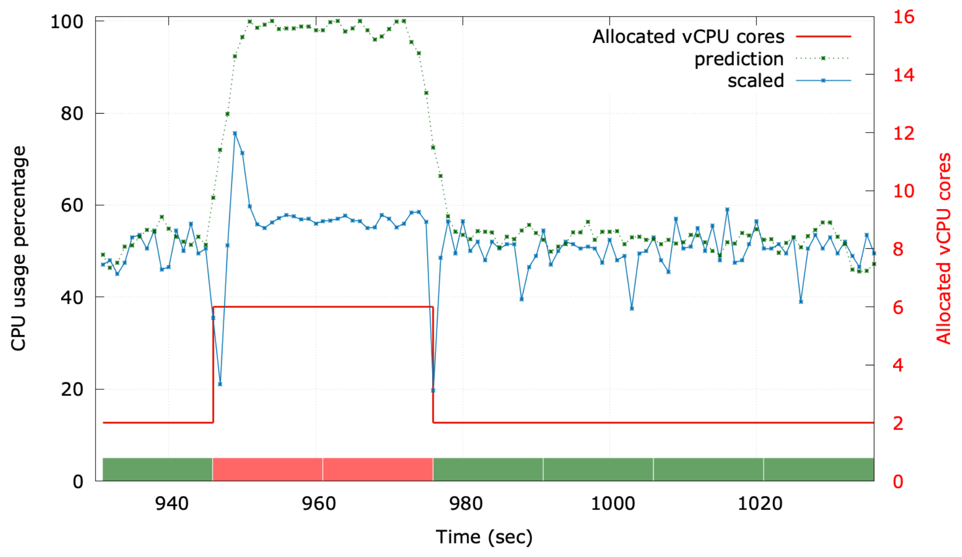


Figure 6.19: CPU usage prediction (green dotted line), CPU usage after scaling actions (blue line) and vCPU cores allocation (red line) using ADA-RP service

In more detail, the green bar shows that during the given time window, ADA-RP service has clustered the predicted CPU usage percentage as Medium demand, thus ADA-RP allocates 2 vCPU cores to $Redis-main$ container. On the other hand, the red bar indicates that ADA-RP service has clustered the predicted CPU usage percentage as High demand, thus ADA-RP allocates 4 additional vCPU cores to $Redis-main$ container leading to 6 vCPU cores during that period of time. As shown in Figure 6.19, ADA-RP service manages to reduce CPU usage percentage utilization between 946 and 976 seconds where the predicted CPU usage percentage would have reached the maximum utilization. It has to be mentioned that the CPU usage percentage consists the average CPU usage of the allocated cores to the container.

Furthermore, to ensure application performance, ADA-RP monitors and collects the operations per second executed in $Redis-main$ container during YCSB workload execution. Figure 6.20 illustrates the operations per second of $Redis-main$ container during the same period of time with Figure 6.18. In more detail, Figure 6.20 red shaded area shows that the performance of $Redis-main$ container significantly drops when $Redis-2$, $Redis-3$ and $Redis-4$ containers execute YCSB workload in parallel. However, ADA-RP proactive approach ensures that 4 additional vCPU cores have been allocated to $Redis-main$ container to incorporate future workload demand. As a result, the average operations per second executed in $Redis-main$ during the overload period increased from 9,173 to 14,850.

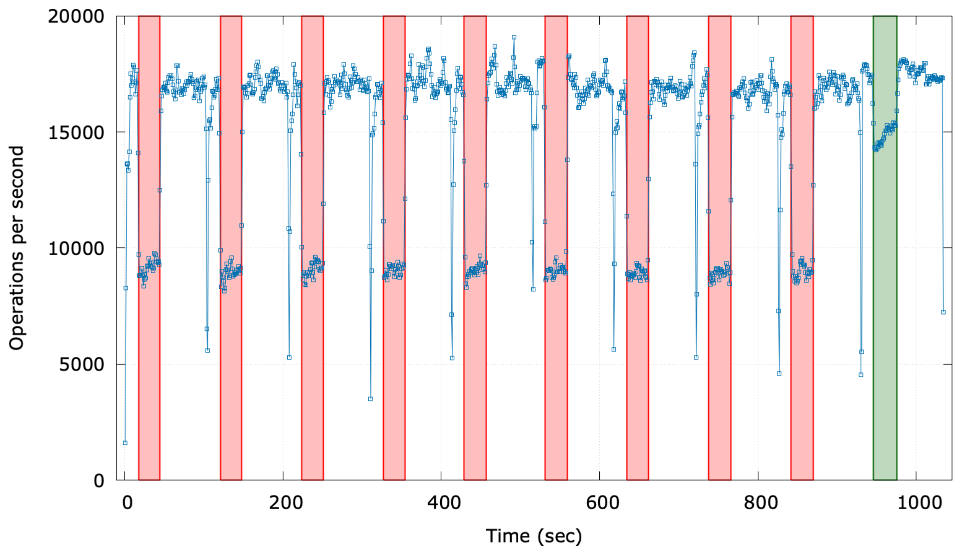


Figure 6.20: Operations per second in $Redis-main$ in multi-tenant environment. Red shaded area indicates system overload period of time and green shaded area shows system recovery.

ADA-RP service enables both reactive and proactive scaling techniques to ensure high application performance under different scenarios. These experiments introduced a threshold-based scaling rule to guarantee application performance and avoid system failures. Moreover, a hybrid technique that is based on time series forecasting and clustering enabled automatic scaling under concurrent workload executions in a multi-tenant cloud environment.

ADA-RP maintained application performance in both experiments and guaranteed QoS requirements.

6.4 SONA Experimental Evaluation

SONA supports constrained performance optimization of cloud applications using a hybrid approach of artificial neural networks and genetic algorithms. The proposed service monitors the source system to identify the optimal configurations that maximize application performance based on genuine workload executions. To demonstrate the effectiveness of the proposed method, a novel dataset collected from various TPC-C runs in a MySQL server (see Chapter 5). Furthermore, SONA uses a cloned containerized environment that replicates the main application to avoid system overhead. This section presents the experimental evaluation of SONA service. It includes a) SONA performance analysis and b) SONA experimental results.

6.4.1 SONA performance analysis

The SONA service enables database constrained optimization and tuning based on user requirements. The service is being evaluated based on functionality, efficiency and performance aspects. Ergo, four parameters are being introduced as follows: (a) cost, (b) cost-performance ratio, (c) baseline performance and (d) average idle CPU resources metrics. Additionally, both ANN and GA evaluated in order to ensure a low error rate and algorithmic convergence.

Cost, cost-performance ratio and TpmC

This subsection introduces the cost attribute as a significant factor that impacts SONA service ability to manage system resources. The cost attribute is a key factor for different applications in cloud environments (Chiang, Ouyang, and Hsu, 2014). However, allocating computing resources which adapt to workload variations and produce user desired performance with minimum cost, is not a trivial task (M. Mao, J. Li, and Humphrey, 2010). Here, the TpmC performance metric in relation to GCP pricing policy is being introduced to describe the relationship between system performance and system expenditures. The cost-performance ($/TpmC) ratio attribute is defined as the monthly GCP cost (in US dollars) divided by the TpmC metric. Introducing a resource management service that that well-balances both is challenging. Figure 6.21 underline the relationship between TpmC and cost during TPC-C workload executions in MySQL systems with different resource and application configurations. Table 6.1 shows the system sizes where each system has 500 GB of magnetic disk and 1024 MB InnoDB buffer pool size.

Table 6.1: System configurations

System Type	A	B	C	D	E	F
CPU Cores	1	2	4	8	12	16
Memory in GB	3.75	7	15	30	45	60

Figure 6.21 illustrates the TpmC score of the TPC-C workload execution on various systems named as A, B, C, D, E and F as in Table 6.1. The TpmC linear performance growth follows a linear growth in relation to system resources while TPC-C workload executed with the same configurations (e.g., number of

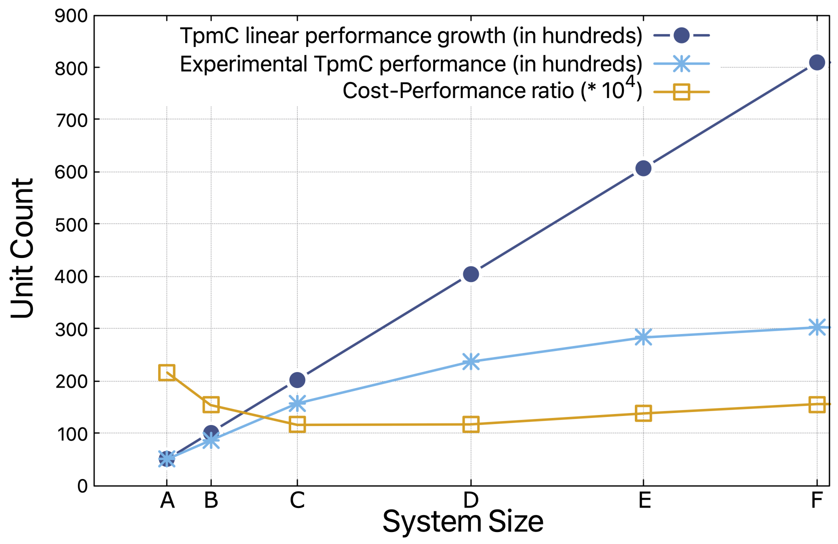


Figure 6.21: Cost-performance ratio and TpmC performance

warehouses). The latter, indicates that if the resources double in size, this will lead to a double number of TpmC. On the contrary, the experimental TpmC defines the TpmC of various systems based on genuine executions of TPC-C workload. Figure 6.21 shows that the experimental TpmC score does not follow a linear performance growth as the TpmC score converges to 30,000 units when executed in system F. Although, an increase to the allocated computing resources maximizes TpmC score, this does not necessarily lead to an efficient solution. A system with low cost-performance ratio, indicates a higher system efficiency since it delivers performance with lower cost. Figure 6.21 shows that the cost-performance ratio has the minimum value for system C with 116 units (the cost-performance ratio has been multiplied by 10^4 for visualization purposes). Thus, cost-performance ratio suggests that system F is not the most efficient solution even if it produces the maximum TpmC score. This example demonstrates that an optimized cost-performance ratio is required to identify the best configurations in terms of both performance and efficiency. SONA service also validates the recommended configurations based on the user's budget and the baseline performance. Thus, a system that exceeds a predefined Budget or a system that produces a TpmC lower than the baseline performance is being penalized.

It has to be mentioned that R_j are tunable penalty coefficients for constraints C_j. High penalty coefficient values indicate a solution that strictly satisfies user requirements while on the contrary low penalty coefficient values cannot guarantee that. SONA consists a flexible framework that enables the user to tune the penalty coefficients. As will be discussed later in this section, SONA could allow a relatively low violation in one requirement in order to gain significant improvement in the overall system efficiency.

CPU idle

This subsection introduces the CPU idle as a performance evaluation metric of SONA service. The impact of the CPU idle has been addressed in various cloud applications (Mei et al., 2010). In this thesis, SONA service measures the CPU idle percentage to allocate a reasonable amount of resources to the running system. SONA extracts the CPU idle metric every second during workload execution. Then, the average CPU idle percentage of each system is calculated for the load and run TPC-C stages respectively.

Figure 6.22 illustrates the CPU idle percentage in the load and run TPC-C stages for systems A, B, C, D, E and F. In the load stage, the CPU idle averages 9.8% for System A with 1 CPU core. The CPU idle rises sharply to 42.9% for system B with 2 CPU cores and reaches 71.2% for system C with 4 CPU cores. The CPU idle continues to increase until it reaches the maximum value of 92.8% for system F with 16 CPU cores. On the other hand, CPU idle is significantly low for systems A, B and C in the run stage. However, the CPU idle rises to 22.4% for system D while reaches a peak of 57.5% for system F. Figure 6.22 shows that running the same TPC-C workload while at the same time increasing the allocated computing resources, leads to high CPU idle levels in both load and run stages.

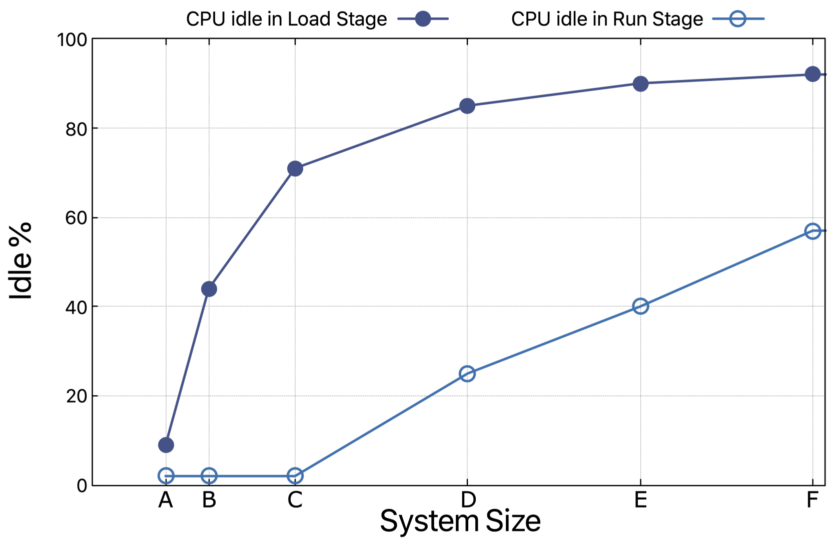


Figure 6.22: CPU idle percentage over Load and Run stage.

Table 6.2 illustrates (a) the CPU utilization percentage (load/run), (b) the CPU idle percentage (load/run) and (c) the TpmC performance of each system during the same TPC-C workload execution. The CPU_{Idle} is being calculated using the overall $CPU_{Utilization}$ Linux metric as shown in equation 6.1:

$$CPU_{Idle} = 1 - CPU_{Utilization} \tag{6.1}$$

Table 6.2 shows that the $CPU_{utilization}$ for system A remains above 90% on average for both load and run stages with a 5,058 TpmC. However, in load stage the $CPU_{utilization}$ is in decline for each system while reaches the minimum value of 7.2% for system F. In the run stage, the CPU is highly utilized for systems A, B and C. On the contrary, $CPU_{utilization}$ started to plummet until reaches a minimum CPU utilization of 42.5% for system F. Although, TpmC performance increases in relation to system size, system F achieves 30,266 TpmC score with only 7.2% of $CPU_{utilization}$ in the load stage and 42.5% $CPU_{utilization}$ in the run stage. As a result, SONA service penalizes underutilized systems with high levels of idle CPU resources as discussed in constraint C4.

Table 6.2: Average system utilization and system idle.

	Load Stage		Run Stage		Performance
System	*Utilisation*	*Idle*	*Utilisation*	*Idle*	*TpmC*
A	90.2%	9.8%	97.4%	2.6%	5,058
B	57.1%	42.9%	97.3%	2.7%	8,645
C	28.8%	71.2 %	96.8%	3.2%	15,695
D	14.5%	85.5%	77.6%	22.4%	23,689
E	9.2%	90.3%	59.1%	40.9%	28,319
F	7.2%	92.8%	42.5%	57.5%	30,266

SONA service introduces a threshold to manage idle CPU resources based on user requirements. On the one hand, a low threshold value will show high tolerance in underutilized systems that may harm overall system efficiency. On the other hand, a high threshold will be less forgiving to idle CPU resources which may affect the TpmC performance. To balance this trade off the recommended CPU idle threshold set to 50%. It should be noted that the threshold can be tuned based on user requirements.

Artificial Neural Network evaluation

ANN model evaluation demonstrates the effectiveness of the model to produce accurate predictions so as to avoid propagating a high error rate to the GA optimization process. The ANN model uses the back propagation algorithm to find the optimal weights that minimize the mean squared error (MSE) loss function as shown in equation 6.2:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{6.2}$$

where $\hat{y}_i$ is the prediction that ANN model produces for the ith observation and y_i is the true response variable.

The dataset has been separated into three parts, the training data, the validation data and the test data based on the following proportions: 70%, 15% and 15% respectively. The model is said to be optimized when the validation score is not further improving over a particular number of epochs according to early stopping technique. Thus, in the training process, the MSE of the ANN model in the training samples was $2.6479e-4$ while the MSE in the validation samples reached $8.8062e-4$ at the epoch of 313. The MSE of the ANN model in the test samples was $4.65032e-3$.

Furthermore, the significance of ANN model is evaluated by utilizing the coefficient of determination (R^2) that clarifies the relation between the experimental and predicted values. Figure 6.23 illustrates the predictions of ANN model over the training and testing observations versus the experimental values of the TpmC performance. Figure 6.23 shows that ANN produces accurate predictions for both training and test set of observations. The blue symbols indicate the predicted TpmC in the training set and the red symbols indicate the predicted TpmC in the test set. The R^2 takes the value of 0.986 which shows the high accuracy of the model. The diagonal green line signifies perfect prediction where $R^2 = 1.00$.

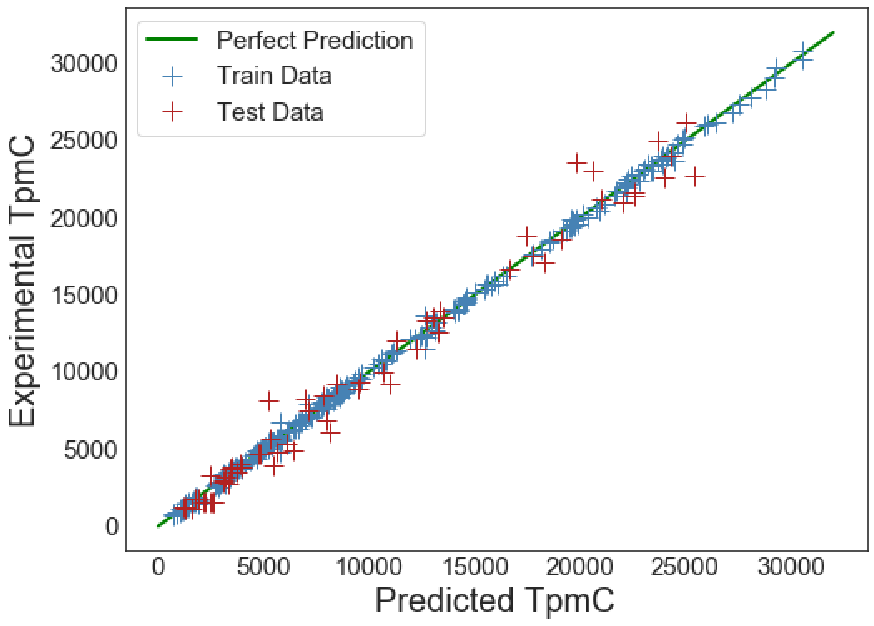


Figure 6.23: Artificial Neural Network TpmC predictions over the training and test set vs the Experimental TpmC.

Genetic algorithm evaluation

In this work, the population size is set to 20 and the number of iterations is set to 250. The crossover probability is set to 0.7 while mutation used with probability equal to 0.001. The fitness function based on the ANN model used to calculate the fitness for each individual in the population. Figure 6.24 illustrates the evolution of generations. The optimized features, produced by the GA, are as following: 16 CPU cores, 60GB Memory, 500GB Solid State Drive, 1024MB InnoDB buffer pool size while TPC-C executes in 5 Warehouses. The aforementioned configurations produced 31,326 TpmC score.

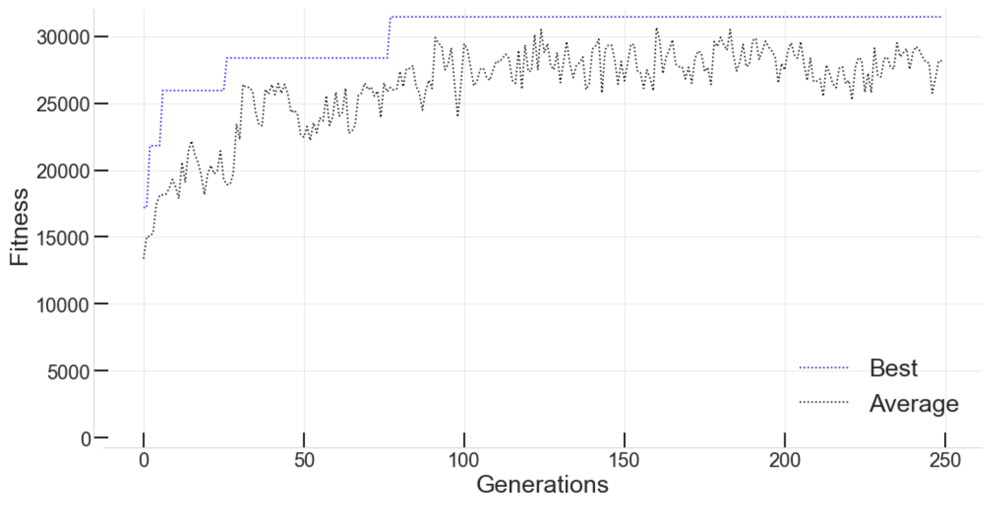


Figure 6.24: Evolution of generations for TpmC optimization.

Figure 6.24 illustrates the convergence of the GA algorithm. It can be observed that the GA optimization result is satisfactory as the population gets better fitness score and reach algorithmic convergence at the end of the evolution.

6.4.2 SONA experimental results

During the experimental evaluation of SONA service, MySQL relational database management system used as the running cloud application deployed in GCP. In addition, TPC-C used as an on-line transaction processing benchmark to evaluate the performance of MySQL based on the TpmC metric. All the experiments executed in a clone system that does not affect the performance of the main application. Three experimental scenarios conducted to show the effectiveness of the proposed service. This includes (a) resource optimization, (b) MySQL application tuning, and (c) resource and application optimization. All the aforementioned scenarios demonstrate the effectiveness of SONA to optimize the target system based on user requirements.

Resource optimization

In this experiment, SONA service enables the neuro-genetic optimization process to identify the requirements of the workload and scale down system resources for improving efficiency in terms of (a) lower cost, (b) lower cost performance ratio and (c) lower idle CPU resources. Figure 6.25 shows the TpmC performance metric (in thousands) of the source system F and the recommended system E while TPC-C workload executed in 5 Warehouses. System F includes 16 CPU cores, 60GB Memory, 500GB Magnetic Disk and 1024MB InnoDB buffer pool size and scores 27,768 TpmC units (27.7 thousands TpmC). On the other hand, System E includes 12 CPU cores, 45GB Memory, 500GB Magnetic Disk and 1024MB InnoDB buffer pool size and scores 27,408 TpmC respectively.

Figure 6.25 shows that decreasing the CPU and Memory resources by 25% leads to 1.3% reduction in the TpmC from 27,768 to 27,408. However, SONA recommends system E since it gains a reduced cost of 20.3% from 408$ to 325$, a lower cost performance-ratio of 19% from 0.0147 to 0.0119 and a lower CPU idle of 15.6% from 57.5% to 41.9%. In this experiment, the user sets the

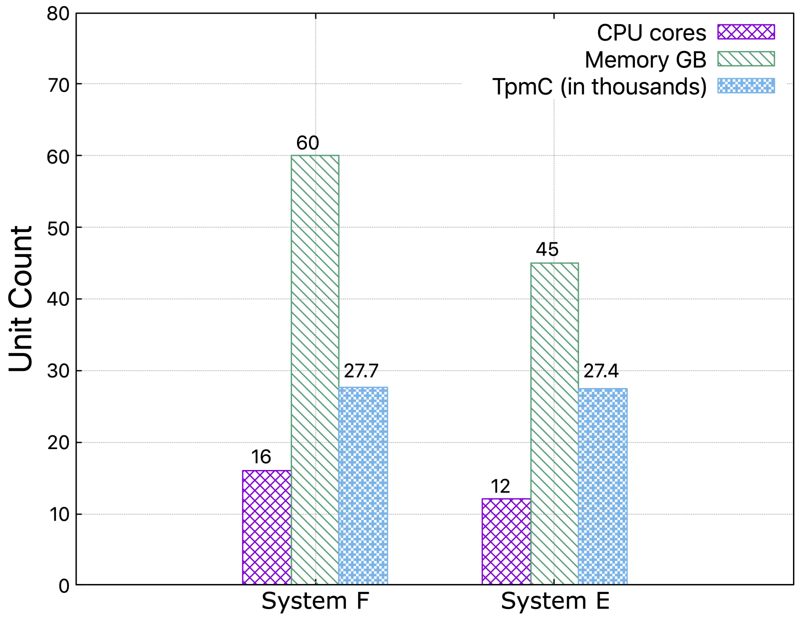


Figure 6.25: Scale down for improving efficiency.

monthly budget to 350$, so SONA does not penalize system E according to C1 constraint since it does not exceed the budget limit. System E has lower cost-performance ratio, so it has a lower penalty term according to C2 constraint. Since the recommended configurations lead to a decrease in the baseline performance, SONA adds a small penalty term to System E according to C3 constraint. As the CPU idle threshold is set to 50%, SONA does not penalize System E according to C4.

Database application tuning

MySQL tuning is a process that adjusts application configurations to maximize the performance. SONA service is able to automatically identify both the system and application configurations that produce the highest TpmC. In this experiment, the user has set a predefined budget limit that does not exceed source system costs. Thus, SONA heavily penalizes all recommended systems that exceed current monthly costs according to C1 constraint, while at the same time explores the optimal application configurations to produce the highest TpmC.

Figure 6.26 shows the TpmC performance of the source system D and the recommended System D*. System D includes 8 CPU cores, 30GB Memory, 500GB Magnetic Disk and 128MB InnoDB buffer pool size (default size). System D* has the same amount of resources with the same monthly costs, however the InnoDB buffer pool size is set to 1024MB. System D and System D* produce 10,500 and 22,234 TpmC while TPC-C workload executed in 5 Warehouses. During the optimization process, System D* is being recommended since outperforms System D in terms of performance and efficiency. Increasing the InnoDB buffer pool size from 128MB to 1024MB results to (a) an increase of 111.7% in TpmC from 10,500 to 22,234, (b) a decrease of 52.9% in cost-performance ratio from 0.0204 to 0.0096 and (c) a decrease of 27% in CPU idle from 53% to 26%.

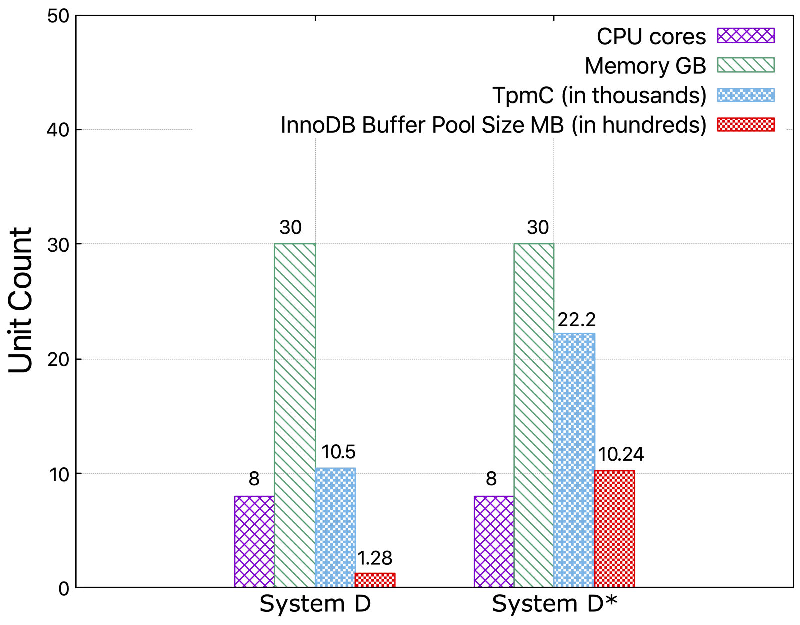


Figure 6.26: Application tuning for TpmC optimization.

In this experiment, System D* allocates the same amount of resources so it is not being penalized according to C1. System D* has lower cost-performance ratio so it has a lower penalty term according to C2. Since the recommended configurations outperform the baseline performance SONA does not penalize the recommended system according to C3 constraint. As the CPU idle threshold is set to 50%, SONA does not penalizes System D* according to C4.

Resource and application optimization

This experiment demonstrates a combination of resource and application optimization. Figure 6.27 shows the TpmC performance metric, the number of CPU cores and Memory size of systems B, E and F respectively. System B is the source system that includes 2 CPU cores, 7.5GB Memory, 500GB magnetic disk and 128MB InnoDB buffer pool size with 6,236 TpmC. System E and System F are two candidates for improving the baseline performance. System E includes 12 CPU cores, 45GB Memory, 500GB Solid State Drive and 1024MB InnoDB buffer pool size. On the other hand, System F includes 16 CPU cores, 60GB Memory, 500GB Solid State Drive and 1024MB InnoDB buffer pool size.

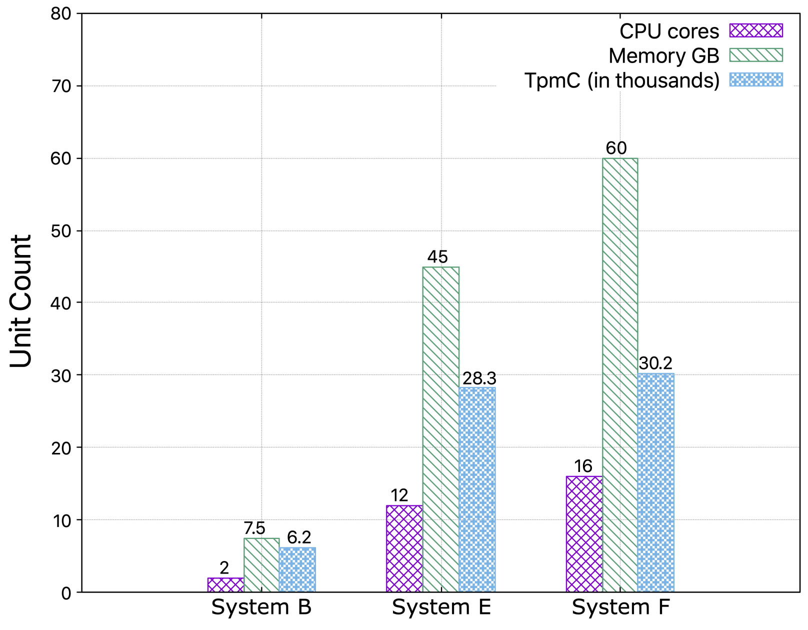


Figure 6.27: Scale up for improving TpmC performance.

System E and System F produce 28,319 and 30,266 TpmC respectively.

System E results to (a) an increase of 354.1% in the baseline TpmC from 6,236 to 28,319, (b) an increase of 473.5% in the cost from 68$ to 390$, (c) an increase of 25.4% in the cost-performance ratio from 0.0110 to 0.0138 and (d) an increase of of 27.6% in the CPU idle from 13.3% to 40.9%. System F results to (a) an increase of 385.3% in the original TpmC score from 6,236 to 30,266, (b) an increase of 595.5% in the cost from 68$ to 473$, (c) an increase of 41.8% in the cost-performance ratio from 0.0110 to 0.0156 and (d) an increase of 44.2% in the CPU idle from 13.3% to 57.5%.

In this experiment, the user sets the monthly budget to 400$, thus, according to C1 budget constraint, SONA does not penalize System E (390$) in contrast to System F (473$). Both E and F systems receive a penalty term according to C2 cost-performance ratio constraint. However, System F has the highest cost-performance ratio so it has the highest penalty term. Since the recommended configurations outperform the baseline performance, SONA does not penalize System E and System F according to C3 constraint. As the CPU idle threshold is set to 50%, SONA penalizes System F (57.5%), while it does not penalize System E (40.9%) according to C4 constraint. As a result, although System F produces higher TpmC, SONA recommends system E with the optimal system and application configurations. Figure 6.28 shows the TpmC performance, the Disk size, the Disk type and the InnoDB buffer pool size of system B and system E respectively.

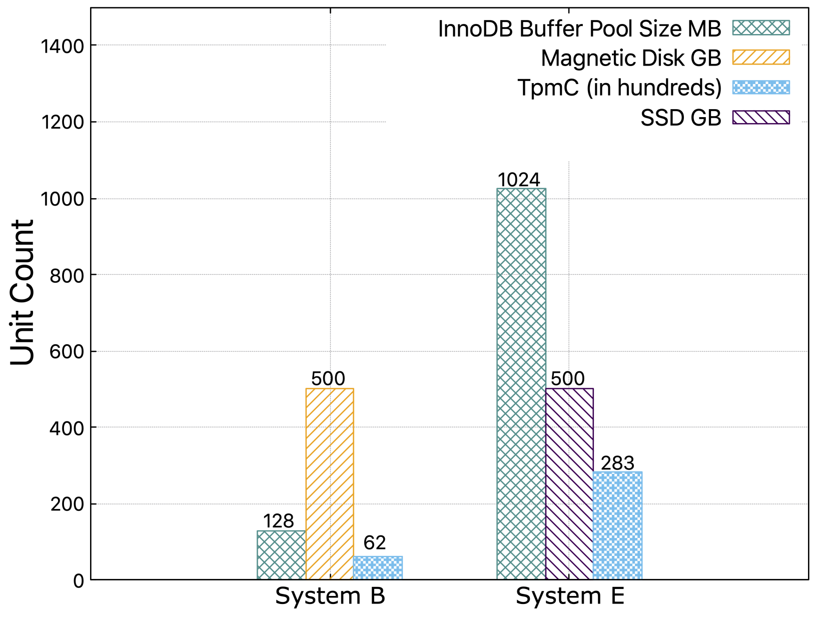


Figure 6.28: System configuration and application tuning for TpmC optimization.

As shown in Figure 6.28 SONA changes the disk type from Magnetic to SSD and increases the InnoDB buffer pool size to achieve higher TpmC. Although, System E increases the TpmC performance within user's budget, is being penalized due to the increase of the cost-performance ratio. This demonstrates the trade off between performance and efficiency, and the need for a balanced solution. SONA consists a flexible framework that enables the user to adjust R_j penalty coefficients based on application demands. In this experiment, the R_2 coefficient adjusted so as to not heavily penalize systems with high cost-performance ratio. Thus, SONA suggests System E configurations that sharply increase TpmC performance within user's budget and with acceptable idle CPU resources.

6.5 Summary

This chapter presents the performance evaluation of PACE framework and its services namely as ADM2, ADA-RP and SONA. It presented experiments conducted in different system and application settings in order to demonstrate the effectiveness of each service. In particular, it includes the experimental evaluation of ADM2 service that enables automatic performance degradation of cloud applications. Furthermore, ADA-RP service evaluated based on its ability to scale up or down containerized cloud applications. Both reactive and proactive techniques have been suggested and evaluated based on different experimental scenarios. Finally, SONA experimental evaluation showed that the proposed technique effectively optimizes cloud resources and tunes application parameter in order to maximize performance based on user requirements.

The experimental achievements showed that PACE framework designed to act in different cloud environments using real-world systems and state-of-the-art benchmarks. The next chapter discusses the conclusion and the limitations of the proposed methodology that further enables a discussion around the future directions of this work.

7 Conclusions and Future Directions

This chapter provides a summary of the research reported in this thesis. In more detail, section 7.1 outlines the major contributions of the thesis. Section 7.2 presents the limitations of this thesis and provides future directions to extend this work. Finally, section 7.3 offers some concluding remarks.

7.1 Major Contributions of the Thesis

The main contribution of this thesis is the development of PACE framework and its services to support reliable and adaptive resource provisioning in cloud computing environments. The following list concludes the major contributions of this thesis:

1. Anomaly detection in cloud systems using LSTM Autoencoder. The proposed methodology effectively monitors and detects aberrant patterns in cloud serving systems. As a result, PACE framework alerts system administrators on-the-fly and ensures that PACE decisions are not affected by abnormal system behavior.

2. Reactive auto-scaling using threshold-based rules. PACE automatically

adjusts cloud resources to avoid application failures and to ensure QoS requirements during intensive workload tasks. Proactive auto-scaling using hybrid machine learning techniques. PACE analyzes monitored data to create scaling policies that incorporate future workload demands. As a result, PACE improves cost efficiency of under-utilized systems and increases application performance during intensive workloads tasks.

3. Constrained optimization of cloud applications based on historical events. PACE explores the optimum configurations that maximize application performance based on user requirements. The decisions are based on historical records from previous workload executions in the running system.

4. An experimental evaluation of PACE framework using state-of-the-art benchmarks for cloud applications. PACE enables a lightweight container-based solution to support fast application deployment and dynamic resource reconfiguration. This also supports the proposed cloning strategy that validates PACE recommendations with zero overhead to the main application.

Having said that, this thesis proposed a framework for reliable and adaptive resource provisioning in cloud computing environments. The proposed methodology effectively adjusts system resources based on workload demand while optimizes system performance to satisfy user requirements. In extension to this, it proposes an anomaly detection technique to ensure that decision making is not prone to errors due to abnormal system behavior.

7.2 Limitations and Future Directions

Although, the research of this thesis addressed the proposed aims and objectives, there are still areas for further development and improvement. This section discusses the limitations of this work and includes recommendations to extent its functionality.

The proposed framework enables constrained optimization of cloud applications to satisfy a variety of user requirements including idle CPU resources which are related to the CPU energy consumption. In that context, a future direction includes the consideration of energy consumption measurements. This requires additional functionalities that calculate the energy consumption of each recommended system configuration. Ergo, the user will be able to set energy consumption constraints that will be included in the optimization process. As a result, the energy consumption measurements will extend the current functionalities that enable efficient utilization and cost reduction.

The proposed methodology is mainly based on historical data to enable reliable resource provisioning and optimization of cloud applications. Having said that, learning-based techniques need a grace period for monitoring, collecting and analyzing data events before triggering scaling decisions. Additionally, this thesis proposes a technique to provision resources based on periodic workloads patterns. As a result, the proposed solution is focused on periodic workload patterns where the system experiences foreseeable utilization patterns over time.

PACE framework proposed an LSTM Autoencoder model to detect performance degradation of cloud systems. The LSTM Autoencoder learns to reconstruct normal throughput patterns while it produces high reconstruction error on new unseen abnormal sequences. As a result, the proposed mechanism classifies throughput observations as normal or abnormal based on a threshold parameters. However, parameter tuning is not a trivial task and often requires domain-expert users. This work could be extended to use machine learning classification algorithms such as Random Forests and Support Vector Machines to classify the encoded sequences as normal or abnormal. As a result, a hybrid approach will enable both LSTM Autoencoder and classification algorithms to detect abnormal system behavior while meeting requirements of less skilled users.

Future directions also include the exploration of different workloads and database systems that will realize different applications scenarios. These systems can be deployed on different cloud platforms including AWS, Azure and IBM Cloud. New application scenarios will enable the collection of additional records to enrich PACE datasets and create new datasets to support the proposed methodologies.

7.3 Concluding Remarks

This thesis proposed PACE, a framework for reliable and adaptive resource provisioning in cloud computing environments. PACE monitors the running system to collect a variety of metrics. These are being analyzed to enable anomaly detection to alert system administrators for abnormal events and to ensure that the resource provisioning process is not prone to incorrect decisions due to abnormal system behavior. Consequently, PACE automatically adjusts system resources to improve application performance and ensure efficient utilization levels. Additionally, PACE collects information about historical runs to identify the configurations that maximize application performance while at the same time satisfy user requirements. The results of this research can be extended with the collection of additional datasets and the use of alternative database systems such as Apache Cassandra and Neo4j.

Bibliography

Abadi, Daniel J, Peter A Boncz, and Stavros Harizopoulos (2009). "Column-oriented database systems". In: *Proceedings of the VLDB Endowment* 2.2, pp. 1664–1665.

Abrahao, Bruno et al. (2006). "Self-adaptive SLA-driven capacity management for internet services". In: *2006 IEEE/IFIP Network Operations and Management Symposium NOMS 2006*. IEEE, pp. 557–568.

Ali-Eldin, Ahmed, Maria Kihl, et al. (2012). "Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control". In: *Proceedings of the 3rd workshop on Scientific Cloud Computing*, pp. 31–40.

Ali-Eldin, Ahmed, Johan Tordsson, and Erik Elmroth (2012). "An adaptive hybrid elasticity controller for cloud infrastructures". In: *2012 IEEE Network Operations and Management Symposium*. IEEE, pp. 204–212.

Amazon (2022a). *Amazon EC2 auto scaling*. URL: https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scale-based-on-demand.html (visited on 09/17/2022).

— (2022b). *Amazon Elastic Compute Cloud reserved instances*. URL: https://aws.amazon.com/ec2/pricing/reserved-instances/ (visited on 09/17/2022).

— (2022c). *Amazon Elastic Container Service (Amazon ECS)*. URL: https://aws.amazon.com/ecs/ (visited on 09/17/2022).

Angles, Renzo and Claudio Gutierrez (2008). "Survey of graph database models". In: *ACM Computing Surveys (CSUR)* 40.1, pp. 1–39.

Arabnejad, Hamid et al. (2017). "A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling". In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, pp. 64–73.

Astillo, Philip Virgil et al. (2022). "Federated intelligence of anomaly detection agent in IoTMD-enabled Diabetes Management Control System". In: *Future Generation Computer Systems* 128, pp. 395–405.

Avula, Raghu Nandan and Cliff Zou (2020). "Performance evaluation of TPC-C benchmark on various cloud providers". In: *2020 11th IEEE Annual Ubiq-*

uitous Computing, Electronics & Mobile Communication Conference (UEMCON). IEEE, pp. 0226–0233.

Baldi, Pierre (2012). "Autoencoders, unsupervised learning, and deep architectures". In: *Proceedings of ICML workshop on unsupervised and transfer learning*, pp. 37–49.

Banker, Kyle et al. (2016). *MongoDB in action: covers MongoDB version 3.0*. Simon and Schuster.

Baresi, Luciano et al. (2016). "A discrete-time feedback controller for containerized cloud applications". In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 217–228.

Barino, Felipe O et al. (2020). "Correlated time-series in multi-day-ahead streamflow forecasting using convolutional networks". In: *IEEE Access* 8, pp. 215748–215757.

Barrett, Enda, Enda Howley, and Jim Duggan (2013). "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud". In: *Concurrency and computation: practice and experience* 25.12, pp. 1656–1674.

Bello, Yahuza et al. (2021). "B5G: Predictive container auto-scaling for cellular evolved packet core". In: *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, pp. 1–7.

Berg, Kristi L, Tom Seymour, Richa Goel, et al. (2013). "History of databases". In: *International Journal of Management & Information Systems (IJMIS)* 17.1, pp. 29–36.

Berndt, Donald J and James Clifford (1994). "Using dynamic time warping to find patterns in time series." In: *KDD workshop*. Vol. 10. 16. Seattle, WA, USA: pp. 359–370.

Bernstein, David (2014). "Containers and cloud: From lxc to docker to kubernetes". In: *IEEE cloud computing* 1.3, pp. 81–84.

Bertero, Christophe et al. (2017). "Experience report: Log mining using natural language processing and application to anomaly detection". In: *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, pp. 351–360.

Bishop, Christopher M and Nasser M Nasrabadi (2006). *Pattern recognition and machine learning*. Vol. 4. 4. Springer.

Biswas, Anshuman et al. (2014). "Automatic resource provisioning: a machine learning based proactive approach". In: *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*. IEEE, pp. 168–173.

Borovykh, Anastasia, Sander Bohte, and Cornelis W Oosterlee (2017). "Conditional time series forecasting with convolutional neural networks". In: *arXiv preprint arXiv:1703.04691*.

Bre, Facundo, Juan M Gimenez, and Vıctor D Fachinotti (2018). "Prediction of wind pressure coefficients on building surfaces using artificial neural networks". In: *Energy and Buildings* 158, pp. 1429–1441.

Brewer, Eric A (2000). "Towards robust distributed systems". In: *PODC*. Vol. 7. 10.1145. Portland, OR, pp. 343477–343502.

Bruneo, Dario (2013). "A stochastic model to investigate data center performance and QoS in IaaS cloud computing systems". In: *IEEE Transactions on Parallel and Distributed Systems* 25.3, pp. 560–569.

Burckhardt, Sebastian (2014). "Principles of eventual consistency". In.

Buyya, Rajkumar, James Broberg, and Andrzej M Goscinski (2010). *Cloud computing: Principles and paradigms*. John Wiley & Sons.

Caruana, Rich, Steve Lawrence, and C Lee Giles (2001). "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping". In: *Advances in neural information processing systems*, pp. 402–408.

Chaisiri, Sivadon, Bu-Sung Lee, and Dusit Niyato (2011). "Optimization of resource provisioning cost in cloud computing". In: *IEEE transactions on services Computing* 5.2, pp. 164–177.

Chandola, Varun, Arindam Banerjee, and Vipin Kumar (2009). "Anomaly detection: A survey". In: *ACM computing surveys (CSUR)* 41.3, pp. 1–58.

Chang, Fay et al. (2008). "Bigtable: A distributed storage system for structured data". In: *ACM Transactions on Computer Systems (TOCS)* 26.2, pp. 1–26.

Chawla, Nitesh V (2009). "Data mining for imbalanced datasets: An overview". In: *Data mining and knowledge discovery handbook*, pp. 875–886.

Cheong, Seung-Kook, Cheol Su Lim, and Byung-Cheol Cho (2012). "Database processing performance and energy efficiency evaluation of DDR-SSD and

HDD storage system based on the TPC-C". In: *2012 International Conference on Cloud Computing and Social Networking (ICCCSN)*. IEEE, pp. 1–3.

Chiang, Yi-Ju, Yen-Chieh Ouyang, and Ching-Hsien Robert Hsu (2014). "An efficient green control algorithm in cloud computing for cost optimization". In: *IEEE Transactions on Cloud Computing* 3.2, pp. 145–155.

Chouliaras, Spyridon and Stelios Sotiriadis (2019). "Real-time anomaly detection of NoSQL systems based on resource usage monitoring". In: *IEEE Transactions on Industrial Informatics* 16.9, pp. 6042–6049.

— (2021). *SONA Dataset*. URL: `https://dx.doi.org/10.21227/azrd-v466`.

— (2022). "Auto-scaling containerized cloud applications: A workload-driven approach". In: *Simulation Modelling Practice and Theory* 121, p. 102654.

Codd, Edgar F (1980). "Data models in database management". In: *Proceedings of the 1980 workshop on Data abstraction, databases and conceptual modeling*, pp. 112–114.

— (2002). "A relational model of data for large shared data banks". In: *Software pioneers*. Springer, pp. 263–294.

Comellas, Josep Oriol Fitó, Inigo Goiri Presa, and Jordi Guitart Fernández (2010). "SLA-driven elastic cloud hosting provider". In: *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. IEEE, pp. 111–118.

Cooper, Brian F et al. (2010). "Benchmarking cloud serving systems with YCSB". In: *Proceedings of the 1st ACM symposium on Cloud computing*, pp. 143–154.

Damousis, Ioannis G, Anastasios G Bakirtzis, and Petros S Dokopoulos (2003). "Network-constrained economic dispatch using real-coded genetic algorithm". In: *IEEE Transactions on power systems* 18.1, pp. 198–205.

Deep, Kusum et al. (2009). "A real coded genetic algorithm for solving integer and mixed integer optimization problems". In: *Applied Mathematics and Computation* 212.2, pp. 505–518.

Dhingra, Mohit, J Lakshmi, and SK Nandy (2012). "Resource usage monitoring in clouds". In: *2012 ACM/IEEE 13th International Conference on Grid Computing*. IEEE, pp. 184–191.

Al-Dhuraibi, Yahya et al. (2017). “Autonomic vertical elasticity of docker containers with elasticdocker”. In: *2017 IEEE 10th international conference on cloud computing (CLOUD)*. IEEE, pp. 472–479.

Dong, Xishuang, Lijun Qian, and Lei Huang (2017). “Short-term load forecasting in smart grid: A combined CNN and K-means clustering approach”. In: *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE, pp. 119–125.

Dumitrescu, Dumitru et al. (2000). *Evolutionary computation*. CRC press.

Elmasri, R et al. (2000). *Fundamentals of Database Systems¡/Title*. Springer.

Eskin, Eleazar et al. (2002). “A geometric framework for unsupervised anomaly detection”. In: *Applications of data mining in computer security*. Springer, pp. 77–101.

Faragardi, Hamid Reza et al. (2019). “GRP-HEFT: A budget-constrained resource provisioning scheme for workflow scheduling in IaaS clouds”. In: *IEEE Transactions on Parallel and Distributed Systems* 31.6, pp. 1239–1254.

Felter, Wes et al. (2015). “An updated performance comparison of virtual machines and linux containers”. In: *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, pp. 171–172.

Feng, Guofu et al. (2012). “Revenue maximization using adaptive resource provisioning in cloud computing environments”. In: *2012 ACM/IEEE 13th International Conference on Grid Computing*. IEEE, pp. 192–200.

Gandhi, Anshul et al. (2016). “Autoscaling for hadoop clusters”. In: *2016 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, pp. 109–118.

Gen, Mitsuo and Runwei Cheng (1996). “A survey of penalty techniques in genetic algorithms”. In: *Proceedings of IEEE International Conference on Evolutionary Computation*. IEEE, pp. 804–809.

George, Lars (2011). *HBase: the definitive guide: random access to your planet-size data*. ” O’Reilly Media, Inc.”.

Gers, Felix A, Jürgen Schmidhuber, and Fred Cummins (2000). “Learning to forget: Continual prediction with LSTM”. In: *Neural computation* 12.10, pp. 2451–2471.

Goldberg, David E (2006). *Genetic algorithms*. Pearson Education India.

Golshani, Ehsan and Mehrdad Ashtiani (2021). "Proactive auto-scaling for cloud environments using temporal convolutional neural networks". In: *Journal of Parallel and Distributed Computing* 154, pp. 119–141.

Gong, Zhenhuan, Xiaohui Gu, and John Wilkes (2010). "Press: Predictive elastic resource scaling for cloud systems". In: *2010 International Conference on Network and Service Management*. Ieee, pp. 9–16.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep learning*. MIT press.

Google (2022a). *Google Cloud Platform*. URL: https://cloud.google.com/ (visited on 09/13/2022).

— (2022b). *Google Cloud Platform Compute Engine*. URL: https://cloud.google.com/compute (visited on 09/13/2022).

— (2022c). *Google Cloud Platform Compute Engine autoscaler*. URL: https://cloud.google.com/compute/docs/autoscaler (visited on 07/14/2022).

— (2022d). *Google Cloud Platform Geography and Regions*. URL: https://cloud.google.com/docs/geography-and-regions (visited on 09/13/2022).

— (2022e). *Google, "Google cloud pricing calculator"*. URL: https://cloud.google.com/products/calculator/ (visited on 09/17/2022).

Görnitz, Nico et al. (2013). "Toward supervised anomaly detection". In: *Journal of Artificial Intelligence Research* 46, pp. 235–262.

Grossman, Robert L (2009). "The case for cloud computing". In: *IT professional* 11.2, pp. 23–27.

Gudivada, Venkat N, Dhana Rao, and Vijay V Raghavan (2014). "NoSQL systems for big data management". In: *2014 IEEE World congress on services*. IEEE, pp. 190–197.

Gulisano, Vincenzo et al. (2012). "Streamcloud: An elastic and scalable data streaming system". In: *IEEE Transactions on Parallel and Distributed Systems* 23.12, pp. 2351–2365.

Guo, Chonghui, Hongfeng Jia, and Na Zhang (2008). "Time series clustering based on ICA for stock data analysis". In: *2008 4th International Conference on Wireless Communications, Networking and Mobile Computing*. IEEE, pp. 1–4.

Gupta, Lalit et al. (1996). “Nonlinear alignment and averaging for estimating the evoked potential”. In: *IEEE transactions on biomedical engineering* 43.4, pp. 348–356.

Győrödi, Cornelia et al. (2015). “A comparative study: MongoDB vs. MySQL”. In: *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*. IEEE, pp. 1–6.

Haerder, Theo and Andreas Reuter (1983). “Principles of transaction-oriented database recovery”. In: *ACM computing surveys (CSUR)* 15.4, pp. 287–317.

Halpin, Terry and Tony Morgan (2010). *Information modeling and relational databases*. Morgan Kaufmann.

Han, Dan and Eleni Stroulia (2013). “Hgrid: A data model for large geospatial data sets in hbase”. In: *2013 IEEE sixth international conference on cloud computing*. IEEE, pp. 910–917.

Han, Rui et al. (2012). “Lightweight resource scaling for cloud applications”. In: *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. IEEE, pp. 644–651.

HBase (2022). *HBase Documentation*. URL: https://hbase.apache.org/ (visited on 09/16/2022).

He, Kaiming et al. (2015). “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034.

Hijazi, Samer, Rishi Kumar, Chris Rowen, et al. (2015). “Using convolutional neural networks for image recognition”. In: *Cadence Design Systems Inc.: San Jose, CA, USA* 9.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). “Long short-term memory”. In: *Neural computation* 9.8, pp. 1735–1780.

Holland, John H (1992a). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

— (1992b). “Genetic algorithms”. In: *Scientific american* 267.1, pp. 66–73.

Horovitz, Shay and Yair Arian (2018). “Efficient cloud auto-scaling with SLA objective using Q-learning”. In: *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE, pp. 85–92.

HPA (2022). *Kubernetes Horizontal Pod Autoscaler (HPA)*. URL: `https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/` (visited on 09/17/2022).

Huang, Gaopan et al. (2016). "Auto scaling virtual machines for web applications with queueing theory". In: *2016 3rd International Conference on Systems and Informatics (ICSAI)*. IEEE, pp. 433–438.

Imdoukh, Mahmoud, Imtiaz Ahmad, and Mohammad Gh Alfailakawi (2020). "Machine learning-based auto-scaling for containerized applications". In: *Neural Computing and Applications* 32.13, pp. 9745–9760.

Iqbal, Waheed, Abdelkarim Erradi, and Arif Mahmood (2018). "Dynamic workload patterns prediction for proactive auto-scaling of web applications". In: *Journal of Network and Computer Applications* 124, pp. 94–107.

Islam, Sadeka et al. (2012). "Empirical prediction models for adaptive resource provisioning in the cloud". In: *Future Generation Computer Systems* 28.1, pp. 155–162.

Jain, Anil K, Jianchang Mao, and K Moidin Mohiuddin (1996). "Artificial neural networks: A tutorial". In: *Computer* 29.3, pp. 31–44.

James, Gareth et al. (2013). *An introduction to statistical learning*. Vol. 112. Springer.

Jiang, Jing et al. (2013). "Optimal cloud resource auto-scaling for web applications". In: *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. IEEE, pp. 58–65.

Kächele, Steffen et al. (2013). "Beyond IaaS and PaaS: An extended cloud taxonomy for computation, storage and networking". In: *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*. IEEE, pp. 75–82.

Kamsky, Asya (2019). "Adapting TPC-C benchmark to measure performance of multi-document transactions in MongoDB". In: *Proceedings of the VLDB Endowment* 12.12, pp. 2254–2262.

Kan, Chuanqi (2016). "DoCloud: An elastic cloud platform for Web applications based on Docker". In: *2016 18th international conference on advanced communication technology (ICACT)*. IEEE, pp. 478–483.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Klinaku, Floriment, Markus Frank, and Steffen Becker (2018). “CAUS: an elasticity controller for a containerized microservice”. In: *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, pp. 93–98.

Koprinska, Irena, Dengsong Wu, and Zheng Wang (2018). “Convolutional neural networks for energy time series forecasting”. In: *2018 international joint conference on neural networks (IJCNN)*. IEEE, pp. 1–8.

Kubernetes (2022). *Kubernetes*. URL: `https://kubernetes.io/` (visited on 09/17/2022).

Larose, Daniel T and Chantal D Larose (2014). *Discovering knowledge in data: an introduction to data mining*. Vol. 4. John Wiley & Sons.

Leutenegger, Scott T and Daniel Dias (1993). “A modeling study of the TPC-C benchmark”. In: *ACM Sigmod Record* 22.2, pp. 22–31.

Li, Linze and Jun Zhang (2021). “Research and analysis of an enterprise E-commerce marketing system under the big data environment”. In: *Journal of Organizational and End User Computing (JOEUC)* 33.6, pp. 1–19.

Lim, Harold C, Shivnath Babu, and Jeffrey S Chase (2010). “Automated control for elastic storage”. In: *Proceedings of the 7th international conference on Autonomic computing*, pp. 1–10.

Lin, Shuyu et al. (2020). “Anomaly detection for time series using vae-lstm hybrid model”. In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Ieee, pp. 4322–4326.

Longari, Stefano et al. (2020). “CANnolo: An Anomaly Detection System based on LSTM Autoencoders for Controller Area Network”. In: *IEEE Transactions on Network and Service Management*.

Lorido-Botran, Tania, Jose Miguel-Alonso, and Jose A Lozano (2014). “A review of auto-scaling techniques for elastic applications in cloud environments”. In: *Journal of grid computing* 12.4, pp. 559–592.

Lu, Hongjun, Hock Chuan Chan, and Kwok Kee Wei (1993). “A Survey on Usage of SQL”. In: *ACM SIGMOD Record* 22.4, pp. 60–65.

Maas, Andrew L, Awni Y Hannun, and Andrew Y Ng (2013). “Rectifier nonlinearities improve neural network acoustic models”. In: *Proc. icml*. Vol. 30. 1, p. 3.

Macedo, Tiago and Fred Oliveira (2011). *Redis cookbook: Practical techniques for fast data manipulation.* " O'Reilly Media, Inc.".

Malhotra, Pankaj et al. (2016). "LSTM-based encoder-decoder for multi-sensor anomaly detection". In: *arXiv preprint arXiv:1607.00148*.

Mao, Ming, Jie Li, and Marty Humphrey (2010). "Cloud auto-scaling with deadline and budget constraints". In: *2010 11th IEEE/ACM International Conference on Grid Computing*. IEEE, pp. 41–48.

Marie-Magdelaine, Nicolas and Toufik Ahmed (2020). "Proactive Autoscaling for Cloud-Native Applications using Machine Learning". In: *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, pp. 1–7.

Marinescu, Dan C (2022). *Cloud computing: theory and practice*. Morgan Kaufmann.

Maurer, Michael et al. (2011). "Revealing the MAPE loop for the autonomic management of cloud infrastructures". In: *2011 IEEE symposium on computers and communications (ISCC)*. IEEE, pp. 147–152.

McCulloch, Warren S and Walter Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4, pp. 115–133.

Mei, Yiduo et al. (2010). "Performance measurements and analysis of network i/o applications in virtualized cloud". In: *2010 IEEE 3rd International Conference on Cloud Computing*. IEEE, pp. 59–66.

Merkel, Dirk et al. (2014). "Docker: lightweight linux containers for consistent development and deployment". In: *Linux j* 239.2, p. 2.

Michalewicz, Zbigniew (1996a). "Evolutionary computation: practical issues". In: *Proceedings of IEEE International Conference on Evolutionary Computation*. IEEE, pp. 30–39.

— (1996b). "Heuristic methods for evolutionary computation techniques". In: *Journal of Heuristics* 1.2, pp. 177–206.

Mireslami, Seyedehmehrnaz et al. (2021). "Dynamic Cloud Resource Allocation Considering Demand Uncertainty". In: *IEEE Transactions on Cloud Computing* 9.3, pp. 981–994.

Mitchell, Tom M (1997). *Machine learning*. Vol. 1. 9. McGraw-hill New York.

Mohan, Aravind et al. (2016). “Scheduling big data workflows in the cloud under budget constraints”. In: *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, pp. 2775–2784.

MongoDB (2022a). *MongoDB Customers*. URL: `https://www.mongodb.com/who-uses-mongodb` (visited on 09/13/2022).

— (2022b). *MongoDB Documentation*. URL: `https://www.mongodb.com/docs/` (visited on 09/13/2022).

Mosberger, David and Tai Jin (1998). “httperf—a tool for measuring web server performance”. In: *ACM SIGMETRICS Performance Evaluation Review* 26.3, pp. 31–37.

Murphy, Kevin P (2012). *Machine learning: a probabilistic perspective*. MIT press.

MySQL (2022a). *MySQL Customers*. URL: `https://www.mysql.com/customers/` (visited on 09/13/2022).

— (2022b). *MySQL Documentation*. URL: `https://dev.mysql.com/doc/` (visited on 09/13/2022).

Nair, Vinod and Geoffrey E Hinton (2010). “Rectified linear units improve restricted boltzmann machines”. In: *Icml*.

Negnevitsky, Michael (2005). *Artificial intelligence: a guide to intelligent systems*. Pearson education.

Nguyen, HD et al. (2020). “Forecasting and Anomaly Detection approaches using LSTM and LSTM Autoencoder techniques with the applications in supply chain management”. In: *International Journal of Information Management*, p. 102282.

Nicodemus, Carlos HZ, Cristina Boeres, and Vinod EF Rebello (2020). “Managing vertical memory elasticity in containers”. In: *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. IEEE, pp. 132–142.

Niennattrakul, Vit and Chotirat Ann Ratanamahatana (2007). “On clustering multimedia time series data using k-means and dynamic time warping”. In: *2007 International Conference on Multimedia and Ubiquitous Engineering (MUE’07)*. IEEE, pp. 733–738.

— (2009). “Shape averaging under time warping”. In: *2009 6th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*. Vol. 2. IEEE, pp. 626–629.

Nikravesh, Ali Yadavar, Samuel A Ajila, and Chung-Horng Lung (2015). “Towards an autonomic auto-scaling prediction system for cloud resource provisioning”. In: *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, pp. 35–45.

Padala, Pradeep et al. (2009). “Automated control of multiple virtualized resources”. In: *Proceedings of the 4th ACM European conference on Computer systems*, pp. 13–26.

Pang, Guansong et al. (2021). “Deep learning for anomaly detection: A review”. In: *ACM Computing Surveys (CSUR)* 54.2, pp. 1–38.

Paraiso, Fawaz et al. (2016). “Model-driven management of docker containers”. In: *2016 IEEE 9th International Conference on cloud Computing (CLOUD)*. IEEE, pp. 718–725.

Petitjean, François, Alain Ketterlin, and Pierre Gançarski (2011). “A global averaging method for dynamic time warping, with applications to clustering”. In: *Pattern recognition* 44.3, pp. 678–693.

Poess, Meikel and Raghunath Othayoth Nambiar (2008). “Energy cost, the key challenge of today’s data centers: a power consumption analysis of TPC-C results”. In: *Proceedings of the VLDB Endowment* 1.2, pp. 1229–1240.

Pramukantoro, Eko Sakti, Dany Primanita Kartikasari, and Reza Andria Siregar (2019). “Performance evaluation of MongoDB, cassandra, and HBase for heterogenous IoT data storage”. In: *2019 2nd International Conference on Applied Information Technology and Innovation (ICAITI)*. IEEE, pp. 203–206.

Ramachandran, Prajit, Barret Zoph, and Quoc V Le (2017). “Searching for activation functions”. In: *arXiv preprint arXiv:1710.05941*.

Rao, Jia et al. (2009). “VCONF: a reinforcement learning approach to virtual machines auto-configuration”. In: *Proceedings of the 6th international conference on Autonomic computing*, pp. 137–146.

Raschka, Sebastian (2015). *Python machine learning*. Packt publishing ltd.

Redis (2022). *Redis Documentation*. URL: `https://redis.io/docs/` (visited on 09/19/2022).

Rosenblatt, Frank (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, p. 386.

Roy, Nilabja, Abhishek Dubey, and Aniruddha Gokhale (2011). "Efficient autoscaling in the cloud using predictive models for workload forecasting". In: *2011 IEEE 4th International Conference on Cloud Computing*. IEEE, pp. 500–507.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1985). *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science.

Sak, Hasim, Andrew W Senior, and Françoise Beaufays (2014). "Long short-term memory recurrent neural network architectures for large scale acoustic modeling". In.

Sakellariou, Rizos et al. (2007). "Scheduling workflows with budget constraints". In: *Integrated research in GRID computing*. Springer, pp. 189–202.

Schuler, Lucia, Somaya Jamil, and Niklas Kühl (2021). "AI-based resource allocation: Reinforcement learning for adaptive auto-scaling in serverless environments". In: *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, pp. 804–811.

Sharma, Prateek et al. (2016). "Containers and virtual machines at scale: A comparative study". In: *Proceedings of the 17th international middleware conference*, pp. 1–13.

Shen, Changyu, Lixia Wang, and Qian Li (2007). "Optimization of injection molding process parameters using combination of artificial neural network and genetic algorithm method". In: *Journal of materials processing technology* 183.2-3, pp. 412–418.

Silberschatz, Avi, Henry F Korth, and S Sudarshan (1996). "Data models". In: *ACM Computing Surveys (CSUR)* 28.1, pp. 105–108.

Singh, Sukhpal, Inderveer Chana, and Rajkumar Buyya (2017). "STAR: SLA-aware autonomic management of cloud resources". In: *IEEE Transactions on Cloud Computing*.

Siow, Eugene, Thanassis Tiropanis, and Wendy Hall (2018). “Analytics for the internet of things: A survey”. In: *ACM computing surveys (CSUR)* 51.4, pp. 1–36.

Sivanandam, SN and SN Deepa (2008). “Genetic algorithms”. In: *Introduction to genetic algorithms*. Springer, pp. 15–37.

Sola, J. and J. Sevilla (1997). “Importance of input data normalization for the application of neural networks to complex industrial problems”. In: *IEEE Transactions on Nuclear Science* 44.3, pp. 1464–1468.

Song, Xiuyao et al. (2007). “Conditional anomaly detection”. In: *IEEE Transactions on knowledge and Data Engineering* 19.5, pp. 631–645.

Sotiriadis, Stelios (2021). “The inter-cloud meta-scheduling framework”. PhD thesis. University of Derby (United Kingdom).

Sotiriadis, Stelios et al. (2016). “Elastic load balancing for dynamic virtual machine reconfiguration based on vertical and horizontal scaling”. In: *IEEE Transactions on Services Computing* 12.2, pp. 319–334.

Stavrinides, Georgios L and Helen D Karatza (2019). “Performance evaluation of a SaaS cloud under different levels of workload computational demand variability and tardiness bounds”. In: *Simulation Modelling Practice and Theory* 91, pp. 1–12.

Strickberger, Monroe W (2000). *Evolution*. Jones & Bartlett Learning.

Sumathi, Sai and S Esakkirajan (2007). *Fundamentals of relational database management systems*. Vol. 47. Springer.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Tan, Pang-Ning, Michael Steinbach, and Vipin Kumar (2016). *Introduction to data mining*. Pearson Education India.

Taylor, Ronald C (2010). “An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics”. In: *BMC bioinformatics* 11.12, pp. 1–6.

TPC (2022a). *TPC*. URL: `https://www.tpc.org/tpcc/` (visited on 09/19/2022).

— (2022b). *TPC*. URL: `https://www.tpc.org/tpcx-iot/default5.asp` (visited on 09/19/2022).

Trinh, Hoang Duy et al. (2019). “Detecting mobile traffic anomalies through physical control channel fingerprinting: A deep semi-supervised approach”. In: *IEEE Access* 7, pp. 152187–152201.

Vora, Mehul Nalin (2011). “Hadoop-HBase for large-scale data”. In: *Proceedings of 2011 International Conference on Computer Science and Network Technology*. Vol. 1. IEEE, pp. 601–605.

VPA (2022). *Kubernetes Vertical Pod Autoscaler (HPA)*. URL: `https://github.com/kubernetes/autoscaler/tree/master/vertical-pod-autoscaler/` (visited on 09/17/2022).

Williams, Ronald J and David Zipser (1989). “A learning algorithm for continually running fully recurrent neural networks”. In: *Neural computation* 1.2, pp. 270–280.

Yu, Jia and Rajkumar Buyya (2006). “A budget constrained scheduling of workflow applications on utility grids using genetic algorithms”. In: *2006 Workshop on Workflows in Support of Large-Scale Science*. IEEE, pp. 1–10.

Zhang, Fan et al. (2019). “Quantifying cloud elasticity with container-based autoscaling”. In: *Future Generation Computer Systems* 98, pp. 672–681.

Zhao, Liang, Sherif Sakr, and Anna Liu (2013). “A framework for consumer-centric SLA management of cloud-hosted databases”. In: *IEEE Transactions on Services Computing* 8.4, pp. 534–549.

Zhao, Siqin, Kang Chen, and Weimin Zheng (2009). “Defend against denial of service attack with VMM”. In: *2009 Eighth International Conference on Grid and Cooperative Computing*. IEEE, pp. 91–96.

Zheng, Yu, Huichu Zhang, and Yong Yu (2015). “Detecting collective anomalies from multiple spatio-temporal datasets across different domains”. In: *Proceedings of the 23rd SIGSPATIAL international conference on advances in geographic information systems*, pp. 1–10.

Zhu, Qian and Gagan Agrawal (2012). “Resource provisioning with budget constraints for adaptive applications in cloud environments”. In: *IEEE Transactions on Services Computing* 5.4, pp. 497–511.