



BIROn - Birkbeck Institutional Research Online

Sotiriadis, Stelios and Bessis, N. (2016) An inter-cloud bridge system for heterogeneous cloud platforms. *Future Generation Computer Systems* 54 , 180 - 194. ISSN 0167-739X.

Downloaded from: <https://eprints.bbk.ac.uk/id/eprint/21790/>

Usage Guidelines:

Please refer to usage guidelines at <https://eprints.bbk.ac.uk/policies.html>
contact lib-eprints@bbk.ac.uk.

or alternatively

Accepted Manuscript

An inter-cloud bridge system for heterogeneous cloud platforms

Stelios Sotiriadis, Nik Bessis

PII: S0167-739X(15)00040-0

DOI: <http://dx.doi.org/10.1016/j.future.2015.02.005>

Reference: FUTURE 2713

To appear in: *Future Generation Computer Systems*

Received date: 29 September 2014

Revised date: 5 February 2015

Accepted date: 11 February 2015



Please cite this article as: S. Sotiriadis, N. Bessis, An inter-cloud bridge system for heterogeneous cloud platforms, *Future Generation Computer Systems* (2015), <http://dx.doi.org/10.1016/j.future.2015.02.005>

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

An Inter-Cloud Bridge System for Heterogeneous Cloud Platforms

Stelios Sotiriadis

*Department of Electronic & Computer Engineering,
Technical University of Crete (TUC), University Campus, Chania,
Crete, GR-73100, Greece,
e-mail: s.sotiriadis@intelligence.tuc.gr*

Nik Bessis

*Department of Computing & Maths, University of Derby,
Kedleston Road, Derby DE22-1GB, UK,
e-mail: n.bessis@derby.ac.uk*

Abstract

Over the years, more cloud computing systems have been developed providing flexible interfaces for inter-cloud interaction. This work approaches the concept of inter-cloud by utilizing APIs, open source specifications and exposed interfaces from cloud platforms such as OpenStack, OpenNebula and others. Despite other works in the area of inter-cloud, that are mainly resource management-centric, we focus on designing and developing a service-centric architecture. We implement an inter-cloud bridge system that is elastic, easy to be upgraded and managed. We develop a prototype composed not only from heterogeneous cloud platforms but also from independent cloud services. These are developed by different cloud service providers and offered as open source Software as a Service (SaaS). The proposed Inter-Cloud Mediation Service uses Future Internet SaaS such as a Context Broker for registrations and subscriptions to services and a Complex Event Processing engine for event management. We present an experimental analysis to show interactions with various heterogeneous cloud platforms and we evaluate the performance of inter-cloud services separately and as a whole.

Keywords: Cloud Computing, Inter-Cloud, OpenStack, FI-LAB, Heterogeneous Inter-Cloud, Cloud Interoperability

1. Introduction

Cloud computing provide a computer-based environment where various services and applications are available to users through the public Internet. Various cloud platforms such as Amazon Web Services (WS), OpenStack and others, provision services to clients on a pay-on-demand model. Recently, the emergence of the Future Internet (FI) concept has promoted the so-called modular cloud service developments that is derived from of the Service Oriented Architecture as in Erl (2005), yet in the area of cloud computing. In particular, this is based on the need to decompose a system in small and easily to control cloud services of basic functionalities (e.g. user authentication) called Generic Enablers (GEs). Each GE is a software block offered as cloud service following by an open specification and flexible API as introduced by the FI-WARE project¹. FI-WARE provides a cloud platform (named as FI-LAB) based on the Datacentre Resource Management System (DCRM) GE² and offers interfaces for future developments (e.g. for developing an inter-cloud). In this platform various cloud service providers deploy and offer cloud services in the context of the FI-PPP programme³.

The work does not aim to alter internal cloud platform processes and possible adaptations that need to be made by cloud providers, as most of the related approaches propose, but on the interfaces and APIs as enablers for remote management of inter-cloud services in a unified manner. Further, the FI era emerges new challenges by building FI applications from services belonging to different providers, thus highlighting a crucial requirement for inter-clouds. We propose an Inter-Cloud Mediation Service by extending the work in Sotiriadis et al. (2014) to involve public clouds forming a collaborative environment for distribution and common management of cloud services using the RESTful protocol as discussed in Schreier (2011). This interoperability vision between heterogeneous resource providers based on open standards in the general concept of the Open Cloud Computing Interface (OCCI)⁴. Lately, it is used widely e.g. by OpenStack and OpenNebula⁵.

¹FI-WARE Project: <http://www.fi-ware.org/lab/>

²FI-WARE DCRM Architecture: <https://forge.fi-ware.org/plugins/mediawiki/wiki/fi-ware/index.php/FIWARE.OpenSpecification.Cloud.DCRM>

³FI-PPP programme: <http://www.fi-ppp.eu>

⁴OCCI: <http://occiwg.org>

⁵OpenNebula: <http://opennebula.org>

We are motivated from the opportunities rising from FI-WARE and OpenStack platforms and particularly from the new horizons of the FI concepts as in Sotiriadis et al. (2014), Galis and Gavras (2013) and in Vandenberghe et al. (2013). Thus we define the Inter-Cloud Mediation Service to support communication with OpenStack, FI-LAB, Amazon WS and VCloud⁶ platforms to demonstrate heterogeneity that extends the work in Sotiriadis et al. (2014) presented in Pop and Potop-Butucaru (2014). We utilise available FI-WARE software, also known as Generic Enablers (GEs) that are open source implementations⁷ to build the inter-cloud bridge system including the Publish/Subscribe Context Broker⁸, the Complex Event Processing (CEP)⁹, the Identity Management (IDM)¹⁰ and the Cloud Store¹¹ offered from FI-WARE. Based on this discussion, Section 2 demonstrates the motivation and the related approaches, Section 3 presents the model of the proposed system and, Section 4 the description of the prototype system, Section 5 details the experimental study, and Section 5 the conclusions and future research steps.

2. Related Works

Today, various cloud vendors aimed to an interoperable cloud effort by jointly establishing federations of clouds. In Sotiriadis et al. (2013a), we presented a study on inter-cloud scheduling model a detailed discussion of related solutions as Lucas-Simarro et al. (2013). In Petcu (2014) a discussion is presented to demonstrate various approaches for inter-cloud bridge systems. The mOSAIC¹² FP7 project focuses on Open-source API and platform for Multiple Clouds for cloud-based application developers, maintainers and users in order to specify the service requirements in terms of Cloud ontology and

⁶VMWARE VCloud: MPLA

⁷FI-WARE catalogue: <http://catalogue.fi-ware.org/enablers>

⁸Context Broker specs: <https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSpecification.Data.PubSub>

⁹CEP specs: http://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/Complex_Event_Processing_Open_RESTful_API_Specification

¹⁰IDM specs: <https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSpecification.Security.IdentityManagement>

¹¹Apps Store open specs: <https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSpecification.Apps.Store>

¹²MOSAIS FP7 Project: <http://www.mosaic-cloud.eu>

an API. The Contrail¹³ FP7 project is focused on cloud software stack of components that are designed to work together for one integrated federated cloud. SeaClouds¹⁴ FP7 project performs a seamless adaptive multi-cloud management of service-based applications, by developing a set of tools to manage complex applications, thus avoiding the problem of Cloud lock-in. Other works as in Mauch et al. (2013) aim to explore cloud computing from the perspective of high performance computing. In the past we have presented a study for Inter-Cloud schedulers as in Sotiriadis et al. (2011), a work that demonstrate the requirements for brokers in Inter-Cloud. Here, we focus on developing a solution in the Future Internet concept, thus we utilize tools and services offered from FI-WARE.

FI-WARE provides the a collection of tools that allow deployment of FI-WARE DCRM infrastructure. The later includes a federation mechanism for accessing multiple systems¹⁵ within a single interface, yet resources are kept separately by provider. In this study we aim to overcome the problem of vendor specific inter-clouds by focusing on the OCCI standard¹⁶. This means that cloud systems developers using such standard (FI-WARE, OpenStack, OpenNebula etc.) will be able to utilize their interfaces to join an inter-cloud. In addition to this, the openings arising and the general concept of cloud datacentre resource management systems that emerge by FI-WARE highlight new challenges. In particular, we attempt to build a cloud interoperable common management architecture based OpenStack API that is suitable to provide inter-cloud management service for FI-WARE based clouds that offer GEs. Lately, various FI-PPP programmes (up to 16 EU funded projects) as in Galis and Gavras (2013) have been promoted to accelerate the development and adoption of Future Internet technologies in Europe.

The experimental demonstration of the architecture is executed in the intellicloud infrastructure of the Technical University of Crete (TUC) and could be offered as a GE service. Intellicloud is an experimental cloud infrastructure for designing cloud-based Internet applications. It is an open testbed environment for researchers and developers that aim to design and deploy cloud based FI applications. The infrastructure is based on OpenStack (an open source cloud computing platform) and offers infrastructural

¹³Contrail FP7 Project: <http://contrail-project.eu>

¹⁴SeaClouds FP7 Project: <http://www.seaclouds-project.eu>

¹⁵XI-FI FP7 Project: <https://www.fi-xifi.eu/fiware-ops.html>

¹⁶OCCI: <http://occi-wg.org>

and platform services that include pre-configured FI-WARE GEs and deployment of Specific Enablers (SEs) for exploration as in Vázquez et al. (2011). SEs are specific enablers that are integrated based on GE functionalities. We utilized various GEs and services to integrate the Inter-Cloud environment motivated by the area of business intelligence in the cloud as in Chang (2014) that highlights new opportunities for innovative services. We anticipate that the Inter-Cloud will spark creation of new services by decoupling users from the actual resources, and allow providers to integrate new resources and services in a seamless manner.

3. Modelling the Inter-Cloud Bridge System

This section defines the model of the inter-cloud bridge system along with the related services. It encompasses an architecture based on cloud platforms that offer APIs like OpenStack¹⁷ which is an open source platform for managing large-scale physical servers in a cloud-computing environment. This includes the transformation of the physical resources to virtual that could be delivered over the Internet as Virtual Machines (VMs). OpenStack offers also a reference API described in Fifield et al. (2014) that works with RESTful protocol and provides remote access to cloud services using the HTTP protocol. In detail, services can make calls to get, post, put or delete actions.

The proposed inter-cloud bridge system is designed on a modular basis of interacting RESTful compliant cloud services. Each module is implemented as a separate service under a different endpoint. This highlights significant advantages including comprehensive structure, easy replacement of modules, flexible configuration of the system according to the needs of the inter-cloud administrator and vendor lock-in minimisation as described in Schreier (2011). Based on that, the next sections present the Inter-Cloud architecture describing the modules of the inter-cloud bridge system (in Section 3.1), the Core Cloud API to map a typical cloud API offerings to the inter-cloud service capabilities (in Section 3.2) and finally detailing the implemented input/output interfaces of each inter-cloud module (in Section 3.3).

¹⁷OpenStack: <http://www.openstack.org>

3.1. The Inter-Cloud Bridge System Services

The design of services aims to ensure modularity and management of key components that are the building blocks of the inter-cloud bridge system. These are as follows:

- (i) Identity Management Service: Its role is to offer capabilities for user credentials through a common authentication point. The work utilizes the Identity Management - KeyRock FI-WARE open specification¹⁸ provided by FI-WARE. It provides interfaces for user and device authentication and secure connection to networks and services by offering authorisation and trust management mechanisms along with a single point of user profile management.
- (ii) Inter-cloud Subscription Service: Its role is to allow configuration of the data subscribed by cloud platforms to the inter-cloud bridge system. This includes interfaces for data retrieval regarding services and resources and context management (subscribe, unsubscribe, create, update and register context). The work uses the Publish Subscribe Context Broker provided by FI-WARE as an open specification¹⁹.
- (iii) Inter-cloud Registry Service: Its role is to allow users to define essential information such as credentials for cloud connectivity, cloud service endpoints (URLs) and ports. The service allows remote authentication, connection and information retrieval on heterogeneous cloud platforms including OpenStack, FI-WARE DCRM GE and Amazon AWS EC2 using RESTful interfaces.
- (iv) Complex Event Processing Service (CEP): Its role is to offer an integrated platform to support the development, deployment, and maintenance of event-driven applications²⁰. It offers interfaces to the inter-cloud mediation to define rules and patterns to react on certain event flows. The work utilizes the CEP - IBM Proactive Technology Online for defining conditional events as in Magid et al. (2008). A simple example is the automatically observation of increasing traffic in a cloud

¹⁸KeyRock IDM GE: <http://catalogue.fi-ware.org/enablers/identity-management-keyrock>

¹⁹Context Broker open specs: <https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSpecification.Data.PubSub>

²⁰FI-WARE CEP GE: <http://catalogue.fi-ware.org/enablers/complex-event-processing-cep-ibm-proactive-technology-online>

platform through time and the generation of an action event to notify subscribed services. The CEP provides a repository namely as inter-cloud Complex Event Rules Repository, to store definitions, which is a collection of events and conditions.

- (v) Context Broker Service: Its role is to offer a service to manage context data along with data availability. The component offers interfaces to register and manage context data (elements and attributes) to any interested party (applications or services) by using a subscription operation. For instance, a Mediation Service is subscribed to Context Broker in order to get notifications on time interval. This service provides subscription services, while CEP provides conditional events. The inter-cloud bridge system utilizes the Orion Context Broker²¹ for implanting this service.
- (vi) Mediation Service: Its role is to act as the core inter-cloud orchestration service including interfaces to cloud platforms, information retrieval for user data (images, instances, flavors as available computational templates, resource usage etc.). It also includes interfaces for the Context Broker, the CEP, the Storage and the Cloud Store service.
- (vii) Storage Service: Its role is to act as the administration module of NoSQL databases (MongoDB²²), e.g. raw data that are collected from heterogeneous cloud platforms using RESTful calls. The data are in JavaScript Object Notation (JSON) format. It allows saving JSON storage objects and to retrieve, remove, update or perform queries on JSON data using simple HTTP methods.
- (viii) Inter-cloud Market (Store API): Its role is to offer capabilities for accounting management, billing and publishing applications into a market place platform. In this work we use the Store - WStore ²³, that offers an API for publishing offerings and resources. By utilizing the administration API the Mediation Service could configure repositories and add market places for users. Further it allows versioning and rating of offered services.

²¹Orion Context Broker: <http://catalogue.fi-ware.org/enablers/publishsubscribe-context-broker-orion-context-broker>

²²MongoDB: <http://www.mongodb.org>

²³WStore GE: <http://catalogue.fi-ware.org/enablers/store-wstore>

3.2. Inter-Cloud Mediator: The Core Cloud API Offerings

This section describes APIs that are available from cloud platforms. With regards to the service models, OpenStack provides support for SaaS so consumers can deploy software and access it usually as a web-based service, PaaS to deploy applications through a programming language or tools and IaaS for instances, network connection and storage. OpenStack architecture is described in Corradi et al. (2014). The proposed inter-cloud service provides interfaces to the following APIs²⁴.

- (i) Identity Service: It provides an API as the means for authentication and authorization by offering a service that generates access tokens for other OpenStack services. This includes a catalogue of endpoints for all OpenStack services. Similarly, identical endpoints are defined in order to authorize inter-cloud communication.
- (ii) Image Service: It provides an API for management of images that are ready and pre-installed VMs that usually include operating systems along with some software configurations.
- (iii) Compute Service: It provides an API for managing the whole lifecycle of instances (that are generated images) in an OpenStack environment. Key responsibilities include spawning, scheduling and decommissioning of VMs on demand. Inter-cloud tenants could generate new instances and store them in their preferred location, this could be offered as an option in case of sensitive data storage as described in Sotiriadis et al. (2013b).
- (iv) Network Service: It provides the network capabilities, e.g. to create new virtual networks and routers for network connectivity. Also, it provides the capability to build private networks of VMs and supports many popular networking vendors.
- (v) Orchestration Service: It provides orchestration capabilities for multiple composite cloud applications. It provides a template-driven engine that allows application developers to describe and automate the deployment of infrastructure.
- (vi) Telemetry Service: It provides usage and performance data across the services deployed in an OpenStack clouds. In the inter-cloud will be the means to manage billing, benchmarking, scalability, and statistical purposes.

²⁴OpenStack API reference: <http://developer.openstack.org/api-ref.html>

3.3. Designing the Inter-Cloud Architecture and Interfaces

In the inter-cloud bridge system each component is specified by interfaces for integrating the system communication. The modules required are the (a) user authentication, (b) registration service for definition of the cloud platform, (c) subscription service for context data, (d) context broker service for managing context data, (e) complex event processing for real time event management, (f) inter-cloud mediation to bridge various cloud systems, (g) storage for unstructured data, (h) marketplace for posting offerings. Initially, the users login using the identity management service that provides interfaces to create and delete user, edit details and create a new application²⁵. Figure 1 shows the interfaces of the service.

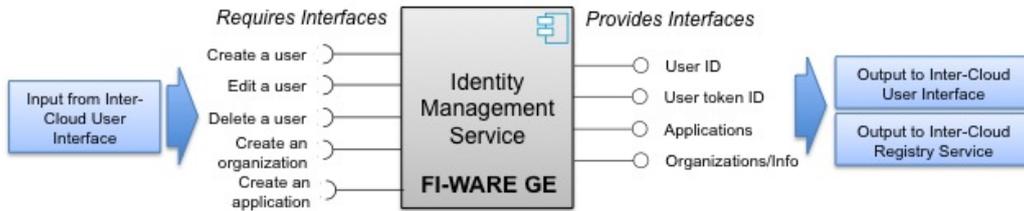


Figure 1: The Interfaces of the Identity Management Service

The registry service collects the data to define cloud platform and to allow remote connections. It offers interfaces for input user data (e.g. credentials and token IDs) and it generates a separate configuration file for each of the clouds. Also, It provides a sanity check test to evaluate given credentials during data input. Figure 2 shows the interfaces of the service.



Figure 2: The Interfaces of the Registry Service

²⁵KeyRock IDM: <http://catalogue.fi-ware.org/enablers/identity-management-keyrock>

The Inter-Cloud Subscription Service defines interfaces for user data along with configurations to subscribe to and unsubscribe from context data. Here the user could select to which data requires notifications, for instance to be notified when a new cloud service (e.g. image or template) is available from a provider. The data input is from the Inter-Cloud Mediation Service. The selected subscriptions are forwarded to the latter, that in response it configures the Context Broker Service according to user requirements. Figure 3 shows the Inter-Cloud Subscription interfaces.



Figure 3: The Interfaces of the User Subscription Service

The Context Broker is responsible for managing context data and availability. It provides interfaces to the Mediation Service in order to input registration, discovery, subscription, update and unsubscribe requests for context data. It allows configuration of conditions (e.g. after passing an interval) to produce notifications in the form of context entities and attributes to the Mediation Service. It uses Next Generation Service Interfaces (NGSI) which is an information model to manage contextualized data including information about entities (such as service in a cloud) or context management (such as which is the provider of the service). In this work, we use it to implement mediation between context producers (e.g. clouds) and the context consumer applications (e.g. the inter-cloud market place or the users that uses the context information). Figure 4 shows the interfaces of the service.

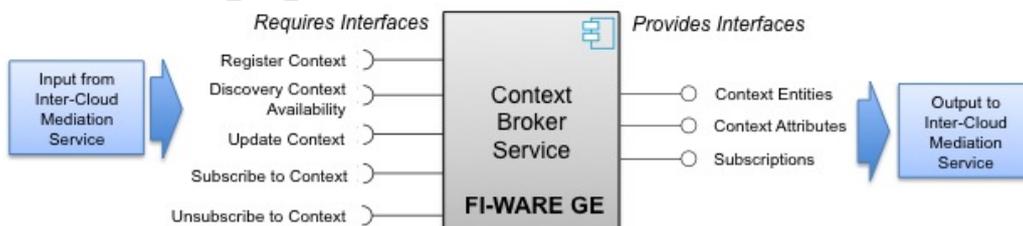


Figure 4: The Interfaces of the Context Broker Service

The CEP Service allows real time event management and generation of new events based on event patterns. The service receives input from and gives output to the Mediation Service. Specifically, the user, (through the service could add, delete and update events and definitions, update the Event engine status and add events using its RESTful interfaces as it provides API interfaces for inter-cloud administrators in order to create new definitions. Figure 5 shows the CEP interfaces.

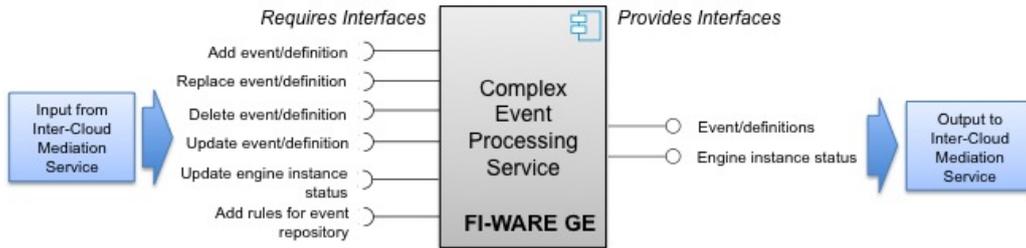


Figure 5: The Interfaces of the Complex Event Processing Service

An example case is the observation of increasing traffic in a cloud provider in real-time and the generation of action events, e.g. to notify user or stakeholders based on a pattern. The service provides an Inter-Cloud Complex Event Rules Repository, to store definitions. The Inter-Cloud Mediation service is the core of the inter-cloud bridge system. It provides input interfaces to the clouds based on the configurations of the user. Figure 6 shows the interfaces of the service.



Figure 6: The Interfaces of the Inter-Cloud Mediation Service

The Inter-Cloud Mediation service retrieves the image lists and information on images (e.g. specifications, containers etc.), user data from the user

interface and from the storage service (e.g. a JSON response of a metric), the networks and related information (e.g. associated floating IP) and the list of servers and information (e.g. CPU, memory etc.). It also provides interfaces for requests to subscribe and un-subscribe context data (connectivity to the Context Broker) and requests for events and definitions (connectivity to the Complex Event Processing). The service outputs to the a) storage service (store JSON objects), b) to the Context Broker (send requests for managing context data), c) Complex Event Processing Service (send requests for events and definitions), d) Marketplace for publishing offers and to the associated stakeholders. The Storage service provides data storage to the inter-cloud mediation service. It particularly utilizes this service in order to keep historic data records. Figure 7 shows the associated interfaces.



Figure 7: The Interfaces of the Storage Service

NoSQL provides a convenient solution for object storage for managing unstructured data (that are usually exported from cloud APIs). It provides input interfaces for add update and delete data (in JSON object format). The Mediation Service retrieves the data and makes it available to other components. The data are divided in two categories that include user data and cloud data. The Mediation Service provides offerings and resources to a common platform where users can access cloud based applications and rate them. Figure 8 shows the service interfaces.



Figure 8: The Interfaces of the Inter-Cloud Marketplace

These are related to repositories, marketplace definitions, organizations, profiles, offerings and resources. The mediation service could post and get all this information in case that the user exports a cloud application implemented in the inter-cloud bridge system. Figure 9 integrates the aforementioned modules of the inter-cloud bridge system. The figure also includes the algorithms and their structure (described in Section 4) regarding their position within the inter-cloud bridge system.

Initially, a user can access the inter-cloud registry using the Identity Management Service. Then, it configures the mediation service using the registry. Then the Mediation Service connects to the configured clouds and collects data regarding generated authentication tokens, images and snapshot, flavors, resource usage information. The collected information is passed to the interacted modules (Context Broker, CEP etc.). Also, the External Third Party Cloud Service includes monitoring of applications and managing composed from services belonging to the inter-cloud bridge system. Further, it provides an endpoint for configuring an inter-cloud market place for subscriptions to inter-cloud offerings.

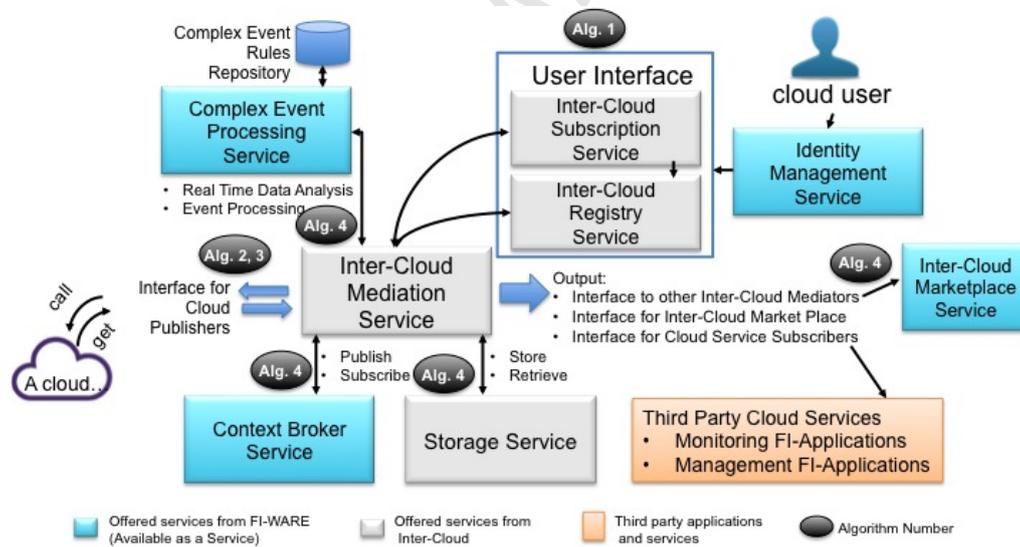


Figure 9: The Inter-Cloud Bridge System modular architecture of associated services

4. Experimental Prototype of the Inter-Cloud Bridge System

This section describes the experimental prototype of the proposed inter-cloud bridge system encompassing the algorithmic structure. This includes the descriptions of RESTful calls, of the tools, services and cloud infrastructures used for implementation and of the prototype solution and the experimental evaluation and results.

4.1. Algorithmic structure of the Inter-Cloud

This section presents the algorithmic structure of the Inter-Cloud modules along with the key operations and interactions. Algorithm 1 shows the user interface modules including authentication using the Identity Management Service and the detail operations of the inter-cloud registry and subscriptions service. In brief, the user configures the registry with data and a series of interactions occur in order to get authentication token and allow management of user subscriptions from the user point.

Algorithm 1: User Interface Module

clouds_count: The number of clouds

i,z: Counters (Integers)

cl_host: The cloud hostname or IP address

port_auth: The cloud authentication port for tokens

port_com: The cloud compute service endpoint port

port_net: The cloud network service endpoint port

t_name: The cloud tenant name

un: The cloud username **pass:** The user password

client_ID: The application client ID

Oauth_URI: The authentication resource location of the identity service

redirect_URI: The redirect location of the identity service

loc_URI: The concatenated location URI of the identity service

client_secret: The client secret ID given by the identity service

user_URI: The user URI of the identity service

response_token: The response token of the user request

user_token: The token ID of the user

sub_id: The token ID of the user

1. **for** clouds_count **to** \leq i **do**
2. i++
3. $\text{conf}_i \leftarrow \{\text{cl_host}, \text{port_auth}, \text{port_com}, \text{t_name}, \text{un}, \text{pass}, \text{port_net}\}$
4. sendMediationServ(conf_{counter}) //send mediation service request

```

5.  end for
6.  loc_URI  $\leftarrow$  {client_ID, OAuth_URI, redirect_URI}
7.  sendReq(loc_URI) //send request to identity service
8.  if callIdServ()  $\neq$   $\emptyset$  then call(  $\Rightarrow$  OAuth_URI)
9.    set_client(client_ID), set_URI(redirect_URI) //set values
10.   set_secret(client_secret), set_decode(client_JSON_decode)
11.   get(response_token)
12. end if
13. user_token  $\leftarrow$  {response_token}
14. sendMediationServ(user_token) //send mediation service request
15. if callMedServ()  $\neq$   $\emptyset$  then
16.   for subs_count  $\leq$  z do
17.     z++
18.     getSubs(sub_idz) //get available subscriptions
19.     setSubs(sub_idz) //set user subscriptions
20.   end if

```

Algorithm 2 demonstrates the Inter-Cloud Mediation module HTTP calls to the various actors of the system, including interacted cloud platforms, and inter-cloud services (e.g. the Context Broker Service). These are calls to get (i) authentication token, (ii) instance information list, (iii) compute (image/services information) list, (iv) telemetry information (resource usage), (v) network information, (vi) cloud flavors (available server configurations) list. In addition, the service provides the requests for posting data to the Storage Service and for flavor a new request for deploying a service.

Algorithm 2: Inter-Cloud Mediation Module OpenStack HTTP Calls

conf: The configuration files of clouds

i,z: Counters (Integers)

token: The cloud user authentication token

tenant_id: The cloud tenant ID

server_endpoint: The instance/server endpoints

server_object: The instance/server object file (JSON)

image_name: The image tenant name

image_object: The image object file (JSON)

res_usage: The resource usage object file (JSON)

[Call to get authentication token]

HTTPCall Token(conf.un, conf.pass, content, conf.cl_host, conf.port_auth)

URI \leftarrow concatenate(URI) //create the URI from the function definition

```

call(URI) //make the call to the URI
get(object) //get response data from the interacted cloud (JSON)
PostStorage(object,storage_service) //post data to the Storage Service
tenant_ID ← ParseObject(tenant_ID) //parse object to get tenant ID
token ← ParseObject(token) //parse object to get token
end HTTPCall

[Call to get instance collection object]
HTTPCall GetInstances(token, tenant_id, port_service)
URI ← concatenate(URI) //create the URI from the function definition
call(URI) //make the call to the URI (servers)
PostStorage(object,storage_service) //post data to the Storage Service
end HTTPCall

[Call to get compute collection object]
HTTPCall GetCompute(token, tenant_id, port_service)
URI ← concatenate(URI) //create the URI from the function definition
call(URI) //make the call to the URI (images)
PostStorage(object,storage_service) //post data to the Storage Service
end HTTPCall

[Call to get telemetry collection object]
HTTPCall GetTelemetry(token, tenant_id, port_service)
URI ← concatenate(URI) //create the URI from the function definition
call(URI) //make the call to the URI (tenant-usage)
PostStorage(object,storage_service) //post data to the Storage Service
end HTTPCall

[Call to get network collection object]
HTTPCall GetNetworks(token, tenant_id, port_network)
URI ← concatenate(URI) //create the URI from the function definition
call(URI) //make the call to the URI (networks)
PostStorage(object,storage_service) //post data to the Storage Service
end HTTPCall

[Call to get cloudflavors collection object]
HTTPCall GetFlavors(token, tenant_id, port_service)
URI ← concatenate(URI) //create the URI from the function definition
call(URI) //make the call to the URI (networks)
PostStorage(object,storage_service) //post data to the Storage Service
end HTTPCall

[Call to post collection object to the Storage Service]

```

```

HTTPCall PostStorage(object,location)
call(storage_Service_Endpoint) //make a call to the storage service
collection ← get(object) //capture call data and stores a collection
end HTTPCall

[Call to post service creation to a cloud]
HTTPCall PostServiceCreation(port_service,tenant_id, name, imageRef, key_name,
flavor, max_count,min_count)
call(storage_Service_Endpoint) //make a call to the storage service
collection ← get(object) //capture call data and stores a collection
URI ← concatenate(port_service,tenant_id, name, imageRef, key_name, flavor,
max_count,min_count) //create the URI from the function definition
call(URI) return(response) //returns the response of the request
end HTTPCall

```

Algorithm 3 demonstrates the Inter-Cloud Mediation module and its calls made to the VMWare VCloud platform. Similar to algorithm 2 we use the HTTP protocol and RESTful calls. This includes calls to receive an authentication token and tenant identification in order to perform further authentication for other services. These are to collect template list (available images), get servers lists and make post for new service deployment (using the extracted data). Finally, the last call defines the retrieval of the related network data (e.g. getting the new assigned floating IP and MAC addresses etc.) for the new instances.

Algorithm 3: Inter-Cloud Mediation Module VMWare VCloud HTTP Calls

```

user: The username
pass The password
token: The cloud user authentication token
tenant_id: The cloud tenant ID
response_data: The response of the request
storage_service: The storage service URI
template_ID: The template ID
service_list: The list of templates (images)
action: The instantiation action (images)
instance_ID: The instance ID

[Call to get authentication token]
HTTPCall GetToken(user:pass, URI)
call(URI) //make the call to the URI
get() //get response data from the interacted cloud (XML)

```

parse(response.XML ⇒ response.JSON) //transform XML to JSON object
 PostStorage(response,storage_service) //post data to the Storage Service
 tenant_ID ⇐ ParseObject(tenant_ID) //Set tenant
 token ⇐ ParseObject(token) //Set token
end HTTPCall

[Call to get template (image list)]
HTTPCall GetTemplate(token, URI, tenant_id)
 call(URI) //make the call to the URI
 get() //get response data from the interacted cloud (XML)
 parse(response.XML ⇒ response.JSON) //transform XML to JSON object
 PostStorage(response,storage_service) //post data to the Storage Service
 template_ID⇐ ParseObject(response) //parse object to get template
end HTTPCall

[Call to get actions (Instantiation command list)]
HTTPCall GetInstances(token, URI, template_ID)
 call(URI) //make the call to the URI
 get() //get response data from the interacted cloud (XML)
 parse(response.XML ⇒ response.JSON) //transform XML to JSON object
 PostStorage(response,storage_service) //post data to the Storage Service
 action ⇐ ParseObject(response) //parse object to get service (instance)
end HTTPCall

[Post for Deployment]
 //[data: name, deployment is TRUE, powerOn is TRUE, template_ID]
HTTPCall PostInstances(token, URI, tenant_ID, action, data)
 call(URI) //make the call to the URI
 parse(response.XML ⇒ response.JSON) //transform XML to JSON object
 PostStorage(response,storage_service) //post data to the Storage Service
 instance_ID ⇐ ParseObject(response) //parse object to get new instance ID
end HTTPCall

[Call to get networks]
HTTPCall GetInstances(token, URI, instance_ID)
 call(URI) //make the call to the URI
 get() //get response data from the interacted cloud (XML)
 parse(response.XML ⇒ response.JSON) //transform XML to JSON object
 PostStorage(response,storage_service) //post data to the Storage Service
 network ⇐ ParseObject(response) //parse object to get network
end HTTPCall

Algorithm 4 shows the Inter-Cloud Mediation operations including calls to (i) Context Broker Service (for updating, querying, registering etc. context entities and their attributes), (ii) CEP Service (to set events, retrieve definitions and update status of the service engine, (iii) Storage Service (for posting, getting, deleting and updating databases and collection) and (iv) Inter-Cloud Market Place for registering application, posting and getting offerings, binding resources to resource providers and provide payment capabilities to users. Finally, the Sanity Check operation shows a typical test to check whether the user information or the interacted clouds are responding to the call requests.

Algorithm 4: Inter-Cloud Mediation Module Operations

BURI: The Context Broker URI

CEPURI: The Context Broker URI

SURI: The Storage Service URI

MURI: The Marketplace URI

b_port: The Context Broker port

c_port: The Complex Event Processing port

s_port: The Storage Service port

m_port: The Marketplace port

data: The data in JSON format (elements, entities or attributes)

acc: An accumulator server of the Context Broker

dur: The duration of the entity

type: The type of the entity (e.g. on time interval action)

int: The interval value

int: The throttling time value

sID: The subscription ID

nc: The notification condition

cv: The condition value

def: The definition of the complex event processing

action: The action of the complex event processing

db: The databases of the Storage Service

col: The collections of the Storage Service

col_name: The collections name of the retrieved collection

col_ID: The collections ID of the retrieved collection

col_ID: The collections ID of the retrieved collection

reg_data: The marketplace registration data (name, version, description)

of_data: The marketplace offering (name, application, image, description)

bind_data: The marketplace binding to provider (name, provider)

pay_data: The marketplace payment data (tax description, payment info)

sanity: A boolean variable for the sanity check

[Module: Context Broker Service]

HTTPCall updateContext(BURI, b_port, data(context elements))
 HTTPCall queryContext(BURI, b_port, data)
 HTTPCall subscribeContext(BURI, b_port, data, acc, dur, type, int, thr)
 HTTPCall updateContextSubscription(BURI, b_port, sID, nc, type, cv)
 HTTPCall registerContext(BURI, br_port, entities and attributes, dur)
 HTTPCall discoverContextAvailability(CBURI, b_port, data)
 HTTPCall discoverContextAvailability(CBURI, b_port, data, acc, dur)

[Module: Complex Event Processing Service]

HTTPCall setEvents(CEPURI, c_port, data)
 HTTPCall RetrieveDefinitions(CEPURI, c_port, def)
 HTTPCall UpdateStatus(CEPURI, c_port, action)

[Module: Storage Service]

HTTPCall post/get/delete/update(SURI, s_port, dbs)
 HTTPCall post/get/delete/update(SURI, s_port, col)
 HTTPCall getCollection(SURI, s_port, col_name)
 HTTPCall getCollectionID(SURI, s_port, col_ID)

[Module: Inter-Cloud Market Place]

HTTPCall register(MURI, m_port, reg_data)
 HTTPCall post/get/updateOffering(MURI, m_port, of_data)
 HTTPCall bindResources(MURI, m_port, bind_data)
 HTTPCall payment(MURI, m_port, offer_ID, pay_data)

[Operation: Sanity Check]

```
call(storage_Service_Endpoint) //make a call to the storage service
if response()  $\neq$   $\emptyset$  then
  sanity  $\leftarrow$  TRUE else
  return sanity
updateContext[sanity_check_status]
end if
```

Algorithm 5 shows a use case example for retrieving information from the inter-cloud bridge system. The algorithm performs a sanity check for connections and if passes it executes the call from Algorithm 2. Also it performs an update to the Context Broker (send a data object) and sends an event to the Complex Event Processing Service following the appropriate call of Algorithm 3.

Algorithm 5: Use Case of Information retrieval from OpenStack system and storage in the Inter-Cloud Mediation Service

1. **for** all clouds **do**
 2. **if** SanityCheck() \Rightarrow TRUE
 3. GetToken(), GetInstances(), GetCompute(), GetTelemetry()
 4. GetNetworks(), GetFlavors, PostStorage(...col.ID...) //Store data
 5. updateContext(userSubscriptions) //post data to Context Broker
 6. setEvents() //pass user options to Complex Event Service
 7. **end if**
 8. **end for**
-

Algorithm 6 shows a use case example for context subscription and querying when new images are deployed to interacted clouds. The assumption is that the user configures a) the type (on change e.g. a new service is available or on time interval) and b) the throttling values (the maximum waiting time for a subscription). Then in case that this change occurs, the Inter-Cloud Mediation Service posts data to the Context Broker Service (which notifies the subscribed consumers (e.g. the user interfaces or post events to the CEP Service)). It also provides capabilities to update context data subscriptions on request of the users. The assumption is that either the Inter-Cloud Mediation Service will allow users to manually subscribe to information offered by the Context Broker (e.g. send notifications to the user interface module) or the service it-self will subscribe to information for performing further actions (e.g. send events to the CEP engine for further analysis and generation of pattern based events).

Algorithm 6: Use Case of Context Subscription and Querying for Images in the Inter-Cloud Mediation Service

1. **for** all clouds **do**
2. GetInstances(), PostStorage(...col.ID...) //Store image data
3. type \leftarrow ONCHANGE
4. //Subscribed consumer actions
5. throttling \leftarrow time_value
6. subscribeContext(entity, duration, type, throttling)
7. //Context Broker Actions
8. updateContext(image) //post image JSON object to Context Broker

```

9.      if interval expired
10.         queryContext() //notify subscribers
11.         updateContextSubscription() //allows the update of subscriptions
12.      end if

```

Algorithm 7 demonstrates a use case of the CEP for events traffic management in the Inter-Cloud Mediation Service (e.g. monitoring on increasing number of requests and provide event response). The assumption is that the Mediation Service is subscribed to the Context Broker for events, and queried data are forwarded to the engine. The user configures the CEP engine with (i) input/output events, (ii) event processing agents along with the event-processing context (the pattern of sequential events), (iii) consumers for output events (in our case this is a RESTful input service for collecting notification implemented within the Mediation Service). The algorithm sends a call in order to start the remote CEP engine and to configure the definitions name. Then, CEP is set to wait for input events coming from the Mediation Service. The CEP engine runs during real-time events submission and the context condition triggers an output event that is forwarded to the pattern subscribers (in our case the Mediation Service). It should be mentioned that the actual pattern has been configured at previous state and stored within the CEP in the so-called Inter-Cloud CEP Repository. An example context condition is to evaluate the event traffic of a cloud and in case of increased traffic (e.g. by 70%); CEP will notify the Mediation Service. This is a simple event pattern, yet users could create more complex based on their requirements. CEP provides a graphical user interface for creating new definitions.

Algorithm 7: Use Case of the Complex Event Processing in the Inter-Cloud Mediation Service

CEP_URI: The URI location of the service

CEP_port: The port of the service

CEP_engine: The engine of the Complex Event Processor Service

app_name: The name of the application

in_event: The input event in the Complex Event Engine

in_vol: The input event volume

out_event: The output event in the Complex Event Engine

event_pr_agent: The event processing agent of the Complex Event Engine

temp_pr_context: The temporary processing context

term_time: The termination time of the context

consumer: The consumer of the pattern (export output report)

event_count: The counter of input events (export output report)

i: An integer variable (export output report)

1. [Calls executed in the Inter-Cloud Mediation Service]
 2. HTTPCall (subscribeContext(data)) //Update event pattern subscribers
 3. HTTPCall (queryContext(data)) //Update event pattern subscribers
 4. [Calls executed in the Complex Event Processing Service]
 5. [Condition: Traffic of events to be increased by 0.7]
 6. SetContext(condition, termination_time) //Set context event and time
 7. SetEventAgent(event_pr_agen, type, context, condition, out_event)
 8. SetConsumer(consumer, format (JSON)) //Set applications consumer
 9. HTTPCall(updateStatus(start)) // Starts the processing engine
 10. **for** event_count \leq i **do**
 11. [Events are coming from Mediation service]
 12. HTTPCall (URI, in_event, in_vol) // e.g. (traffic, 1000)
 13. **end for**
 14. HTTPCall (out_event, consumer) // Consumer: Mediation Service
 15. [Calls executed in the Inter-Cloud Mediation Service]
 16. HTTPCall (updateContext(data)) //Update event pattern subscribers
-

4.2. The Inter-Cloud Bridge System Implementation

To develop the inter-cloud bridge system we utilized heterogeneous cloud platforms like OpenStack, FI-WARE FI-LAB²⁶ and Amazon WS Simple Storage Service (S3) API for experimental purposes. The interactions of services are based on RESTful protocol as in Schreier (2011). This offers major benefits to a heterogeneous inter-cloud bridge system as the interacted parties that exchange RESTful calls do not require knowing the structure of the service APIs. The responses are in JSON²⁷ information model and the transfer of data is happening using the HTTP protocol. The later is a key lightweight syntax for efficient communication and is supported by different clouds e.g. OpenStack, OpenNebula, FI-LAB, Amazon WS etc. and by different services like Context Broker and CEP. The storage service is based on a MongoDB²⁸ as an agile and scalable open-source document database

²⁶FI-LAB: <http://www.fi-ware.org/lab/>

²⁷JavaScript Object Notation: <http://json.org>

²⁸MongoDB: <http://www.mongodb.org>

for storing JSON objects. It offers an easy to use schema-free information model that allows storing data in the form of objects. The Mediation Service is deployed in an OpenStack platform and connectivity to cloud platforms and services happens at real-time. This includes that the events (that are RESTful calls) are exchanged either on a time interval or on user request.

4.3. Description of interacted heterogeneous cloud platforms

The prototype system has been developed to interact with various cloud platforms including OpenStack, FI-WARE FI-LAB and Amazon WS as shown in Figure 10.



Figure 10: The Inter-Cloud Mediation Service interacted Cloud Platforms

The prototype system utilizes mainly a) the intellicloud of Technical University of Crete²⁹ that is an operative OpenStack datacentre with a capacity of 128 CPU Cores, 284 GB RAM, 12 TB HDD, b) the Seville XIFI node³⁰ that totally includes 832 CPU Cores, 6656 GB RAM and 300 TB HDD. We also use Amazon S3 API, and a second OpenStack system (4CPU Cores, 8 GB RAM, 1 TB HDD) for experimentation purposes. It should be mentioned that we have used instances of IDM, Context Broker, CEP and Marketplace Service deployed in FI-LAB platform. The rest of the inter-cloud services are implemented and deployed in the Intellicloud platform.

5. Inter-Cloud Experimental Analysis

The experimental analysis aim to explore the performance of the various components. The target is to show that metrics perform efficiently and do not increase the processing times compared to the execution time of the requests within the cloud, as executed by the cloud platform. Also another target is to

²⁹Intellicloud of TUC: <http://cloud.intellicloud.tuc.gr>

³⁰XIFI FP7 project: <https://fi-xifi.eu>

show that the inter-cloud services do not increase the traffic within the clouds or cause bottlenecks and increasing delays in communication. Specifically we have produced a performance analysis of:

- (i) The Context Broker Service for context data update, query and to subscribe and unsubscribe on data attributes (as in Algorithm 4). We execute 100 consequent requests and we compare the trends on real time responses to explore whether the service offers stability regarding the increasing number of requests and identify what is the tendency between the last call in correlation to the first one.
- (ii) The CEP event management, as in Algorithm 4, similarly to experiment (i) the CEP performance analysis aims to explore the stability of the service with respect to (a) the increasing number of calls and (b) for the case that this it causes increasing traffic to the CEP engine.
- (iii) The various requests for services, as in Algorithm 3. The aim is to compare the real times spent within a cloud for consuming a request and producing a response with the actual real times of the service. Another useful comparison is to explore the trends on real-time responses to clarify whether or not the consequent number of requests causes increasing traffic or bottlenecks within an OpenStack platform.
- (iv) The various requests in a FI-LAB platform as in Algorithm 3. Similar to the previous experiment (ii) we explore the performance of the FI-LAB when certain calls occur and we identify increasing traffic and bottlenecks. Further, we compare the performance of the same service calls executed in Intellicloud and FI-LAB using the inter-cloud Mediation Service to explore possible deviations on real-times and the significance of these results.
- (v) The various requests executed in an inter-cloud bridge system regarding tenant authentication in 4 clouds (2 OpenStacks, Amazon WS S3 and FI-LAB). The aim is to explore the deviation of performance comparing the real-times for authentication for 1 to 4 clouds respectively.
- (vi) Presentation of a use case of Inter-Cloud Mediation for context data requests and submissions to the Storage Service and posting data to the Context Broker Service and CEP. This demonstrates the evaluation of the deviation of the real-times of the various calls. Also the aim is to explore the combination of each of which in the Inter-Cloud Mediation Service and the significance of this performance.

The experimental metrics are as follows: real, user and sys that are I/O time and CPU time statistics. The real metric represents the wall clock time (the time needed from start to finish of a call), the user is the amount of CPU time spent in user-mode code (outside the kernel within the process) and the sys is the amount of CPU time spent (inside the kernel within the process).

5.1. Experimental Analysis of the Context Broker Service Calls

The Context Broker experimental analysis includes the exploration of the real times for actions related to (a) subscribe/unsubscribe of context data and (b) create, query and register context to the broker. The tests aim to explore the behaviour of the service regarding to the overall stability on the response times made by the Mediation Service. Figure 11 demonstrates the increasing number of 50 requests to subscribe and un-subscribe to context data from the Mediation Service to the Context Broker Service.

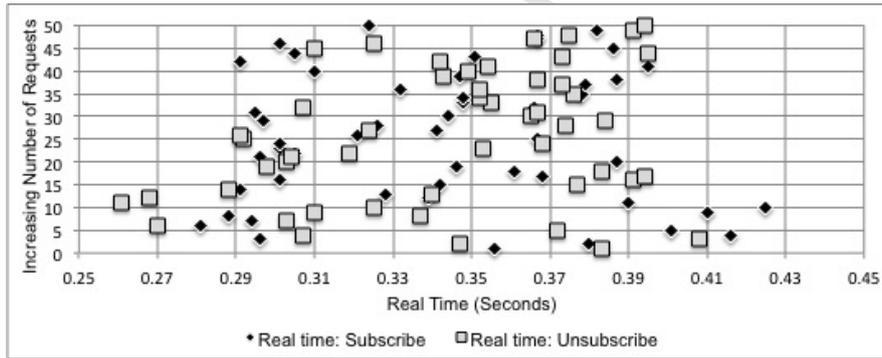


Figure 11: The Mediation Service calls to the Context Broker (subscribe and unsubscribe)

We have utilized the Google Data API Services³¹ for extracting benchmarks regarding user authentication and information retrieval. We tested 50 call requests that have been executed thoroughly and we have calculated an average of 0.58 seconds for authentication and 0.31 for information retrieval. Based on Figure 11, the assumption is that the user manually subscribe/unsubscribe to the broker using the Mediation Service, get notification periodically (e.g. every 5 seconds) or on user request. We have tested it for consequent requests over time in order to explore whether the

³¹<https://developers.google.com/gdata/>

service behaves stable during the requests. It is shown that the real times of subscriptions/un-subscriptions remain under the 0.43 seconds. This is considered an efficient system response for RESTful calls compared also to the typical call times offered by the Google Data API (e.g. the value of 0.31 seconds). The diagram also demonstrates that there are not highly increasing peaks during the calls, thus behaves efficient in terms of stability. Further, we can observe that that the trend lines for both calls do not increase significantly their tendency rapidly.

Figure 12 demonstrates the increasing number of 100 calls to create, query and update context data requests from the Mediation Service to the Context Broker Service. In this case the create context calls utilize the update context calls of Algorithm 2 and along with the context registration performs efficient as the most real time result remains as a whole remain between 0.5 and 0.7 seconds. In general, these results are acceptable compared to the typical call on the Google Data API where the analogous value is 0.58 seconds.

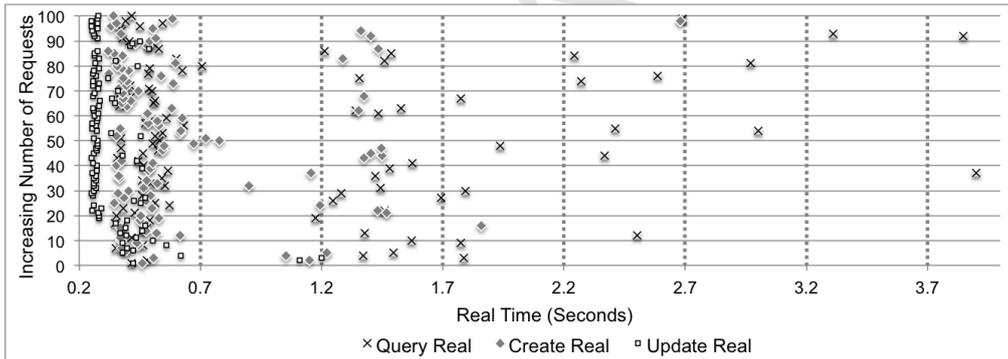


Figure 12: The Mediation Service calls to the Context Broker (create, query and update)

5.2. Experimental Analysis of the Complex Event Processing Service Calls

The CEP Service accepts call requests from the Mediation Service to trigger patterns and rules for conditional events. Figure 13 demonstrates the actions regarding getting definitions (a predefined event management pattern), changing definitions as in Figure 13(a) and finally the real time of posting events to the Event Engine as in Figure 13(b). Specifically, in Figure 13(a) we executed 10 requests as the assumption is that each service will have a certain number of pattern definitions.

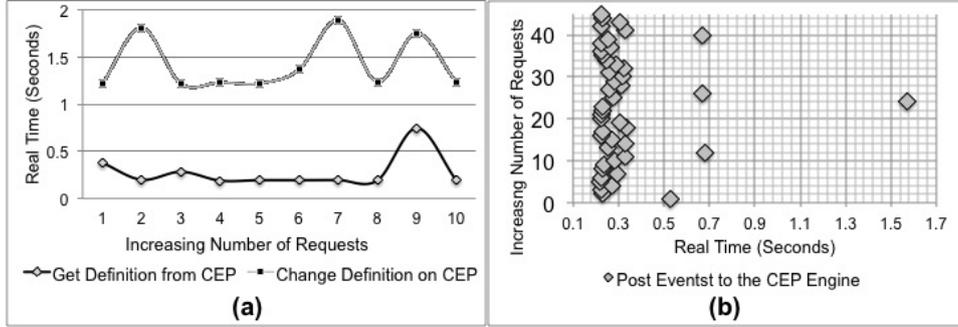


Figure 13: The Mediation Service calls to the Complex Event Processing Service

Within each definition there is data regarding events, types, data types and conditions. Thus a user creating a rule adds a significant amount of information. In our case we have configured 4 event types and their configurations and 1 conditional event per definition. Thus the number of 10 requests is considered as an adequate experimental value (around 50 configurations totally). Both diagrams show satisfactory responses, as the definitions management calls using RESTful remain under the 2 seconds boundary. Similarly, Figure 13(b) shows that 89% of 50 consequent events are posted within 0.3 seconds while 99% remain under the 0.7 seconds, a result that shows a highly stable service.

5.3. Experimental Analysis of the OpenStack Service Calls

The performance analysis shows the response times of a typical OpenStack platform for data retrieval requests such as information of authentication, credentials, available images, instances (servers), flavors and resources. The analysis aims (a) to identify the performance of the cloud in terms of internal time for responding a request and (b) to explore whether the increasing number of requests cause bottlenecks or delays in the actual cloud platform. The benchmarks are based on the works of Paradowski et al. (2014) and in Sotiriadis et al. (2014). Specifically, in Paradowski et al. (2014) the benchmarking study concludes that the OpenStack times for deployment of a new VM varies among 16 and 24 seconds (tested with 1 to 2 CPUs, 100 MB to 1024 MB RAM and 100 MB to 20GB HD). Also in Steinmetz et al. (2012) are presented times for VM creation within the OpenStack platform. In Sotiriadis et al. (2014) we measured the average time of a service call within the OpenStack (e.g. for authentication is 0.22 seconds).

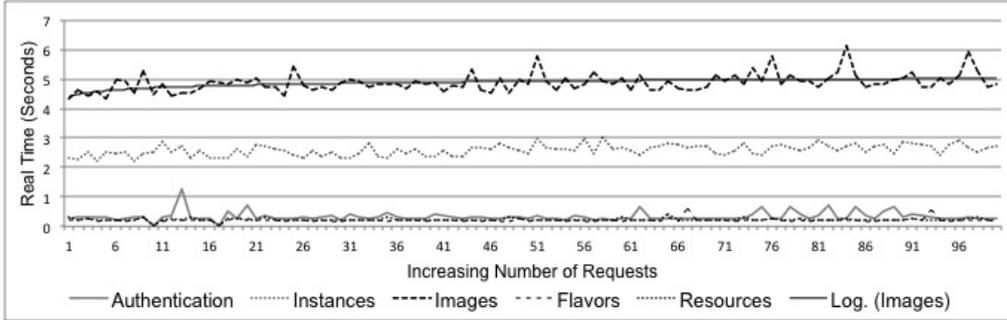


Figure 14: The variation of real times of Mediation Service calls to OpenStack platform

Figure 14 demonstrates the variation of real times for 100 calls made by the Mediation Service (as in Algorithm 2) to the interacted OpenStack platform. It is shown that the resource information retrieval calls are the fastest, while the images retrieval is executed between 4 and 6.2 seconds (yet in this case the response includes a high number of data regarding the specification of the images). We consider the times for authentication, flavors and resources as efficient since the average result (0.5 seconds) remains close to the 0.23 seconds for the internal calls. Yet the times needed for instances and images data are higher because these actions include authentication, posting tenant information and retrieval of large datasets. In the tested environment this includes retrieval of 15 images and 15 instances along with a list of information.

The responses in this case include data retrieval in JSON format. It should be mentioned that the calls are executed once; in the first serve instantiation, then the Mediation Service subscribes to context and sends notifications using a criterion (interval or on user request) for new data. In general, these calls are executed in efficient times (80% of all the calls executed under 3 seconds). Also the diagram lines show an overall stability of the OpenStack cloud and do not cause an increasing tendency on delays in the platform. For instance, we observe that the logarithmic trend line shows a stable tendency over the 40 requests.

Figure 15 demonstrates three cases of OpenStack calls made by the Mediation Service. The calls include the total time for performing authentication, getting available images, instances, flavors and resources as a whole. The real time figures show that the most of calls (91.2%) are executed under 8.6 seconds, in user time, 75% remain under 0.3 seconds, and in sys time the

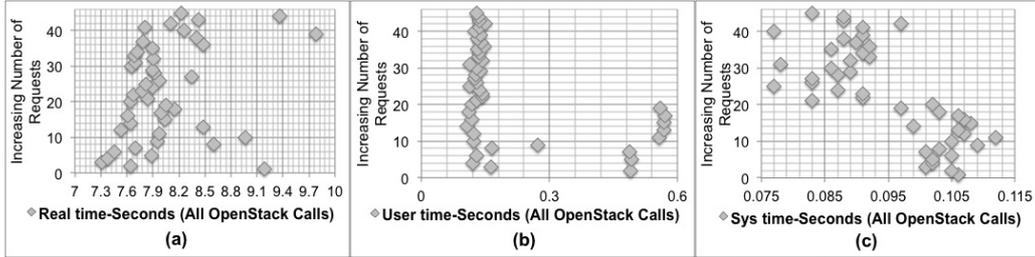


Figure 15: The real, user and system time of OpenStack calls from Mediation Service

percentages are shared, yet under the 0.12 seconds. Especially, in 15(a), we observe that most of calls behave stable as are executed within the boundary of 7.3 and 8.5 seconds. Also, 96% of the calls executed as a whole remain under the 9.4 seconds thus showing a deviation of around 2 seconds for OpenStack platform responses. Based on this small deviation, we consider that this value does not cause high delays in the system.

5.4. Experimental Analysis of the FI-LAB Service Calls

The FI-LAB experimental analysis demonstrates the exploration of the response times for calls as presented in the three diagrams of Figure 16. For real time, 95.5% is near 5.4 seconds compared to the 7.9 seconds of the OpenStack cloud (Figure 16(a)). Regarding user and sys metrics results remain at low levels (Figure 16(b),(c)). It is also shown that the Mediation Service does not cause any delays, e.g. real times remain close to the 5 seconds boundary.

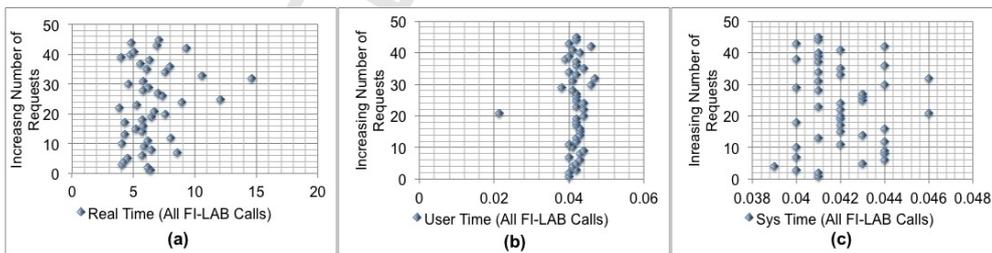


Figure 16: The variation of real times of Mediation Service calls to FI-LAB platform

Figure 17 demonstrates the comparison of average time of service calls between OpenStack and FI-LAB made by the Mediation Service. It is shown

that in most of the cases OpenStack outperforms FI-LAB, however for instance calls, FI-LAB achieves better times. An overall conclusion is that the Mediation Service performs efficiently as it retrieves data under 2.5 seconds per call, with an average time spend for calls to be 0.95 seconds. This compared to the benchmark value of 0.25 seconds for internal calls is higher, yet it remains efficient since this includes the time consumed during communication (e.g. due to the distance of the datacentre).

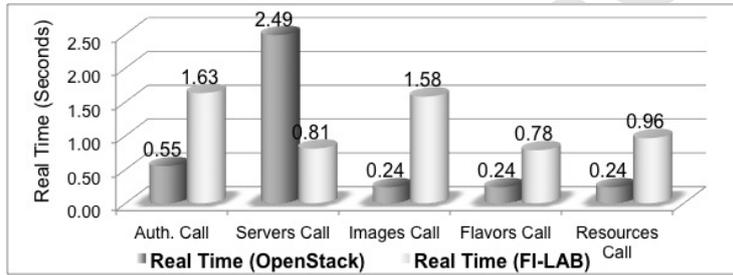


Figure 17: The comparison of calls from Mediation Service to OpenStack and FI-LAB

Table 1 shows the times (in seconds) of the calls where FL-RT stands for real times for FI-LAB and OS-RT for OpenStack real times of intellicloud. The last row shows the average (AVG) times as in Figure 17.

Calls	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	AVG
FL-RT: Auth	1.6	3.1	1.4	1.2	1.3	1.4	2.1	1.5	2.2	1.4	1.1	3.1	1.2	1.6	1.6	1.3	1.4	1.2	1.3	1.6	1.63
FL-RT: Instances	0.7	0.8	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	1.9	0.8	1.0	0.8	0.8	0.6	0.7	0.9	0.8	0.8	0.81
FL-RT: Images	1.4	1.3	1.1	1.0	1.6	1.5	3.8	1.2	1.8	1.0	2.2	2.0	1.1	1.5	1.2	1.1	1.0	1.3	2.2	2.1	1.58
FL-RT: Flavors	2.3	0.4	0.4	0.7	0.5	0.6	1.4	0.5	0.6	0.5	0.5	0.4	0.4	1.1	0.4	0.5	0.7	0.6	0.5	2.6	0.78
FL-RT: Resources	0.5	0.5	0.6	0.7	0.4	1.4	0.6	2.5	0.4	0.5	0.4	1.8	0.6	0.6	1.1	2.3	0.5	1.6	1.8	0.5	0.96
OS RT: Auth	1.4	0.7	0.3	0.3	0.6	0.2	0.6	0.3	0.6	1.2	0.7	0.3	0.7	0.2	0.7	0.3	0.7	0.3	0.7	0.3	0.55
OS RT: Instances	2.6	2.4	2.4	2.4	2.4	2.5	2.4	2.5	2.6	2.6	2.5	2.4	2.3	2.6	2.6	2.5	2.3	2.6	2.6	2.5	2.49
OS RT: Images	4.7	4.2	4.1	4.3	4.4	4.2	4.2	5.3	4.4	4.7	4.3	4.4	5.1	4.4	4.3	4.3	4.5	4.8	4.3	4.4	4.47
OS RT: Flavors	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.3	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.20
OS RT: Resources	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.23

Table 1: Various calls executed in the OpenStack and FI-LAB platforms

5.5. Experimental Analysis of the Inter-Cloud Bridge System (4 Cloud Platforms)

The experiment demonstrates an Inter-Cloud connection to four clouds to demonstrate heterogeneity and the variation of the metrics in a large-scale environment. The inter-cloud sequence of calls include authentication in Intellicloud, CloudLab, FI-LAB, and Amazon WS S3. To show efficiency we introduce a new metric, namely as the factor which is the actual performance

of the metric to the compared value (division of worst by best performance value). The factors for comparison of the four clouds are demonstrated below. The calls are made from the Mediation Service executed in FI-LAB. We compare factors of four (2 OpenStacks, FI-LAB, AWS), to three clouds (2 OpenStacks and FI-LAB), in this case the three clouds achieve 53% for real metric and 83% for user of the two clouds performance. This variation is mainly related to the distance of the remote AWS datacentre. Compared to one cloud performance, the four clouds achieve 20% of the benchmark performance (same requests executed within the cloud). Yet, realistically, the inter-cloud authentication service is executed in less than 2 seconds, a result that is considered as acceptable. Figure 18 shows that the real time remains slightly averagely under the 2 seconds. To our view, this is a highly acceptable value by considering that the numbers of interactions have been increased.

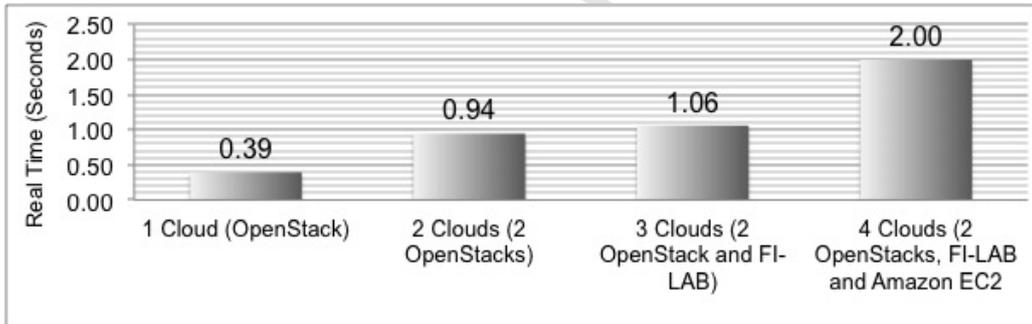


Figure 18: Real times for combination of clouds (1-4 clouds)

5.6. Use Case of Inter-Cloud Mediation Service Calls

This section presents a use case for experimentation study to demonstrate the whole set of call requests executed by the Mediation Service in a typical Inter-Cloud scenario. It includes the calls including the Context Broker for context data registration, subscription and querying, CEP to set/get/alter definition and posting events, and the Mediation Service calls to two interacted clouds (intellicloud and FI-LAB). The use case also includes a request from the Mediation Service for a new cloud instance. The assumption is that the user has already created a security key using the OpenStack dashboard. Figure 19 shows the real times of various interactions.

Figure 19(a) shows the comparison of the calls as a whole. It is shown that the highest times are for the instance creation service (in average of 10 seconds), while the Context Broker and the CEP are the lowest times (in average 2.5 seconds). Figure 19(b) demonstrates that the Mediation service executes the sequence of steps in an average of 27 seconds.

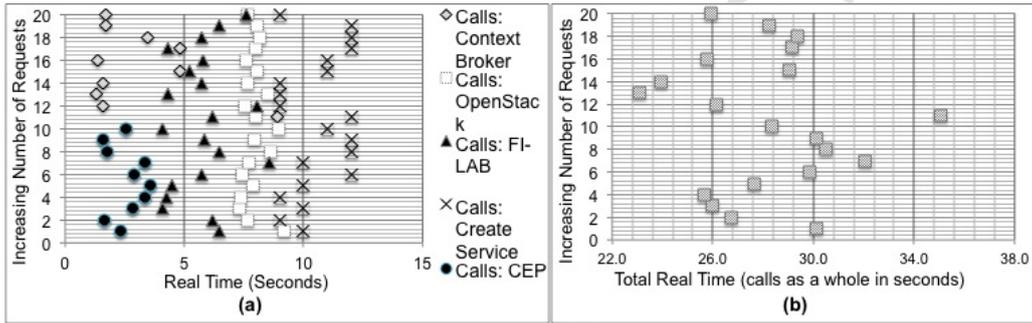


Figure 19: The Mediation Service Calls to Context Broker, Complex Event Processing and interacted clouds (OpenStack and FI-LAB)

We consider it as an efficient time when compared to the benchmarks presented in Paradowski et al. (2014) where analogous deployment request requires an average of 22.5 seconds. Further, compared to Steinmetz et al. (2012) where the average VM deployment is 5 seconds (internal OpenStack process), the Mediation Service achieves an average of 10 seconds using the API. Also, the interactions among the services that are not depending on the performance of the actual clouds offer low times, e.g. from the average 27 seconds of service creation; around 5 seconds belonging to the interaction of the services. The conclusions of the Inter-Cloud use case are as follows.

- (i) FI-WARE services including Context Broker and CEP, show stability and do not increase significantly the execution time of the Mediation Service during the calls. As shown in 19(a) the services are executed under the 5 seconds boundary.
- (ii) The Inter-Cloud consisting of two clouds (Intellicloud and FI-LAB) perform efficiently and show a consistency with regards to the time spent within a cloud for executing the call.
- (iii) The calls do not cause bottleneck or delays either in the interacted clouds, FI-WARE services or the Mediation Service itself. The highest consumed calls are these related with the creation of a new service, yet

the high times are clearly related with the time spent within the Intellicloud or FI-LAB for generating a new service (actions include, selection of available image retrieved previously, configuration of parameters related with flavors, security groups and security key).

- (iv) The Inter-Cloud as a whole demonstrates a stability with regards to the high number of requests, e.g. for the sequence of 20 requests the real times are between 26 and 30 seconds (where each one encompasses total 4 calls to Context Broker, CEP, Intellicloud and FI-LAB respectively, where again each call includes other calls e.g. for images, flavors etc.).
- (v) The new service deployment times are effective. The comparison is based on the deployment of a VM of the same configuration with the one presented in Paradowski et al. (2014). The Inter-Cloud Mediation Service completes an analogous action in 27 seconds as compared to the 22.5 seconds of the benchmark study. Yet, in our case this time also includes communication with Context Broker, CEP, communication with two clouds and deployment of a VM in intellicloud.

To conclude, the experimental evaluations demonstrate the effectiveness of the system. The solution could be compared with a federated cloud system in the case that the interactions are made within platform of the same type (e.g. OpenStack). An advantage of this work is that it includes the part of cloud federation (e.g. showing interactions between two different OpenStack systems), yet it highlights new advantages such as interoperability with heterogeneous platforms (e.g. Amazon EC2) and provision of subscription and event processing to increase performance.

6. Conclusions and Future Work

This study presented an Inter-Cloud Mediation Service utilizing OpenStack, FI-LAB and AWS APIs. The proposed solution provides communication with already deployed services like the Context Broker and the Complex Event Processing Service. The final platform service can retrieve data for available services e.g., instances, images, resource and deploy a new service in the Inter-Cloud. The experimental prototype demonstrates the initial configuration and the basic operations of the various modules. We presented various experimental analysis studies to test the interactions among the FI-WARE services, the cloud platforms and the Mediation Service and to demonstrate the performance of the fundamental prototype.

The analysis shows efficiency and consistency as real times remaining in acceptable levels as most of the time is consumed within the remote cloud providers rather than in the system it self. Future work will focus on the development of algorithms to optimize the communication and to provide new patterns and rules that will be used automatically by the system e.g. in CEP. In addition, this includes remote management of the user cloud resources (e.g. suspend/resume instances). We aim, also, to explore the messaging queuing protocol and extensions of it, such as the energy aware messaging model as in Bessis et al. (2013) to optimize communication and resource usage.

References

- Bessis, N., Sotiriadis, S., Pop, F., and Cristea, V. (2013). Using a novel message-exchanging optimization (meo) model to reduce energy consumption in distributed systems. *Simulation Modelling Practice and Theory*, 39(0):104 – 120. S.I.Energy efficiency in grids and clouds.
- Chang, V. (2014). The business intelligence as a service in the cloud. *Future Generation Computer Systems*, 37(0):512 – 534. Special Section: Innovative Methods and Algorithms for Advanced Data-Intensive Computing Special Section: Semantics, Intelligent processing and services for big data Special Section: Advances in Data-Intensive Modelling and Simulation Special Section: Hybrid Intelligence for Growing Internet and its Applications.
- Corradi, A., Fanelli, M., and Foschini, L. (2014). {VM} consolidation: A real case based on openstack cloud. *Future Generation Computer Systems*, 32(0):118 – 127. Special Section: The Management of Cloud Systems, Special Section: Cyber-Physical Society and Special Section: Special Issue on Exploiting Semantic Technologies with Particularization on Linked Data over Grid and Cloud Architectures.
- Erl, T. (2005). *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA.
- Fifield, T., Fleming, D., Gentle, A., Hochstein, L., Proulx, J., Toews, E., and Topjian, J. (2014). *OpenStack Operations Guide*. O'Reilly Media, Inc., 1st edition.
- Galis, A. and Gavras, A. (2013). *The Future Internet: Future Internet Assembly 2013 Validated Results and New Horizons*. Springer Publishing Company, Incorporated.

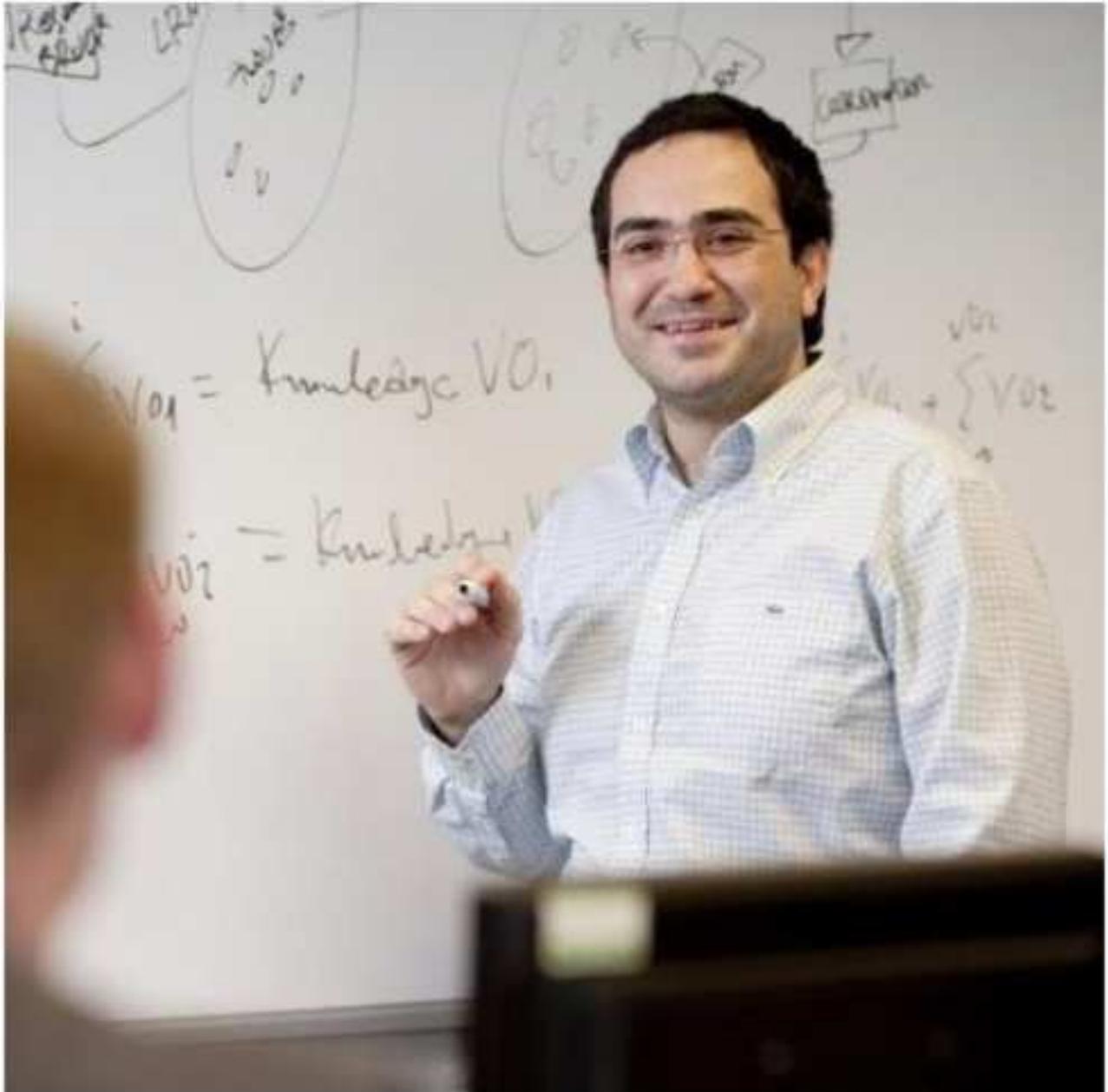
- Lucas-Simarro, J. L., Moreno-Vozmediano, R., Montero, R. S., and Llorente, I. M. (2013). Scheduling strategies for optimal service deployment across multiple clouds. *Future Gener. Comput. Syst.*, 29(6):1431–1441.
- Magid, Y., Oren, D., Botzer, D., Adi, A., Shulman, B., Rabinovich, E., and Barnea, M. (2008). Generating real-time complex event-processing applications. *IBM Syst. J.*, 47(2):251–263.
- Mauch, V., Kunze, M., and Hillenbrand, M. (2013). High performance cloud computing. *Future Generation Computer Systems*, 29(6):1408 – 1416. Including Special sections: High Performance Computing in the Cloud; Resource Discovery Mechanisms for {P2P} Systems.
- Paradowski, A., Liu, L., and Yuan, B. (2014). Benchmarking the performance of openstack and cloudstack. In *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2014 IEEE 17th International Symposium on*, pages 405–412.
- Petcu, D. (2014). Consuming resources and services from multiple clouds. *J. Grid Comput.*, 12(2):321–345.
- Pop, F. and Potop-Butucaru, M., editors (2014). *Adaptive Resource Management and Scheduling for Cloud Computing*. First International Workshop, ARMS-CC 2014, held in Conjunction with ACM Symposium on Principles of Distributed Computing, PODC 2014, Paris, France, July 15, 2014, Revised Selected Papers, Vol. 8907, 2014, XII, 217 p. 68 illus, Lecture Notes in Computer Science. Springer International Publishing.
- Schreier, S. (2011). Modeling restful applications. In *Proceedings of the Second International Workshop on RESTful Design, WS-REST '11*, pages 15–21, New York, NY, USA. ACM.
- Sotiriadis, S., Bessis, N., and Antonopoulos, N. (2011). Towards inter-cloud schedulers: A survey of meta-scheduling approaches. In *Proceedings of the 2011 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC '11*, pages 59–66, Washington, DC, USA. IEEE Computer Society.
- Sotiriadis, S., Bessis, N., Kuonen, P., and Antonopoulos, N. (2013a). The inter-cloud meta-scheduling (icms) framework. In *Proceedings of the 2013 IEEE 27th International Conference on Advanced Information Networking and Applications, AINA '13*, pages 64–73, Washington, DC, USA. IEEE Computer Society.

- Sotiriadis, S., Bessis, N., and Petrakis, E. (2014). An inter-cloud architecture for future internet infrastructures. In Pop, F. and Potop-Butucaru, M., editors, *Adaptive Resource Management and Scheduling for Cloud Computing*, Lecture Notes in Computer Science, pages 206–216. Springer International Publishing.
- Sotiriadis, S., Petrakis, E., Covaci, S., Zampognaro, P., Georga, E., and Thuemmler, C. (2013b). An architecture for designing future internet (fi) applications in sensitive domains: Expressing the software to data paradigm by utilizing hybrid cloud technology. In *Bioinformatics and Bioengineering (BIBE), 2013 IEEE 13th International Conference on*, pages 1–6.
- Steinmetz, D., Perrault, B. W., Nordeen, R., Wilson, J., and Wang, X. (2012). Cloud computing performance benchmarking and virtual machine launch time. In *Proceedings of the 13th Annual Conference on Information Technology Education, SIGITE '12*, pages 89–90, New York, NY, USA. ACM.
- Vandenberghe, W., Vermeulen, B., Demeester, P., Willner, A., Papavassiliou, S., Gavras, A., Sioutis, M., Quereilhac, A., Al-Hazmi, Y., Lobillo, F., Schreiner, F., Velayos, C., Vico-Oton, A., Androulidakis, G., Papagianni, C., Ntofon, O., and Boniface, M. (2013). Architecture for the heterogeneous federation of future internet experimentation facilities. In *Future Network and Mobile Summit (FutureNetworkSummit), 2013*, pages 1–11.
- Vázquez, A. G., Soria-Rodríguez, P., Bisson, P., Gidoín, D., Trabelsi, S., and Serme, G. (2011). Fi-ware security: Future internet security core. In *Proceedings of the 4th European Conference on Towards a Service-based Internet, ServiceWave'11*, pages 144–152, Berlin, Heidelberg. Springer-Verlag.

S.Sotiriadis received his PhD in Inter-Clouds from the University of Derby, UK. He is currently a research collaborator at the Technical University of Crete (TUC), and a member of the Intelligent Systems Laboratory. He works for the Future Internet Social Technological Alignment Research (FI-STAR), which is an FI-PPP programme. His research interests are related to Clouds, Future Internet Application development, inter-clouds and cloud federations, high performance computing systems and grids. He has won 2 best paper awards from IEEE and Springer conferences, has published over 50 papers in conference proceedings and international journals.

N. Bessis is the Director of Distributed and Intelligent Systems (DISYS) research centre, a full Professor and a Chair of Computer Science in the School of Computing and Mathematics at University of Derby, UK. His research interest is related to social graphs and dynamic resource provisioning in distributed environments such as grids, inter-clouds, and Internet of Things. He is involved in and led a number of funded research and commercial projects in these areas. Prof. Bessis has published over 200 papers, won 4 best paper awards and is the editor of several books and the Editor-in-Chief of the International Journal of Distributed Systems and Technologies (IJDST).





1. We model a service-centric service for Inter-Cloud communication.
2. The architecture relies on different cloud providers and their open SaaS.
3. We examine Inter-Cloud using cloud platform API interfaces (OpenStack and FIWARE).
4. Approach shows efficiency regarding service stability and minimizes delays.
5. Experiments show effective Inter-Cloud request, response and deployment times.

ACCEPTED MANUSCRIPT