QoS-aware and Semantic-based Service Coordination for Multi-Cloud Environments

Amir Vahid Dastjerdi

Submitted in total fulfilment of the requirements of the degree of

Doctor of Philosophy

March 2013

Department of Computing and Information Systems
THE UNIVERSITY OF MELBOURNE

QoS-aware and Semantic-based Service Coordination for Multi-Cloud Environments

Amir Vahid Dastjerdi

Supervisor: Prof. Rajkumar Buyya

Abstract

The advantages of Cloud computing, such as cost effectiveness and ease of management, encourage companies to adapt its services. However, In a Multi-Cloud environment, the wide range of Cloud services and user specific requirements make it difficult to select the best composition of services. An automated approach is required to deal with all phases of service coordination including discovery, negotiation, selection, composition, and monitoring. To simplify the process of Cloud migration, this thesis proposes an effective architecture to provide automated QoS-aware deployment of virtual appliances on Cloud service providers. The architecture takes advantage of ontology-based discovery to semantically match user requirements to Cloud services. Then, it applies a set of negotiation, selection, and optimization strategies to pick up the best available services from the list of discovered services. Finally, this thesis shows how monitoring services have to be described, deployed (discovered and ranked), and executed to enforce accurate penalties. The key contributions of this thesis are:

1. An ontology-based Cloud service discovery is proposed that works based on modeling virtual units into Semantic Web services. This helps users to deploy their appliances on the fittest providers when providers and users are not using the same notation to describe their services and requirements.
2. A scalable methodology to create an aggregated repository of services in Web Service Modeling Ontology (WSMO) from service advertisements available in XML.
3. A negotiation strategy that acquires user preferences and provider's resource utilization status and utilizes time-dependent tactic along with statistical methods to maximize the profit of Cloud providers while adhering to deadline constraints of users and verifying reliability of providers' offers. The proposed negotiation strat-

egy is tested to show how our approach helps Cloud providers to increase their profits.

4. A QoS criteria model for selection of virtual appliances and units in Cloud computing. In addition, two different selection approaches, genetic-based and Forward-checking-based backtracking (FCBB), are proposed to help users deploying network of appliances on Clouds based on their preferences.
5. A ranking system for Cloud service composition that let users express their preferences conveniently using high-level linguistic terms. The system utilizes evolutionary multi-objective optimization, and a fuzzy inference system to precisely capture the preferences for the ranking purpose.
6. An approach to help non-expert users with limited or no knowledge on legal and virtual appliance image format compatibility to deploy their services flawlessly. For this purpose , Cloud services are enriched with experts knowledge (from lawyers, software engineers, system administrators, etc). The knowledgebase then is used in a scalable algorithm for reasoning that identifies whether a set of Cloud service consisting of virtual appliance and units are compatible or not.
7. A semantic SLA template that can be used as a goal for discovery of necessary monitoring services. In addition, SLA dependencies are modeled using WSMO to build a knowledgebase that is exploited to eliminate the effects of SLA failure cascading on violation detection.

Declaration

This is to certify that

1. the thesis comprises only my original work towards the PhD,
2. due acknowledgement has been made in the text to all other material used,
3. the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Amir Vahid Dastjerdi, March 2013

Acknowledgements

Throughout my PhD study, I received support from amazing people whom I wish to acknowledge here. First and foremost, I would like to express my sincere gratitude to my supervisor Professor Rajkumar Buyya for the continuous support, advice, and guidance throughout my candidature. He has built and directed an environment that granted me an opportunity to learn and practice research skills, meet and collaborate with brilliant researchers, and transfer the long journey of PhD to a great and lovely experience.

I wish to extend my gratitude to the members of PhD committee: Prof. Christopher Andrew Leckie and Dr. Rodrigo N. Calheiros for their encouragement and insightful comments on my research. In particular, it has been all the time helpful to discuss initial research ideas with Rodrigo. He has generously helped both in building the testbed for my experiments and proof-reading of the thesis.

I would also like to thank all past and current members of the CLOUDS Laboratory, at the University of Melbourne. In particular, I would like to express my gratitude towards Dr. Saurabh Kumar Garg, Adel Nadjaran Toosi, and Yoganathan Sivaram who closely collaborated with me. I would like to thank Nikolay Grozev for proof-reading of this thesis and for his extensive comments. My thanks to fellow members: Mohsen Amini, Anton Beloglazov, Atefe Khosravi, Sare Fotouhi, Deepak Poola, Linlin Wu, Mohammed Alrokayan, Yaser Mansouri, Dr. Marco Netto, Dr. Mustafi zur Rahman, Dr. Mukaddim Pathan, Dr. Suraj Pandey, Dr. Rajiv Ranjan, Dr. Christian Vecchiola, and Dr. Marcos Dias de Assuncao for their friendship and supports.

A special thanks to Professor Omer F. Rana and Dr. Sayed Gholam Hassan Tabatabaei. It has been such a pleasure and a privilege to work with you all.

I wish to acknowledge Australian Federal Government, the University of Melbourne,

the School of Engineering, Australian Research Council (ARC), IEEE Victoria, Google, and CLOUDS laboratory for granting scholarships and travel supports which enable me to pursue doctoral study and attend international conferences.

I would like to give thanks to my parents, my brothers and sister for their endless help, supports, and love. Specially my father who is no longer with us, but his words, memories, and hard working attitude always inspired and encouraged me in all stages of life.

Finally, I would like to extend my heartfelt thanks to my wife Elahe for her encouragements during rough times. Her patience and assistance to deal with challenges we faced during last 4 years has been always impressive and extraordinary.

Amir Vahid
Melbourne, Australia
March, 2013

This thesis is dedicated to the memory of my father,
Iraj Vahid,
whose passion for science and learning was inspiring and contagious

Contents

List of Figures

List of Tables

Chapter 1

Introduction

ENTERPRISES are constantly searching for novel and creative approaches to maximize their profits and reduce their costs. They need technologies that let them grow and still do not strain them financially. Among the existing ones, Cloud computing has emerged as a promising solution providing on-demand access to virtual computing resources, platforms, and applications in a pay-as-you-go manner. Cloud service customers only use what they require and pay for what they use. As a result, Cloud computing has raised the delivery of IT services to a new level that brings the comfort of traditional utilities such as water and electricity to its users. The advantages of Cloud computing, such as cost effectiveness, scalability, and ease of management, encourage more and more companies and service providers to adapt it and offer their solutions via Cloud computing models. According to a recent survey of IT decision makers of large companies, 68% of the respondents expect that by 2014, more than 50% of their company's IT services will be migrated to Cloud platforms [128].

Clouds provide three types of services [122], namely Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). IaaS Clouds offer computing resources such as processing power, storage, networks, and other fundamental computing resources. The underlying Cloud infrastructure is managed by a provider. However, users have the flexibility to select their virtual machine images and to deploy these applications. In the PaaS model, providers supply clients with tools and services to develop software applications. In addition to the IaaS restrictions, PaaS users do not have the ability to manage or control their virtual machine images and servers. SaaS providers allow customers to use the applications such as web-based email, calendar or word edi-

tor running on a Cloud infrastructure. Neither the infrastructure nor the application are controlled by users in this model.

Moreover, Clouds, based on their ownership, exposure, and deployment model can be classified as either private, community, public, or hybrid Clouds. A Private cloud is utilized by an organization and is neither shared with other organizations nor with the general public. In contrast, A public cloud's services are accessible to the general public and in the case of community Cloud, the infrastructure is shared by a number of organizations. A Hybrid Cloud offers services deployed on two or more Clouds. Therefore, it enables data and application portability among participating Clouds. When the underling Cloud infrastructures are restricted to public Clouds (as shown in Figure 1.1) this model is called Multi-Cloud. Multiple providers are offering different virtual appliances and computing units with different pricing and Quality of Service (QoS) in the market. Therefore, it is important to exploit the benefit of hosting virtual appliances on multiple providers to not only reduce the cost and provide better QoS but also achieve failure-resistant deployment.

In order to offer their solutions in the Cloud, service providers can either utilize Platform-as-a-Service offerings such as Google App Engine [1], or develop their own hosting environments by leasing virtual machines from Infrastructure-as-a-Service providers like Amazon EC2. [2]. However, most PaaS services have restrictions on the programming language, development platform, and databases that can be used to develop applications. For example, Google App Engine supports applications developed using Java or Python only. Such restrictions encourage service providers to build their own platforms using IaaS service offerings. By offerings, we mean virtual machines and virtual appliances, which are a virtual machine image that have necessary software components to meet a specific business objective.

Therefore, if we make the assumption that service providers prefer IaaS and Multi-Cloud, service providers have to go through a process to select the most suitable Cloud offerings to host their services. This process, which is called Cloud service coordination, consists of four phases, namely discovery, Service Level Agreement (SLA) negotiation,

[1] Google App Engine. `http://appengine.google.com/`
[2] Amazon EC2. `http://aws.amazon.com/ec2/`

selection, and SLA monitoring as shown in Figure 1.1. In the service discovery phase, user requirements are used as input for discovering the best suited Cloud services among various repositories of Cloud providers. For SLA Negotiation, discovered providers and the user negotiate on the quality of services. An SLA contract will be selected from a set of made agreements. Then, the acquired service will be continuously monitored in the SLA monitoring phase. This thesis introduces an architecture for service coordination,

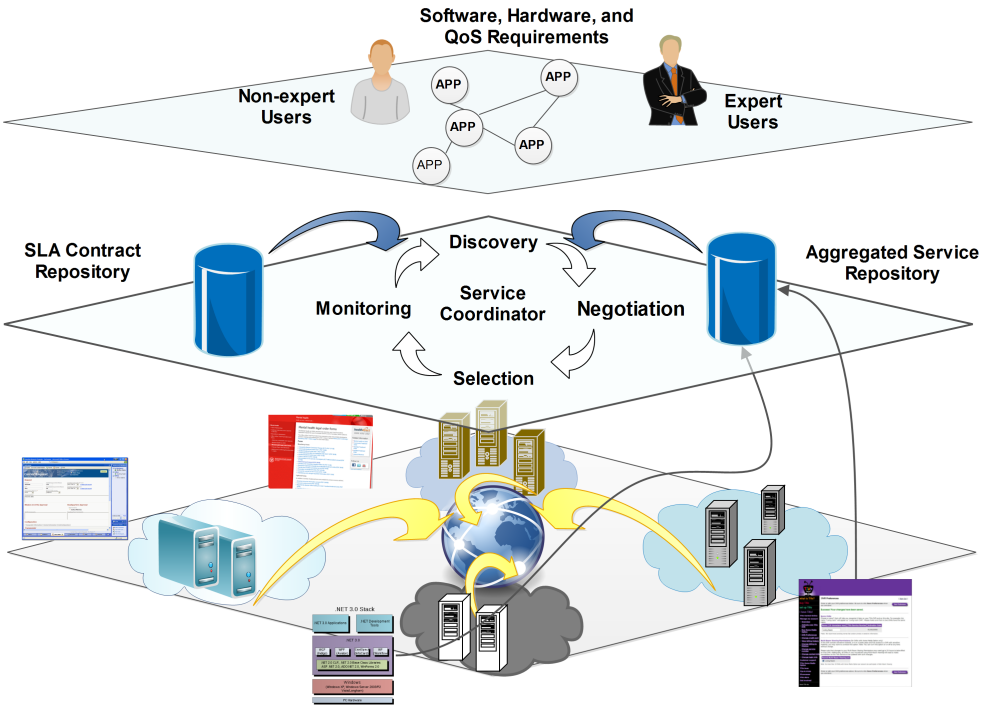


Figure 1.1: Service coordination in a Multi-Cloud environment.

and investigates algorithm and methodologies for each phase to simplify service deployment in Multi-Cloud environments. The reminder of this chapter details the need for service coordination and discusses the research problem, objectives, contributions, and organization of this thesis.

1.1 Motivation and Scope

Clusters, supercomputers and partially Grids relied on non Service Oriented Architecture (SOA) application, while Cloud focuses on Web 2.0 and SOA technology [60]. Although Clouds adopted some common communication protocols such as HTTP and SOAP, the integration, interoperability, and coordination of all services remain the biggest challenges. Service deployment and coordination, the process of making a service ready for use, often include deploying multiple, interrelated software components into heterogeneous environments. Different technologies and tools try to satisfy user requirements in terms of software and hardware and to address these complexities by describing the environments, abstracting the dependencies, and automating the process. Nevertheless, most of previous works [2,24] focused on satisfying user requirements using SOA architecture and virtualization, neglecting the consideration of Cloud computing environment as a resource supplier. In order to satisfy user requirements in terms of software and hardware, virtual appliances and virtual machines (virtual units) are considered as two fundamental offerings of IaaS providers. Virtual appliances consist of optimized operating systems and pre-built, pre-configured, ready-to-run applications. They are emerging as a breakthrough technology to solve the complexities of service deployment. Virtual appliances are proved to provide a more convenient and speedy service deployment mechanism [158].

Migrating network of connected applications such as web applications to Cloud services is a complex task. The main challenges in mapping those applications to Cloud offerings is to select and compose the most appropriate and compatible set of virtual appliances and virtual machine (also called virtual unit throughout the thesis) that satisfies the QoS requirement of users. The process of mapping, which involves all phases of Cloud service coordination, has to consider semantic heterogeneity of Cloud service interfaces and QoS criteria, and complex compatibility and dependency among cloud services that are not possible to resolve manually. Figure 1.2 shows the complexity of migrating multi-tier applications to Multi-Cloud. In addition to the service cost and reliability factors, the provider of choice is obliged to comply with law and regulations of the country where its datacenter is located. An example is the restriction imposed by United State on ex-

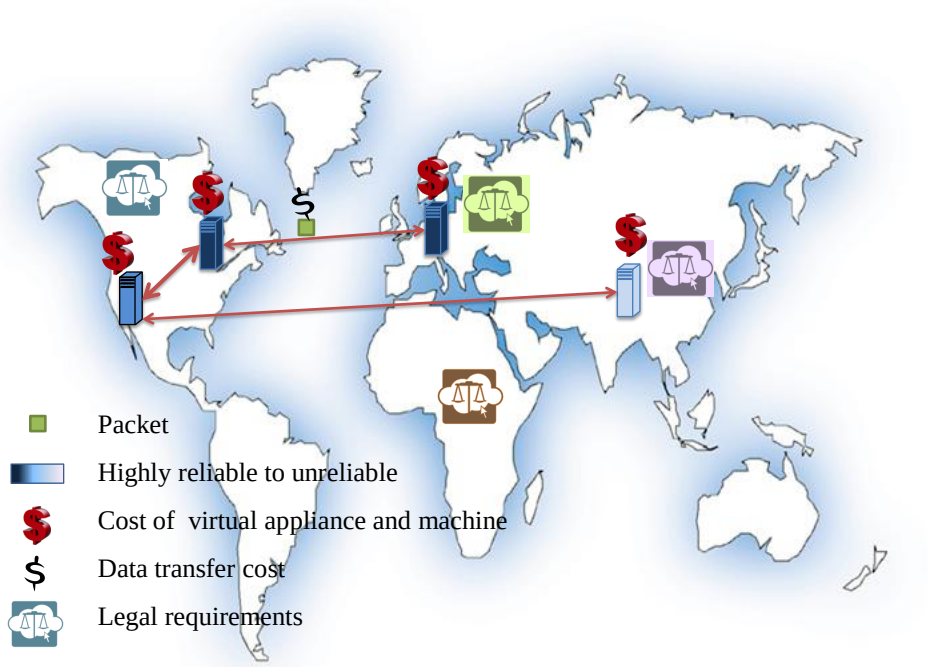


Figure 1.2: Complexities of service coordination in Multi-Cloud environments.

porting encryption technology [10], that prevents the export and deployment of such U.S developed software abroad. Therefore, migration is troublesome unless the aforementioned complexities are automatically handled by a third party system similar to what we offer in this thesis. Consequently, we propose an architecture consisting of a number of components which simplifies the process of service coordination and deployment in a Multi-Cloud environment where virtual appliance and machines are considered as offerings of Cloud providers.

1.2 Research Problems and Objectives

On a journey towards automated Cloud service deployment, in each phase of service coordination in Multi-Cloud environments, following challenges will arise:

- **Cloud service discovery challenges**
  - How to search for services, when in heterogeneous environment like Multi-

Cloud (as shown in Figure 1.1) applying symmetric attribute-based matching is not possible?

- How to automatically build an integrated repository of Cloud services so that their functional and QoS properties are understood by all parties (users, Cloud service providers, monitoring service providers) to maximize accuracy of Cloud service discovery?

- **Cloud service selection and composition issues**

  - When migrating a network of applications to the Cloud, what is the best strategy for placing them across Cloud providers? Should they be placed based on the traffic they exchange, therefore placing those with higher connectivity closer to each other to decrease latency and data transfer cost?
  - If all Cloud services and their related deployment meta-data such as auto-scaling policies and security configuration are placed on the same provider, and that provider fails, the access to deployment information would not be guaranteed. Consequently, the recovery process would be significantly delayed. How could we avoid this from happening?
  - How to identify which images are compatible with different instance types of different Clouds? Moreover, some countries like USA impose restrictions, on the location of imported and exported images, and computing units which introduce legal constraints that have to be taken into account when services form different Cloud providers are composed.
  - The Priority of end users is avoiding systems which incur complexity in capturing their constraints, objectives, and preferences. How can the system reduce the complexity of capturing user preferences for non-expert users to comply with the ease of use promise of Cloud?

- **SLA management challenges**

  - How to resolve conflicting objectives of Cloud providers and users through negotiation strategies that can achieve consensus while satisfying requirements

of both parties?

- How to create a standard model for describing SLAs in different layers of the Cloud as a major requirement for discover of necessary SLA monitoring services?
- How monitoring services have to be described, deployed (discovered and ranked), and then how they have to be executed to enforce accurate penalties?

1.2.1 Objectives

Driven by the aforementioned challenges, the following objectives have been identified:

- Investigate discovery techniques that give enough flexibility to end users to discover their required virtual appliances and machines from a range of providers and dynamically deploy them on different IaaS providers. The objective is to look for an approach that can work efficiently when the providers and users are not using the same notation to describe their services and requirements. Moreover, to guarantee the success of discovery algorithm, we investigate an approach to automatically build an aggregated repository of Cloud services from advertised offerings.

- Identify the user preferences and QoS criteria of Cloud services in the migration problem and then search for the right optimization technique that satisfies user objectives and constraints.

- Look into approaches to model the knowledge of experts on the compatibility of services (e.g legal and image and virtual machine compatibility) in the system. Then, identify an approach to query the knowledgebase for filtering the non-compatible services.

- Investigate approaches that can handle vague preference of users to improve ease of use and accuracy of selection techniques.

- Build a Cloud agnostic metadata to persist deployment configurations.

- Examine techniques and languages to build a standard SLA template that makes it convenient to discover necessary monitoring services that have the required capabilities to monitor service level objectives in SLAs.

- Model SLA dependency knowledge and use the knowledge to enhance the dependability of the monitoring service.

- Investigate SLA negotiation strategies that are capable to automatically and dynamically generate offers by considering reliability of providers offers for users and resource utilization for providers.

1.3 Contribution

The steps taken to tackle challenges in each phases of service coordination are listed as follows:

1. A taxonomy and survey that provides key background information, comments, and categorization of existing solutions for service coordination.

2. A semantic-based Cloud service discovery in Multi-Cloud environments.

    - An ontology-based and QoS-aware Cloud service discovery is proposed, which works based on modeling virtual units into one of the most prominent initiatives in Semantic Web services, i.e., Web Service Modeling Ontology (WSMO) [161]. This helps users to deploy their appliances on the fittest IaaS providers based on their QoS preferences when both sides (the providers and users) are not using the same notation to describe their services and requirements.

    - An approach is presented that builds semantically enriched Cloud services (along with their non-functional properties). It creates an aggregated repository of service in WSMO from service advertisements information available in XML. The performance of translation technique is measured for different repository size to prove its scalability.

- A novel semantic-based deployment descriptor is offered to persist deployment configurations.

3. A service (virtual appliance and virtual unit) selection and composition approach for migrating web application to Cloud.

   - Relevant QoS criteria, namely latency, cost (data transfer cost, virtual machine, and appliance cost), and reliability for selection of the best virtual appliances and units in Cloud computing environments.
   - Two different selection approaches -genetic-based and Forward-checking-based backtracking (FCBB)- have been proposed to help users deploying network of appliances on multiple Clouds based on their QoS preferences. For that purpose, various types of requests (with different network load between appliances) are generated and data from 12 real Clouds was collected.
   - Effects of factors such as latency requirements and data communication on the cost of appliance placement and the selection of providers are investigated.
   - A ranking system is proposed for Cloud service (i.e. virtual appliance and unit) composition that let users express their preferences conveniently using high-level linguistic terms. The system then utilizes evolutionary multi-objective optimization approaches and a fuzzy inference system to precisely capture the entered preferences for the ranking purpose.
   - An approach is presented to help non-expert users with limited or no knowledge on legal and virtual appliance image format compatibility to deploy their services flawlessly. For this purpose, we first automatically build a repository of Cloud services using WSMO and then enrich it with expert's knowledge (from lawyers, software engineers, system administrators, etc) on the aforementioned constraints. The knowledgebase then is used for reasoning that identifies whether a set of Cloud services consisting of virtual appliance and units are compatible or not. Furthermore, execution time of the compatibility checking approach is measured for different number of appliances in the composition and for different number of discovered appliances, which demon-

strates the scalability of approach.

4. A dependable SLA management system.

   - A semantic SLA template is proposed that is understood by all parties including providers, users, and monitoring services. The semantic SLA contract is defined in a way that can be used as a goal for discovery of necessary monitoring services.
   - SLA dependencies are modeled using Web Service Modeling Ontology (WSMO) to build a knowledgebase that is exploited to eliminate the effects of SLA failure cascading on violation detection. The proposed approach was tested in a case study which proves its effectiveness.
   - A negotiation strategy is proposed that acquires user preferences and provider's resource utilization status and utilizes time-dependent tactic along with statistical methods to maximize the Cloud providers profit while adhering to deadline constraints of users and verifying reliability of providers' offers. Through a series of experiments, we investigate the effect of modifying parameters of the time-dependent tactic, such as initial offer value and deadline, on negotiation outputs including social welfare and success of negotiation in Cloud environments. In addition, the offered negotiation strategy is tested for multiple workloads and in diverse market conditions to show how time-dependent tactics settings can dynamically change to help Cloud providers to increase their profits.

1.4 Thesis Organization

Works presented in this thesis have been to a degree or completely derived from the set of papers published during the course of the PhD candidature. Figure 1.3 shows the relationship among the chapters and the category they belong to. The remainder of this thesis is organized as follows:

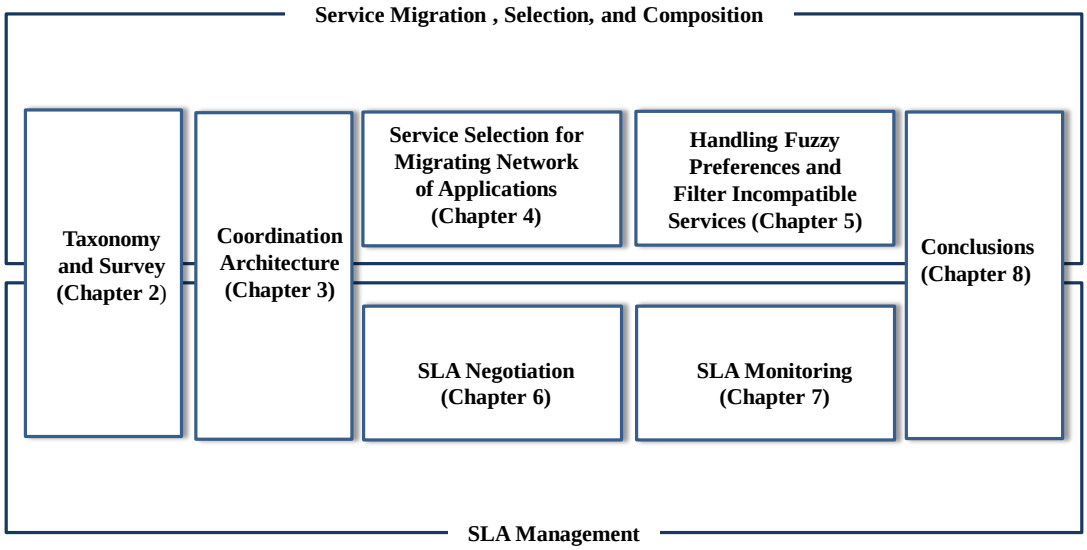


Figure 1.3: Thesis organization.

- Chapter 2 presents a taxonomy and survey of Cloud service coordination, requirements analysis, scope of this thesis, and its positioning in regards to existing works. The chapter is partially derived from:

  – **Amir Vahid Dastjerdi** and Rajkumar Buyya, A Taxonomy of QoS Management and Service Selection Methodologies for Cloud Computing, Cloud Computing: Methodology, Systems, and Applications, L. Wang, Rajiv Ranjan, Jinjun Chen, and Boualem Benatallah (eds), ISBN: 9781439856413, CRC Press, Boca Raton, FL, USA.

- The proposed architecture used for service coordination in multi-Cloud along with an ontology-based Cloud service discovery are described in Chapter 3 which is partially derived from:

  – **Amir Vahid Dastjerdi**, Sayed Gholam Hassan Tabatabaei, Rajkumar Buyya. An Effective Architecture for Automated Appliance Management System Applying Ontology-Based Cloud Discovery, The 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2010), IEEE Computer Society Press, May 17-20, 2010, Melbourne, Australia.

- Chapter 4 describes a service selection mechanism for migrating network of applications to Multi-Cloud. This chapter is derived from:

  - **Amir Vahid Dastjerdi**, Saurabh Kumar Garg , Omer F. Rana, and Rajkumar Buyya, CloudPick: a Toolkit for QoS-aware Service Deployment Across Clouds, Journal of Automated Software Engineering, 2012, in review.
  - **Amir Vahid Dastjerdi**, Saurabh Kumar Garg and Rajkumar Buyya, QoS-aware Deployment of Network of Virtual Appliances across Multiple Clouds, Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science (IEEE CloudCom 2011, IEEE CS Press, USA), Athens, Greece, Nov. 29 - Dec. 1, 2011.

- Cloud service composition challenges, and a mechanism which handles vague preferences of users and filter incompatible services are investigated in Chapter 5. Chapter 5 derives from:

  - **Amir Vahid Dastjerdi**, Yoganathan Sivaram, and Rajkumar Buyya, Multi-objective and Ontology-based Cloud Service Composition Under Fuzzy Preferences of Users in Clouds, Journal of Future Generation Computer Systems , 2013, in review.

- Chapter 6 investigates strategies that are capable of automatically and dynamically conduct SLA negotiation in multi-Clouds and is derived from:

  - **Amir Vahid Dastjerdi**, and Rajkumar Buyya, An Autonomous Time-dependent Negotiation Strategy for Cloud Computing Environments, IEEE Transactions on Parallel and Distributed Systems, 2012, in review.
  - **Amir Vahid Dastjerdi**, and Rajkumar Buyya, An Autonomous Reliability-aware Negotiation Strategy for Cloud Computing Environment, Proceedings of the 12th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2012), Ottawa, Canada, May 13-16, 2012.

- A dependable SLA monitoring approach in a multi-Cloud environment which is

capable of monitoring service discovery and handling of SLA dependency is presented in Chapter 7. This Chapter is derived from:

  - **Amir Vahid Dastjerdi**, Sayed Gholam Hassan Tabatabaei, Rajkumar Buyya. Dependency-aware Ontology-based Approach for Deploying SLA Monitoring Services in Cloud, Journal of Software: Practice and Experience, Volume 42, Issue 4, pp. 501-518.

- This thesis is concluded in Chapter 8 by discussing conclusions and future research directions.

Chapter 2

Taxonomy and Survey of Cloud Service Coordination Methodologies

This chapter presents an overview of services coordination concepts and methodologies. Service coordination has been applied in different computing paradigms namely SOA, and Grid. A number of approaches with variety of architectures and algorithms have been proposed to tackle the challenges in service coordination. The aim of this chapter is to create a comprehensive taxonomy to categorize theses approaches and present them in such a way that identifies gaps in this area of research. Furthermore, this chapter indicates how the approaches in SOA and Grid can share their contributions to deal with service coordination challenges in Cloud.

2.1 Background

CLOUD [18] is a type of parallel and distributed system consisting of a collection of virtualized computing resources that are provisioned on-demand and offered as one or more unified service(s) based on service-level agreements established through negotiation between the Cloud provider and users. Clouds can be classified according to their service types and deployment models. There are three types of services as shown in Figure 2.1:

1. **Infrastructure as a Service (IaaS):** It provides resources that have been virtualized (virtual units) such as virtual computer, database system, or even a virtual cluster. Among IaaS providers, Amazon Elastic Compute Cloud (Amazon EC2) has attracted considerable attention. Amazon EC2[1] provides the flexibility to choose

[1]Amazon EC2. `http://aws.amazon.com/ec2/`

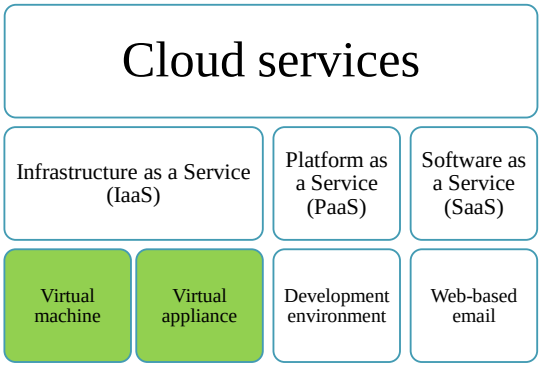


Figure 2.1: Cloud service models.

from a number of different virtual machine (instance) types to meet various computing needs. Each instance provides a predictable amount of dedicated compute capacity and is charged per instance-hour consumed.

2. **Platform as a Service (PaaS):** It provides developers with an environment that supports all phases of software development from coding to deployment. Therefore, developers can concentrate on their main duty of creating efficient software instead of building an environment for running their software. For example, Google App Engine[2] makes it easy to build an application that runs reliably, even under heavy load.

3. **Software as a Service (SaaS):** SaaS is the closest layer of Cloud Computing to the end users and offers software in utility-based model. A SaaS user is not aware of underlying infrastructure. The main advantage of adopting SaaS for users is saving on upfront license or infrastructure costs. A distinguished example for SaaS is Salesforce[3] which can keep track of a business customers and manage its budget at the same time.

Furthermore, Clouds can be classified based on their deployment models as:

1. **Public Cloud:** It offers Cloud services on the Internet to clients or other service providers. In addition, public Cloud services are not restricted and are available to general public.

[2]Google App Engine. `http://appengine.google.com/`
[3] Salesforce. `www.salesforce.com/au`

2. **Private Cloud:** In contrast to public Clouds, Private Cloud services can be accessed and managed exclusively by people of an organization.

3. **Hybrid Cloud:** It offers services which are deployed on two or more Clouds. Therefore, it enables data and application portability among participating Clouds.

4. **Multi-Cloud:** Similar to Hybrid Cloud, it offers services deployed on two or more Clouds. However, the underling Cloud infrastructures are restricted to public Clouds.

As it has been mentioned in Chapter 1, this thesis investigates service deployment in Multi-Cloud environments. We mainly focus on IaaS services such as virtual machines and virtual appliances to satisfy user requirements as shown in Figure 2.1. Service deployment in Multi-Cloud (the process of making a service ready for use) often includes deployment of multiple interrelated software components into heterogeneous environments. Service deployment in Cloud consists of multiple phases which mainly focus on discovering application required by users, and deploying them on the appropriate IaaS Providers. These phases altogether called Service coordination [98], which effectively and consistently provides discovery, selection, and SLA management solutions for a given context. In Service Discovery, user requirements are used as an input for discovery of the best suited Cloud services among various service repositories of providers. For SLA Negotiation, the user and the discovered providers negotiate on the quality of services. In the selection phase, an SLA contract will be selected from a set of made agreements. Further, the acquired service will be continuously monitored in the Monitoring phase.

We provide the key background information and categorize and comment on existing solutions for all service coordination phases. While a number of approaches with variety of architectures and algorithms have been proposed for service coordination in Grid and SOA contexts, Cloud service coordination is in its infancy and there are many open problems. Therefore, we investigate new challenges in service coordination specially in the context of Cloud computing. Next, we identify what can be extracted from previous works in the context of Grid and SOA to tackle those challenges. In addition, survey and taxonomy are presented in a way that identifies gaps in this area of research, and helps up to position this thesis. We start the classification by reviewing service discovery

techniques.

2.2 Discovery

Service discovery is a procedure of searching for required services which their functional and non-functional semantics satisfy a users goal. In the SOA context, the typical architecture of a web services includes three roles, namely service user, service broker and service provider. Once the user sends a service request to the broker, matching service providers should be sent back by the broker. If the broker finds a set of services, which all satisfy the functional requirements of a user, the crucial issue is how to select the fittest service based on the user preferences. Current approaches to service discovery can be divided into categorizes as depicted in Figure 2.2. In following, we survey existing approaches for service discovery based on the presented classification in Figure 2.2.

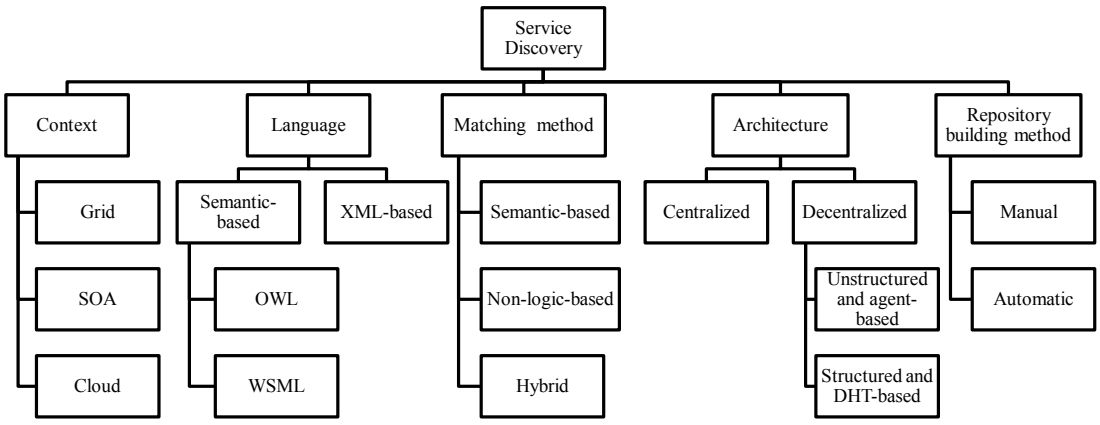


Figure 2.2: Taxonomy of discovery approaches.

2.2.1 Non-logic Based Discovery

Approaches in this category of service discovery apply symmetric attribute-based matching between requirements and a request. For example, IBM smart Cloud provides syntactic matching discovery system to choose the appropriate Cloud provider, instance type, and asset catalog for the deployment. A similar discovery strategy is offered by Amazon EC2, GoGrid and Rackspace. For Globus [151] the resource discovery service is part of

the so called Monitoring and Discovery Service (MDS). MDS makes use of WSRF (Web Services Resource Framework) standards and has a centralized information indexing service similar to UDDI [169], which cannot support complex queries. Similarly, Condor [111, 142], Gridbus broker [17], and Inter-Cloud [19] apply a centralized attribute-based matchmaker to match requests to resources.

2.2.2 Semantic-based

Non-logic based discovery systems in grid and Cloud (IBM Smart Cloud Catalog search, Amazon EC2 image search) require exact match between a client's goal and a provider's service description. In a heterogeneous environment such as Cloud, it is difficult to enforce syntax and semantics of QoS descriptions of services and user requirements. Therefore, applying symmetric attribute-based matching between requirements and a request is impossible. Building semantics of Cloud services, user requirements, and data would provide an inter-Cloud language which helps providers and users share common understanding regarding the Cloud service functionalities, QoS criteria, and their measurement units. A semantic service is a result of a procedure in which logic-based languages over well-defined ontologies are used to describe functional and non-functional properties of a service. Discovery approaches can be further categorized based on logic-based languages they support as following:

WSMO-based Matchmaking

Web Service Modeling Ontology (WSMO) [145] defines a model to describe Semantic web services, based on the conceptual design set up in the Web Service Modeling Framework (WSMF) [54]. WSMO identifies four top-level elements as the main concepts [55]:

- Ontologies, provide the (domain specific) terminologies used and are the key elements for the success of Semantic Web services. Furthermore, they use formal semantics to connect machine and human terminologies.
- Web services, are computational entities that provide some value in a certain domain. The WSMO Web service element is defined as follows:

  - Capability: This element describes the functionality offered by a given service.
  - Interface: This element describes how the capability of a service can be satisfied. The Web service interface principally describes the behavior of Web services.

- Goals, describe aspects related to user desires with respect to the requested functionality, i.e. they specify the objectives of a client when consulting a web service.

- Mediators, describe elements that handle interoperability problems between different elements, for example two different ontologies or services. Mediators can be used to resolve incompatibilities appearing between different terminologies (data level), to communicate between services (protocol level), and to combine Web services and goals (process level).

Besides these main elements, non-functional properties such as cost, deployment time, security, scalability, and, reliability are used in the definition of WSMO elements. Furthermore, there is a formal language to describe ontologies and Semantic Web services called WSML (Web Service Modeling Language) which contains all aspects of Web service descriptions identified by WSMO. In addition, WSMX (Web Service Modeling eXecution environment) is the reference implementation of WSMO, which is an execution environment for business application integration [74].

In addition, WSMX provides centralized semantic-based service discovery [92] which uses WSMO descriptions of goals and web services. The discovery engine in WSMX enables set-based discovery, and further enhances the discovery by adding a caching mechanism [157] and two-phase discovery technique to improve the discovery performance. The two-phase discovery technique advances the previous approach by differentiating two types of goals. First a generic objective and preference description is created at design time which is called goal template. Next, at runtime the goal template is instantiated with clients input values. Therefore, matched services for goal templates are identified and stored at the design time and stored services are additionally filtered based on user inputs at runtime. By reducing number of reasoning and matchmaking operations, this

approach improves the performance compared to the previous WSMX discovery technique.

OWL-S based Matchmaking

Enabling automatic web service discovery was one of objectives of creating OWL-S [118]. To achieve this, a software agent needs a computer-interpretable description of the service. Therefore, OWL-S offers a semantic web markup language which establishes a framework within which these descriptions are made and shared. OWLSM [83] and similar other efforts [130] proposed discovery (Input Output matching) for services which are described in OWL-S. OWLSM [83] was among the first efforts to offer ranking results based on different matching degrees instead of retuning only success or fail.

Claiming that previous OWLS-based matchmakers fail to fully capture the functionality of semantic services, Averbakh et al. [6] proposed a method that further improves the discovery results by incorporating users' feedbacks. The method is capable of dealing with limited amount of feedbacks by acquiring required information from similar requests. Once experimentally evaluated and compared with OWLS-MX using a standard semantic service collection, results confirm that utilizing users' feedbacks improves the matchmaking quality.

2.2.3 Building Semantic-based Service Repository

XML Schemas are used to describe the data model of WSDL services, however the data model for semantic services is described using the conceptual model provided by ontologies. Thus, in order to build semantic service repository we require mapping between XML data and its corresponding representation in ontology. The mapping can be achieved manually, which is time consuming and error-prone or automatically, which is called data Grounding. Data Grounding has been investigated intensively [77, 95], however we only mention two fundamental technologies which were implemented and their efficiencies were verified. Ferdinand et al. [56] proposed a Grounding approach with two kinds of data transformations that maps XML Schemas to OWL ontologies and

XML documents into RDF graphs. Likewise, WSMO Grounding [100] applies a similar method but with only one single set of mappings between XML Schema elements and WSMO ontology meta-model.

2.2.4 Hybrid Matchmaking

Computational expensive nature of logic-based reasoning required for semantic-based discovery has motivated researchers to build hybrid matchmakers. Hybrid matchmakers are claimed to enhance the quality of service discovery by incorporating both syntactic and semantic matching. WSMO-MX [97] and OWLS-MX [96] are examples of hybrid matchmaker.

WSMO-MX is a hybrid and IOPE (input, output, precondition, and effect) matchmaker built for services described in WSML. WSMO-MX applies varieties of logic-based and non-logic based techniques to search for services which are semantically close to a given goal. Services in WSMO-MX are described in WSML-MX which is an extension of WSML. The objective of WSML-MX is to enrich WSML-RULE with additional language elements that allow users to impose relaxation constraints and set preferences for matchmaking. The performance of the hybrid matchmaker is measured based on precision, recall, and computation time. Experimental results show that the hybrid approach outperforms both logic-based and syntactic-based approaches. It decreases the computation time and at the same time performs similar to the logic-based approach with regards to precision and recall.

2.2.5 Decentralized P2P Discovery

When service descriptions are distributed over repositories located in different locations, this class of discovery approaches are helpful.

Structured and DHT based

Distributed Hash Table (DHT) operates on a group of services descriptions which are distributed among repositories in a network. It controls content distributions for efficient

routing queries. DHT provides a distributed and key-based lookup protocol which is similar to hash tables.

An approach for logic-based and QoS-aware discovery of semantic services, which are advertised in P2P-based repositories, is proposed by Vu et al. The objective of the work is to map semantic services to repositories in such a way that finding the repositories with the best matching services is much quicker. The idea is to group services that are operating on specific set of concepts (services which their inputs and outputs concepts are similar) in to a same repository and thus giving them a same hash key. Distributed hash table is used by them to search for the repository which corresponds to a particular key. The proposed discovery approach is scalable in terms of the number of registries, users, and service properties.

Basic DHT is efficient for single-dimensional queries. However, as a Cloud service is characterized by several functional and non-functional properties a search query is always multidimensional. These search dimensions or attributes can include service type, processor speed, architecture, installed operating system, available memory, and network bandwidth. Therefore, Ranjan et al. [144] proposed a Cloud service discovery by extending the DHTs for indexing complex Cloud service provisioning information.

Unstructured and agent-based

This family of discovery techniques have built a unique decentralized discovery approach by combining agent and semantic technology. This gives the discovery approach the flexibility of allowing resources (which are represented by agents) to interact semantically and form a composition which is capable of completing a given task. In the approach offered by Han et al. [75], to form the composition, the first resource agent is chosen randomly and all the other consecutive agents are selected by calculating semantic similarity. The experimental results show that success probability of the discovery algorithm increases with the lower similarity threshold, higher task load, and larger resource network scale.

Service discovery on its own cannot map user requirements to a specific Cloud service. This is because a result of discovery phase can be a set of candidates which all satisfy

the functional requirements of users. Therefore, a service selection and ranking methodology is desirable to find and select the most prominent service based on user preferences on non-functional properties. Next section presents and classifies major service selection approaches in the literature.

2.3 Service Selection Taxonomy

Service selection has been applied in different computing paradigms, namely SOA, and Grid. This is because a resource can be represented by web services and web service selection approaches help to assign each request in the queue to the most proper resource in a fair manner. In addition, recently Cloud Computing has emerged as a promising paradigm in which the platform and even the infrastructure are represented as service, therefore Cloud selection plays a significant role and will attract lots of attention. Considering the popularity of Cloud computing, it is highly possible to conduct redundant researches in this area. A sizable body of literature exists in the context of SOA and Grid that can share their contributions to tackle selection problems in the Cloud. Consequently, the aim of the taxonomy presented in this section is to identify state-of-the-art challenges in web service selection and extract a general model for service selection. In addition, it aims at classifying works based on how they model QoS attributes and how they approach selection.

In the next sections, a general model for service selection is presented. The model consists of two parts. The first part describes steps that have to be considered for QoS management and the second part deals with the process of web service selection based on the QoS preferences and service descriptions provided.

2.3.1 QoS Management

As depicted in Figure 2.3, several steps have to be taken into account for QoS management. In general, a QoS management approach for QoS-aware Web service selection includes the following phases:

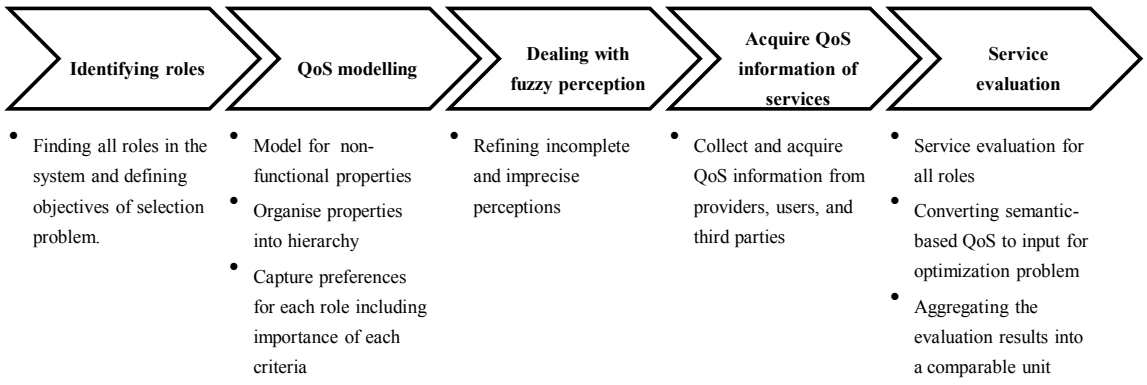


Figure 2.3: QoS management process.

1. **Identifying Roles:** Provider and client (requester) are the two basic roles in all service selection problems. The objective of selection is determined based on the defined roles in the problem. Selection solutions usually try to maximize profit of providers, requesters, or both.

2. **QoS Modeling:** QoS can be described in user preferences to express their expectations. It also can be included in a service advertisement when there are different service providers that present diverse versions of services to answer varying requirements of their customers. User preferences and service offerings have both functional and complex non-functional aspects, which need to be matched against each other [182]. Moreover, the model should give the user the ability to express which QoS criteria are more important for them and provides a way to define the relations between QoS criteria. For example, one user may prefer a service with better response time compared to a service with a lower price and the other may prefer the less costly service. In addition, it increases the transparency if we place QoS properties into hierarchical structure. For example, in the hierarchical structure, throughput and response time are performance aspects while security and privacy are safety aspects. commonly ontology is used to build the hierarchy [164,165].

3. **Taking Care of Users Fuzzy Perceptions**: Dealing with fuzzy perceptions of users is another important task in QoS management. The reason is that ranking system are used by non-experts who find it difficult to express their preference through utility functions precisely[172].

4. **Collecting QoS Information:** In this phase, a proper approach to collect and acquire QoS information is needed. Even though, some works [161] believe users are responsible for development of QoS information, others [170, 174] assume that service providers are supposed to offer QoS information along with their service descriptions.

5. **Aggregating the Evaluation Results into a Comparable Unit**: This phase includes transferring semantically described QoS to an input for an optimization algorithm [62]. It is essential to aggregate QoS criteria and sub-criteria scores to gain a final score for the service. In this step, a suitable aggregation method needs to be selected [182].

2.3.2 Process of Service Selection

After QoS values are acquired, they are used as an input for the selection approach which finds the most preferable services. Following selection phases are commonly considered:

- In the first phase, formulating and modeling of the problem is accomplished. This includes finding constraints and objectives for the selection problem. For example, in a work done by D. Tsesmetzis et al. [165] the bandwidth has been considered as a constraint and the objective is to minimize the cost.

- Next, the selection problem is tackled by a proper optimization or decision making technique which suits the modeled problem.

2.3.3 Service Selection Context

Service selection has been investigated in different computing paradigms and most importantly for Grid, and SOA. This section shows how characteristics of the problem in each realm differ from the others. The objectives of studies in each context are summarized and illustrated in Figure 2.4.

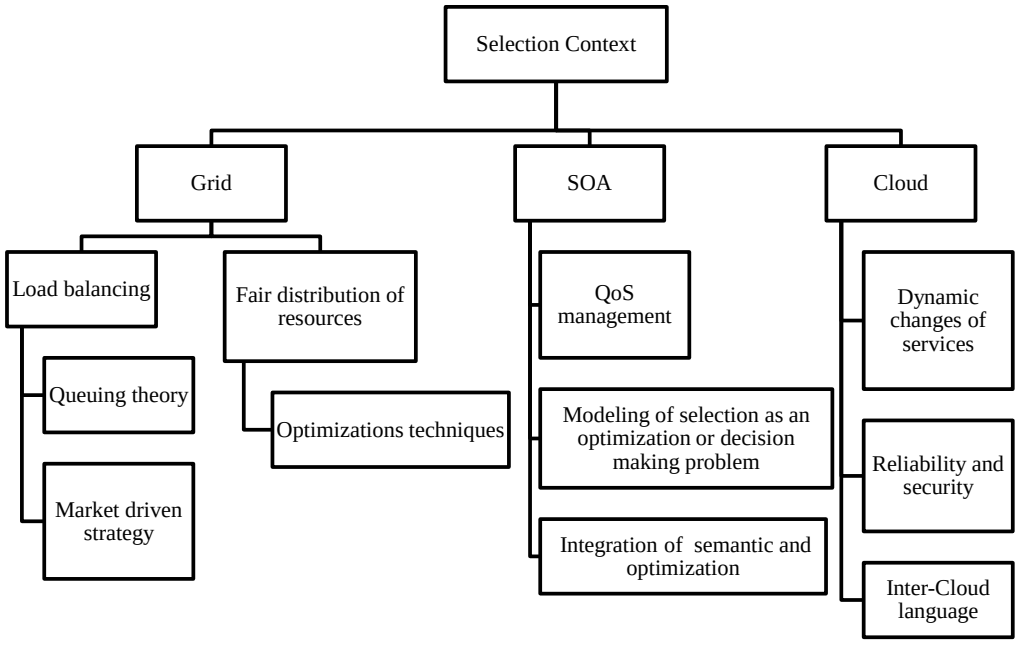


Figure 2.4: Selection researches in different contexts.

Grid Computing

Grid Computing aims to enable resource sharing and coordinated problem solving in dynamic, multi-institutional virtual organizations [59, 61]. The goal of such a paradigm is to enable federated resource sharing in dynamic and distributed environments. QoS management and selection works in this context mainly focus on load balancing [16] and fair distribution of resources among service requests. Besides that, there are cases where a service on its own can not satisfy user requirements, and a composition of services is required. The problem of finding a composition which satisfies user requirements have been also studied extensively in this context.

Load Balancing As Xiaoling et al. [174] quoted that because web services are highly dynamic and distributed, it is likely that loads on service providers are not distributed symmetrically. Consequently, the work approaches the problem by applying the queuing theory to assign requests to service providers that have the least load. Hence, the system obtains all providers which can serve the request and choose

the one which has the minimal expected waiting time. The Expected Waiting Time (EWT) is calculated based on Equation 2.1.

$$EWT = E(w) = \frac{1}{\mu}\left(\frac{\rho}{1-\rho}\right) = \frac{\lambda}{\mu(\mu-\lambda)} \tag{2.1}$$

where λ is the arrival rate, which specifies the number of requests the provider served in the time unit. This can be calculated by Equation 2.2.

$$\lambda = \frac{NUM}{elapstime} \tag{2.2}$$

In the Equation $elapstime$ is calculated from the point when the web service provider is started, and NUM is number of requests that satisfied during $elapstime$. Performance evaluation for the approach shows the selection strategy reduces total waiting time in general and works better than random selection when the provider's capacity is limited and low.

Wancheng et al. [170] uses the price-demand rule in commodity-market to improve load balancing. They discuss that using best-effort strategy to serve users leads to an unbalanced system because of greedy competition for the best services. In their approach, users and providers are considered as buyers and sellers in a commodity-market to adjust the supply and demand [16]. In addition, the selection strategy offers "enough" QoS as a substitute for "best QoS". The evaluation model for web service selection introduces the relationship between price of services and value of their attributes. The service selection problem is modeled as a 0-1 multi-dimensional knapsack problem. The demands of web services are determined by their invocation rate. Therefore, it increases the price when the invocation rate is high and vice versa to obtain a balanced system.

Fair Distribution of Resources Guha et al. [73] proposed an algorithm, which enables optimal selection while considering fair distribution of resources among users. First, they analyzed traditional matchmaking algorithms [114]. The algorithms take the first request and then map it to the service with the highest score. Then, they se-

lect the second best for the second request in the list. The major drawback of this kind of algorithm is that requests placed later in the queue are mapped to services with poor match scores. One way to tackle this problem is applying a constraint-satisfaction-based matchmaking algorithm (CS-MM). The algorithm's effectiveness appears when several requests are assigned to a single service which can only serve one request at a time. Therefore, the algorithm selects the service for a request, which has the lower second highest match score.However, to achieve better results when Grids process a large number of concurrent requests for particular services the Multiple Objective based Particle Swarm Optimization Algorithm using Crowding Distance (MOPSO-CD) is adopted. The algorithm is swarm based artificial intelligence algorithm [33, 79, 133] and was inspired from the nature of bird flocking. The experimental results in the paper show that, although the MOPSO-CD takes considerably more time to be executed, it produces a far more accurate solution than CS-MM.

With the aim of fair distribution of Grid resources (services) among requests, Ludwig et al. [114] presented a service selection approach for Grid computing based on a Genetic Algorithm (GA) and compared its performance with traditional matchmaking methods. They are five quality of service criteria: execution duration, price, reputation, reliability, and availability. First, the paper proposed a scenario in which many service requests coming into the system are to be served at the same time. Next, it compares matchmaking and genetic algorithm approaches. The matchmaking algorithm uses a weighting approach for selection and chooses the service with the highest score for each request. The main weakness of this method is that requests which are placed later in the list are likely to be mapped to services with poor match scores. To tackle this problem the paper applies the NSGA-II genetic algorithm. The genetic algorithm was tested for several populations and the average matching score was compared with the matchmaking algorithm. Results show that if a proper population size is chosen for NSGA-II, it can deliver better performance compared to traditional matchmaking algorithms. The work solved the fairness problem, however introduced a new problem of finding a proper population size

for the GA.

Service Oriented Architecture

In this paradigm, the sizable body of literature has focused on describing QoS for services and user preferences, building QoS ontology, and proposing optimization approaches for multi-criteria web service selection. For QoS description, the semantic web service has been applied [62] in SOA to increase expressiveness, flexibility, and accuracy. Semantically described QoS information then has to be converted to comparable units to be proper inputs for optimization techniques[62] for selection.

Quality of Service Description Toma et al. [161] discuss various approaches and their advantages and disadvantages for modeling QoS. Initially, the work starts with discussing three main approaches to process QoS in SOA, namely: combined broker, separate QoS-broker, and direct negotiation. The combined broker approach [152] is an extended version of UDDI in which a broker is extended to process QoS information. In the second method [119, 171] the devoted broker is responsible for processing QoS. And the third one requires provider and user to negotiate and agree on service level agreement [113,163]. Subsequently, the paper talks about current supports of QoS in WSMO with the help of non-functional properties. WSMO proposes a number of non-functional properties for each element in a service description. The description of QoS is based on the Dublin Core Metadata Initiative [175]. Finally, the article offers three techniques to extend WSMO/WSML support for QoS. The first method specifies each QoS property through the use of relations in WSML. Therefore, there is no need to add separate vocabulary for QoS. The second technique is to define a new concept for QoS. And through the appropriate property, we can include relationship between non-functional properties. The third approach is to model them like capabilities in WSML. The main disadvantage of this approach is the need for extending WSML syntax. Among the three, the third approach seems to be the most suitable one, as it can provide the full support of QoS. However, this approach has to be developed from scratch to deliver clear syntax and proper ontology for reasoning.

Integration of Semantic and Optimization In addition, with the emergence of semantic web services, some of the works in this context concentrate on filling the gap between semantic-based solutions and optimization solutions [62]. Furthermore, some efforts [3,112,170] describe the selection problem as a part of the service composition problem.

J. Garcia et al. [62,63] present a framework to transform the user preferences which are in the form of ontologies to an optimization problem's inputs. The authors argue that semantic web services define ontologies that help web services to be discovered and composed automatically. As they use description logic for those purposes, they are infertile in dealing with QoS-based selection. The reason is that QoS-based selection approaches require optimization that cannot be accomplished by applying description logic. The authors build their selection approach on their previous efforts [64], which apply semantically described utility functions to define user requirements. They propose a selection approach that transforms user preferences into an optimization problem. That optimization problem can be tackled using various techniques, such as constraint programming or dynamic programming. The selection approach consists of the following phases. In the first phase, QoS values are retrieved from the semantic description of web services. Proceeding to the next stage, the previous phase results are linked to the user preferences, which express utility functions for related QoS criteria. In the last stage, an XSL transformation has been applied to the utility function to acquire the specification of the desired optimization problem. The optimization problem chosen for the work is the Constraint Satisfaction Optimization Problem (CSOP) [146]. However, other optimization techniques can be supported by designing a proper XSL style sheet.

2.3.4 Service QoS Modeling Taxonomy

Currently there are two ways to define QoS attributes for web services namely, extended Universal Discovery, Description, and Integration (UDDI), and semantic web services. However, UDDI and Web Services Description Language (WSDL) [103] do not support modeling of QoS properties of web services. Therefore, works [12, 143] such as UDDIe

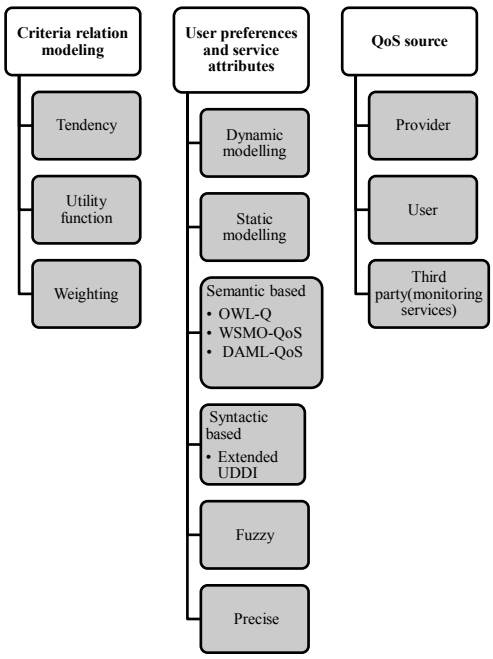


Figure 2.5: Web service QoS modeling taxonomy.

[152] and web service level agreement [113] were proposed to enrich web services with QoS properties. In the following subsections classifications for QoS management are presented and their summary is illustrated in Figure 2.5.

User Preferences and QoS Criteria Relation and Tendency Modeling

When users express their expectations from services, they identify functional and non-functional (QoS) characteristics of the required services. In addition, they have to identify which of the QoS criteria are more important than the others. A simple way to perform this is to ask users to give assign to each criterion. Using weights to achieve a decision matrix is one of the primary ways of modeling importance of criteria in user preferences. This approach has been applied in many works [54, 118, 173] as it is simple and computationally efficient. However, the major drawback concerning this approach is the complexity of finding proper weight coefficients in real world applications. In addition to the importance of criteria, [173] it has to be identified whether a parameter value is more desirable for a particular user when it is smaller or greater. There can be some other tendencies assigned to QoX properties like "exact" and "close" [164].

Moreover, modeling of relations between QoS criteria have been also investigated. For example, Tsesmetzis et al. [165] discussed the importance of QoS consideration for service providers and users to help providers attract more customers. As the first step, the work creates a QoS vocabulary ontology in OWL [152] with the maximum height of two to reduce the complexity. The QoS vocabulary covers a wide range of non-functional properties from performance to security and reliability. Then, it applies a standard generic model [131] for defining an association between QoS attributes and the approach for measuring them. A comprehensive work on the relation modeling is done

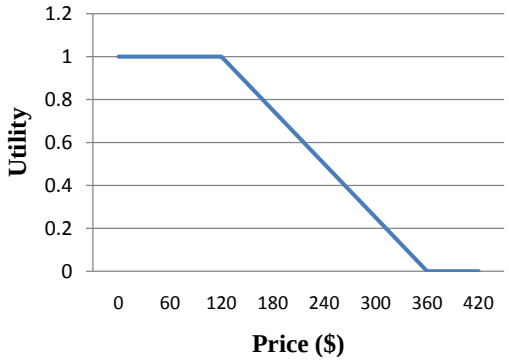


Figure 2.6: Price utility function.

by Qing [181]. In the work, a mechanism for ranking web services using logic scoring preferences (LSP) [46] and ordered weighted averaging (OWA) [22, 57] is proposed. The authors pointed out that current ranking algorithms ignored relations between individual criteria and the simple arithmetic metric is incapable of representing relations such as simultaneity and replaceability. Authors claim that those drawbacks can be addressed by adapting LSP and OWA. The work makes use of LSP, which was originally developed for solving hardware selection, and considers the relation between criteria of selection such as replaceability, simultaneity, and mandatory-ness. However, since the work is based on LSP, it cannot deal with selection problem with many QoS criteria.

Utility function recently has been used [63, 104, 106, 146] and it is said [63] to be the most appropriate way to expressively model user preferences. Therefore, works in Cloud computing environments can adopt it for service selection. Utility function is a normalized function and shows which values of QoS criteria are preferable. For the selection of the best alternative, all of the utility functions have to be aggregated to compute the

global utility value. J. Garcia et al. [63] proposed a new approach of ranking based on the description of user preferences in the form of a utility function. Authors presented a hybrid architecture for service ranking by adding support for Web Service Modeling Ontology (WSMO) [40, 145] to describe QoS. For dealing with several non-functional properties, each utility function was associated with a relative weight. Therefore, to solve a multi-criteria ranking problem, the user preferences were calculated as a weighted composition of the associated utility functions. It is worth to mention that, user preferences definitions are inserted as part of a goal in the form of WSML. Figure 2.6 shows a sample of utility function extracted from their work. As depicted in Figure 2.6, the highest utility value is returned by the function if the price is lower than 120 and the returned value drops when the price increases.

In addition to those methods, Tran et al. [164] adopted Analytic Hierarchy Process (AHP) [148] for QoS criteria relation modeling and service selection. AHP consists of three main phases, problem breakdown, comparative evaluation, and priority composition. In the first phase, each problem is decomposed to three elements, the overall goal, its criteria and sub-criteria. Next, pairwise comparisons for all criteria and sub-criteria will be done to obtain their relative importance for decision making. Subsequently, all solutions are ranked locally applying those sub-criteria. In the final stage, all relative local ranks of solutions are combined to obtain the overall rank for the solution. Since the method uses pairwise ranking, its performance decreases when the selection problem consists of large number of criteria. Moreover, the authors enhance flexibility of ranking algorithm by allowing two options of mandatory and optional QoS constraints. It offers more options (exact and close) for tendency characteristics of QoS criteria in addition to negative and positive options considered by other studies [63, 162]. The "exact" option shows that the property value should be equal to the defined value for the request. And the "close" option says that a value close to the requested value is more desirable.

Furthermore, Wang et al. [172] presents a novel resolution process for determining the linguistic weight of QoS criteria. First, it creates the framework for evaluation QoS criteria, which summarizes QoS requirements of web services from opinions of decision makers and market surveys. Next, it determines the importance of criteria by aggregating

all opinions of participants. The approach is a complement to other previous studies [39,159], which help to select web services in market places based on QoS.

QoS Source: Provider, User, or Third Party

The majority of studies assume that QoS information is supplied by providers along with services description. However, some believe [172] that not all providers are willing to supply the related QoS information for comparison, or that they are likely to advertise their services exaggeratedly. That is why considering consumers feedback on their experience of using web services determine QoS values more accurately. In addition, there are values that cannot be determined by users or providers, such as network related QoS information, reliability, and trust, which is usually evaluated by a third party (monitoring services). To collect QoS information, Zhenyu et al. [112] built an approach based on distributed agents. The main contribution of the work is building a procedure for processing the quality of web service from multiple locations. It presents a distributed approach to acquire the QoS data from users in different locations with the help of agents scattered in the network.

Context-Aware

There are cases where QoS information of web services vary based on the users' contexts. The context-aware QoS information allows service providers or third parties to publish QoS values for web service based on the user context. Qing et al.[181] adopts the inContext project[4] for providing dynamic context information. This information includes location, budget of users, and availability of services. Therefore, the selection is based on reasoning on the context data. In addition, a selection approach offered by Lamparter et al. [106] is context sensitive as it adopts utility function policies to model context dependent user preferences. For example, there might be a case where a web service selected as the best service in the list is not available in a particular location. Therefore, selecting it for the user in that context is not acceptable.

[4]inContext project.`http://www.in-context.eu`

Context information plays an important role In Multi-cloud service selection. This is because, context information of users helps the service coordinator to improve user's quality of experience. For example, service coordinator can use a client's location information to determine which Cloud is the closest and thus provides service with less latency and higher throughput. Apart form that, there are some restrictions applied by law for deploying on Clouds in specific geographic locations. For example, according to Data Protection Act (DPA), there are cases where transferring data to Clouds located outside the European Union are unlawful for European located companies.

As Papakos et al. [132] discussed, another important QoS information (for service selection in Cloud) to be considered is the user's device. As they have mentioned, requirements of a client with a mobile device can change because of changes in the context of the device. These status changes encompass hardware resources, environmental variables, or user preferences. Binding to a random service offering may lead to excess consumption of mobile resources such as battery life. Therefore, the authors propose VOLARE the middleware that dynamically provision Cloud service based on the context of a user.

Dynamic Versus Static Modeling of User Preferences and Web Service QoS Attributes

The value of QoS properties for a web service can remain constant and therefore identified once, or it can be updated regularly. In addition, user preferences also can be specified once or change during interactions with the system. For example, the service response time is heavily dependent on network traffic. Therefore, it can have a short response time at a moment in a day and then increase dynamically to a certain level when it is not available.

Lamparter et al. [106] pointed out shortcomings of current works like the WS-Agreement [1] in modeling dynamic preferences of users. Moreover, it shows how they can be tackled by a mechanism using utility [93, 107] which are presented in the form of OWL ontology. In addition, authors concluded that the performance of their modeling depends on how expressive services and requests are described. Consequently, if service selection happens in runtime, one way to increase the performance is limiting the expressiveness

of the biding language.

Andrzej et al.[185] have taken the advantage of state-full web services to deal with dynamic changes of Cloud services. They proposed a higher layer of abstraction which provides selection based on QoS criteria values that describe dynamically the state and characteristics of Cloud Services (cluster as a service). In addition, they have implemented their proposed solution to prove its feasibility.

Semantic-Based Versus Syntactic-Based Service Description

There are two ways to describe entities of a web service, namely semantic-based and syntactic-based (extended UDDI). The syntactic-based approach uses numerical and key word values for web service QoS properties. Although an extended version of UDDI can encompass the QoS information of web services, it is not machine understandable, hence not suitable for automatic selection. The automatic selection enables service providers and users to be decoupled, which means they do not have to be aware of each other before execution phase. In addition, different service providers and users can apply a variety of models for describing QoS attributes, and then it is essential to acquire a solution to understand different QoS representations. That solution, which covers the mentioned drawbacks, is semantic web service that increases expressiveness, flexibility, and accuracy by applying ontology for representing QoS properties. A number of QoS ontologies were proposed for web service, which are based on three semantic languages, namely OWL-Q [104, 149], WSMO-QoS [120, 161, 173], and DAML-QoS [188].

OWL-Q is built on top of the OWL-S [118] language and has the strength of not only modeling and measuring static QoS attributes but also dynamic properties. Nonetheless, it does not provide a way of defining importance of QoS criteria and whether they are mandatory or optional. Tran et al. [164] developed a comprehensive OWL-based QoS ontology and an AHP-based ranking algorithm to dynamically rank services.

WSMO-QoS is another promising approach for QoS modeling. It specifies an upper level ontology that describes each attribute of a web service in detail. Moreover, a new category of properties with the name of "non-functional" was added to Goal and semantic service descriptions. The category provides attributes for describing QoS type, metric,

dynamicity level, tendency, and importance.

Finally, DAML-QoS was developed as a complement to DAML-S to be capable of specifying QoS ontology. More specifically, the ontology consists of three layers of QoS profile, QoS property, and QoS metric. The QoS-profile layer is developed mainly to describe services and request ontology for providers and users. The QoS property deals with name, domain, and range of QoS attributes that can be defined in DAML-QoS. And finally, the QoS metric layer is used for identifying how to measure QoS.

Fuzzy Versus Precise Preferences

Commonly, non-expert users have vague preference because of the complex nature of QoS properties [172]. Therefore, Wang et al. [172] proposed new selection algorithms based on MaX-Min-Max composition of intuitionistic fuzzy sets (IFS) under the vague information. The work expects fuzzy perception of users and providers. Therefore, authors suggest a fuzzy multi-criteria decision making solution with following aspects:

- Capable of handling imprecise preferences of users;
- A clear weighting strategy for QoS criteria;
- A QoS-aware service ranking ability.

They use the QoS criteria presented by the W3C working group in 2003. In addition, they mention the fact that not all providers are willing to provide the related QoS information for comparison. That is why they have to consider feedbacks of users on their experience of using web services. As mentioned earlier, the web service selection approach in this work is based on IFS. IFS was introduced in 1989 by Atanassov [5] and can be considered as a generalization of the concept of fuzzy set that is effective in dealing with vagueness. In addition to IFS, this type of problem can be tackled by Fuzzy multi-objective and multi-level optimization [189].

Identifying Roles in the Problem

It is important to determine the objective of the selection algorithm. The selection might aim at maximizing provider profit or satisfying user's objectives. Furthermore, the selection approach [164] can be flexible enough to act for both parties. In details, it can give weight to objectives of users and providers in the utility function. Therefore, by adjusting the weight in the utility function, it can work in favor of either of two parties.

2.3.5 Taxonomy of Web Service Selection Approach

As it is illustrated by Figure 2.7, research on service selection are based on two types of approaches, namely optimization and decision making. Decision-making can be defined as the process of identifying and choosing alternatives based on values and goals of decision makers. Therefore, the assumption is that there are many candidates to be chosen and the aim is to select the one that best fits our goal, desires, and constraints. This process is depicted in Figure 2.8. When there is one single criterion, the selection can be made by identifying the alternatives with the best value for that criterion. However, when there are several criteria as it is depicted in Figure 2.9, it is necessary to define which criteria have higher priorities to users. And this is where the decision making techniques offer approaches such as AHP to help users assign the comparative importance to those criteria. In the case of multiple criteria and a sufficiently small number of explicitly given alternatives, when there is no existing scale of measurement for the criteria, the problem can be solved using decision making approaches such as the Analytical Hierarchy Process (AHP) and the Multi-Attribute Utility Theory (MAUT).

However, if there are large number of alternatives, multiple criteria optimization techniques can be applied. These techniques can be categorized into evolutionary-based and non-evolutionary based techniques. Evolutionary-based techniques (for multi-objective problems) are based on the Pareto solution, which is an economic concept and applied for the condition when a better value for an attribute can only happen once the value of at least one other attribute gets worse. AI solutions, such as the Non-dominated Sorting Genetic Algorithm-II (NSGA-II), Strength Pareto Evolutionary Approach II (SPEA-II),

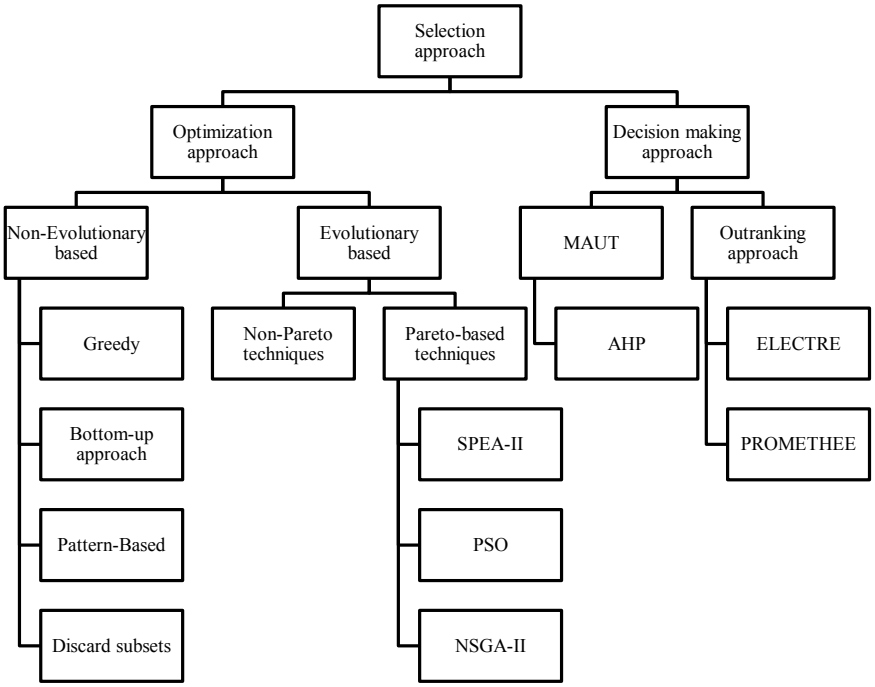


Figure 2.7: Web service selection approaches.

and particle swarm optimization (PSO) are among the most used techniques in this area. In the following sections, each of these selection approaches (from decision making to optimization) will be discussed in detail.

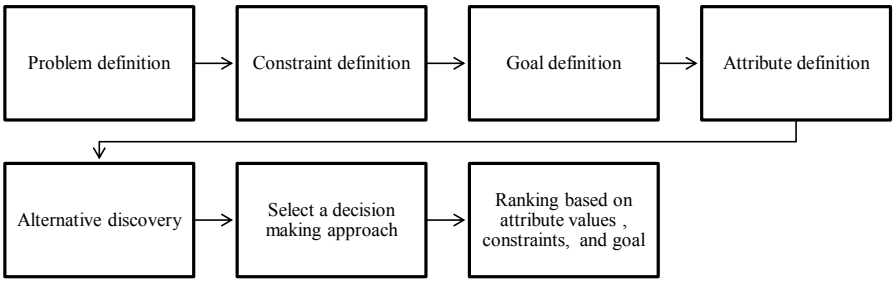


Figure 2.8: Process of decision making.

Decision Making Formulation and Analytical Hierarchy Process (AHP)

The problem of multi-criteria decision making (MCDM) can be considered in the following form: $C_1...C_m$ are the criteria, $A_1...A_n$ are the alternatives, $W_1..W_m$ are weights assigned to criteria, a is a matrix whose a_{ij} element shows the score of alternative A_j

against the criteria C_i, and $X_i...X_n$ are aggregative scores of the alternative A_i. MCDM approaches can be classified into two main categories, namely Multi-Attribute Utility Theory (MAUT) and outranking approach. MAUT is benefited from applying a utility function that has to be maximized. Moreover, it allows the complete payoff between criteria that can show relative importance of each attribute in alternative evaluations. From other work in this category, the distance-minimizing approach [67] can be named. However, the Analytical Hierarchy process (AHP) is one of the most applied methods in the MAUT category, therefore in this section we will investigate it in detail. The AHP method was suggested by Satty in 1998 and is based on a pairwise comparison of criteria to determine their weights in utility function.

The major contribution of AHP is to convert subjective assessments of the relative importance to numerical values or weights. The methodology of AHP is based on pairwise comparisons of the criteria by asking the question of "how important is criterion C_i compared to criterion C_j ?". The answer to this question determines the weight for each criterion. Figure 2.9 depicts the process of choosing the best service provider when there are four criteria (cost , reliability, security, and trust) to be considered. The answers to the question can be one of the responses listed in Table 2.1.

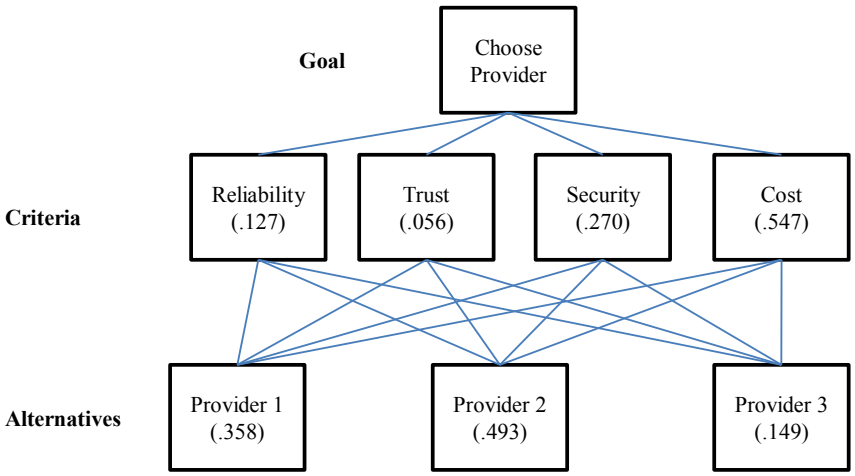


Figure 2.9: Choosing the fittest provider using AHP.

After the pairwise comparison as it is shown in Figure 2.9, relative importance is given to each criterion. In the next step, similar questions have to be asked to evaluate

Table 2.1: Major scales for pairwise comparisons.

Scores	Response to the question
1	Equal importance or preference.
3	Moderate importance or preference of one over another.
5	Strong or essential importance or preference.
7	Very strong or demonstrated importance or preference.
9	Extreme importance or preference.

the scores for alternatives on the subjective criteria. Then, based on the results of this phase, alternatives are ranked and the best provider is selected.

The next category in decision making approaches is named Outranking, which was introduced by Roy in 1968. In this approach, all alternatives are compared in a way that alternative A_i outranks A_j if on the majority of criteria A_i performs as well as A_j, and at the same time it achieves sufficiently acceptable scores in other criteria. ELECTRE [70] and PROMETHEE [15] are among the most famous approaches in this category.

Optimization Methods

Optimization methods search for the most suitable service, which is usually the one that maximizes or minimizes one or several criteria, such as cost and deployment time. The optimization problem can be complicated when more than one criterion are considered and there are constraints imposed by users. By considering constraints in the selection, the definition of optimization can be rewritten as "finding the most suitable services for the clients or providers, which maximizes or minimizes one or several criteria and still adheres to the constraints". For example, assume that the best service is the one with minimum cost, highest availability, and least deployment time when there is a limitation for providers to serve the users [165,170] due to bandwidth constraint. Then, the problem of selection can be formulated as the "Selective Multiple Choice Knapsack Problem" (SMCKP) which is an NP-hard problem. Studies in the area of selection faced the problem in different ways, some trying to find the optimal solution [137], others aiming at finding the semi-optimal solution by using heuristics [3,114,124,172]. Dominant approaches

in this area can be classified in two main categories, non-evolutionary and evolutionary optimization methods, which will be investigated next .

Non-Evolutionary Optimization Method Four classes of selection approaches in this category are: pattern based, selection using discarding subset results, bottom-up selection, and greedy algorithms. This classification is done by Jaeger et al. [82] to solve the problem of multi-criteria selection. It compares several algorithms for selecting the best web service candidates. They considered four QoS categories introduced by Zeng et al. and Menasce [123,184]. The categories are: execution time, cost, reputation, and availability. After that, an approach for aggregating QoS [85] of individual web services was applied. At the comparison stage, the work applied the Simple Additive Weighting (SAW) approach, which was extracted from the context of Multiple Criteria Decision making (MCDM) [81]. They have compared algorithms' performances, and the results are reported as follows:

- A greedy selection is not able to handle constraints. Instead, it can find the candidate that scored the highest among all the other candidates.
- A Bottom-up approach [84] relies on the fact that the selection problem for composition (selection in composition) of web services shows similarities to Resource Constrained Project Scheduling Problem (RCPSP). In RCSP, a project is divided into individual tasks and each task has to be assigned to an available worker to complete the project in a way that meets the constraints, such as deadline. Bottom-up selection results in the second worse QoS and its computation effort is negligible.
- Pattern-Based selection [71] considers each composition pattern separately and then tries to find the best assignment. This approach offers the best achieved QoS compared to all heuristic approaches. The computational effort of this selection is reasonable, and depends on the composition structure.
- The selection by discarding subsets is a kind of backtracking-based algorithm that uses a search tree consisting of nodes, each representing a possible pair of a candidate and task. It results in the best QoS possible and also meets the

constraints.

Evolutionary Multi-objective Optimization Methods Evolutionary Multi-objective Optimization methods are based on the principle of natural selection, which is called survival of the fittest and originally characterized by Charles Darwin [38]. Since evolutionary approaches have shown desirable potential for solving optimization problem, this class of search strategy has been utilized for multi-objective optimization from mid-1980s. Evolutionary multi-objective optimization is in fact a combination of the evolutionary computation and traditional multiple criteria decision making. Evolutionary approaches follow two major concepts. The first concept is the competition for reproduction, which is called selection. And second one mimics the ability of producing new generation by mutation, which is called variation.

A considerable number of evolutionary multi-objective optimization (EMOO) techniques have been developed in recent years [31, 166]. Two of the most applied methods in the selection literature, NSGA-II and SPEA-II are briefly explained below.

NSGA Non-dominated Sorting Genetic Algorithm (NSGA) [156] was proposed by Srinivas and Deb. The algorithm modifies the ranking procedure originally proposed by Goldberg [69] based on several layers of classifications of the individuals. NSGA was highly computational intensive and had several other drawbacks which led to the rise of NSGA-II [41]. In the first step, NSGA-II constructs a space of solutions, then performs sorting based on non-domination level, and applies the crowded-comparison operator to create a new pool of offspring. It applies a fast non-dominated sorting approach which has $O(MN2)$ computational complexity, where M is the number of objectives and N the population size. The algorithm is capable of outperforming many other genetic optimization algorithms [41].

SPEA The Strength Pareto Evolutionary Algorithm (SPEA) is presented by Zitzler and Thiele [190]. The method is a result of integrating different EMOO techniques. It has the unique feature of archiving non-dominated solutions already found in order to not to lose certain portions of the current non-dominated front due to ran-

dom effects. For ranking (calculating the strength) of non-dominated solution an approach similar to MOGA is used. In addition, fitness of individuals is calculated based on strengths of all non-dominated solutions in archive which can dominate it. Moreover, for maintaining diversity a method called "average linkage method" [126] is used. Numerous studies [137,190] applied SPEA to solve the 0/1 knapsack problems.

2.4 Service Level Agreement Management

As shown in Figure 2.10, research in this area has mainly focused on three aspects of SLA, namely SLA language specification, SLA negotiation techniques [192], and SLA monitoring approaches [49]. An example is the research conducted by Kotsokalis et al. [102] that proposed a generic architecture, and tries to address all phases of the SLA management lifecycle, from negotiation and establishment to termination, with respect to the existence of SLA interdependencies. The goal is to ensure that operations of existing services would not be affected. Emeakaroha et al. [49] proposed a framework called "LoM2HiS" to provide a mapping from low-level resource metrics to high-level SLA parameters. The proposed framework aimed to develop an infrastructure for autonomic SLA management and enforcement. Moreover, it is capable of detecting a potential SLA violation based on predefined threat thresholds. Although multi-level SLA issues have been discussed in the work, no solution for modeling dependency knowledge was proposed. Theilmann et al. [160] discusses the need for multi-level SLA management approaches in order to fuel the next step towards a service-oriented economy. The authors proposed a conceptual architecture, which consistently bridges and mediates the various views of stakeholders and business/IT layers. However, no implementation is provided to verify the applicability of the proposed architecture.

2.4.1 SLA Negotiation Techniques

Negotiation techniques define how a party generates offers and counteroffers during the negotiation. The following families are the most commonly used negotiation techniques

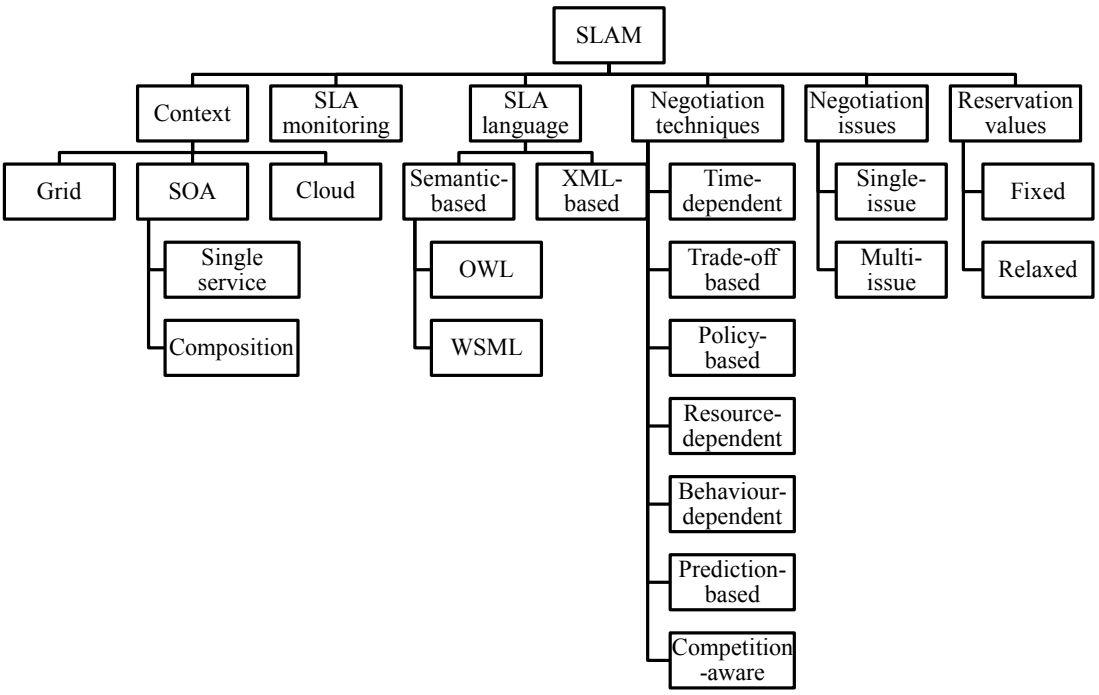


Figure 2.10: Service Level Agreement Management (SLAM) taxonomy.

in the literature.

Time-Dependent

If parties have deadline in the negotiation, these techniques are the appropriate choice. This category of techniques concedes faster as the deadline approaches. Faratin et al. [52] have extensively studied the behavior of this family by considering different time-dependent functions and investigating the effect of modifying their parameters such as the initial offer value and the deadline on negotiation outputs that include ratio of deals made. Based on the work, time-dependent functions later have been adopted for problems in different contexts such as Grid [110] and Cloud computing.

Resource-Dependent

This family of negotiation strategies are particularly helpful to reach a consensus, when resource constraints such as the remaining bandwidth (for providers) and the budget (for

users) are imposed on the negotiation problem. This family generates offers and counter-offers based on the availability of resources. For example, they concede faster when resources such as the bandwidth is tighter and vice-versa. In addition to the work of Faratin et al. [52], similar studies [108, 154, 177] have applied the pure resource-dependent techniques or its combination with other techniques for negotiation problems in the Grid computing context.

Policy-Based

They aim at defining required protocols and languages to capture user preferences in the form of polices, and then proposing transformation approaches to map high-level policies to low-level offer values. Common Open Policy Service for Service Level Specification (COPS-SLS) [129] provided a service level negotiation protocol that helps establishing a contract in the context of policy-based networks [180]. The work uses the COPS (Common Open Policy Service) [47] protocol to enable negotiation message exchange between clients and service providers for intra-, inter domain, and end-to-end service level negotiation. COPS offers a query and a response protocol that are used for exchanging the policy information between a Policy Decision Point (PDP) and its clients. A policy-based negotiation broker middleware framework is presented by Zulkernine et al.[191]. In their work, the WS-Policy specification is used to capture high-level business goals and preferences. The captured preferences form inputs for a mathematical model that converts them to low-level negotiation function parameters.

An SLA management framework for Cloud computing environments is offered by Chhetri et al.[25], that uses a policy-based model to support the automated establishment of SLA. The approach uses WS-policy [9] to create a set of rules that can be later queried by Cloud users and providers to automatically choose the most appropriate interaction protocol in a given context. This work lacks any negotiation strategy, but it can be adopted by Cloud providers and users to enable an SLA negotiation.

Behavior-Dependent

When there is no deadline in a negotiation, agents can adopt this class of techniques. This way they can imitate behaviors of opponents to perform at least as well as other parties in the negotiation. Techniques in this category are different in the degree of imitation which could vary from proportional to absolute. Both Axelrod et al. [7] and Faratin et al. [52] have studied this category. Axelrod et al. [7] found this class of negation technique particularity effective when it is applied to cooperative problem-solving negotiation. In addition, Faratin et al. investigated its performance extensively and in comparison to other negotiation techniques under different negotiation scenarios.

Trade-off-Based

Trade-off based techniques generate a counter offer that keeps the utility value high for the negotiation agent and improves the utility value of the opponent. Fuzzy similarity technique was proposed by Faratin et al. [53] for this purpose. It investigates a situation where an agent is willing to offer more attractive SLA contracts to its opponent, however it does not intend to decrease its utility function. The work presented an approach that determines how much (in a scale of 0 to 1) two contracts match. Then, by applying the hill climbing technique, feasible contracts are searched for a contract that has the highest similarity with the opponent's offer and still has the same utility value as the previously offered contract.

Prediction-Based

This category offers a learning mechanism that predicts of behavior an agent based on its previous offers. A prediction-based negotiation technique can use regression analysis [78] to predict a negotiation agent's behavior. The regression technique can increase the chance of reaching agreements with higher utility function values without any prior knowledge about the other agents. Zheng et al. [187] applied game theory to 1-to-1 web services negotiation and considered both parties' preferences. Techniques based on game theory assume that each agent is aware of all information about the possible

strategies and the corresponding outcomes of its opponents. However, this assumption is not always true for real-world applications which makes this category of techniques less practical.

Competition-Aware

The amount of relaxation in this class is decided based on the success rate in reaching the agreement and the demand intensity. The rate of concession is decided based on the opportunity function [155] that is used when Clouds and users are concurrently negotiating with each other. The opportunity function works out the probability of reaching an agreement based on the number of alternative negotiation parties and the difference between its offers and the received counter offers. Consequently, if the aforementioned probability is high, then a smaller amount of concession will be made in each round.

2.4.2 Negotiation for Multiple Services

In SOA context, an approach to provide support for reaching agreements between the service consumer and providers is crucial. However, reaching the agreement is even more challenging in the case of service composition where the consumer negotiates with multiple providers. Yun et al. [178] tackled this issue by adopting the agent technology and extending the Foundation for Intelligent Physical Agents (FIPA) protocol [138]. The work chooses the utility function for developing a decision making model for agents and uses the concept of fuzzy similarity [53] in its negotiation strategy. The presented negotiation strategy makes an effort to increase the chance of reaching an agreement by generating more similarity between offers and counter offers. The language used for negotiation is WSDL, and the prototype implementation shows the feasibility of the approach.

Single Versus Multiple Criteria Negotiation

When SLA negotiation consists of multiple issues such as service time and price, approaches that are purely time-dependent [52] are no longer effective. Coehoorn et al. [30]

proposed a multi-issue negotiation approach by gathering information regarding opponents' preferences across negotiation issues using kernel density estimation.

Relaxed Versus Fixed Reservation values

Sim et al. [155], in contrast to majority of previous works, proposed a negotiation strategy tactic with relaxed reservation values. In that strategy, the consumers and providers are allowed to slightly relax their reservation values. As we explained earlier, the amount of relaxation is decided based on the success rate in reaching the agreement and the demand intensity. In their work, the rate of concession is decided based on the opportunity function that is used when Clouds and users are concurrently negotiating with each other. Mansour et al. [117] described a case where a service client negotiates with multiple providers and for multiple negotiation issues. Their presented negotiation strategy allows flexible reservation values and outperformed strategies with static reservation values under different negotiation environments.

Cardinality

Cardinality in negotiation indicates the number of concurrent negotiation instances (none, one, or many) of an entity in relation to another entity. Values for cardinality can be one-to-one, one-to-many, many-to-one, and many-to-many. For example, if cardinality of negotiation for Cloud service providers and clients is one-to-many it means that a Cloud provider concurrently negotiates with many clients.

An architecture for one-to-many negotiation was proposed by Rahwan [139], which buyer agents coordinates a set of sub-negotiators, each can flexibly have a distinct negotiation strategy. The work identified four coordination strategies for a concurrent negotiation. In the first one, the buyer agent ends negotiations with others once it reaches a consensus with a seller. In another strategy, the buyer agent holds temporary agreements with the seller agents during negotiation. Once all negotiations end, the buyer selects the most prominent agreement. Similarly, a strategy which is called "optimized patient" holds a temporary agreement, and does not accept a new agreement with less utility.

The forth proposed strategy changes the negotiation techniques of its sub-negotiators dynamically during the bargaining period.

2.4.3 SLA Monitoring

SLA monitoring starts form a fundamental task of identifying proper QoS parameters in SLA and fine-grained metrics to measure them. Lawrence et al. [109] proposed an approach for SLA management which is built using WS-Agreement and is a part of the project called "OPTIMIS" [58]. Their work is designed for environments where SLAM is needed between service providers and infrastructure providers. They defined several novel negotiation criteria, but no negotiation tactic that can be utilized in Cloud environments. Similarly, Goiri et al. [68] developed a fine-grained QoS metric for CPU performance that can be used in SLA, and avoids fake SLAs violations to help providers achieve higher resource utilization.

The next step after defining the set of required QoS criteria and metrics is identifying and modeling the dependency between those criteria. This helps in improving the accuracy of the monitoring system by leading it to the root cause of a failure. Winkler et al. [176] proposed an approach for automated management of SLAs to support composite services. In that approach, the explicit knowledge about a set of dependencies to automate the tasks of SLA negotiation, renegotiation, and the handling of SLO violations is taken into account.

Bodenstaff et al. [13] proposed an approach called "MoDe4SLA" to monitor dependencies between SLA when managing composite services. In this case, different types of dependencies between services and the impact that services have on each other are analyzed during the development phase. Therefore, the approach proposed a way to analyze the monitoring results based on the impact and dependency knowledge.

2.4.4 SLA Language

No standard language is defined for building SLA offers and contracts. However, in the context of SOA, the Web Service Level Agreement (WSLA) language and the Web Ser-

vices Agreement Specification (WS-Agreement) are commonly referred. WS-Agreement is a web service protocol for building agreements between service providers and users and uses an XML-based languages as the template for agreements. IBM proposed the Web Service Level Agreement (WSLA) language and framework in 2001. The language is based on XML and particularly created for web services that define service interfaces in the WSDL. Similarly, SLAng proposed an XML-based language [105] to enable QoS-aware service provisioning.

WS-Negotiation [80] is another language that enables negotiation between web services providers and requestors. It consists of three parts namely Negotiation Message, Negotiation Protocol, and Negotiation Decision Making. While the Negotiation Message introduces an XML-based format for message exchange, the Negotiation Protocol defines rules and mechanisms that providers and clients have to follow during the negotiation. The Negotiation Decision Making describes the negotiation strategy of a party and shows how the party generate, accepts, or rejects an offer.

The BREIN project [127] uses ontologies and semantic technologies to overcome obstacles to automate the whole SLA lifecycle. The objective is building a commonly understood conceptual model in OWL-S for QoS criteria to deal with the semantic heterogeneity. Similarly, Fakhfakh et al. [51] proposed an ontology-based model for establishing SLAs between parties that automates their monitoring as well.

2.5 Analysis and Positioning

2.5.1 Requirement Analysis

This section presents requirements for service coordination and deployment in a Multi-Cloud environment.

- **Toolkit for service coordination and deployment:** In order to simplify cross-Cloud deployment and coordination, a toolkit that focuses on QoS modeling and deployment optimization is required. By a service we mean a virtual appliance or a virtual machine (virtual unit), which are two fundamental offerings of IaaS providers.

- **Semantic interoperability and multi-Cloud service discovery:** In a Multi-Cloud environment, it is difficult to enforce syntax and semantics of service (virtual machine and appliance) descriptions and user requirements. Therefore, applying symmetric attribute-based matching between requirements and request is impossible. In order to tackle this problem, we require a semantic-based discovery that works on top of well-defined ontologies and semantic-based service repositories.

- **Automatic reliability and infrastructure aware SLA negotiation:** We require a negotiation strategy that assesses the reliability of offers for users and considers infrastructure management concerns of providers. In a Cloud computing context, we need to consider infrastructure management issues (such as resource utilization balancing) in the bargaining strategy. It means that Cloud providers are willing to concede on the price of resources that are less utilized, and it has to be reflected in the negotiation tactics.

- **QoS-aware service selection in multi-Cloud environments for network of applications:** Multiple providers are offering different appliances and virtual units with different pricing in the market, it is important to exploit the benefit of hosting appliances on multiple providers to reduce the cost and provide better QoS. However, this could be only possible if high throughput and low latency can be guaranteed among different selected Clouds. Therefore, the latency constraint between nodes has to be considered as key QoS criteria in the selection problem. Amazon EC2, GoGrid, Rackspace, and other Key players in the IaaS market, although they constitute different deployment models using virtual appliances and units (computing instances), however none of them provide a solution for composing Cloud services based on users functional and non-functional requirements such as cost, reliability and latency constraints.

- **Cloud service compatibility verification in a composition:** A service provider will require more than one virtual appliance and unit and a composition of them that can meet all the requirements of users needs to be built. However, the selection of the best composition is a complex task. The best choices found for individual ap-

pliances cannot be simply put together as some appliances will not be compatible with the hosting environment. For example, if an appliance format is OVF it cannot currently be deployed on Amazon EC2 as it only accepts appliance with AMI format. In addition to that, there exists legal constraints imposed by countries such as USA on importing and exporting appliances from a provider to another. Therefore, to simplify the process of deployment for non-expert users, and fulfill the great promise of Cloud which is ease of use, we require an approach to automatically check the compatibility of services in a composition.

- **User preference modeling:** Minimizing effort of users in expressing their preferences is pivotal for the success of the proposed solution for selection of Cloud services. Therefore, we require a methodology to let users expressing their needs in high-level linguistic terms. This brings a great comfort to them compared to systems that force them to assign exact weight to each preference.

- **Reliable SLA monitoring:** In Multi-Cloud environments where various providers are involved in satisfying user requirements, we face a set of difficulties in SLA monitoring. Firstly, the existence of different SLA offers, counter offers, and contract templates makes it difficult to discover the necessary monitoring services that have required capabilities to monitor service level objectives in SLAs. Therefore, creating a standard model for describing SLAs in different layers of the Cloud has been considered as a major requirement in this area of research. In Cloud computing environments, there are dependencies between different service's performances. It means that if one of the lower-layer services (infrastructure layer) is not functioning properly, it can affect performance of higher-layer services. Whereas SLA dependency has been considered by several works [13, 176], no practical approach has been presented to model the dependencies among services.

2.5.2 An Investigation of Existing Work

In this section, we review recent and major developments in service coordination in the Cloud computing context. The key works in this area along with their fundamental char-

acteristics are summarized in Table 2.2.

We start with OPTIMIS [58], whose main contribution lies on optimizing the whole service life-cycle, from service construction and deployment to operations in Cloud environments. OPTIMIS consists of a set of tools that provide trust, risk, carbon emission efficiency, and cost assessment of Cloud services. In addition, it offers tools to create, deploy, and run services including image construction and licensee management. The OPTIMIS architecture offers an XML-based extension of OVF to build a service manifest that describes a request for the Cloud service in detail. The SLA management framework in OPTIMIS is based on WS-Agreement and the SLA negotiation is carried out manually. The selection of Cloud providers is accomplished through an adoption of Analytical Hierarchy Process (AHP).

The mOSAIC project [43] is proposed to enable multi-Cloud application development and deployment. The project [136] aims at facilitating Cloud portability by providing a set of open APIs to offer an additional level of freedom that avoids vendor lock-in. To achieve even higher degree of flexibility, mOSAIC proposes the CloudWare API that utilizes semantic and negotiation technologies to postpone selection of Cloud services to deployment time. In mOSAIC, Cloud ontology [35–37] plays an essential role, and expresses the application's needs for Cloud resources in terms of SLAs and QoS requirements. It is utilized to offer a common access to Cloud services in Cloud federations. mOSAIC uses perfCloud [141] and WS-Agreement for management of user authentication and authorization to a Cloud Provider.

Contrail [21] is another European project that builds a federation that allows users to utilize resources belonging to different Cloud providers. Contrail's components tackle a number of challenges in Cloud computing environments. Contrail PaaS (ConPaaS) provides Platform as a Service for web and high-performance computing applications. In addition, Contrail SLA manager component defines SLA in federation level and reuses monitoring and SLA management features that have been developed for the SLA@SOI project [5]. Other components provide security, virtual private network, and virtualized execution platform for federated Clouds.

[5] SLA@SOI Project. `http://sla-at-soi.eu/`

STRATOS [134] is a broker at an early stage of development that tackles the Cross-cloud deployment problem by mapping it to a multi-criteria optimization problem. The problem is then solved by utilizing Service Measurement Index (SMI) [66]. SMI introduces a set of QoS criteria and metrics for performance measurement of Cloud services, and uses AHP to rank them accordingly. In addition, STRATOS proposed an XML-based Topology Descriptor File (TDF) to describe the specification of the deployment topology configuration.

CloudGenius [125] is a framework that focuses on migrating single tier Web application to the Cloud by selecting the most appealing Cloud services for users. CloudGenius considers different sets of criteria and dependencies between virtual machine services and virtual appliances to pick up the most appropriate solution. Like the majority of the works in the Cloud computing context, it chooses AHP for raking Cloud services. Since pair-wise comparisons for all Cloud services are computing intensive, the selection criteria were restricted to numerical criteria. The framework was validated through experiments, which also studied the time complexity of AHP.

Cloud services usually do not have characteristics like Input,Output, Post conditions, and Effect (IOPE). Therefore, IOPE-based discovery approaches can not be directly applied in this context. Instead Cloud services can be described and later discovered based on their functional (eg. CPU power, Bandwidth, Storage size) and non-functional properties (e.g price, location). However, only limited literature is available on Cloud service discovery, which mostly focuses on building ontologies for Cloud services. The proposed discovery approaches are mostly inspired from OWL-MX [96] and WSML-MX [97]. The Cloud Service Discovery System (CDCS) [76] claims to be a pioneer in offering agent-based Cloud service discovery, which uses Cloud ontology to identify similarities among services. Building Cloud service ontology and reasoning about Cloud service relations are stated as the main contributions of the system. The discovery mechanisms (e.g. similarity reasoning) used in the paper are very similar to what offered by Han et al. [75]. Therefore, the strength of work is mostly in constructing Cloud ontology.

Table 2.2: Analysis of existing works.

Project	Requirement Fulfillment and Methodology		Outstanding Characteristics
OPTIMIS	Architecture	Manage service life Cycles in Cloud	Enabling self-management of services deployed across multiple Clouds; Novel QoS criteria (e.g. eco-efficiency, security, trust)
	Selection	AHP	
	Preference Modeling	Pair-wise comparison	
	Negotiation	Manual	
	SLA language	WS-Agreement	
Mosaic	Architecture	Offers Cloud portability	Abstract API for connecting to multiple Clouds; Extensive Ontology describing services, actors, etc
	Semantic Interoperability	Ontology created manually	
	Negotiation	Manual	
	SLA language	WS-Agreement	
Contrail	Architecture	Federation as a third party	Mechanisms for identity management in federated Clouds; Abstract API for connecting to Clouds
	SLAM	Extending the SLA@SOI project for SLA management	
	Negotiation	Manual	
	Deployment descriptor	OVF extension	
STRATOS	Selection	AHP	A novel deployment descriptor
	Preference modeling	Pair-wise comparison	
	Deployment descriptor	XML-based meta-data	
CDCS	Semantic interoperability	Ontology created manually	Building Cloud ontology
	Discovery	OWLS-MX	
CloudGenius	Selection	AHP	Considering novel sets of criteria and dependencies for Cloud service selection
	Preference modeling	Pair-wise comparison	

Kanagasabai et al. [90] proposed a brokering system that utilizes OWL-S to semantically match a user's request to required Cloud services. In addition, their work is benefited from the hybrid discovery approach presented in OWL-MX [96] by enabling service discovery to handle constrains. Therefore, the contribution of the paper lies in its attempt to apply the hybrid match making in the Cloud computing context. In addition, they conducted experiments that reveal that adding non-logic based matching dose not impose a considerable overhead to the system, but rather makes the system adoptable for the Cloud context. To further improve the performance of the discovery in terms of running time, investigating alternative semantic databases such as AllegroGraph was considered as a future work.

2.5.3 Scope and Positioning of This Thesis

In this thesis, to simplify the process of service deployment and coordination in the Cloud, we present a novel framework that deals with all challenges involved in Cloud service coordination and satisfies the mentioned requirements. Table 2.3 describes methodologies that have been applied for each requirement. In the rest of this section, for each requirement, we define the methodology and scope of our thesis and how it is different from related works.

- This thesis proposes an approach that gives enough flexibility to end users to discover their needed appliances from a range of providers and dynamically deploy it on different IaaS providers. Most of related works [2,24] focused on satisfying user requirements using SOA architecture and virtualization, neglecting the proper consideration of Cloud computing as a resource provider. Our proposed architecture offers a unified solution that uniquely applies state of the art technologies of semantic services, agent negotiation, and multi-objective and constraints optimization to satisfy the requirements of whole service deployment life cycle.

- Resource matching is the process of selecting resources based on application requirements. Traditional resource matching, as exemplified by the Condor Matchmaker [142] or Portable Batch System [11] are considered as inflexible and difficult

to extend to new characteristics or concepts. In this thesis, unlike the traditional Grid resource selectors that describe resource/request properties based on symmetric flat attributes, separate ontologies are created in WSML to declaratively describe Cloud resources and user requests using an expressive ontology language. Instead of exact syntax matching, the ontology-based matchmaker performs semantic matching using terms defined in those ontologies. Compared to our work, mOSAIC [43] and CDCS [76], which also adopted ontology, are not capable of creating ontologies automatically from Cloud service descriptions provided through API calls. In addition, the provided semantic-based Cloud service descriptions, in contrast to ours, do not contain QoS information.

- This thesis addresses the issue of migrating inter-related components such as multi-tier web applications to Cloud. The problem of selecting the required virtual appliances and infrastructure services in a migration process has been investigated in two different scenarios. In the first one, the thesis looks into the optimization problem of selecting Cloud services which adhere to reliability and latency constraints and minimize the deployment cost. The problem is the multidimensional knapsack problem and was tackled by a genetic algorithm and a heuristic which is based on the discarding subset algorithm. In the second scenario, infrastructure services are allowed to be picked up from one provider. The objective is to minimize the deployment cost, time, and maximize the total reliability together. The selection problem is solved using multi-objective evolutionary algorithms. As discussed in the previous section, majority of recent works [58, 125, 134] have utilized AHP for service selection. In comparison with our approach for Cloud service selection, those works can only perform well when the number of given alternatives is small and the number of objectives is limited. In contrast, our approach can deal efficiently with a large number of Cloud services in the repository. CloudGenius has considered Cloud service selection for migrating single tier web applications. However, it has not considered the complexity of service selection for multi-tier applications, where we are restricted by constraints such as latency.

- In addition, the thesis proposes an approach to help non-expert users with lim-

ited or no knowledge on legal and virtual appliance image format compatibility issues to deploy their services flawlessly. For this purpose, we automatically build a repository of Cloud services in WSML and then enrich it with experts' knowledge (lawyers, software engineers, system administrators, etc) on the aforementioned compatibilities. The knowledgebase is used for reasoning in an algorithm that identifies whether a set of Cloud services, consisting of virtual appliances and units are compatible or not. None of existing approaches [58,125] for Cloud service selection so far have taken the compatibility factor into account.

- Many existing Cloud service ranking system require users to assign weights to their objectives [63,104,106,146]. In this case, users have to find a way to prioritize their preferences and then map them to weights. After that, the ranking system has to find out how precise users have gone through the process of weight assignment. To tackle this complexity, a major objective of this thesis is to offer a ranking system for Cloud service (i.e. virtual appliance and unit) composition that lets users express their preferences conveniently using high-level linguistic terms. Our system utilizes evolutionary multi-objective approaches and a fuzzy inference system to precisely capture the supplied preferences for the ranking purpose.

- Creation of a standard model for describing SLAs in different layers of Cloud has been considered a major challenge in this area of research. In this thesis, this matter has been addressed by a form of semantic SLA (created in WSML) which brings a common language and understanding to all parties involved in service provisioning. While SLA dependency has been considered by several works in SLA@SOI project [102,176], no practical approach has been presented to model the dependencies among services. Consequently, this thesis shows how dependency knowledge can be modeled using semantic technology, and how that knowledge can be used in discovery of monitoring services and SLA failure detection. This eliminates the effects of SLA failure cascading on violation detections.

Table 2.3: Positioning of this thesis.

Requirement	**Applied Methodology**	**Uniqueness**
Coordination Toolkit	Mixing SOA and virtulaization technologies to enable multiple Cloud deployment	- A Cloud agnostic deployment descriptor - Handling the whole cycle of the coordination
Semantic Interoperability and Discovery	- Centralized - Ontology-based - WSML-based Hybrid Cloud service discovery	Automatic creation of required ontologies
Service Selection	- Enhance version of discarding subset algorithm - Multi-Objective evolutionary approaches	- Migrating a multiple inter-related components such as multi-tier web applications to Cloud
User Preference Modeling	- Multi-Objective evolutionary approaches - Fuzzy inference system	Increase the accuracy of the system
Compatibility Checking	Reasoning on knowledgebase which is automatically created	Ease of use for non-experts
SLA Negotiation	Time-dependent technique	- Autonomous - Utilization and reliability aware
SLA Management	- Dependency knowledge was built in WSML - Semantic-based matchmaking for monitoring service discovery	- Discovery and ranking of monitoring Services - Eliminating SLA failure cascading effects

- With the advances of Cloud technology, operations such as discovery, scaling, monitoring, and decommissioning are accomplished automatically. Therefore, negotiations between Cloud services clients and providers can be a bottleneck if they are carried out manually. Hence, in contrast with existing works such as OPTIMIS, mOSAIC, and Contrail, this thesis offers a solution that automates the negotiation process. Our proposed approach utilizes a time-dependent negotiation strategy which is capable of assessing the reliability of offers to increase the dependability of the strategy, and to fill the gap between decision making and bargaining. In addition, our strategy is more flexible compared to trade-off [186] approaches, as it does not require any prior knowledge regarding opponent's utility functions. In addition, in comparison with recent works in Cloud computing context [25, 186], for Cloud providers, the strategy uniquely considers utilization of resources when generates new offers and automatically adjusts the tactic's parameters to concede more on the price of less utilized resources.

2.6 Conclusions

This chapter described the concepts, background, and methodologies of service coordination. We have investigated and classified systems that enable each phase of service coordination in contexts of Grid and SOA. This helps us to find out what could be inherited from other paradigms and what has to be done uniquely for the Multi-Cloud environment considering its special characteristics. Then, the chapter described the detail of major requirements for each phase of Cloud service coordination and reviewed how the state-of the-art developments have addressed them.

In addition, we positioned this thesis with regards to existing work and described how different we address the challenges of Cloud service coordination. As discussed in this chapter, our approach provides: ease of use for non-experts, semantic interoperability, more precise discovery and selection, more reliable SLA monitoring, and automatic negotiation strategy.

In the next chapter, we describe an architecture that enables effective service coordi-

nation in a Multi-Cloud environment. In addition, next chapter further describes how the architecture is benefited from semantic-based Cloud discovery.

Chapter 3

An Architecture for Automated Cloud Service Coordination

This chapter focuses on an automated approach to deploy required virtual appliances on the most suitable Cloud infrastructure. We propose an effective architecture to simplify and automate service coordination and deployment in Multi-Cloud environments. The architecture harnesses the ontology-based discovery to provide QoS-aware deployment of applications on the fittest Cloud service provider. The approach is tested in a case study and the result shows its efficiency and effectiveness.

3.1 Introduction

AS explained in Chapter 1 and 2, service coordination is considered a major challenge in Multi-Cloud environments. Different technologies and tools try to satisfy user requirements in terms of software and hardware. They accomplish this by describing the environments, abstracting the dependencies, and automating the process. Nevertheless, most of previous works focused on satisfying user requirements using SOA architecture [2,24] and virtualization [91,168], neglecting the consideration of Cloud computing environment as a resource supplier. In order to simplify cross-Cloud deployment and coordination, an architecture that focus on QoS modeling and deployment optimization is presented in this chapter. By Cloud service, we mean virtual appliances and virtual machines (virtual unit), which are two fundamental offerings of IaaS providers. The architecture is designed to help migrating user applications to Cloud by mapping them to the fittest compatible Cloud offerings. Apart from the Cloud service discovery component, other components for SLA management and Cloud service selection and composition will be explained in subsequent chapters.

Aside from the architecture, this chapter proposes a flexible approach for performing ontology-based discovery of Cloud services. This provides clients with enough flexibility to discover their required appliances from a range of providers and dynamically deploy them on different IaaS providers. The discovery approach is particularly useful where providers and users do not use the same notation for describing their services and requirements. To enable the ontology-based discovery, the chapter proposes a translation and an advertisement approach for IaaS providers based on modeling virtual units into one of the most prominent initiatives in Semantic Web services, i.e., Web Service Modeling Ontology (WSMO) [72].

3.2 Architecture

As mentioned in previous chapters, our proposed architecture offers a unified solution that uniquely applies state of the art technologies of semantic services, agent negotiation, and multi-objective and constraints optimization to satisfy the requirements of whole service deployment life cycle. The main goal of the architecture is to provide: ease of use for non-experts, semantic interoperability, more precise discovery and selection, more reliable SLA monitoring, and automatic negotiation strategy. The proposed architecture is depicted in Figure 3.1 and its main components are described below:

1. **User Portal:** All services provided by the system are presented via the Web Portal to service clients. This component provides graphical interfaces to capture users' requirements such as software, hardware, QoS requirements (including maximum acceptable latency between tiers, minimum acceptable reliability, budget), firewall, and scaling settings. Moreover, it contains an account manager, which is responsible for user management. It provides authorization and authentication for users and keeps the history of all users activities in the system.

2. **Translator:** Since WSMO is used for service discovery, Cloud services information are translated to WSML format by the Translator component . This component takes care of building and maintaining an aggregated repository of Cloud services and is explained in detail in Section 3.3.1.

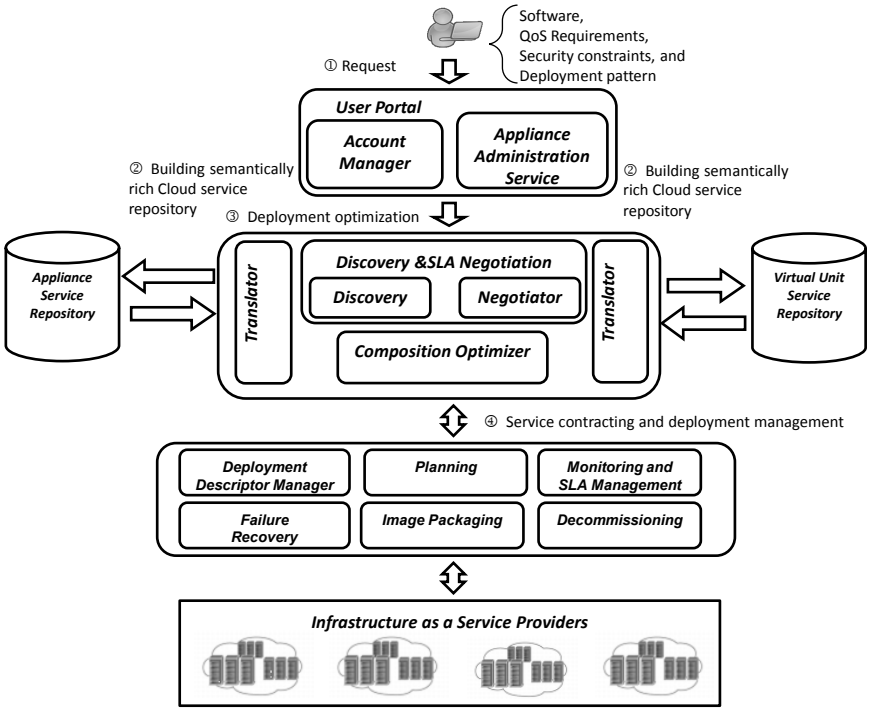


Figure 3.1: Architecture's main components that enable cross-Cloud deployment of user applications.

3. **Cloud Service Repositories:** They allow IaaS providers to advertise their services. An advertisement of a virtual unit can contain descriptions of its features, costs, and the validity time of the advertisement. From standardization perspective, a common metamodel that describes IaaS provider's services has to be created. However, due to the lack of standards, we developed our own metamodel based on previous works and standards in this area using WSMO.

4. **Discovery and Negotiation Service:** This component maps user's requirements to resources using the ontology-based discovery technique. It acts in user's interest to satisfy quality of service (QoS) requirements by selecting the set of eligible IaaS providers. The negotiation service uses a time-dependent negotiation strategy that captures preferences of users on QoS criteria to maximize their utility functions while only accepting reliable offers. the proposed Negotiation strategies are described in detail in Chapter 6.

5. **Planning:** The planning component determines the order of appliance deployment on the selected IaaS providers and plans for the deployment in the quickest possible manner.

6. **Image Packaging:** The Packaging component builds the discovered virtual appliances and the relevant meta-data into deployable packages, such as Amazon Machine Image (AMI) or Open Virtualization Format (OVF) [45] packages. Then the packages are deployed using the deployment component.

7. **Deployment Component:** It configures and sets up the appliances and virtual units with necessary configuration such as firewall and scaling settings in the destination. For example in a web application, specific connection details about the database server need to be configured.

8. **Deployment Descriptor Manager:** This component persists specifications of required services and their configuration information such as firewall and scaling settings in a format called Deployment Descriptor. Besides, it includes the mapping of user requirements to the instances and appliances provided by the Cloud. The mapping includes instance description (e.g. name, ID, IP, status), image information, etc. This meta-data is used by the appliance administration service to manage the whole stack of services deployed across multiple Clouds. Formally described using WSML, the Deployment Descriptor is located in our system (as a third party service coordinator), and in a Cloud-independent format that is used for discovering and configuring alternative deployments in case of failures. An example of a Deployment Descriptor is shown in Appendix A (Section A.2). It identifies how firewall and scaling configurations have to be set for Web server appliances. In addition, Deployment Descriptor can help to describe the utility function of users for provisioning extra Cloud services when scaling is required. This helps to create scaling policies that utilize the optimization component on the fly to provision services that maximizes the user's utility functions. For example, providers that have the lowest price, latency, and highest reliability are going to be ranked higher.

9. **Appliance Administration Service:** After the deployment phase, this component

helps end users to manage their appliances (for example starting, stopping, or redeploying them). It uses the Deployment Descriptor to manage the deployed services.

10. **Monitoring and SLA Management:** This component provides health monitoring of deployed services and provides required inputs and data for failure recovery and scaling. A monitoring system is provided by this component for fairly determining to which extent an SLA is achieved as well as facilitating a procedure taken by a user to receive compensation when the SLA is violated. The monitoring is based on the copy of signed SLA which is kept in SLA repository. The component provides an approach to discover and rank necessary third party monitoring services. Third party monitoring results can be similar to what the CloudStatus[1] service reports. Hyperic's CloudStatus is the first service to provide an independent view into the health and performance of the most popular Cloud services, including Amazon Web services and Google App Engine. CloudStatus gives users real-time reports and weekly trends on infrastructure metrics including service availability, response time, latency, and throughput that affect the availability and performance of Cloud-hosted applications. More detail on this component is given in Chapter 7.

11. **Failure Recovery:** It automatically backs up virtual appliance data and redeploys them in the event of Cloud service failure.

12. **Decommissioning:** In the decommissioning phase, Cloud resources are cleaned up and released by this component.

13. **IaaS Providers:** They are in both fabric and unified resource level [60] and contain resources that have been virtualized as virtual units. Therefore, they expose their services as virtual units that can be a virtual computer, database system, or even a virtual cluster. IaaS provider's advertisements of virtual units are converted to the WSML notation using the Translator Component. Among IaaS providers, Amazon Elastic Compute Cloud (Amazon EC2) has attracted considerable attention. Amazon EC2 [167] provides the flexibility to choose from a number of different

[1] Hyperic. `http://www.hyperic.com/products/Cloud-monitoring.html`

instance types to meet various computing needs. Each instance provides a predictable amount of dedicated compute capacity and is charged per instance-hour consumed. Figure 3.3 shows how an instance type of Amazon EC2 is modeled as a Web service in WSML.

In the next section we provide a detailed description of the Discovery Component along with a case study that proves its effectiveness. Then, we investigate the performance of the translation approach.

3.3 Matchmaker Architecture

The matchmaker consists of two components:

1. Ontologies, which provide the domain model and vocabulary for expressing Cloud service advertisements and service client requirements.

2. Matchmaking algorithm, which determines when an advertised virtual unit matches a requester requirement description.

In this work two ontologies have been developed using WSML. We use WSMO studio [44] for editing ontologies. WSMO Studio is an open source Semantic Web service and Semantic Business Process modeling environment for the Web Service Modeling Ontology. WSMO Studio is available as a set of Eclipse plug-ins. The most useful WSMO Studio features include: Ontology editor with integrated WSML Reasoner (for consistency checks and querying of ontologies Editor for WSMO elements (Web services to advertise virtual units, goals to define user requirements, mediators). Each ontology domain defines functional and non-functional properties and their elements. These two ontologies are:

- **Requirements ontology:** The ontology, as depicted in Figure 3.2 captures a requester virtual unit requirements which are defined as functional properties (e.g., number of CPU, memory size) and non-functional properties (e.g., budget, location) that represent QoS requirements. The majority of our notation uses the Common Information Model (CIM)[2], and in particular OVF, for describing a resource

[2]CIM standards. `http://www.dmtf.org/standards/standard-cim`

```
/*****************************
*  GOAL
*****************************/
goal  _ "http://example.org/GetVirtualUnit"
nonFunctionalProperties
 dc#title has Value "Goal of getting a virtual unit"
 dc #type hasValue _ "http://www.wsmo.org/TR/d2/v1.2/#goals"
 wsml#version has Value "Revision: 1.1 $"
endNonFunctionalProperties
importsOntology { _"http://www.example,org/ontologies/OpSys"?
_"http://www.example.org/ontologies/VirtualHardware"?
_"http://www.wsmo.org/ontologies/location"?
_"http://www.wsmo.org/ontologies/cost"}

capability _ "http://example.org/GetVirtualUnit#capl"
sharedVariables { ?dep? ?budget? ?loc}
effect havingAVirtualUnit
 nonFunctionalPropertie
 VU#DeploymentLatency  hasValue ?dep
 VU#Budget  hasValue ?budget
 VU#Location  hasValue ?loc
 endNonFunctionalProperties
 definedBy
[Info hasValue "Guest Operating System"?
  Description  hasValue "Unix"]
memberOf Vu#OperatingSystemSection [ hasId  hasValue  "99"] and
[      [ElementName  hasValue "Virtual Hardware Family"?
  InstanceID  hasValue "0"?
  VirtualSystemType hasValue "vmx-04" ] memberOf VHS#System and
[Description  hasValue "Number of virtual CPUs"?
  ElementName  hasValue "1 virtual Cpu"?
  ResourceType hasValue "3"?
  VirtualQuantity  hasValue "1"] memberOf  VHS#Item and
[AllocationUnits  hasValue  " byte * 2^20"?
  Description  hasValue "Memory Size"?
  ElementName  hasValue "256 MB of memory"?
  InstanceId  hasValue "2"?
  ResourceType  hasValue "4"?
  VirtualQuantity  hasValue "256"] memberOf  VHS#Item and
[AutomaticAllocation  hasValue " true "?
  Connection  hasValue " VM Network"?
  ElementName  hasValue  " Ethernet adapter on 'VM Network' "?
  InstanceID  hasValue  "3" ?
  ResourceType  hasValue "10" ] memberOf  VHS#Item and
[Elementname  hasValue "SCSI Controller 0 - LSI Logic"?
  InstanceID  hasValue "4" ?
  ResourceSubType  hasValue "LsiLogic"?
  ResourceType  hasValue "6" ] memberOf  VHS#Item and
[ElementName  hasValue "Harddisk 1"?
  HostResource  hasValue "ovf:/disk/lamp"?
  InstanceId  hasValue "5"?
  Parent  hasValue "4"?
  ResourceType  hasValue "17" ] memberOf  VHS#Item and
]     memberOf  VH#VirtualHardwareSection [hasInfo hasValue
"Virtual Hardware Requirements:  256Mb? 1 CPU? 1 disk? 1 NIC"]
```

Figure 3.2: Requirements ontology.

```
/*****************************
*  WEBSERVICE
*****************************/
nonFunctionalProperties
 dc#title  hasValue "A virtual unit service"
dc#type  hasValue _"http://www.wsmo.org/TR/d2/v1.2/#services"
wsml#version  hasValue "$Revision:  1.1  $"
endNonFunctionalProperties
importsOntology { _"http://www.example.org/ontologies/opSys"?
_"http://www.example.org/ontologies/VirtualHardware"?
_"http://www.wsmo.org/ ontologies/location"?
_"http://www.wsmo.org/ ontologies/cost"}
Capability __"http://example.org/ VirtualUnit#capl"
sharedVariables {?dep? ?price? loc}
effect
nonFunctionalProperties
VU#DeploymentLatency  hasValue ?dep
VU#Budget  hasValue ?budget
VU#Location  hasValue ?loc
endNonFunctionalProperties
definedBy
[ Info  hasValue " Supported Operating System"?
Description  hasValue "Red Hat Enterprise Linux"?
Description  hasValue "Windows Server 2003" ?
Description  hasValue "Oracle Enterprise Linux"?
Description  hasValue "OpenSolaris" ?
Description  hasValue "OpenSUSE Linux"?
Description  hasValue "Ubuntu Linux"?
Description  hasValue "Fedora"?
Description  hasValue "Gentoo Linux"?
Description  hasValue "Debian" ]
memberOf  VU#OperationgSystemSection and
[    [ ElementName  hasValue "Virtual Hardware Family"?
InstanceID  hasValue "0" ?
VirtualSystemType  hasValue "Small Unit"] memberOf  VHS#ECType and
[ Description  hasValue "Number of virtual CPUs"?
ElementName  hasValue "1 virtual CPU " ] memberOf  VHS#Item and
[ AllocationUnits  hasValue "byte * 2 ^ 30"
Description  hasValue "Memory Size"?
ElementName  hasValue "1.7 GB of memory " ] memberOf  VHS#Item  and
[ AutomaticAllocation  hasValue "true"?
Connection  hasValue " VM  Network " ?
ElementName  hasValue "Ethernet adapter on 'VM Network' " ]
memberOf  VHS#Item and
[ ElementName  hasValue "Storage"?
StorageCapacity  hasValue " byte * 160 * 2^30 "?
HostResource  hasValue "ovf:/disk/lamp" memberOf  VHS#Item and
[ElementName  hasValue "Platform"?
PlatformType  hasValue "bit * 32"] memberof  VHS#Item and
 [ElementName  hasValue "I/O"?
 Performance  hasValue "Moderate"] memberOf VHS#Item and
]  memberOf VU#VirtualHardwareSection [hasInfo  hasValue
"Offering  Virtual Hardware: 1.7 GB? 1 CPU? 1 disk? 1 NIC"]
Interface _ "http://example.org/VirtualAppliance#capl"
Choreography  _"http://example.org/tobedone"
orchestration _"http://example.org/tobedone"
```

Figure 3.3: Cloud service (virtual units) ontology.

management that is impartial to IaaS providers and implementations.

- **Cloud service ontology:** The ontology provides an abstract model for describing virtual appliances and units and their capabilities to enable IaaS providers to advertise their services. Our initial model concentrates on modeling of computational Cloud services as depicted in Figure 3.3.

The service description shown in Figure 3.3 has been created manually as an example in WSMO studio. However, virtual appliances and units meta-data are currently described in XML format. Therefore, the manual translation of Cloud appliance and virtual unit offerings' descriptions is time consuming and not practical. Therefore, next section offers an approach to semantically enrich Cloud offerings that minimizes human intervention.

3.3.1 Automoted Construction of Semantic-based Cloud Services and Their Quality of Services

Currently, there is no integrated repository of semantic-based services for virtual appliances and units (virtual machines). The first step towards describing services and their QoS is to communicate with Clouds and the Cloud monitoring services through their APIs and gather required meta-data for building the repository. The process of metadata translation is demonstrated in Figure 3.4. The components involved in this process are:

Integrity Checking

This component first merges output messages of API calls for acquiring Cloud services description using Extensible Stylesheet Language Transformations (XSLT)[3] and then compares them with the previously merged messages using a hash function. If the outputs of the hash function are not equal, the component triggers the Sync component to update the semantic repository.

[3]XSLT. `http://www.w3.org/TR/xslt`

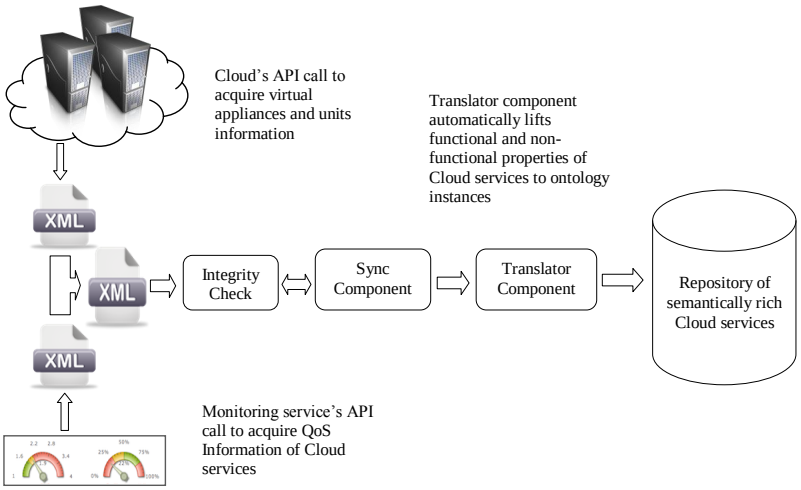


Figure 3.4: The process of translation of the virtual appliances and units descriptions to WSML.

Sync Component

The goal of this component is to keep the semantic-based repository consistent with the latest metadata provided by Cloud providers. As the synchronization is computing intensive, it is avoided unless the integrity checking component detects any inconsistency. It receives the output message that is required for synchronization and finds the corresponding semantically rich services and updates them with the output of the translator component.

Translator Component

During the communication of a semantic-level client and a syntactic-level web service, two directions of data transformations (which is also called grounding) are necessary: the client semantic data must be written in an XML form that can be sent as a request to the service, and the response data coming back from the service must be interpreted semantically by the client. We use our customized Grounding technique on WSDL operations (that are utilized to acquire virtual appliance and unit metadata) output to semantically enrich them with ontology annotations. WSMO offers a package, which utilizes Semantic Annotations for WSDL (SAWSDL) for grounding [101]. It provides two extensions

attribute namely as Lifting Schema Mapping and Lowering Schema Mapping. Lowering Schema Mapping is used to transfer ontology to XML and lifting Schema Mapping does the opposite. In our translator component, the lifting mapping extension has been adopted to define how XML instance data coming from Clouds API calls is transformed to a semantic model.

As the first step in grounding, from output message schema, the necessary ontology is created for virtual units and appliances. The basic steps to build the ontology from XML schema using WSMO grounding is explained by Kopecky et al. [101]. In this step our contribution lies on building the ontology from multiple output message schemas. It means that the monitoring service output message schema is used to extend the ontology to encompass the non-functional properties. This can be accomplished by merging two schemas to construct an output message that not only describes the format of the element that has functional properties of services but also non-functional properties such as price, and reliability.

Modelreferences are attributes whose value is a list of URIs that points to corresponding concepts in the constructed ontology in the last step. Having the ontology available, the next step is adding the necessary modelReferences for all element declarations or type delimitation for output messages in the Semantic Annotations for WSDL (SAWSDL) format. Subsequently, we need to add schema mappings that point to the proper data lifting transformation between XML data and semantic data. For this purpose two attributes, namely liftingSchemaMapping and loweringSchemaMapping, are offered by SAWSDL. These aforementioned attributes are then utilized to point from Cloud virtual appliance meta-data schema to a XSLT, which shows how meta-data is transferred from XML to WSML. We tested this approach for Cloud service repositories with variety of sizes, and will present the experimental result in Section 3.5. The ontology listed in Section A.1 of Appendix A was partially created by the described translator component. For example it shows how an appliance meta-data with ID of "aki00806369" has been translated to WSML format.

3.3.2 Matchmaking Algorithm

In order to consider whether a goal $\mathcal{G}$ that represents user requirements and a Web service $\mathcal{W}$ that represents advertised virtual units match on a semantic level, the sets $\mathcal{G}$ and $\mathcal{W}$ describing these elements have to be interrelated; precisely speaking, we expect that some relationship between $\mathcal{G}$ and $\mathcal{W}$ exists. The service matching in the proposed architecture is based on Description Logics (DLs) [8], which are a family of knowledge representation formalisms that are able to represent the structural knowledge of an application domain through a knowledge base including a terminology and a world description. The basic formalism of a DL system comprises three components: 1) Constructors, which represent concepts and rules, 2) Knowledge base (KB), which consists of the TBox (terminology) and the ABox (world description). The TBox presents the vocabulary of an application domain, while the ABox includes assertions about named individuals in terms of this vocabulary, 3) Inferences which are reasoning mechanisms of Tbox and Abox. Before we proceed to define discovery, we need to introduce the goal and five matching operations described below:

Definition 3.1 ***(Goal)*** *Let τ be an acyclic Tbox. A Goal $\mathcal{G}$ for τ is defined in the form of $\mathcal{G} = (\mathcal{C}_\mathcal{G}, I_\mathcal{G}, \mathcal{N}_\mathcal{G})$, where:*

- *$\mathcal{C}_\mathcal{G}$ is the set of capabilities of Web services including goal constraints, which the user would like to have.*
- *$I_\mathcal{G}$ is the set of interfaces of Web service, which the user would like to have and interact with.*
- *$\mathcal{N}_\mathcal{G}$ is the set of Nonfunctional properties, which is similar to that attached to Web services.*

Definition 3.2 ***(Exact matching)*** *Suppose that a requested capability of a Goal $\mathcal{C}_{\mathcal{G}1} \in \mathcal{G}_1$ is given. Let a capability of a Web service $\mathcal{C}_{\mathcal{W}1} \in \mathcal{W}_1$ and Nonfunctional properties of a Web service $\mathcal{N}_{\mathcal{W}1} \in \mathcal{W}_1$. If $(\mathcal{N}_{\mathcal{G}1} \equiv \mathcal{N}_{\mathcal{W}1}) \sqcap (\mathcal{C}_{\mathcal{G}1} \equiv \mathcal{C}_{\mathcal{W}1})$ then $\mathcal{G}_1$ can "exactly" match $\mathcal{W}_1$,i.e. $\mathcal{G}_1 \equiv \mathcal{W}_1$.*

Definition 3.3 ***(PlugIn matching)*** *Suppose that a requested capability of a Goal $\mathcal{C}_{\mathcal{G}1} \in \mathcal{G}_1$ is given. Let a capability of a Web service $\mathcal{C}_{\mathcal{W}1} \in \mathcal{W}_1$ and Nonfunctional properties of a Web service $\mathcal{N}_{\mathcal{W}1} \in \mathcal{W}_1$. If $(\mathcal{N}_{\mathcal{G}1} \sqsubseteq \mathcal{N}_{\mathcal{W}1}) \sqcap (\mathcal{C}_{\mathcal{G}1} \sqsubseteq \mathcal{C}_{\mathcal{W}1})$ then $\mathcal{G}_1 \sqsubseteq \mathcal{W}_1$. This match is called "PlugIn".*

Definition 3.4 ***(Subsumption matching)*** *Suppose that a requested capability of a Goal $C_{G1} \in G_1$ is given. Let a capability of a Web service $C_{W1} \in W_1$ and Nonfunctional properties of a Web service $N_{W1} \in W_1$. If $(N_{W1} \sqsubseteq N_{G1}) \sqcap (C_{W1} \sqsubseteq C_{G1})$ then $W_1 \sqsubseteq G_1$. This match is called "Subsumption".*

Definition 3.5 ***(Intersection matching)*** *Suppose that a requested capability of a Goal $C_{G1} \in G_1$ is given. Let a capability of a Web service $C_{W1} \in W_1$ and Nonfunctional properties of a Web service $N_{W1} \in W_1$. If $\neg(N_{G1} \sqcap N_{W1} \sqsubseteq \perp) \sqcap \neg(C_{G1} \sqcap C_{W1} \sqsubseteq \perp)$ then $\neg(G_1 \sqcap W_1 \sqsubseteq \perp)$. This match is called "Intersection".*

Definition 3.6 ***(Non-matching)*** *Suppose that a requested capability of a Goal $C_{G1} \in G_1$ is given. Let a capability of a Web service $C_{W1} \in W_1$ and Nonfunctional properties of a Web service $N_{W1} \in W_1$. If $(N_{G1} \sqcap N_{W1} \sqsubseteq \perp) \sqcap (C_{G1} \sqcap C_{W1} \sqsubseteq \perp)$ then $G_1 \sqcap W_1 \sqsubseteq \perp$. This match is called "Non-matching".*

Based on the above definitions, we propose the Web service discovery algorithm which is shown in Algorithm 1 and is defined as follows:

Definition 3.7 ***(Discovery)*** *Suppose that a requested capability of a Goal $C_{G1} \in G_1$ is given. Let a capability of a Web service $C_{W1} \in W_1$ and Nonfunctional properties of a Web service $N_{W1} \in W_1$. Discovery is defined as to find a set of Web services such that:*
$N_{G1} \equiv N_{W1}) \sqcap (C_{G1} \equiv C_{W1}) \sqcup (N_{G1} \sqsubseteq N_{W1}) \sqcap (C_{G1} \sqsubseteq C_{W1}) \sqcup (N_{W1} \sqsubseteq N_{G1}) \sqcap (C_{W1} \sqsubseteq C_{G1}) \sqcup$
$\neg(N_{G1} \sqcap N_{W1} \sqsubseteq \perp) \sqcap \neg(C_{G1} \sqcap C_{W1} \sqsubseteq \perp)$

3.4 Case Study

In this section our approach is validated on a case study to show the effectiveness of the proposed algorithm. To show its applicability, it has been tested in Web Service Modeling Toolkit (WMST) [94]. The Audio Video Devices (AVD) online store has a powerful website for selling digital gadgets. Their business is initially based on Europe and they have just expanded it to US. They have leased a dedicated server from a data center in US. Nevertheless, due to a business plan for announcement of twenty percent discount in some category of items, an exceptional load is predicted to build up on the server. As they have doubt about the ROI, they are not going to acquire another server. Recently,

Algorithm 1: Discovery

Input: g_{user} is the user's goal, $\mathcal{G}_{exist}$ is a set of existing goals, $\mathcal{W}$ is the set of Cloud services

1 **forall the** $g \in \mathcal{G}_{exist}$ **do**
2 **if** $g_{user} \equiv g$ **then**
   /* g is an existing WSMO goal */
3 **return** $(g_{user}, matchType(g))$
4 **forall the** $w \in \mathcal{W}$ **do**
   /* w is an existing WSMO service */
5 **if** $\mathcal{C}(g_{user}) \equiv \mathcal{C}_w$ ***and*** $\mathcal{N}(g_{user}) \equiv \mathcal{N}_w$ **then**
   /* $\mathcal{C}$ and $\mathcal{N}$ stands for capability and non-functional properties respectively */
6 **return** $(w, Exact)$
7 **else if** $\mathcal{C}(g_{user}) \sqsubseteq \mathcal{C}_w$ ***and*** $\mathcal{N}(g_{user}) \sqsubseteq \mathcal{N}_w$ **then**
8 **return** $(w, PlugIn)$
9 **else if** $\mathcal{C}_\mathcal{W} \sqsubseteq \mathcal{C}(g_{user})$ ***and*** $\mathcal{N}_\mathcal{W} \sqsubseteq \mathcal{N}(g_{user})$ **then**
10 **return** $(w, Subsumption)$
11 **else if** $\neg(\mathcal{C}(g_{user}) \sqcap \mathcal{C}_w \sqsubseteq \bot)$ ***and*** $\neg(\mathcal{N}(g_{user}) \sqcap \mathcal{N}_w \sqsubseteq \bot)$ **then**
12 **return** $(w, Intersection)$
13 **return** $(guser, \mathcal{N}onMatch)$

they have been informed about Cloud computing and its pay-as-you-go usage model and found it very useful for their case. That is because they can lease a virtual server even for an hour and terminate it when the load returns to the normal level. However, they are seeking for a solution to deploy their application automatically on the most suitable IaaS provider.

We show how our work can help them to achieve their goal. Their virtual unit requirements are depicted in Table 3.1 the and Cloud services' specification that were advertised by providers are shown in Table 3.2. The operating systems supported by each

Table 3.1: Case study Cloud service request.

Requestor	CPU (core MHz)	Memory	Storage	Platform	Budget	Location	Deployment-time (Sec)	OS
AVD	800	1.5 GB	140 GB	32-bit	0.15 $	US	79	Unix compatible

Table 3.2: Cloud services listed for the case study.

Requestor	CPU (core MHz)	Memory (GB)	Storage (GB)	Platform	Budget	Location	D-time (Sec)
A	1000	1.7	160	32-bit	0.12$	US	79.5
B	1000	2	120	64-bit	0.38$	US	80
C	1000-1200	1.7	170	32-bit	0.10$	US	76.5
C	1000-1200	1.7	170	32-bit	0.11$	EU	76.5
A	4000	7.5	850	64-bit	0.138$	US	78

IaaS provider are depicted in Table 3.3. First, the AVD IT officer connects to the portal and expresses their software, hardware, and other requirements. AVD needs an Apache server to be installed on a virtual unit with specification illustrated in Table 3.1. Consequently, the Appliance Administration Service discovers a suitable Apache appliance that will be packed according to the destination Cloud VM image format standard. Next, the matchmaker, as explained in Section 3.3, checks the capabilities of both virtual units Web services against the resource requirements. Since the knowledge base (KB) specifies that both "Linux family" and "OpenSolaris" are types of "Unix", not only IaaS provider A but also IaaS provider C, pass the operating system requirement criteria.

Both providers A and C services (which is located are US) pass functional require-

Table 3.3: Supported operating systems for the case study.

Providers	Operating Systems
A	UNIX, Debian 5.0, 4.0 ; Ubuntu 9.04, 8.10 ; Windows Web Server 2008
B	Windows Server 2008 ; Windows Server 2003
C	OpenSolaris ; OpenSUSE Linux ; Ubuntu Linux ; Windows Server 2003

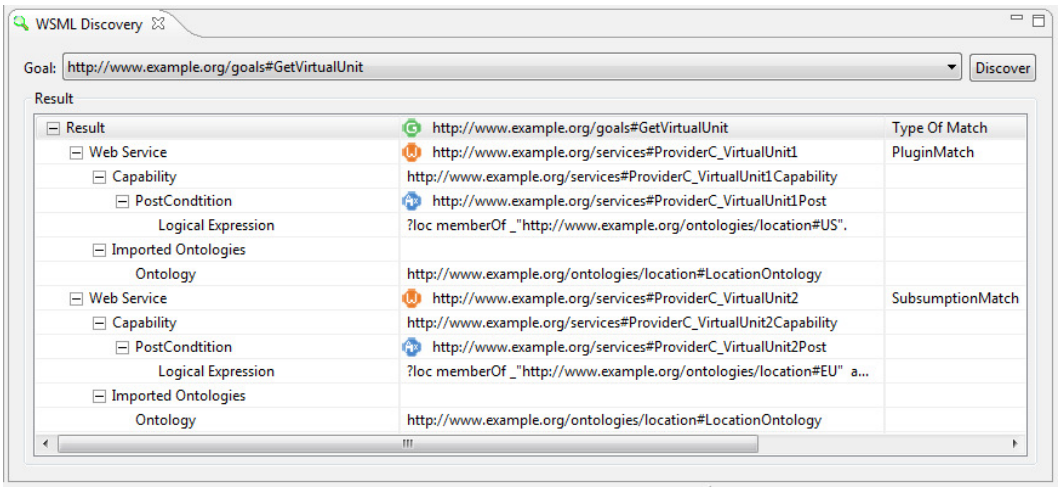


Figure 3.5: Case study validation in WSMT environment.

ments criteria based on Definition 3.3 . As it is shown in Figure 3.5, providers C in EU and A match type with user requirements is Subsumption as they cannot satisfy location and deployment time criteria correspondingly. However, the provider C in US is the most preferable as it can satisfy all requirements and its match type is PlugIn. Next, as described in Section 3.2 the signed SLA will be sent to the third party for monitoring. In this case, the third party realized that the deployment time was 80 seconds which is 3.5 seconds more that what both parties have been agreed on. Therefore, the third party informs them, and AVD is found eligible for receiving compensation as the SLA was violated.

3.5 Performance of the Translation Approach

Major Cloud providers have large repository of virtual appliance and unit services. For example, Amazon Web Service's repository's size alone is greater than 10.6 megabytes. To increase the efficiency of the approach we will only synchronize when the translation service is triggered by integrity checking component. We increased the number of services in the repository by merging repositories from various Cloud providers to investigate

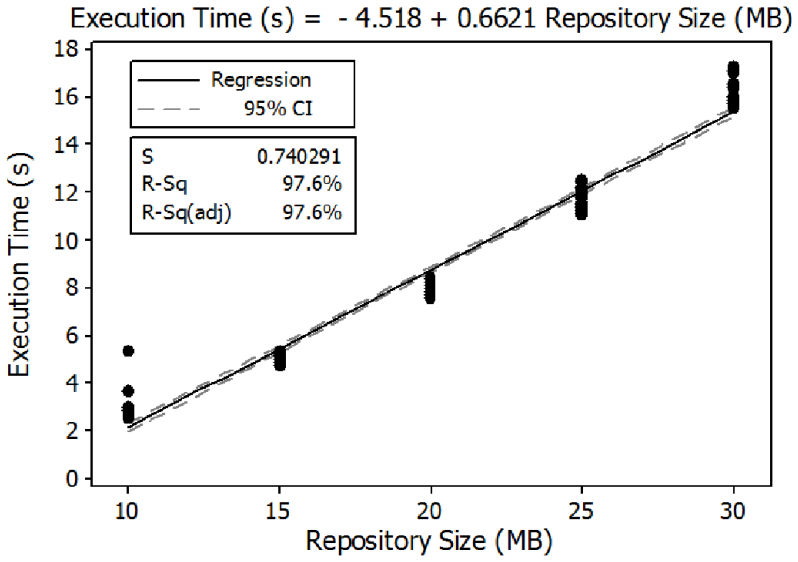


Figure 3.6: Execution time of translation for different repository sizes.

the scalability of our approach in terms of execution time needed for the translation. For each case of repository size, we repeated the experiment 30 times and the results are plotted in Figure 3.6. Regression analysis shows that there is positive and linear relationship between the repository size and the translation time. The evidence confirms that the regression coefficient is 0.6621, which suggests that if the data size to be translated increases by a mega byte, translation time increases roughly by 0.6 second. Consequently, the synchronization function can be executed online in an acceptable time even if a considerable percentage of virtual appliance and unit properties are updated.

3.6 Related Work

Recently, many works have targeted satisfying end user requirements by offering solutions based on virtualization [91, 168]. These works did not consider Cloud providers as resource suppliers. Therefore, they did not offer any discovery and selection techniques for resource provisioning. In this section, two of the most recognized works are reviewed. Keahey et al. [91] presented the idea of virtual workspace (VW), which allows users to define an environment in terms of their requirements (such as resource requirements or software configuration), manage and then deploy it in the Grid and Cloud. They have

their own Cloud, named Nimbus Cloud, for deployment of VW which. It provides virtualization in the form of Xen virtual machine and can be used to make a request to deploy a workspace based on a specified VM image. It is worth mentioning that that they have not considered the user QoS requirements and in general SLA. In addition, Cloud discovery and selection is missing from the work.

A related work has been done in North Carolina State University. The project name is Virtual Computing laboratory (VCL) [168], which was originally described in February 2004. VCL claims that it is an ideal product to support all kinds of Cloud solution. VCL services vary from virtual computer laboratory seats or desktops, to single applications on demand, to high-performance computing services and clusters. However, VCL support for virtual machine images discovery and selection is limited and it cannot adequately capture QoS requirements.

3.7 Conclusions

In this chapter, an effective architecture for Cloud service coordination is presented. The presented approach includes three main improvements: persisting deployment configuration using a standard semantic language, proposing an advertisement approach for IaaS providers, and applying ontology-based discovery to find the best suited providers. One of desired attributes of our architecture is to allow users to present their requirements in terms of high-level and general software and hardware characteristics, which will be mapped to appliances and virtual units.

In the next section, we propose an approach to migrate a multi-tier application to Multi-Cloud. The approach considers necessary QoS criteria and optimization algorithms to move user applications to Cloud services that have the minimum cost and the highest reliability.

Chapter 4

Migrating Multi-tier Applications to Multi-Cloud

Multiple Cloud providers are offering different virtual appliances and computing units with different pricing and Quality of Service (QoS) in the market. Thus, it is important to exploit the benefit of hosting virtual appliances on multiple providers to not only reduce the cost and provide better QoS but also achieve failure resistant deployment. This chapter presents an approach to simplify cross-Cloud deployment and particularly focuses on QoS modeling and deployment optimization. An optimization approach is required for deploying networks of appliances to guarantee minimum cost, low latency, and high reliability. We propose and compare two different deployment optimization approaches: genetic-based and Forward-checking-based backtracking (FCBB). They take into account QoS criteria such as reliability, data communication cost, and latency between multiple Clouds to select the most appropriate combination of virtual machines and appliances. We evaluate our approach using a real case study and different request types. Experimental results suggest that both algorithms reach near optimal solution. Further, we investigate effects of factors such as latency and reliability requirements and data communication between appliances on the performance of the algorithms and placement of appliances across multiple Clouds. The results show that the efficiency of optimization algorithms depends on the data transfer rate between appliances.

4.1 Introduction

ONE of the key challenges in building a platform for deploying applications is to automatically select, configure, and deploy the necessary application infrastructure that consists of number of different components. If we consider the deployment requirements of a web application service provider, it includes security devices (e.g. Firewall), load balancers, web servers, application servers, database servers, and storage de-

vices. Setting up such a complex combination of applications is costly and error prone even in traditional hosting environments [158], let alone in Clouds. Virtual appliances provide an elegant solution for this problem.

A virtual appliance is a virtual machine image that has all the necessary software components to meet a specific business objective pre-installed and configured [150] and can be readily used with minimum effort. Virtual appliances not only eliminate the effort required to build these appliances from scratch, but also avoid any associated issues such as incorrect configuration. To overcome deployment problems such as root privilege requirements and library dependencies, virtual appliance technology is adopted as a major Cloud component.

In general, Cloud deployment includes two main phases: discovery which was explained in the previous chapter, and selection (deployment optimization). In the discovery phase, all virtual units and appliances, which are called Cloud services throughout this thesis, that satisfy users goals and QoS requirements are retrieved. Then, in the selection phase, all the combinations of virtual units and appliances are evaluated and ranked based on user preferences and the top combination in the ranked list is returned as the best composition. The availability of wide range of virtual appliances and units (computing instances) and the specific QoS requirements of users make it difficult to select the best combination of cloud services. In addition, multiple providers are offering different appliances and virtual units with different pricing in the market, therefore it is important to exploit the benefit of hosting appliances on multiple providers to reduce the cost and provide better QoS. However, this could be only possible if high throughput and low latency can be guaranteed among different selected Clouds. Therefore, the latency constraint between nodes has to be considered as key QoS criteria in the optimization problem. Amazon EC2, GoGrid, Rackspace, and other Key players in the IaaS market constitute different deployment models using virtual appliances and units (computing instances). However, they do not provide a solution for composing Cloud services based on users functional and non-functional requirements such as cost, reliability and latency constraints.

The first step to enable cross-Cloud deployment optimization is to model the appli-

ances, virtual units, and QoS requirements of users, which was achieved in Chapter 3. In the next step, the service coordinator has to deal with requests (group of connected applications) with different latency, reliability and budget constraints, and the objective of minimizing the deployment cost. The problem is to find a composition of combinations of virtual appliances and units that adheres to the user constraints and minimizes the cost of deployment.

This chapter addresses the modeling of relevant QoS criteria, namely latency, cost (data transfer cost, virtual unit, and appliance cost), and reliability for selection of the best virtual appliances and units in Cloud computing environment. In addition, it presents and evaluates two different selection approaches (genetic-based and Forward-checking-based backtracking (FCBB)) to help users in migrating network of application or multi-tier applications to multiple Clouds based on their QoS preferences. For this purpose, various types of requests (with different network load between appliances) are generated and data from 12 real Clouds was collected. Furthermore, the chapter investigates effects of factors such as latency requirements and data communication on the cost of appliance placement and the selection of providers.

4.2 Motivation Scenario

To study user requirements and concerns for deploying a network of applications on Clouds, we give an example of a real world case study with known network traffic between appliances. A good example of network of virtual appliances (a set of appliances in the form of a connected graph which have data communication among them) is multi-tier applications supporting web-based services. Each tier has communication requirements as characterized by Diniz Ersoz et al. [50]. They considered a data center with 11 dedicated nodes of a 96-nodes Linux cluster and host an e-business web site encompassing 11 appliances: 2 front-end Web-Servers (WS) in its web tier, 3 Databases (DB) in its database tier and 6 Application Servers (AS) in between. As they have characterized network traffic between tiers, we selected their work to build our case study. Assume that the administrator of the e-Business application is interested in migrating the appliances

to the Cloud in order to save on upfront infrastructure and maintenance costs, as well as to gain the advantage of on-demand scaling. In addition, to allow disaster recovery and geography-specific service offering, he/she would prefer multiple Cloud deployment. For such deployment, he/she faces several challenges such as:

1. What is the best strategy for placing appliances across Cloud providers? Should they be placed based on the traffic they exchange, therefore placing those with higher connectivity closer to each other to decrease latency and data transfer cost?

2. Is it economically beneficial to do so?

3. If appliances are placed across multiple providers, how the latency between different providers affects the performance?

4. How can the most reliable Cloud services be selected for the deployment?

4.3 QoS Criteria

The three QoS criteria considered in the deployment optimization problem are reliability, cost, and latency.

1. **Reliability:** We consider reliability as a QoS objective. To measure the IaaS providers' reliability, as shown by Equations (4.1) and (4.2), Service Level Agreement (SLA) Confidence Level (SCL) is introduced. It measures how reliably a provider is offering their services based on the binding SLA. SCL can be computed by a third party, who is responsible for monitoring the SLA.

$$SCL = \sum_{j=1}^{k} I_j \times SCL_j \quad where \quad SCL_j = MSQ_{jt} - SLO_{jt} \tag{4.1}$$

$$\text{Total Reliability } TR = \sum_{v \in V} SCL_v \tag{4.2}$$

where SCL_j is the SLA confidence level for QoS criteria j of a Cloud service; I_j is the importance of QoS criteria j for user; MSQ_{jt} is the monitored value of QoS criteria

j for a period of t; SLO_{jt} is promised QoS criteria j value in SLA for the period of t, and k is number of QoS criteria monitored.

Although our SCL metric can model various Cloud specific QoS criteria, in this work we consider only availability as it is the primary criteria specified by current IaaS providers in their SLAs.

2. **Cost:** Cost is a non-functional requirement of a user who wants to deploy a network of applications. In our problem, minimization of deployment costs is considered as the objective of users. The deployment cost includes monetary cost of leasing virtual units as well as appliances and communication costs. The communication monetary cost for connected virtual appliances depends on how much data they exchange and can be determined by the following factors: 1) One time communication message size and 2) Communication rate (how often two appliances communicate), which can be calculated based on request inter-arrival rate.

3. **Latency:** Latency can have a significant impact on e-Business web sites performance and consequently on the end user experience. Therefore, we have considered it in the problem as one of the users constraints. It is assumed that customers have different constraints for the latency between appliances which have to be satisfied with the selection of proper Cloud providers

4.4 Deployment Problem Formulation

4.4.1 Provider Model

Let m be the total number of providers. Each provider is represented in Equation (4.3).

$$P_k : (\{a\}, \{vm\}, Cdata_{in}(Pk), Cdata_{out}(Pk)) \tag{4.3}$$

Where a, vm, $Cdata_{in}(Pk)$ and $Cdata_{out}(Pk)$ denotes appliance, virtual machine, cost of internal data transfer and cost of external data transfer respectively. A virtual appliance a can be represented by a tuple of four elements: appliance type, cost, license type,

and size as represented in Equation (4.4).

$$a : \{ApplianceType; Cost; LicenseType; Size\} \tag{4.4}$$

A virtual machine vm can be formally described as a tuple with two elements as shown in Equation (4.5).

$$vm : \{MachineType; Cost\} \tag{4.5}$$

4.4.2 User Request Model

The user request for deployment of his application can be translated into a graph $G(V, E)$ where each vertex represents a server (virtual appliance running on a virtual unit). Server corresponding to a vertex v is represented as Equation (4.6).

$$S_v = \{appliance, virtualunit\} = \{a_v, vm_v\}, \forall v \in V \tag{4.6}$$

The edge $e\{v, v'\}$ indicates that vertex v and v' are connected. The data transfer between these connected vertexes (i.e., one server to another) is given by "D".

The objective of a user is to minimize the deployment cost of his whole application on multiple Cloud providers' infrastructures, given a lease period of "T" and budget B. The users have constraint for reliability (SCL_v) of the provider on which the server should be hosted and also latency constraint $(L(e\{v, v'\}) where \quad v, v' \in V)$ that represents the maximum acceptable latency between servers. The cost of renting a server includes the cost of virtual unit and virtual appliance.

Let appliance for S_v be rented from provider P_k and virtual unit from provider P_l. The cost of server S_v as shown in Equation (4.7) is the cost of appliance $(cost(a_{v,p_k}))$ and virtual unit $(cost(vm_{v,p_l}))$ plus cost of transferring the appliance if the appliance and virtual unit are not acquired from a same provider.

$$Cost\left(S_v\right) = \begin{cases} \left(Cost\left(a_{v,P_k}\right) + Cost\left(vm_{v,P_l}\right)\right) \times T & \text{if } k = l; \\ \left(Cost\left(a_{v,P_k}\right) + Cost\left(vm_{v,P_l}\right)\right) \times T + Size\left(a_{v,P_k}\right) * & \\ Cdata_{out}\left(P_l\right) & \text{if } k \neq 1. \end{cases} \tag{4.7}$$

Let $S_v = \left\{a_{v,Pk}, vm_{v,pl}\right\}$ and $S'_v = \left\{a_{v',Pk'}, vm_{v',pl'}\right\}$ be two connected vertexes (servers) by edge $e\left\{v, v'\right\} \in E$; and P_k , P_l, $P_{k'}$ and $P_{l'}$ are the providers that using their resources Servers S_v and $S_{v'}$ are deployed. The data transfer cost between two servers is given by Equation (4.8).

$$DCost\left(e\left\{v, v'\right\}\right) = \begin{cases} DSize\left(e\right) \times \left(CData_{out}\left(p_l\right) + CData_{out}\left(p_{l'}\right)\right) \times T & \\ \text{if } l \neq l' & \\ DSize\left(e\right) \times CData_{in}\left(p_l\right) \times T & \\ \text{if } l = l' \end{cases} \tag{4.8}$$

Therefore, the total cost of hosting users application is given by Equation (4.9).

$$TC = \sum_{v \in V} Cost\left(S_v\right) + \sum_{\substack{e \in E \\ v, v' \in V}} DCost\left(e\left\{v, v'\right\}\right) \tag{4.9}$$

4.4.3 Deployment Optimization Objectives

The objective of the user is to minimize the deployment cost of his whole application on multiple cloud infrastructures $(P_k, \quad 0 < k < m)$. Thus, the mathematical model is given by Equations (4.10), (4.11) , and (4.12).

$$Min\left(TC\right) Subject \quad to \quad 0 < TC < B \tag{4.10}$$

$$Latency\left(S_v, S_{v'}\right) < L\left(e\left\{v, v'\right\}\right) where \; \forall \; e\left\{v, v'\right\} \; \in E \tag{4.11}$$

$$SCL\left(S_v\right) < SCL_v \; where \; \forall \; v \in V \tag{4.12}$$

Where, $Latancy\,(S_v, S_{v'})$is the latency between Cloud infrastructures where server S_v and $S_{v'}$ are hosted, and $SCL(S_v)$ is the reliability of the Cloud infrastructure where server S_v is hosted.

4.5 Deployment Optimization Algorithms

To tackle the mentioned problem, one may consider a greedy algorithm [27]. However, it cannot be directly adopted to solve the deployment problem, as it is not capable of satisfying the budget constraint and latency constraints between vertices. Another approach that can be used to solve the problem is finding all possible compositions using exhaustive search, comparing their overall cost, and selecting the composition with the lowest cost that satisfies budget, reliability, and latency constraints. This approach can find the optimal solution; however, the computation cost of the algorithm is high due to NP hardness of the problem [82]. In order to deal with the aforementioned challenges, in the rest of this section, we describe two algorithms: Forward-checking-based backtracking (FCBB) and genetic-based Cloud virtual appliance deployment optimization.

4.5.1 Forward-Checking-Based-Backtracking (FCBB)

In FCBB the process of searching providers begins from a start node (vertex) S_v that has minimum deployment cost (including appliance and virtual unit cost) and for all its children there can be found at least one provider that satisfies all constraints (partial forward checking) [Algorithm 3 lines:12-14]. The partial forward checking on the problem constraints is added to the algorithm to avoid back jumps in the circumstances where latency constraints of the users are comparatively tight.

Then, S_v is added to the processed node list. After that, the algorithm processes all the children of S_v that are not processed, and for each child of $S_{v'}$, providers are selected using the selection function [Algorithm 2 lines:8-11] such that 1) latency and SCL constraints are satisfied with all the connected processed nodes (backward checking), 2) they can pass forward checking and 3) they have minimum communication (to already processed nodes) and combination cost [Algorithm 3 lines:16-19]. After selection of all the

unprocessed children of the start node $S_{v'}$, the similar search and selection process is applied recursively for all the grandchildren of the start node S_v [Algorithm 3 lines: 12-13]. If the selection function does not find any set of providers, it moves back and replaces the parent node with the second best set of providers in the *Combination* list (Backtrack) [Algorithm 2 lines: 7 and 11].

Algorithm 2: FCBB

```
   Input: S_v, RequestG(V, E)
1  if S_v = theFirstStartNode then
2      S_v ← getStartNode(RequestG(V, E), processedSet);
3      processedSet ← processedSet ∪ S_v ;
4      selection(S_v);
5      if selection(S_v') = null then
6          backtrack;
7  connectedNotProcessed ←
   getConnectedNotProcessed(parentNode, RequestG(V, E), processedSet);
8  foreach S_v' in connectedNotProcessed do
9      selection(S_v');
10     if selection(S_v')=null then
11         backtrack;
12 foreach S_v' in connectedNotProcessed do
13     FCBB(S_v');
```

4.5.2 Genetic-Based Virtual Unit and Appliance Provider Selection

Since genetic approaches have shown potential for solving optimization problems [31], this class of search strategies was utilized in our problem. The adoption of genetic-based approaches for the deployment problem involves 4 steps.

The *first* step is to plan the chromosome, which consists of multiple genes. In our problem, each vertex in the graph of request is represented by a gene. The *second* step is to

Algorithm 3: Selection

Input: S_v

Output: *selected* []

```
1  minCost ← ∞; constraintsViolated ← false; feasible ← true;
2  foreach combination in getAllCombinationSorted(S_v) do
3          ▷ getAllCombinationSorted returns combinations sorted using quick sort.
4      if SCL(S_v) < SCL(combination.getVUProvider()) and
       SCL(S_v) < SCL(combination.getAppProvider()) then
5          connectedProcessed ←
           getConnectedProcessed(startNode, RequestG(V, E), processedSet);
6          if connectedProcessed = null then
7              foreach S_v' in connectedProcessed do
8                  if Latency(S_v, s_v') > L(e{S_v, s_v'}) then
9                      constraintsViolated ← true;
10             if constraintsViolated = false then
11                 connectedNotProcessed ←
                   getConnectedNotProcessed(startNode, RequestG(V, E), processedSet);
12                 foreach s_v' in connectedNotProcessed do
13                     if ∉ combination in getAllCombinationSorted(S_v) that
                       Latency(S_v, s_v') > L(e{S_v, s_v'}) then
14                         feasible ← false          ▷ Forward Checking;
15                 if feasible = true then
16                     cost ← communicationCost + combination.getCost();
17                     if cost < minCost and cost + totalCost < request.getBudget()
                       then
18                         minCost = cost;
19                         selected[S_v] ←
                           {combination.getVUProvider(), combination.getAppProvider()};
```

create the population, hence each gene represents a value that points to a combination of virtual unit and appliance service (which satisfies requirements of corresponding vertex) in a sorted (based on the combination cost) list.

Implementation of fitness function is the *third* step. The fitness values are then used in a process of natural selection to choose which potential solutions will continue on to the next generation, and which will die out. The fitness function, as shown in Equation 4.13, is equal to the total cost of the solution. However, if constraints are violated, the penalty function is applied. Designing penalty function for genetic-based approach is not a trivial task. Several techniques have been applied in our work until a proper penalty function was found that is capable of handling constraints in the problem.

The penalty function is constructed as a function of the sum of the number of violations for each constraint multiplied by constants as shown in Equation 4.14. In the penalty function, Age is the age of chromosome, ki is a constant, NVi is number of cases that violates the constraints, and $NNVi$ is the number of cases that do not violate the constraints. In addition, to discard the infeasible solutions in early generations, infeasible solutions with lower age are penalized heavier. Finally, the last step is the evolution of the population based on the genetic operator. The genetic operator adopted for our work is the Java Genetic Algorithm Package (JGAP) natural selector [121].

$$fitness\left(Chromosome\right) = \begin{cases} \left(\sum_{i \in V} Cost\left(Gene_i\right) + \sum_{\substack{e \in E \\ i,j \in V}} DCost\left(e\left\{Gene_i, Genej\right\}\right)\right) * T \\ \text{if constraints are not violated} \\ \left(\sum_{i \in V} Cost\left(Gene_i\right) + \sum_{\substack{e \in E \\ i,j \in V}} DCost\left(e\left\{Gene_i, Genej\right\}\right)\right) * \\ T + Penalty\left(\right) \\ \text{if constraints are violated} \end{cases} \tag{4.13}$$

$$Penalty\left(\right) = \sum_{i=1}^{n} \left(\frac{NVi}{NVi + NNVi} \times ki\right) \times \left(\frac{1}{Age}\right) \times fitnessvalue \tag{4.14}$$

4.6 Experimental Testbed Modeling

To evaluate the proposed algorithms and study the placement of appliances, essential input data using real experiments was collected. The collected data can be classified either in data for providers modeling, and data for user request modeling.

1. Providers Modeling: A set of 12 real Cloud providers are selected namely: Amazon, Zerigo, Softlayer, VMware, Bitnami, rpath, Turnkeylinux, Rackspace, GoGrid, ReliaCloud, Lindoe, and Prgmr. Their virtual units and appliances have been modeled in our system. In addition, latency data between Cloud providers and SCL for each of them have been measured. The following subsections describe the data collected in detail.

2. Virtual Unit and Appliance Modeling: We built an aggregated repository of virtual appliance and virtual unit services based on the advertised services by Cloud providers. Services contain information regarding cost, virtual appliance size, and data communication cost inside and outside Clouds.

3. Latency and reliability (SCL) calculation: The latency data between Cloud providers has been collected for three months using the Cloud harmony service [29]. Data collection was conducted twice daily at random times. Tests consist of pinging to determine latency. Table 4.1 shows mean latency between EC2 and 3 different virtual unit providers as a sample. Max, min and average of latency between providers are 58.94, 2.51 and 29.88 (ms) respectively. In addition, Panopta (a monitoring tool) is used to supply SCL input data. Table 4.1 demonstrates how a sample of SCL input data looks like for three Cloud providers service uptime for a 365 days period.

4.6.1 Generation of Requests for Experiments

The request generation involves three steps. Firstly, number of servers requested by the user and requirements of each server in terms of virtual unit and appliance types are determined. Next, connected vertices in the request are identified. Finally, data trans-

Table 4.1: Latency between Clouds and SCL input data.

Cloud A	Cloud B	Latency (ms)	Cloud B Monitored Availability	Cloud B Promised Availability
EC2	Rackspace	49.8	99.996%	100%
EC2	GoGrid	8.9	99.996%	100%
EC2	Lindoe	5.01	99.996%	100%

Table 4.2: Request types.

Request Type	Request Graph Density	Request Inter Arrival Rate DB $\leftrightarrow$ AS	Request Inter Arrival Rate WS $\leftrightarrow$ As
Strongly connected	0.85	Log-normal (1.4719,2.2075)	Weibull (0.70906,10.185)
Moderately connected	0.5	Log-normal (1.1695,1.9439)	Weibull (0.41371,1.1264)
Poorly connected	0.25	Log-normal (0.8912,1.6770)	Weibull (0.24606,0.03548)

fer rates between connected appliances are identified. For experimental evaluation, two classes of requests are used, i.e., a real case study and randomly generated requests.

1. **Modeling user requests using a real case study:** For the real case study example, we use the three-tier data centre scenario presented by Ersoz et al. [50]. The required virtual units and appliance types for each vertex is assigned based on the scenario. They implemented an e-Business web site that encompasses 11 appliances: 2 front-end web-servers (WS) in its web tier, 3 databases (DB) in its database tier, and 6 application servers (AS) in between. In their work, a three-tier data centre architecture was used to collect the network load between appliances. Two different workloads, RUBiS [23] and SPECjApp-Server2004, are used by them. However, our focus is on the RUBiS, which implements an e-Business web site. That web site includes 27 interactions that can be carried out from a client browser. Their analysis of experiments results has been represented by various distributions of request inter-arrival times, and data size between tiers for 15 minutes runs of the RUBiS workload with 800, 1600, and 3200 clients. This data, which is shown in

Table 4.2, is used to calculate the network traffic between connected appliances.

2. **Modeling user requests for extensive experiment study:** Three classes of user requests (network of appliances) namely strongly, moderately, and poorly connected are created as shown in Table 4.2 which differs from each other in communicated message sizes, message inter-arrival rates, and graph density (proportion of the number of edges in request graph to total possible number of edges) of the request graph. The reason for building 3 classes of requests is to study the effect of network traffic and request graph density on performance of algorithms. For each vertex, we randomly assign a required virtual unit and appliance type, and then we use a random graph generation technique to identify which vertices are connected. All generated network of appliances follow the topology presented by Ersoz et al. [50]. Based on appliances that are connected to each other, data transfer rates are assigned. For example, if one appliance is a database and the other one is an application server and the request is in category of strongly connected, then the request inter-arrival rate is Log-normal (1.4719, 2.2075). In addition, to investigate effects of message size on the performance of algorithms, two classes of requests with different message sizes are created using workload "a" [50] (e-Business application with small message size) and "b" [4] (98 World cup with large message size).

4.7 Experimental Results

The experiments aim at:

- comparing the proposed heuristics with Exhaustive Search (ES) using the real case study,
- evaluating effects of variation in request types on algorithms performance,
- analyzing effects of variation in request types and constraints on distribution factor and deployment cost,
- and investigating effects of number of iterations an population size on the algorithm performance.

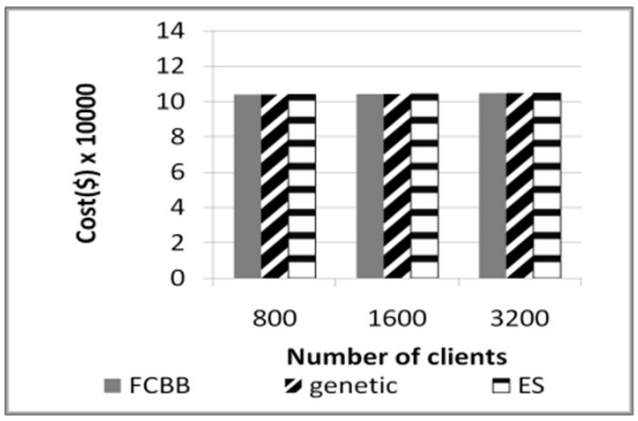


Figure 4.1: Performance evaluation for case study.

Table 4.3: Mean execution time for case study.

Algorithm	Mean Execution time(s)
FCBB	0.102
genetic	36.393
Exhaustive Search (ES)	3248.152

4.7.1 Comparison with Exhaustive Search (ES)

Figure 4.1 shows how close the proposed algorithms are to the Exhaustive Search (ES) for the case study. Both of them could reach the same solution achieved with ES. As evidenced by Table 4.3, the mean execution time for finding the solution using exhaustive search of the solution space is extremely high comparing to our proposed algorithms. The execution time for the ES approach rises further exponentially with the computational effort for larger number of servers and providers. Therefore, it cannot be considered as a practical solution for the problem. To further examine the near-optimality of FCBB and the genetic approach, we conducted experiments with 10 different requests (in terms of service requirements, graph density, message size, and request inter-arrival time) for each category of 10, 15, and 20 servers. The results are shown in Table 4.4, where we observe that on average, the difference in deployment cost compared with ES is 7% for the FCBB and 1% for genetic approach. Therefore, both FCBB and genetic approach can reach a near-optimal solution without much computational cost.

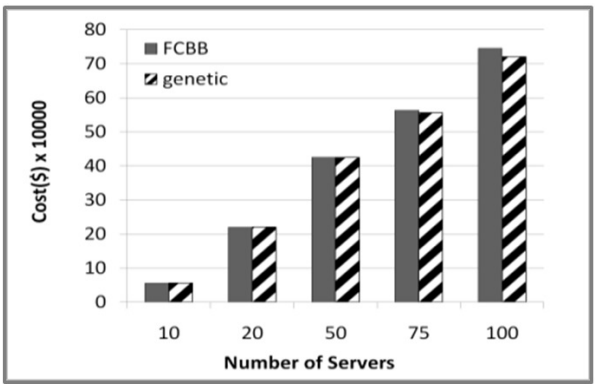
Table 4.4: Mean exhaustive search(es) costs/algorithms costs.

Algorithm	**Number of servers**		
	10	**15**	**20**
ES/FCBB	0.9841	0.9175	0.9013
ES/genetic	0.9952	0.9868	0.9923

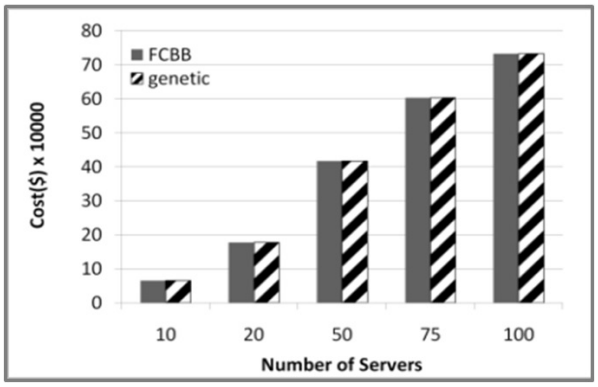
4.7.2 Results of Variation in Request Types on Algorithms Performance and Execution Time

Figures 4.2 and 4.3 depict the performance of the proposed algorithms for different request types (strongly, moderately, and loosely connected) with different number of servers. In the case of workload "a", as the message size is small, differences are comparatively small, except for strongly connected requests (Figure 4.2a and especially for the case of 100 servers where genetic-based approach can save approximately 3% of cost. In other cases of workload "a", when vertices are moderately or poorly connected, the genetic-based approach has better or relatively same performance (regarding the cost) compared to the FCBB algorithm. However, when the message size is larger (workload "b"), as shown in Figure 4.3a, the genetic algorithm in almost all cases outperforms the FCBB algorithm. In Table 4.5, mean execution times for 20 experiments in relation to the number of servers for groups of requests is given. It shows that the execution time of FCBB is negligible compared to the genetic's one. It also shows that adding "forwards checking" feature successfully decreases execution time, especially for the requests that require more than 10 servers and therefore it outperforms the "discard subset" algorithm offered in [82] (the algorithm proposed to solve the web service composition optimization problem with multiple constraints) regarding the execution time while they both could result in the same objective values for all cases.

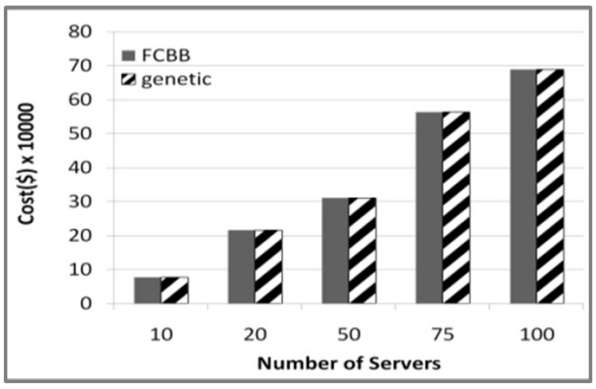
Therefore, the performance of the algorithms differs from one workload to another and when there exists a workload with small message size (like the e-Business workload "a"), performance differences of algorithms are low. In such cases, FCBB can be used to save execution time. However, when the message size increases, they show comparatively higher differences. As a result, when users look for minimizing cost instead of



(a) Strongly connected.

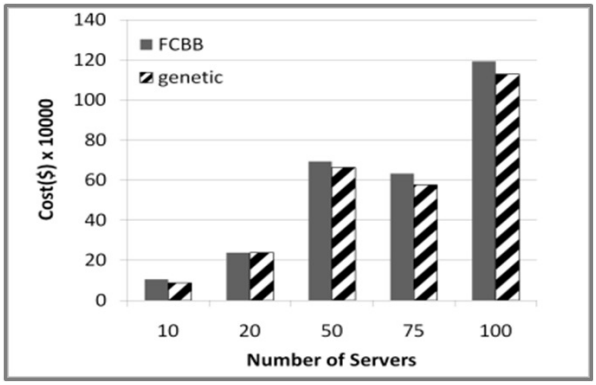


(b) Moderately connected.

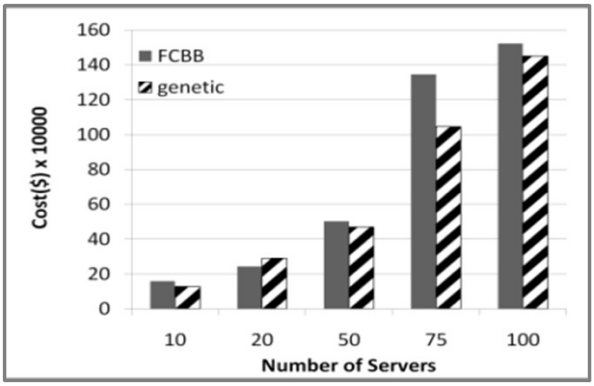


(c) Loosely connected.

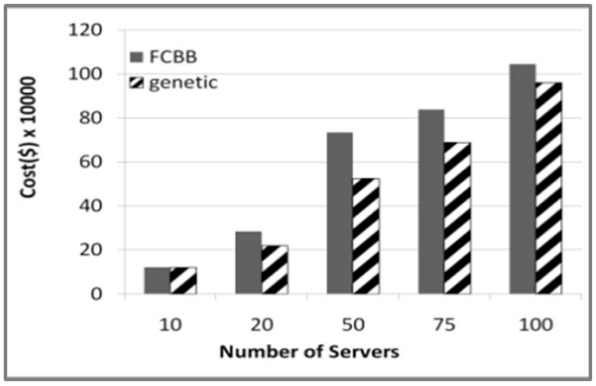
Figure 4.2: Change in connectivity for workload a.



(a) Strongly connected.



(b) Moderately connected.



(c) Loosely connected.

Figure 4.3: Change in connectivity for workload b.

Table 4.5: Mean execution time(s).

Algorithm	Number of servers				
	10	**25**	**50**	**75**	**100**
FCBB	0.103	0.115	0.288	0.407	0.841
Discard subset	0.138	0.271	0.849	2.339	6.091
genetic	31.997	144.426	497.377	1288.056	1814.488

execution time, the genetic-based approach is the more appropriate solution.

4.7.3 Effects of Variation in Request Types and Latency Constraints on Distribution Factor

In this experiment, the objective is to study the possibility of placing a network of appliances on different providers rather than on a single one. For this purpose a metric named "distribution factor" is designed, which shows the proportion of the number of different providers selected to the total number of providers. Table 4.6 shows how a request type (data transfer rate, and graph density as explained in Table 4.2) affects the distribution factor. For the loosely connected requests with loose latency requirement, we conclude that considering multiple Cloud providers decreases the deployment cost while still maintaining the minimum performance requirements (by adhering to latency constraint). For all cases from 10 to 100 servers, when there is a higher data transfer and number of connection between vertices, the distribution factor decrease dramatically. For the majority of cases, it is decreased by more than 75%. It means that FCBB selection algorithms have a tendency to select the same virtual unit provider for all vertices to save on communication cost. The same trend can be observed for the genetic-based approach. However, when the latency constraints are tight, if we consider multiple providers for deployment, the cost will be lower. But still the distribution factor decreases by 25%. Consequently, the experiments show that network of appliances with higher graph densities and data transfer are less likely to be distributed across multiple providers and they are expected to have higher deployment cost.

Table 4.6: Distribution factor.

Request type	Number of servers				
	10	**25**	**50**	**75**	**100**
Loosely connected & Loose latency	44%	55%	55%	55%	44%
Strongly connected	11%	11%	11%	11%	11%
Tight latency	22%	44%	33%	33%	33%

Table 4.7: Effects of the deployment constraints on the cost.

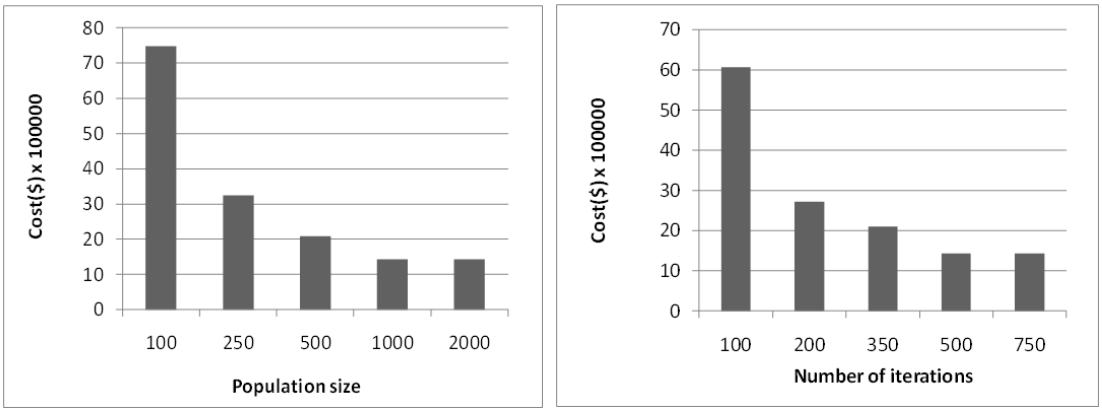
Request type	Average percentage of cost increase
High reliability	5.8117414
Tight latency	10.1966957

4.7.4 Consequence of Variation of Reliability Constraints on Deployment Cost

This experiment is designed to help us understand how characteristics of network of appliances affect the deployment cost when they are migrated to the Cloud. As illustrated in Table 4.7, the deployment cost increases by almost 10% on average when latency requirement is tighter as less number of providers could satisfy latency requirement (lower distribution factor). In addition, demanding providers with higher reliability has slightly increased the cost of deployment which is less than the increase in the case when the latency constraint is tighter.

4.7.5 Varying Iteration Number and Population Size

Figure 4.4a and 4.4b represent the effects of increasing the number of iterations and population size on improvement of objective function. The examined request has 100 highly connected vertices from workload (b). The aim is to show to what extent increasing the number of iterations and population size improves performance of the genetic approach. It can be observed that increasing the number of iterations and population size contribute to the objective function. However, from a certain point (for population size 1000 and for iteration number 500), the improvement is marginal and negligible.



(a) Population size versus cost.

(b) Number of iteration versus cost.

Figure 4.4: Varying iteration number and population size.

4.8 Conclusions

This chapter mainly focused on cross-Cloud Quality of service service modeling and deployment optimization. We investigated the Cloud provider selection problem for deploying a network of appliances and proposed new QoS criteria. The problem of deployment is formulated and tackled by two approaches namely FCBB and genetic-based selection. We evaluated the proposed approaches by a real case study using real data collected from 12 Cloud providers, which showed that the proposed approaches deliver near-optimal solution. Next, they were tested with different types of requests. The results show that when message size increases, approaches present comparatively higher differences, and if execution time is not the main concern of users, genetic-based selection in most cases achieves better value for the objective function. In contrast, if the massage size between appliances is small, FCBB can be used to save on execution time. Further, based on the conducted experiments, we found out that network of appliances with higher graph density and data transfer are less likely (in contrast to requests with lower data transfer) to be distributed across multiple providers. However, for requests with tight latency requirements, appliances are still placed across multiple providers to save on deployment cost. Further, we show how the iteration number and population size affect the performance of the genetic algorithm.

In the next chapter, a Cloud migration scenario is investigated where user look for a Cloud service composition which maximizes reliability and minimizes deployment time and cost. The chapter proposes a selection methodology for this purpose that also simplifies the process of migration for non-experts who do not have clear idea regarding their preferences and compatibility of Cloud services in the composition.

Chapter 5

Cloud Service Composition Under Fuzzy Preferences of Users

This chapter investigates challenges of Cloud service composition, where a single Cloud service, on its own, cannot satisfy all the user requirements. Cloud service composition, which includes several tasks such as discovery, compatibility checking, selection, and deployment, is a complex process and users find it difficult to select the best one among the hundreds, if not thousands, of possible compositions available. In this chapter, we develop an ontology-based approach to analyze appliance and virtual unit compatibility constraints to simplify the process of deployment for unskilled users by applying reasoning on the expert knowledge. In addition, minimizing effort of users in expressing their preferences is pivotal for the success of the proposed solution. Therefore, we have applied combination of evolutionary multi-objective algorithms and fuzzy logic for composition optimization to let users express their needs in high-level linguistics terms which brings a great comfort to them compared to systems that force users to assign exact weight for all preferences. Our approach is validated using a real Cloud service composition request to show its effectiveness and applicability. In addition, performances of several multi-objective algorithms for the composition problem have been compared and the effectiveness of applying fuzzy logic to deal with vague preferences of users in the problem is tested.

5.1 Introduction

IN order to offer their solutions in the Cloud, service providers can either utilize the Platform-as-a-Service (PaaS) offerings such as Google App Engine [1], or develop their own hosting environments by leasing virtual machines from Infrastructure-as-a-Service (IaaS) providers like Amazon EC2[2]. However, most PaaS services have restrictions on

[1]Google App Engine. `http://appengine.google.com/`
[2]Amazon EC2. `http://aws.amazon.com/ec2/`

the programming language, development platform, and databases that can be used to develop applications. For example Google App Engine mainly supports applications developed using Java or Python only. Such restrictions encourage service providers to build their own platforms using IaaS service offerings.

One of the key challenges in building a platform for deploying applications is to automatically compose, configure, and deploy the necessary application infrastructure that consists of number of different components. If we consider the deployment requirements of a web application service provider, it will include security devices (e.g. Firewall), load balancers, web servers, application servers, database servers, and storage devices. Setting up such a complex combination of appliances is costly and error prone even in traditional hosting environments [158], let alone in Clouds. Virtual appliances provide an elegant solution for this problem.

Furthermore, a user may require more than one virtual appliance and machine, and a composition of them that can meet all the requirements of users is required. However, the selection of the best composition is a complex task. The best choices found for individual appliances cannot be simply put together as some appliances will not be compatible with the hosting environment. For example, if an appliance format is OVF it cannot currently be deployed on Amazon EC2 as it only accept appliance with AMI format[3]. In addition to that, there exist legal constraints imposed by countries such as the USA on importing and exporting of appliances from a provider to another.

In this chapter, to simplify the process of selecting the best virtual appliance and unit (computing instance) composition, a novel framework is presented. The composition problem that we have considered is to find the best combination of compatible appliances and virtual units that minimizes the **deployment cost** and **deployment time**, and maximizes the **reliability**. In this framework, first we model the compatibility of appliances using Web Service Modeling Ontology (WSMO) [145] and utilize Web Service Modeling Language reasoner (WSML-reasoner) [145] to check the compatibility of services in a composition. Then we consider three user QoS objectives: the lowest cost, quickest deployment time, and the highest reliability. After that, different algorithms are utilized to

[3]Amazon Machine Images (AMIs). `https://aws.amazon.com/amis`

solve this multi-objective problem and their performances are compared. Finally we refine the selected appliance compositions based on users' preferences by utilizing a fuzzy logic approach.

To further highlight the contribution of this chapter first we describes issues with current research and practices and then how those have been addressed in this chapter.

5.1.1 Issues with Current Virtual Appliance Management Systems

Public Clouds have different deployment model, in IBM smart Cloud model[4] users first have to select their software solution form what is called asset catalogs and then select the instance configuration for that virtual appliance. In addition, IBM smart Cloud provides an Image composition toolkit which builds required virtual appliance from images in IBM smart Cloud repository or by importing images from other Cloud repositories. However, IBM smart Cloud does not provide any ranking system to choose the best Cloud provider, instance type and asset catalogue for the deployment. In addition, it also lacks an approach to identify which images are compatible with different instance type of different Clouds. Moreover, some countries like USA impose restriction on the location of imported and exported images, and computing units which introduces legal constraints which have to be considered when services form different Cloud providers are composed. Amazon EC2, GoGrid, Rackspace, and other Key players in the IaaS market, although constitute different deployment models, however none provides a solution for composing Cloud services based on user's functional and non-functional requirements as cost, the total deployment time, reliability, and compatibility constraints. Therefore, in this chapter we propose:

1. A Cloud service composition optimization technique that allows non-expert Cloud users set their preferences using high level if-then rules and get user friendly recommendations on the composition solution's prominence. Majority of end users is avoiding systems which incur complexity in capturing their constraints, objectives and preferences. An example of such systems is the one which require users to as-

[4]IBM smart Cloud. `http://www-935.ibm.com/services/au/en/cloud-enterprise/`

sign weights to their objectives. In this case, users have to find a way to prioritizes their preferences and then map them to weights. Then, the system has to find out how precise users have gone through the process of weight assignment. To tackle this issue, a major objective of this research is to offer ranking system for Cloud service (i.e. virtual appliance and machine) composition that let them expressing their preferences conveniently using linguistic high-level rules. Our system then utilizes Multi-Objective evolutionary approaches, and fuzzy inference system to precisely capture the entered preferences for ranking purpose.

2. An approach to helps non-expert users with limited or no knowledge on legal and virtual appliance image format compatibility issues to deploy their services flawlessly. For this purpose, we first automatically build a repository of Cloud services in WSML and then enrich it with experts' knowledge (lawyers, software engineers, system administrators, etc) on the aforementioned constraints. The knowledgebase then is used for reasoning in an algorithm that identifies whether set of Cloud service consists of virtual appliance and units are compatible or not.

5.2 Composition Problem

The first step in service composition is to model the appliances (based on cost, size, functionality, etc.) and QoS requirements of users. This step not only allows appliance and virtual unit providers to advertise QoS of their services, but also provides a way for end users to express their QoS preferences. As shown in Chapter 3, in this work, WSML is extended and used for appliance and virtual unit QoS modeling. Unlike syntax-based modeling, WSML, which is a semantic based language, offers a common language for all parties involved in the composition to express and understand each other's requirements. Similarly, in this chapter we have developed ontology-based modeling for all the compatibility and QoS requirements.

Currently there is no single directory that lists all the available virtual appliances and machines. Hence as a first step we built a directory by aggregating the virtual appliance

and unit information from the vendors such as VMware Virtual Appliance Marketplace[5], AMI directory[6], Parallels Virtual Appliance Directory [7], GoGrid[8], and TurnKey Linux[9]. The information gathered includes the name of the virtual appliance and unit, purpose, operating system and other software utilized, price, virtual unit characteristics, size of the appliance, Cloud service provider from whom it needs to be downloaded and any restrictions and compatibility requirements in order to use the appliance. To build this database we have utilized the Application Programming Interfaces (API) and web services provided by vendors (e.g. Amazon EC2 API) and for others (e.g. VMWare) we crawled the web pages where the appliance information is listed. One of the important aspects to consider is that the available appliance information and the provider details are constantly changing and rapidly increasing. Hence the data gathering module, as explained in Chapter 3, has been designed so that it can keep the data up-to-date and is extensible in future for any new appliance vendors. For more information on this module refer to Chapter 3.

5.2.1 Evaluation of Composition Criteria

The composition problem is to find the best combination of compatible and composable appliances and virtual machines that minimizes the deployment cost and deployment time, and maximizes the reliability while adhering to composability constraints. A formal description of the problem is given below.

Provider Model

Let m be the total number of providers. Each provider, can provide virtual appliances, virtual units or both, and is represented as shown in Equation (5.1).

[5]VMware Virtual Appliance Marketplace. `www.vmware.com/appliances/`
[6]Amazon Machine Images (AMIs). `https://aws.amazon.com/amis`
[7]Parallels Virtual Appliance Directory. `http://www.parallels.com/ptn/download/va/`
[8]GoGrid. `http://www.gogrid.com`
[9]TurnKey Linux. `http://www.turnkeylinux.org/all`

$$\begin{aligned} \text{Provider } P_k : \left\{ \{a\}, \{vm\}, C_{d_{ext}}, C_{d_{int}} \right\} \\ where\, 0 < k \leq m \end{aligned} \tag{5.1}$$

where a, vm, $C_{d_{ext}}$, $C_{d_{int}}$ denotes appliance, virtual machine, Cost of external data transfer and Cost of internal data transfer respectively. A virtual appliance a can be represented by a tuple of five elements (Equation (5.2)) : appliance type, cost, license type, compatibility list, and size.

$$a : \{ApplianceType, Cost, LicenseType, CompatibilityList, Size\} \tag{5.2}$$

A virtual machine vm can be formally described as a tuple with two elements as shown in Equation (5.3).

$$vm : \{MachineType, Cost\} \tag{5.3}$$

The user request for the appliance composition can be translated into a weighted graph $G(V, E)$ where each vertex represents a server that consists of a virtual appliance running on a virtual machine. Server corresponding to vertex v is represented by Equation (5.4).

$$S_v = \{a_v, vm_v\}, \forall\, v \in V \tag{5.4}$$

Each edge $e\{v, v'\} \in E$ indicates that corresponding servers communicate. The Data Transfer Rate between these connected vertexes (i.e. one server to another) is given by the weight associated to edge e.

Compatibility

When multiple Cloud services (i.e. virtual appliances and units) are composed together, they should be compatible with each other. In this work, we consider legal and image format compatibility constraints. However, it should be noted that in reality there will be

other compatibility constraints such as compatibility between the products installed on the appliances.

- **Virtual appliance image format compatibility:** Before we finalize the deployment plan, we have to find out whether the image formats of chosen set of virtual appliances are compatible with the destination virtual unit provider. As it is illustrated in the first row of Table 5.1, a sample ontology-based reasoning on the built knowledgebase (as shown in Appendix A in Section A.1) shows that image with ID of "aki00806369" is compatible with the large instance type provided by Amazon EC2.

- **Legal requirements:** In Cloud infrastructure, virtual machines can be deployed in data centers located in different parts of the world. However, there are legal requirements, for example US restrictions on exporting encryption technology [10], that prevent the export and deployment of software developed in one country to another. Hence, we need to ensure that the virtual appliances can be legally deployed on the selected virtual units. The second row of the Table 5.1 presents a query which sets appliance with the ID of "aki00806369" compilable to only virtual units provided by Clouds located in a same country where the appliance provider is situated.

To evaluate the compatibility requirements, first the Ontology-based vocabulary is created using WSML, as shown in the Appendix A.1[line 151-156] in which compatibility constraints are imposed by experts in the form of axioms in the ontology or alternatively by reasoning on the ontology. For example, the ontology listed in Appendix A.1 shows how an axiom (set by an expert) enforces that appliances can only be deployed on virtual units provided by Cloud providers which are located in the same country as the appliance provider. After building required ontology, we can exploit the advantages of Description Logic (DL) to query the knowledgebase and check compatibility constraints of composition candidates. A WSML-DL query sample given in Table 5.1 shows which virtual unit is compatible (legally and regarding image format) to the appliance with the ID of "aki00806369".

Table 5.1: Compatibility reasoning for Cloud service composition.

Query String	**Result**
?x[isCompatipleWith hasValue ?y] :- ?x memberOf virtualAppliance and ?x [hasName hasValue ?v] and ?v=aki00806369 and ?y memberOf virtualUnit and ?y[hasProvider hasValue?pvu] and ?pvu [supportVmFormat hasValue ?supportedFormat] and ?x[hasFormat hasValue ?format] and ?format[hasName hasValue ?formatName] and ?supportedFormat[hasName hasValue ?supportedFormatName] and ?supportedFormatName=?formatName.	*?formatName: "AMI" ?v: "aki00806369" ?pvu: AmazonCalifornia ?p: AmazonCalifornia ?supportedFormatName: "AMI" ?format: AMI ?supportedFormat: AMI ?y: largeInstance ?x: aki00806369*
?x [isCompatipleWith hasValue ?y]:- ?x memberOf virtualAppliance and ?x [hasName hasValue ?v] and ?v=aki00806369 and ?x[hasProvider hasValue ?p] and ?y memberOf virtualUnit and ?y[hasProvider hasValue ?pvu] and ?p[hasCountry hasValue ?capp] and ?pvu [hasCountry hasValue ?cvu] and?capp[hasName hasValue ?cappName] and ?cvu[hasName hasValue ?cvuName] and ?cappName=?cvuName.	*?v: "aki00806369" ?pvu: AmazonCalifornia ?p: AmazonCalifornia ?capp: USA ?cappName: "USA" ?cvu: USA ?cvuName: "USA" ?y: largeInstance ?x: aki00806369*

Our objective is to achieve full compatibility among the appliances in the composition. Based on the compatibility constraints considered in our work, the compatibility (C) can be calculated based on Equation (5.5).

$$C = \begin{cases} 0 & \text{if there exists at lease one pair of incompatible services;} \\ 1 & \text{otherwise.} \end{cases} \tag{5.5}$$

In addition, Algorithm 4 illustrates the process to evaluate the compatibility of the services in a composition. The algorithm checks the compatibility constraint (by sending the related query to WSML-reasoner) for each pair of appliance and virtual unit in the

composition, and the only compositions where all constraint queries are satisfied would be returned as valid.

Algorithm 4: Compatibility evaluation algorithm

Input: Composition c, Constraint List (cl) such as Image Format and Legal Compatibility

Output: Composition Validity

```
1  if CompositionValidity(c, cl) Exists in Cache then
2      ValidComposition = GetCompositionValidtyFromCache();
3  ValidComposition=True;
4  foreach Appliance a and Virtual Unit vu In Composition do
5      foreach Constraint t in ConstraintList do
6          if Compatibility (t, a, vu) Exists in Cache then
7              ValidComposition = Get Compatibility FromCache();
8          else
9              ValidComposition = CheckCompatibilitybyReasoning(t, a, vu);
10         Insert Compatibility toCache(t, a, vu);
11     if ValidComposition=False then
12         break;
13 InsertCompositionValiditytoCache();
14 return ValidComposition;
```

Cost

The costs involved in procuring and using virtual appliances can be categorized as follows:

- **Acquisition Cost:** Costs involved in purchasing the virtual appliance, such as licensing cost, cost of the virtual machine and any costs associated with deployment such as the data transfer costs to transfer the appliances to the virtual machine at

the IaaS provider.

To build a server S_v, lets assume appliance a_v is obtained from provider P_k and virtual machine vm_v is obtained from provider P_l. If this server will be running for a duration of T (lease period), the Equation (5.6) shows the Acquisition Cost.

$$AqCost\left(S_v\right) = Cost\left(a_{v,P_k}\right) \times T + Cost\left(vm_{v,P_l}\right) \times T + AppTransCost\left(k,l\right) \tag{5.6}$$

where $Cost\left(a_{v,P_k}\right)$ is the cost of appliance per unit of time, $Cost\left(vm_{v,P_l}\right)$ is the cost of virtual machine per unit of time, and the cost of appliance transfer from appliance provider k to virtual unit provider l is given by Equation (5.7).

$$AppTransCost\left(k,l\right) = \begin{cases} 0 & \text{if } k = l; \\ Size\left(a_{v,P_k}\right) & \\ \times C_{d_{ext}}\left(P_l\right) & \text{if } k \neq l. \end{cases} \tag{5.7}$$

- **Ongoing Cost:** This will include the costs of running the virtual appliance, such as the cost of data transfers. In this work we consider only the costs associated with data transfers as ongoing costs.

  Let v, v' be two vertexes (servers) on provider l, connected by edge $e\left\{v,v'\right\}$. The data transfer cost between the two servers is given by Equation (5.8):

$$TransCost\left(e\left\{v,v'\right\}\right) = Size\left(Data_{e\{v,v'\}}\right) \times C_{d_{int}} \tag{5.8}$$

- **Decommissioning Cost:** Decommissioning cost will primarily include archival and removal costs of the data at the end of the application life cycle such as the data sanitisation, and henceforth as shown in Equation (5.9) will depend on the size of the data stored. The amount of data stored will vary from server to server.

$$DecomCost\left(S_v\right) = CostPerUnit \times SizeOfData_v \tag{5.9}$$

Based on the costs calculated above, Total Cost (TC) can be computed as shown in Equation (5.10),

$$TC = \sum_{v \in V} AqCost\left(S_v\right) + \sum_{e \in Ev, v' \in V} TransCost\left(e\left\{v, v'\right\}\right) + \sum_{v \in V} DecomCost\left(S_v\right) \tag{5.10}$$

Deployment Time

Virtual appliances significantly minimize the time required to build and configure the necessary independent components. However, the size of the virtual appliances will range from a few megabytes to tens of gigabytes depending on the applications installed on them and will impact the time required to transfer and deploy the appliances from the appliance provider to the virtual machine provider. Hence, we consider the deployment time as one of the composition objectives. The time required to deploy a given appliance a_v obtained from provider P_k on a virtual machine vm_v obtained from provider P_l is given by Equation (5.11),

$$DT\left(a_{v,P_k}, vm_{v,P_l}\right) = \begin{cases} InitTime\left(vm_{v,P_l}\right) \text{ if } k = l; \\ InitTime\left(vm_{v,P_l}\right) + \\ \frac{Size\left(a_{v,P_k}\right)}{DataTransferRate(P_k,P_l)} \text{if } k \neq l. \end{cases} \tag{5.11}$$

where $InitTime\left(vm_{v,P_l}\right)$ is the time required to initialize the appliance on the virtual unit. As appliances can be deployed in parallel, Total Deployment Time (TD) will be given by Equation (5.12),

$$TD = Max\left\{DT\left(a_{v,P_k}, vm_{v,P_l}\right)\right\} \tag{5.12}$$

Reliability

Finally, we consider the reliability as another QoS objective. To measure the IaaS providers' reliability we introduce Service Level Agreement (SLA) Confidence Level (SCL) in Chapter 4, which measures how reliably a provider is offering their services based on the

Table 5.2: Virtual appliance composition objectives.

Criteria	**Metric**	**Type**	**Requirement**
Compatibility (C)	C	Constraint	To be equal to 1
Total Cost (TC)	$	Objective	To be minimized
Total Deployment Time (TD)	mS	Objective	To be minimized
Total Reliability (TR)	SCL	Objective	To be maximized

binding SLA. Total reliability can be computed according to Equations (5.13) and (5.14) by a third party, which is responsible for monitoring the SLA.

$$SCL = \sum_{j=1}^{k} I_j \times SCL_j \quad where \quad SCL_j = MSQ_{jt} - SLO_{jt} \tag{5.13}$$

$$\text{Total Reliability } TR = \sum_{v \in V} SCL_v \tag{5.14}$$

where SCL_j is the SLA confidence level for QoS criteria j of a Cloud service; I_j is the importance of QoS criteria j for user; MSQ_{jt} is the monitored value of QoS criteria j for a period of t; SLO_{jt} is promised QoS criteria j value in SLA for the period of t, and k is number of QoS criteria monitored.

Although, our SCL metric can model various Cloud specific QoS criteria, in this work we consider only availability as it is the primary criteria specified by current IaaS providers in their SLAs.

5.2.2 Overall Objectives

The final objective is to find a fully compatible service composition that minimizes the deployment cost and time, and improves the reliability of the composition while adhering to compatibility constraints as shown in Table 5.2.

5.3 Composition Optimization Technique

In our problem we consider three user objectives, the lowest cost, quickest deployment time, and the highest reliability. This makes it infeasible to find an optimal composition as these objectives can conflict with each other. One way to address this problem is to convert the appliance composition problem to a single-objective problem by asking users to give weights to all the objectives. However, that approach is error prone and impractical, as not all the users will have the knowledge to appropriately assign weights to objectives. Furthermore, as the composition will depend on the capability of users to assign proper weight to the objectives, we have to find a way to evaluate the knowledge of users about each objective to assure the performance of the approach. As shown in Figure 5.1, we have tackled these challenges in two steps. First we find the Pareto front [179] composition solutions using different multi-objective algorithms (OMOPSO, NSGA-II, and SPEA-II). We have used Jmetal [48] for this purpose which allows us to define problems by defining each gene (variable) to point to Cloud service candidates as show in Figure 5.1. Then, we defined necessary fitness functions for all the defined objectives and choose the algorithm to solve the problem. In the next step, we help users to describe their preferences using high level "if-then" rules, which builds our fuzzy engine rule-base to rank the Pareto front acquired from the previous step. By doing this we acquire Pareto front once and then filter it according to the user preferences. This has a great advantage when compared to the method offered in [14] as our method does not require to search the solution space each time the user preferences change. Next, we overview our optimization approach and then explain our fuzzy logic based ranking approach.

Evolutionary algorithms have been effectively applied for solving optimization problems. Among them NSGA-II [41] and SPEA-II [190] outperform many other genetic optimization algorithms [41]. Nevertheless, recently other meta-heuristics which work based on swarm intelligence such as Particle Swarm Optimization (OMOPSO) [153] are also used to tackle multi-objective optimization problems. In this work, we perform a comparative study between the algorithms NSGA-II, SPEA-II, and OMOPSO for the appliance composition problem to determine which of them will be suitable for our problem. To the best of our knowledge none of previous works have compared multi-objectives

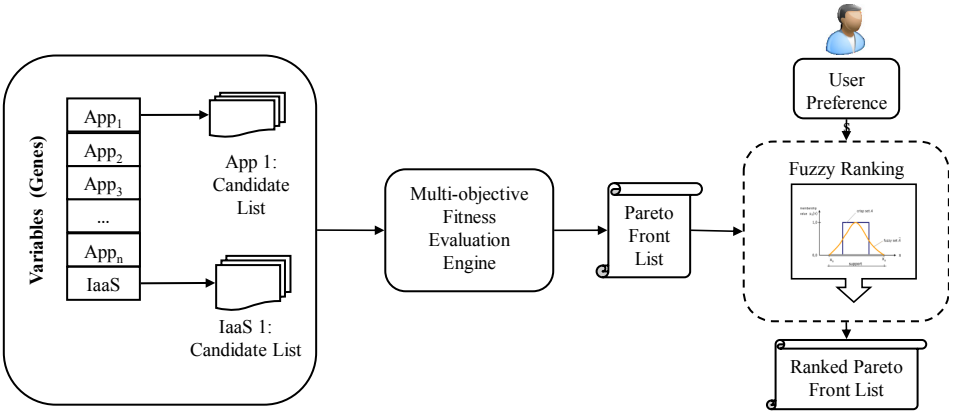


Figure 5.1: Virtual appliance composition optimization approach.

algorithms and then combined them with fuzzy logic for Cloud service composition.

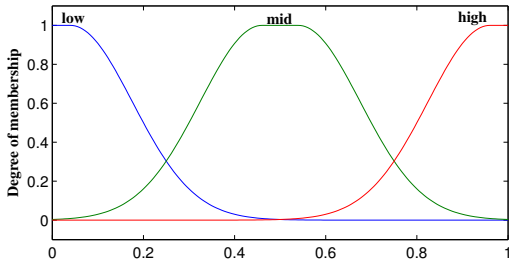
Fuzzy logic based ranking: Our proposed fuzzy inference engine includes three inputs and one output. Inputs of the system are normalized Deployment time (DT), Deployment Cost (DC), and Reliability of composition, which are all described based on the same membership functions in Figure 5.2a. Output of the fuzzy engine as shown in Figure 5.2b represents how desirable the current set of inputs are based on the fuzzy rule-based indication. For example, the value "0" in output means the solution is highly undesirable whereas the value "1" shows that the solution is highly desirable. Fuzzy rules should be defined by the user to describe their QoS preferences. For example a rule can be defined as:

if **DT** is *low* and **DC** is *low* and **Reliability** is *high*, composition is **highly desirable**.

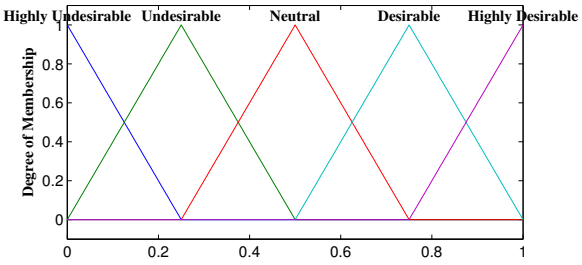
Table 5.3 shows some sample rules that can be expressed by the users. In this work we use a fuzzy engine based on Mamdani inference system [116] with Centroid of area defuzzification strategy. Readers can refer to [183] for detailed information on fuzzy inference systems. Once rules and defuzzification strategy are defined, a fuzzy inference system can map the inputs of fuzzy engine to its output. Figure 5.3 demonstrates how deployment time, cost, and reliability are mapped to composition desirability when all rules are set.

Table 5.3: Sample high level rules set by users.

DC	DT	Reliability	Composition Desirability
Low	Low	Low	Undesirable
Low	Low	Medium	Neutral
Low	Low	High	Highly Desirable
High	Low	High	Not sure
Low	High	High	Not sure

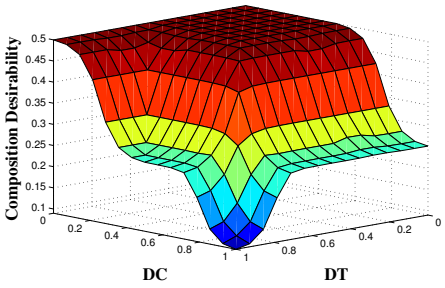


(a) Input fuzzy sets.

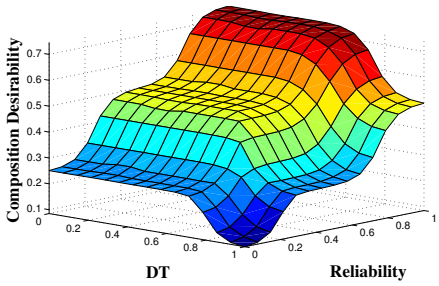


(b) Output fuzzy sets.

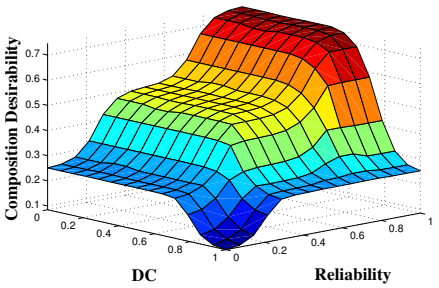
Figure 5.2: Fuzzy engine input and output fuzzy sets.



(a) Mapping of DT and DC to composition desirability.



(b) Mapping of DT and reliability to composition desirability.



(c) Mapping of DC and reliability to composition desirability.

Figure 5.3: Mapping from QoS criteria values to composition desirability value.

5.4 Performance Evaluation

In order to realize and evaluate the proposed approach, a number of components and technologies are utilized as explained in Table 5.4. Most importantly, multi-objective algorithms are implemented in jMetal [48] and then Pareto Front composition solutions have been passed to our fuzzy-logic based ranking components, which utilizes jFuzzyLogic [28] to define the membership functions and rules.

Table 5.4: Technologies used for appliance composition tasks.

Component	**Supported by**	**Description**
Inter-Cloud API	jclouds [10]	jclouds presents cloud-agnostic abstractions, with stable implementations of Compute Service which simplifies the task of managing machines in the cloud.
Monitoring	Cloud Harmony API [29]	Providing objective, measurable, and unbiased analysis on cloud services.
Discovery	WSMO Discovery Engine	Developed to exploit semantic descriptions of services based on WSMO and WSML for the dynamic discovery
Compatibility Evaluation	WSML-Reasoner	WSML-Reasoner
Translator	WSMO Grounding [101]	to transfer description of Clouds services to WSML
Fuzzy Inference	jFuzzyLogic [28]	jFuzzyLogic is a fuzzy logic package written in java. It implements Fuzzy control language (FCL) specification.
Multi-Objective Optimization	jMetal [48]	jMetal is an object-oriented Java-based framework aimed at the development, experimentation, and study of metaheuristics for solving multi-objective optimization problems.

5.4.1 Request Modeling and Data Collection

In this experiment we consider a case study of a web-based collaboration application that allows users to store, manage, and share documents and drawings related to large construction projects. The composition of 14 appliances required for this application is described in Table 5.5. To meet these requirements, our objective is to find the best Cloud service composition.

Appliance Discovery

Table 5.6 shows an example of the data gathered during the appliance discovery phase. The last column of Table 5.5 shows the number of matching appliances available for our request model just from one provider (VMWare Market Place). This information alone shows the scale of choices available to the user and the complexity of the problem. We expect the number of available appliances and providers will increase rapidly over time.

Cloud Service Provider Information

Cloud service providers details, such as the promised availability and monitored availability (for calculating SCL), data transfer costs, and data throughput between two Cloud service providers (to estimate the transfer time of virtual appliances) are obtained using the CloudHarmony service[11]. When the required data is not available from them, it is directly obtained from the Cloud service provider. Similar to the virtual unit and appliance directory, the module to obtain the Cloud service provider information also can be used to update and extend the information in future.

5.4.2 Results

There are three main experiment results presented in this section: 1) a performance comparison between the OMOPSO, NSGA-II, and SPEA-II algorithms for the real case study, 2) an evaluation of the effectiveness of fuzzy inference system on handling imprecise user preferences, and 3) an analysis of the execution-time of the ontology-based compatibility

[11] CloudHarmony. `http://cloudharmony.com/`

Table 5.5: Application composition for the request model.

Appliance	**Purpose**	**Required**	**Available**
Firewall	For packet filtering and port blocking	01	97
Intrusion Detection	Pattern based intrusion detection and malicious file scanning	01	22
Load Balancer	Distributes the load between several web application servers	01	41
Web Server	Hosts the web application and accepts end user requests	04	386
Application Server	Implements the business logic	03	492
Database Server	Hosts all the data related to the application	01	130
Database Reporting Server	Dedicated server used to produce reports	01	19
Email Server	Email server is used to send outgoing notifications (E.g. Availability of a new document) and process incoming emails (E.g. Document submissions)	01	134
Server Health Monitoring	Monitors each component in the composition and takes any required corrective action including sending alerts to system administrators	01	12

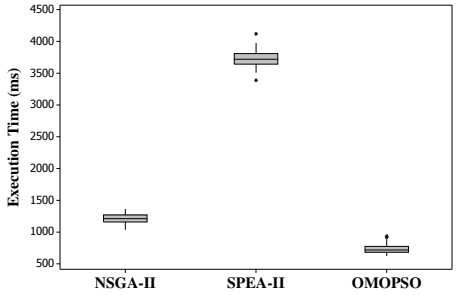
checking algorithm. NSGA-II and SPEA-II algorithms use a population of size of 100, and maximum number evaluations of 25000. OMOPSO is configured with 100 particles, with a maximum of 100 leaders and maximum number of iterations of 250. We have carried out 40 independent runs per experiment and then statistically analyzed results.

Table 5.6: Sample appliance directory.

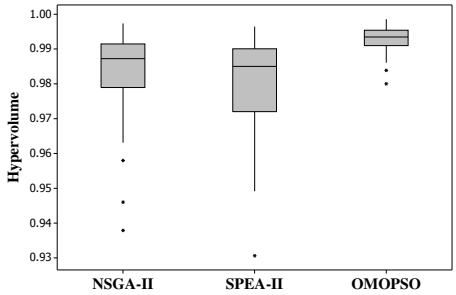
Name	Purpose	Software	Price	Size	Provider
Tomcat on Apache	Web server	Ubuntu, Apache, Java, and MySQL	Free	149MB	TurnKey
IPCop Firewall	Firewall	Linux	Free	40MB	VMWare

Performance of single-objective optimization algorithms can be evaluated by analyzing the best value achieved by the algorithms. However, in the case of multi-objective optimization, it is not practical. There are quality indicators, which can be used to evaluate the quality of the obtained set of solutions, and determine the convergence and diversity properties of algorithms. In our experiments the algorithms are compared using Execution Time, Inverse Generational Distance (IGD) [190] which determines convergence of algorithms, and Spread which determines diversity [42] and Hypervolume which determines both. In addition, it has been complemented by the application of statistical tests to ensure the significance of the results. Otherwise, the drawn conclusions may be incorrect as the differences between the algorithms could have occurred by chance.

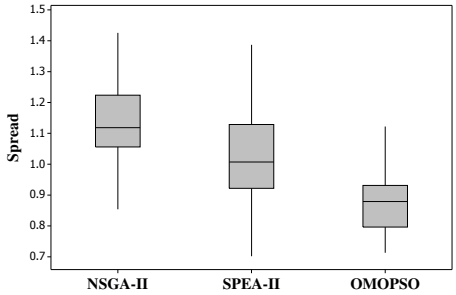
As in our problem the Pareto fronts are not known, applying the aforementioned quality indicators is not possible. As a common approach, we build a reference front by collecting all the results of 100 runs of the algorithms. This approach helps us to compare the relative performance of algorithms (Jmetal [48] provides an automatic way of obtain the reference front). The Hypervolume [166] indicator has been widely used and is strictly Pareto-compliant [32]. If the indicator values a solution higher than the other, and that solution set dominates the other one's set, the indicator is called Patreto-complaint. The Hypervolume calculates the volume of dominated portion of the objective space. In addition to Hypervolume, we have also used other two widely used and recommended indicators [42], namely IGD and spread. As shown in Equation (5.15), IGD works out the average distance of the obtained solution points from the optimal Pareto fronts. Where the d_i is the Euclidean distance between the obtained solution points and the closest member of optimal Pareto front and n is the number of points in the optimal Pareto front. Hence, when the achieved solution is in the optimal Pareto front the IGD is



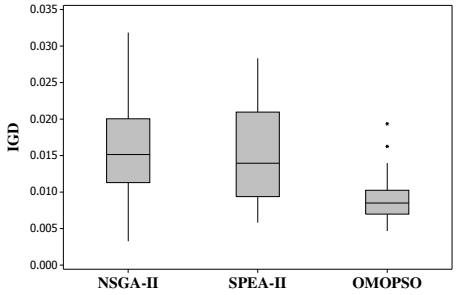
(a) Execution time.



(b) Hypervolume.



(c) Spread.



(d) IGD.

Figure 5.4: Comparison of algorithms

Table 5.7: Statistical comparison of algorithms.

95% CI for Difference	**ANOVA Test**			
	Execution time	**Spread**	**IGD**	**Hypervolume**
NSGA-II-SPEA-II	(-2469.5,-2579.4)	(0.0476,0.1820)	Not Significant	Not Significant
NSGA-II-OMOPSO	(420.4, 531.1)	(0.2002,0.3355)	(0.003222,0.009074)	(0.00599,0.01848)
SPEA-II-OMOPSO	(2944.8, 3055.5)	(0.0854,0.2206)	(0.002913,0.008765)	(0.00372,0.01620)

equal to 0.

$$IGD = \frac{\sqrt{\sum_{i=1}^{n} {d_i}^2}}{n} \tag{5.15}$$

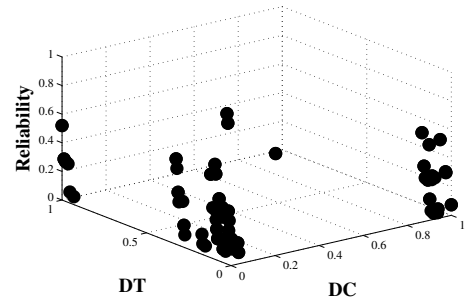
In addition, spread is a metric to calculate the broadness and calculated based on Equation (5.16).

$$Spread = \frac{d_f + d_l + \sum_{i=1}^{N-1} \left| d_i - \bar{d} \right|}{d_f + d_l + (N-1)\bar{d}} \tag{5.16}$$

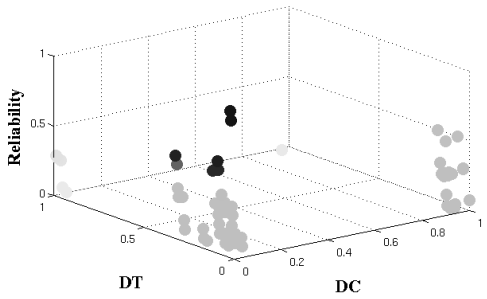
where d_i is the Euclidean distance between consecutive solutions in the obtained solutions. d_f and d_l are the Euclidean distance between the boundary solutions of the obtained pareto front set. When spread is equal to zero, it means obtained solutions are well diverse.

The spread, IGD, Hypervolume, and execution time of three algorithms are compared using Analysis of Variance (ANOVA) test, as the quality values are normally distributed and there is no strong evidence to indicate that variance is not constant. From the ANOVA table the P-value is less than 0.001, which strongly suggests that there are differences in mean spread, IGD, Hypervolume, and execution time between the algorithms. The algorithm that obtains smaller spread is capable of acquiring a set of non-dominated composition solutions with better diversity. In addition, if an algorithm achieves smaller value for IGD, it is better in converging to Pareto-optimal front. However, algorithms with larger value of Hypervolume are more desirable.

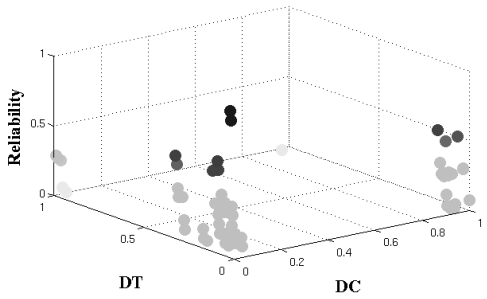
In our experiment, when Hypervolume is used for comparison, OMOPSO outperforms NSGA-II and SPEA-II as shown in Table 5.7 and Figure 5.4b. In addition, if spread is used for diversity comparison, as shown in Table 5.7 and Figure 5.4c, 95% confidence



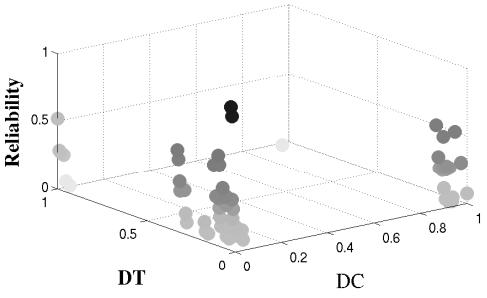
(a) NSGA-II pareto front appliance composition solutions without any user preferences.



(b) Coloring NSGA-II output using fuzzy Logic based on 9 user preference rules.



(c) Coloring NSGA-II output using fuzzy Logic based on 18 user preference rules.



(d) Coloring NSGA-II output using fuzzy Logic based on 27 user preference rules.

Figure 5.5: Appliance composition optimization results

interval (CI) for the difference between NSGA-II - OMOPSO and SPEA-II - OMOPSO are (0.2002, 0.3355) and (0.0854, 0.2206) respectively. This shows the better suitability of OMOPSO in achieving higher diversity. In addition, SPEA-II outperforms NSGA-II in terms of Spread. Moreover, as Figure 5.4a illustrates OMOPSO has the lowest execution time. Furthermore NSGA-II performs better than the SPEA-II in terms of execution time. For the convergence comparison, IGD has been considered. As shown in Figure 5.4d and in Table 5.7, 95% CI for differences between NSGA-II - OMOPSO and SPEA-II - OMOPSO are reported as (0.003222, 0.009074) and (0.002913, 0.008765) respectively, which shows OMOPSO is better in converging to Pareto-optimal front.

Figure 5.5a shows the output of NSGA-II for the case study appliance composition without considering any user preferences. In this case users would receive a set of non-dominated composition solutions. It indicates that all the composition solutions have their own outstanding characteristics and none of the points could dominate the others.

In reality, Cloud users at least have some vague idea regarding their objectives which can be captured by asking them to set high level linguistic rules. To emulate the user behavior, in our experiment, 27 sample fuzzy rules are designed similar to the one described in Section 5.3. Based on these rules, by applying fuzzy inference system, we mapped points in Figure 5.5a to a number between 0 (least desirable) and 1 (most desirable). That number was used to color the points in a way that if the composition solution is more appealing to the user, the corresponding point is darker. As shown in Figures 5.5b and 5.5c when the number of preferences defined by the user using the if-then rule increases, the solution's prominence also becomes clearer. As it can be seen in Figure 5.5d, when all the 27 rules are set (none of them set as "not sure"), the majority of points have distinct colors, however as the number of rules set to "not sure" increases more and more solutions will have the same color. As Figure 5.5d shows two points at the center are the darkest and therefore are the most appealing solutions. Using these techniques non-expert Cloud users are now able set their preferences using high level if-then rules and get user friendly recommendations on the solution's prominence. This experiment also shows how the system could be useful for end user with different levels of knowledge. Traditionally when end users are asked to assign weights to the objectives, they

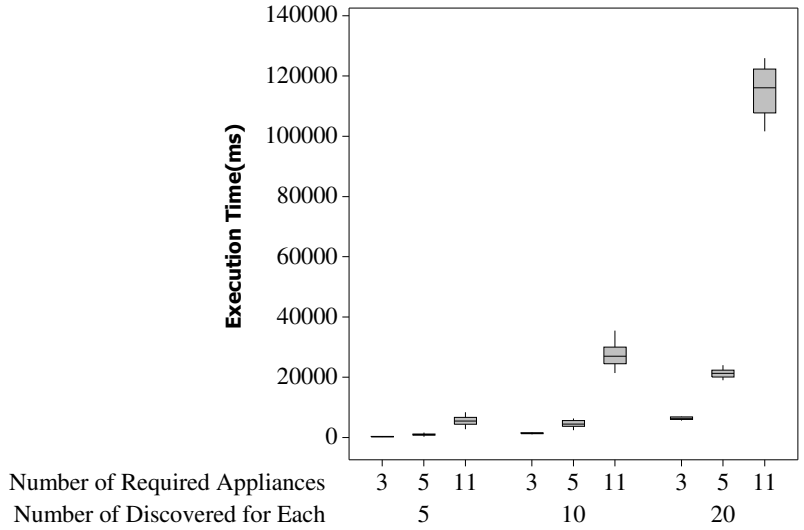


Figure 5.6: Execution-time analysis of the compatibility checking algorithm.

have to be aware of their preferences precisely, however our approach can help users with different levels of knowledge.

Execution-time Analysis of the Ontology-based Compatibility Checking Algorithm

As the reasoning is computationally intensive, we have measured the execution-time of the ontology-based compatibility checking algorithm. As shown in Figure 5.6, the impacts of the number of required appliances in the composition and the number of the discovered possible candidate appliances for each of the required appliances, on the execution time have been investigated. For each case the experiment has been repeated for 40 times with different request requirements. As illustrated in Figure 5.6, when the number of appliances in the composition and the number of discovered appliances increase the execution time grows. However, on average it never goes beyond two minutes, which makes the algorithm eligible to be used for online services.

5.5 Conclusions

In this chapter we have investigated the virtual appliance composition problem and identified the key challenges. To tackle these challenges, we presented an ontology-based

approach to describe appliances and their QoS properties, which helped us to build a composition with a set of compatible appliances. Our system helps non-expert users with limited or no knowledge on legal and image format compatibility issues to deploy their services faultlessly. In addition, we offered a technique to optimize the appliance composition based on user preferences such as deployment time, cost, and reliability. The approach exploits the benefits of evolutionary algorithms such as OMOPSO, NSGA-II, and SPEA-II for optimization and fuzzy logic to handle vague preferences of users. Results show that for the proposed case study, we can effectively help an unskilled user to identify the appliance compositions which are closest to their preferences. Furthermore, experimental results show OMOPSO outperforms NSGA-II and SPEA-II in terms of execution time, composition solution's diversity, and convergence. In addition results show the compatibility checking algorithm has acceptable execution time as number of discovered appliances candidate and number of appliance in composition grow.

Now that we have concluded or contributions for Cloud service discovery and composition optimization, we are going to to discuss SLA management component. Particularly, the next chapter focuses on our proposed negotiation techniques which after the discovery phase helps providers and users to reach consensus on a set of QoS and non-functional values in SLA.

Chapter 6

An Autonomous Negotiation Strategy for Cloud Computing Environments

Cloud Service Level Agreement (SLA) Negotiation is a process of joint decision making between Cloud clients and providers to resolve their conflicting objectives. As shown in previous chapters, Cloud service coordination operations such as discovery, selection, and composition are accomplished automatically. Therefore, negotiation between Cloud clients and providers can be a bottleneck if it is carried out manually. Our objective is to offer a state-of-the-art solution to automate the negotiation process in Cloud environments. The proposed negotiation strategy is based on a time-dependent tactic. For Cloud providers, the strategy uniquely considers utilization of resources when generates new offers and automatically adjusts the tactic's parameters to concede more on the price of less utilized resources. In addition, while the previous negotiation strategies in literature trust offered QoS values regardless of their dependability, our proposed strategy is capable of assessing reliability of offers received from Cloud providers. Furthermore, to find the right configuration of the time-dependent tactic in Cloud computing environments, we have investigated the effect of modifying its parameters such as initial offer value and deadline on negotiation outputs that include ratio of deals made, and social optimality. The proposed negotiation strategy is tested with different workloads and in diverse market conditions to show how the time-dependent tactic's settings can dynamically adapt to help Cloud providers increase their profits.

6.1 Introduction

CLOUD SLA Negotiation is a process of joint decision-making between Cloud users and providers to resolve their conflicting objectives. Cloud services have cost, availability, and other non-functional properties on one hand, and generate profits on the other hand. In Cloud environments, both clients and providers have cost-benefit

models for negotiation and decision making. Therefore, SLA negotiation automation requires mapping of the knowledge and objectives of policy makers to lower-level decision-making techniques. The first step towards the automation is finding, capturing, and modeling goals and objectives of parties involved in the negotiation. The second step is finding a proper strategy to use those goals in the low-level negotiation process. In this chapter, the negotiation target is a Cloud virtual machine service and the negotiated parameters are listed below.

- Hard Disk (functional requirement and fixed)
- CPU (functional requirement and fixed)
- RAM (functional requirement and fixed)
- Cost (QoS requirement and negotiable)
- Availability (QoS requirement and negotiable)
- Deadline (non-functional requirement and fixed)

As described in Table 6.1, users aim for the lowest price and the highest availability while Cloud providers would like to sell their services in the highest possible price and at the lowest QoS guarantee. Besides, users have time constraints when they are participating in the negotiation. That is because if they do not acquire the required resources by a particular time, they are not able to satisfy their end users expectations or reach their business objectives. Furthermore, Cloud providers have to consider the utilization of resources when they are offering prices during negotiation. It means that they are willing to concede on the prices of resources which are less utilized. Moreover, when service requestors are conceding in such multi-issue negotiation, the negotiation strategy has to prioritize the criteria which are more important to users. Both negotiation strategies for users and providers have the objectives of maximizing the chance of signing the contract during the negotiation.

Automated SLA negotiation has attracted a great deal of interest in the context of Service Oriented Architecture (SOA), Grid Computing, and recently Cloud Computing.

Table 6.1: Negotiation objectives.

Objectives	**Negotiation Parties**	
	Requestors	*Providers*
Cost	To be minimized	To be maximized
Availability	To be maximized	To be minimized
Other	- Acquiring the resource by deadline - Conceding on less important QoS - Verifying providers offer reliability	- Maximizing number of agreements - Maximizing Profit

Works in these contexts mainly focused on offering negotiation strategies which maximize the user's utility values and the number of signed contracts. However, they have not considered infrastructure management issues (such as resource utilization balancing) in the bargaining strategy. It means that Cloud providers are willing to concede on the price of resources which are less utilized, and that has to be reflected in the negotiation tactics. In addition, previous works have not considered reliability in the negotiation process. These works assume that service requestors would trust whatever QoS criteria values providers offer in the process of negotiation. Nevertheless, providers may offer a QoS value during the negotiation that was not achieved according to the monitored QoS data. To address such issues, this chapter proposes a negotiation strategy that acquires user's preferences and provider's resource utilization status and utilizes time-dependent tactic along with theory of statistics to maximize the Cloud providers profit while adhering to deadline constraints of users and verifying providers offer reliability. Through a series of experiments, we investigate the effect of modifying parameters of the time-dependent tactic such as initial offer value and deadline on negotiation outputs including social welfare and success of negotiation in Cloud environments. In addition, the offered negotiation strategy is tested for different workloads and in diverse market conditions to show how time-dependent tactic's settings can dynamically change to help

Cloud providers to increase their profits.

6.2 Motivations

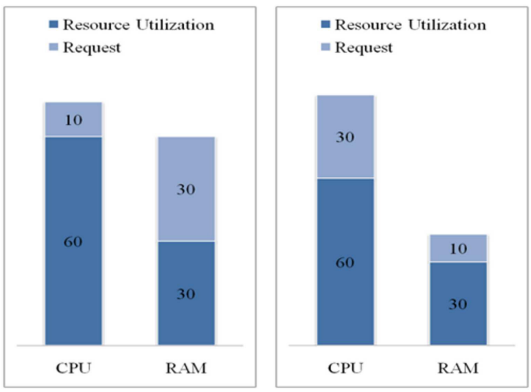
6.2.1 Offers Reliability

In the negotiation models presented in the literature [52, 53, 192], a method for determining the reliability of offers and counter offers is missing. Since in parallel negotiations in Multi-Cloud a party makes a decision based on the presented QoS values in SLA offers, there has to be a way to know how reliable the provider is in delivering the promised QoS values. The recorded data from monitoring services can be analyzed and converted to reliability information of offers. As explained in Chapter 3, the monitoring is based on the copy of the signed SLA, which is kept in the SLA repository. Third party monitoring results can be similar to what Cloud harmony [29] services report. To make inference from the observed data, we use the theory of statistics (Beta Density Function), which will be explained in Section 6.4.

6.2.2 Balancing Resource Utilization to Host More Virtual Machines

Previous works [178, 192] believe that, when service providers are concurrently negotiating with multiple users, using the same negotiation strategy for all incoming requests would maximizes providers' profits. However, we argue that in the Cloud context, providers are interested in a SLA negotiation strategy that balances the available resources, which helps them to host more virtual machines by avoiding resource fragmentation. To achieve that, providers have to concede more on the price of resources that are less utilized (or have more free capacity) and less on the price of resources that are more utilized. Consequently, providers offer more attractive prices in earlier stages of negotiation for clients whose requested virtual machines' allocation would balance resource utilization. For example, the request shown in Figure 6.1a will be offered more attractive deals in early stage of negotiation compared to request demonstrated in Figure 6.1b, because request shown in Figure 6.1b leads to a more balanced utilization of CPU

and RAM than request represented in Figure 6.1a.



(a) (b)

Figure 6.1: Requests and their effects on balancing resource utilization.

6.2.3 Investigating Behavior of the Time-dependent Function in the Cloud Computing Context

Time-dependent tactics [52] are proper candidates to be adopted for Cloud computing environments as users have deadlines for acquiring resources when they are participating in a negotiation. However, with the best of our knowledge, their applicability has not yet been evaluated for the Cloud context. Therefore, first we create a testbed that allows us to implement time-dependent functions for an environment consisting of multiple Clouds and brokers, and then we modify negotiation parameters such as deadline of requests, initial offer values, and type of tactic (polynomial or exponential) to study the behavior of time-dependent tactics for our problem.

6.3 Negotiation Framework

Figure 6.2 shows the major components in the negotiation framework and Figure 6.3 briefly describes the sequence of interactions between the Cloud and the requester negotiation service. A service requester specifies certain requirements such as hardware specifications like CPU, storage, and memory. In addition, the requester provides prefer-

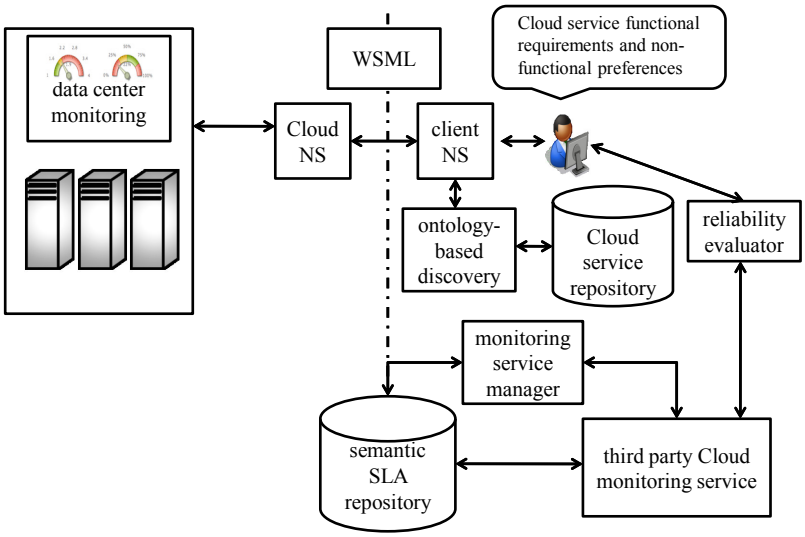


Figure 6.2: The proposed negotiation framework.

ences on the QoS criteria. Afterward, functional and QoS requirements are used as input for discovering suitable Cloud services. In the sequence, the client Negotiation Service (NS) starts negotiating with the discovered service providers' NSes on QoS criteria (price and availability) based on the requester's preferences. It is worth mentioning that client budget and deadline for acquiring resources are used by the client NS to make a decision on accepting or rejecting an offer. Client NS uses a time-dependent tactic that takes the client's preferences as an input and automatically generates an initial and then following offers. Once Cloud NS receives the offer, it uses request functional requirements, QoS requirements, and Cloud resources utilization from the monitoring system to generate counter offers. On the arrival of providers' offers, the client NS uses the reliability evaluator components and the time-dependent tactic to accept or reject the offer, or otherwise reply with a counteroffer.

If the negotiation is successful, the SLA contract will be signed by both parties and the obtained contract, which includes the set of expected QoS values (service level objectives), is kept in the SLA contract repository. Afterward, as it will be discussed in Chapter 7, the monitoring service manager discovers and selects the required monitoring services based on user preferences (cost and reliability) for each SLA contract in the repository. They constantly monitor the SLA and notify the reliability evaluator on any violation of

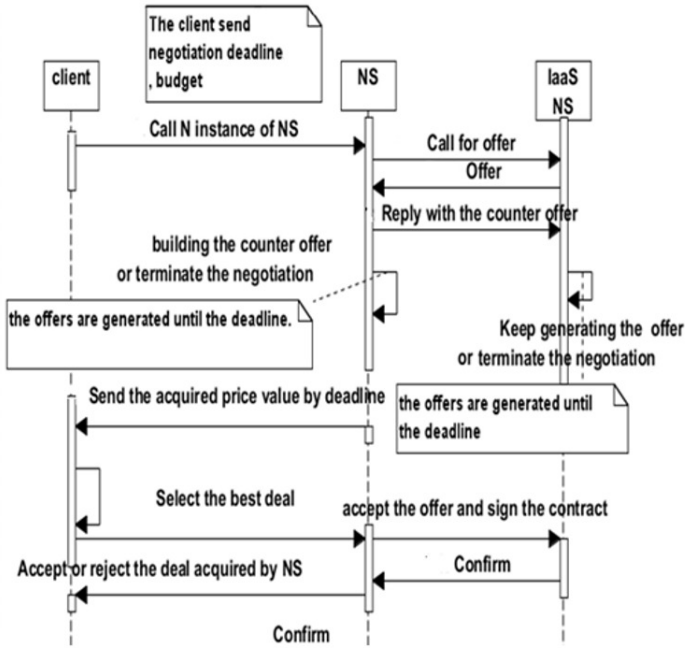


Figure 6.3: Negotiation sequence diagram.

service level objectives.

Moreover, as explained in Chapter 3, QoS ontology, Cloud services, their QoS, user requests, and SLA contracts in the proposed framework are described using Web Service Modeling Ontology (WSMO) in Web Service Modeling Language (WSML) [145]. Consequently, the Cloud service and monitoring service discovery component can perform semantic matching using defined terms in ontology. The ontology-based discovery can increase the level of flexibility and automation, allowing the two parties to use their own terminology as long as it is related to the commonly understood conceptual model.

6.4 Negotiation Strategy

Prior to explaining the negotiation strategies for each party, a brief description of the negotiation model and the applied negotiation tactics are given. In addition, descriptions of symbols used for expressing the negotiation process are listed in Table 6.2.

6.4.1 Negotiation Model

To create a negotiation model, we extended the model proposed by Raiffa [140] to incorporate the reliability of offers. In the model, the negotiation service receives requestor preferences on the importance (W_i) of n negotiation issues, min_i and max_i (reservation values), which are the acceptable range of values for issue i (VI_i), and negotiation deadline (t_{max}). The service then measures the utility of offers received from the other negotiation service based on the Equations (6.1) and (6.2).

$$UV = \sum_{i=1}^{n} W_i VI_i(y_i) \quad where \quad \sum_{i=1}^{n} W_i = 1 \tag{6.1}$$

$$VI_i(y_i) = \begin{cases} \frac{max_i - y_i}{max_i - min_i} & VI_i \text{ increases as } y_i \text{ decreases;} \\ \\ \frac{y_i - mini}{max_i - min_i} & VI_i \text{ decreases as } y_i \text{ decreases.} \end{cases} \tag{6.2}$$

Next, as shown in Equation (6.3), the offer is accepted if its utility value is greater than or equal to the utility of the counter offer that will be sent by the negotiation service. Otherwise, the negotiation service generates a new counter offer. In addition, if the timestamp of the received offer (t_{offer}) is greater than the deadline, the service terminates the negotiation.

$$Response = \begin{cases} terminate & \text{if} \quad t_{offer} > t_{max}; \\ accept & \text{if} \quad UV_{offer} > UV_{\text{counter offer}}; \\ new\ counter\ offer & \text{otherwise.} \end{cases} \tag{6.3}$$

6.4.2 Time-dependent Negotiation Tactic

As cited by Faratin et al. [52], time-dependent negotiation tactics are a class of functions that compute the value of a negotiation issue by considering the time factor. Therefore, they are particularly helpful when the NS receives a deadline (t_{max}) as an input, and has to concede faster as the deadline approaches. For this family of tactics, Equation (6.4)

is used by NS "a", which represents either a Cloud service requestor or a provider to generate a new counter offer for NS "b" for the negotiable issue i.

$$O^t_{a\to b}[i] = \begin{cases} min^a_i + \alpha^a_i(t)\left(max^a_i - min^a_i\right) & \text{if} \quad V^a_i \text{ is decreasing;} \\ min^a_i + \left(1 - \alpha^a_i(t)\right)\left(max^a_i - min^a_i\right) & \text{if} \quad V^a_i \text{ is increasing.} \end{cases} \tag{6.4}$$

Numerous functions have been defined for calculation of α^a_i(t) such as polynomial and exponential [52]. As it can be figured out from Equation (6.5), by changing the value of β (convexity degree) in both functions, the behavior of the negotiation tactic changes. If $\beta > 1$ then the tactic reaches its reservation's value at the early stage of negotiation. On the contrary, in the case of $\beta < 1$, it concedes to its reservation value only when the deadline is approaching. We adopt this family of the negotiation functions and change β dynamically to maximize the NS utility function.

$$\alpha^a_i(t) = \begin{cases} k^a_i + \left(1 - k^a_i\right)\left(\frac{min(t,t_{max})}{t_{max}}\right)^{1/\beta} & Polynomial; \\ \\ e^{\left(1-\frac{min(t,t_{max})}{t_{max}}\right)^{\beta} lnk^a_i} & Exponential. \end{cases} \tag{6.5}$$

6.4.3 Providers Strategy

For providers, the negotiation service input is composed of the Cloud resource utilization, minimum and maximum resource prices, and amounts of requested resources. The output of NS can be a SLA contract with a detailed description of a provider, a client, a service, and service level objectives. Providers are interested in an SLA negotiation strategy that balances the available resources and gives the attractive offers while keeps their utility functions high. To achieve that, providers have to concede more (by adjusting time-dependent function parameters) on the price of the resources that are less utilized (or have more free capacity) and less on the resources that are more utilized.

Table 6.2: Description of symbols.

Symbols	Description	Symbols	Description
a,b	negotiation parties	RP_{jt}	price of a resource j (e.g. RAM) at t
W_i	importance of issue i	αRP_j	time-dependent function for price of resource j
VI_i	offer value for issue i	IRP_j	initial price for resource j
UV	utility value of the offer	A_j	portion of resource j that is available
t_{offer}	offer timestamp	β_j	convexity degree for price of resource j
t_{max}	negotiation deadline	RUBO	resource utilization balancing oriented tactic
$O^t_{a\to b}$ [i]	offer sent from a to b for issue i	PO	priority oriented tactic
min^a_i	minimum acceptable value of issue i for a	γ_1	relative importance of $RUBO$
max^a_i	maximum acceptable value of issue i for a	γ_2	relative importance of PO
y_i	defines the range of values for an issue i	$RC_{offerVI_i}$	reliability constraint for issue i
α^a_i (t)	time-dependent function of issue i for a	$R_{offerVI_i}$	reliability of an offer's value of issue i
V^a_i	value offered for issue i by a	COD	consensus desirability
K^a_i	initial offer value for issue i by a	CF	conceding factor
β	convexity degree	ρ, τ	beta distribution parameters
P_t	price of virtual machine instance at t		

Unlike the majority of works that require time-dependent function parameters to be given explicitly, Zulkernine et al. [192] proposed a method to derive the parameters from the high level negotiation policy. Inspired by their work, we propose an approach to derive a price for the next offer based on the Cloud resource utilization. In comparison to their work, we argue that our approach is more suitable for parallel negotiation in Cloud context. The reason is we are discriminating regarding the pattern of concession when negotiating concurrently with multiple clients, while Zulkernine et al. apply the same pattern of concession for all clients. As shown in Equations (6.6), (6.7), (6.8), (6.9), and (6.10), we first define a total price of a VM instance as the sum of prices of its individual resources Equation ((6.6)). In the next step, for each resource, a time-dependent function (Equations (6.7) and (6.8)) is defined, and its parameter is adjusted (Equations (6.9) and (6.10)) based on its underutilized capacity compared to average resources' idle capacity ($\overline{A}$) for m type of resources.

$$P_t = \sum_{j=1}^{m} RP_{jt} \tag{6.6}$$

$$RP_{jt} = MinRP_j + \alpha RP_j \left(MaxRP_j - MinRP_j\right) \tag{6.7}$$

$$\alpha RP_j = IRP_j + \left(1 - IRP_j\right) \left(\frac{min\left(t, t_{max}\right)}{t_{max}}\right)^{1/\beta_j} \tag{6.8}$$

$$\overline{A} = \frac{\sum_{j=1}^{m} A_j}{m} \tag{6.9}$$

$$\beta_j = CF \times e^{C\left(A_j - \overline{A}\right)}$$

$$where\ CF = \omega \times COD\ and, \tag{6.10}$$

$$\omega \text{ and } C \text{ are constants and } c,\omega > 0.$$

As shown in Equation (6.10), when the idle capacity of a resource is greater than the average free capacity of resources in the data center, $A_j - \overline{A} > 0$ and $\beta_j > 1$, and therefore the negotiation strategy is conceding on the price of that resource. As a result, providers offer a more attractive price in earlier stages of negotiation for clients whose requested virtual machines' allocations would balance resource utilization. This increases the chance of reaching an agreement with the preferred request. However, in this tactic β is calculated based on the resources utilization and does not reflect the preferences of provider regard-

ing the importance of price and guaranteed availability criteria. The tactic based on [192] is adopted in Equation (6.11) to derive β from provider's preferences. Consequently, in order to satisfy all providers' objectives, the negotiation strategy has to be built as a mixture of those aforementioned tactics as shown in Equation (6.12).

$$\beta_j = e^{C\left(\frac{1}{n} - W_i\right)} \tag{6.11}$$

Where n is the number of criteria in the negotiation, C is a constant, and W_i is the importance of issue i and $\sum_{i=1}^{n} W_i = 1$.

$$O^t_{a\to b}[i] = \gamma_1 RUBO^t_{a\to b}[i] + \gamma_2 PO^t_{a\to b}[i] \tag{6.12}$$

Where $\gamma_1 + \gamma_2 = 1$, and $RUBO$, PO are offers' issue values generated by Resource Utilization Balancing Oriented tactic and Preference Oriented tactic respectively.

6.4.4 Cloud Client NS

The client NS receives user preferences on budget, deadline, and importance of QoS criteria and maps them to low level time-dependent parameters as described in the previous section and based on Equation (6.11) [192]. It means that β is defined in a way that the NS concedes less if the criteria are more important to the user and concedes more otherwise. In order to capture the importance of the criteria for the user, Analytic Hierarchy Process (AHP) [147] is adopted. Similar to the provider NS, the output can be a SLA contract with full specification of services, provider, client, and service level objectives. In this strategy, our contribution lies in the probabilistic assessment of offers reliability in negotiation.

The client NS assesses providers' offers in a probabilistic approach based on their past adherence level to SLA contracts. Therefore, as shown in Equation (6.13), the client NS only accepts offers when similar previous accepted offers have achieved a certain level of reliability (based on the monitored data) for each issue. For example, if in a multi-criteria negotiation a provider concedes in availability, and its reliability in such criteria is not high, users should not consider that an attractive offer.

$$\text{Offer acceptance conditions} = \begin{cases} UV_{offer} > UV_{\text{counter offer}} & and \\ \text{for each } VI_i \quad R_{\text{offer } VI_i} > RC_{\text{offer } VI_i} \end{cases} \tag{6.13}$$

We used the beta reputation system [89] to assess the reliability of offers. The reason is that Monitoring Outcome (MO) of a particular SLA contract can be modeled as in Equation (6.14), and therefore is a binary event. Consequently, the beta density function, which is shown in Equation (6.15), can be efficiently used to calculate posteriori probabilities of the event. As a result, the mean or expected value of the distribution can be represented by Equation (6.16).

$$MO = \{\text{SLA not violated,SLA violated}\} \tag{6.14}$$

$$\begin{gathered} f(x|\rho,\tau) = \frac{\Gamma(\rho+\tau)}{\Gamma(\rho)\Gamma(\tau)} x^{\rho-1}(1-x)^{\tau-1} \\ where \quad 0 \leq x \leq 1, \rho > 0, \tau > 0 \end{gathered} \tag{6.15}$$

$$\mu = E(x) = \rho/(\rho+\tau) \tag{6.16}$$

As mentioned in Section 6.3, in our architecture a component is responsible for monitoring SLA contracts. If we assume that the monitoring component has detected that SLA violation occurred v times for provider p (for a total number of n monitored SLAs). Considering that $\rho = n - v + 1$ and $\tau = v + 1$, the reliability is equal to probability expectation of SLA is not going to be violated and is calculated as shown in Equation (6.17). Once $R_{\text{offer } VI_i}$ is calculated for all issues, NS can only accept the offer if, for all the issues, $R_{\text{offer } VI_i}$ is greater than $RC_{\text{offer } VI_i}$.

$$R_{\text{offer } VI_i} = \frac{n - v + 1}{n + 2} \tag{6.17}$$

6.5 Performance Evaluation

For our performance evaluation, we extended CloudSim [20], a discrete event Cloud simulator, to build a new environment for testing negotiation techniques for Cloud computing environment. The extended package's main classes are illustrated by a Class diagram in Figure 6.4. The package enables data centers and brokers to have distinct negotiation strategies that describe the detailed sequence of actions in negotiation and the series of condition for accepting, replying, and rejecting offers. Each negotiation strategy consists of a single negotiation tactic (e.g. time-dependent) or a combination of them, which is responsible for generating a new offer in each round of negotiation.

The inter-arrival time of requests would not affect the performance of the negotiation strategies. Therefore, it is simply considered as a uniformly distributed value between 0.0 and 1.0 second. The simulation period is 1 hour and when the Demand to Supply Ratio in the experiment (DSR) is less than 1, the datacenters capacity is set to 100,000 Hosts. When DSR is greater than 1, Data center capacity is set to 10,000 Hosts. Each Host has 12 CPU cores, each 1.7 GHz; 12 GB RAM, and disk capacity of 4 TB. Reservation values for Cloud clients are set to \$5 per resource unit for the minimum price and \$15 for the maximum. For Clouds, however, reservation values are \$10 and \$20 per resource unit for minimum and maximum price, respectively. Using the request generator class, brokers send requests simultaneously to data centers. The request generator randomly generates requests with different deadlines and required instance types, which are formally represented by Equation (6.18).

$$Instance = N_{CU} + N_{RU} + N_{HDU} \tag{6.18}$$

Where N_{CU} is the number of CPU units requested, N_{RU} is the number of RAM units requested, and N_{HDU} is the number of hard disk units requested. Requests generated for experiments can be classified into two classes, namely balanced and unbalanced. In a balanced request, $N_{CU} = N_{RU} = N_{HDU}$, while in an unbalanced requests $N_{CU} \neq N_{RU} \neq N_{HDU}$. The balanced and unbalanced requests for our experiments have been designed

according to Amazon EC2 instances types[1]. In addition, they can be further categorized to requests with tight (from 20 to 40 rounds), moderate (from 40 to 50 rounds), and loose (from 50 to 100 rounds) deadline. The experiments are repeated 30 times.

The conducted experiments are mainly going to investigate:

- How modifying deadline of requests, initial offer values, and time-dependent function type affect the consensus rate and social welfare(Section 6.5.1 and 6.5.2);
- How successful the proposed strategy for Cloud NS is in accommodating more requests and thus increasing Cloud providers' profits, which is calculated based on the number of VMs allocated and the achieved price in the negotiation (Section 6.5.3);
- How to react to different market conditions (demand-to-supply ratio) to increase the profitability of the negotiation strategy (Section 6.5.4).

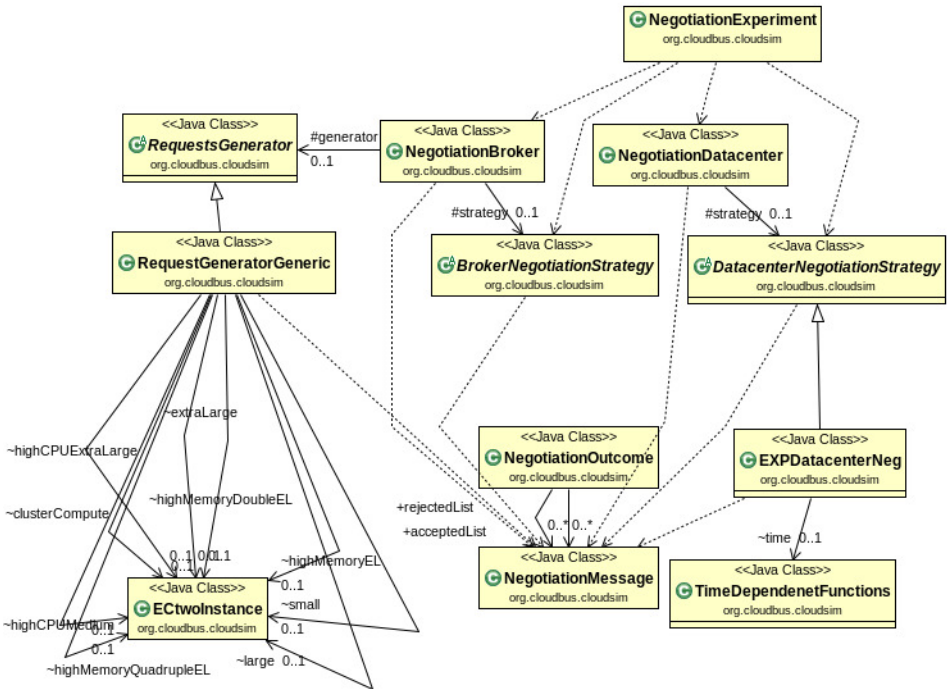
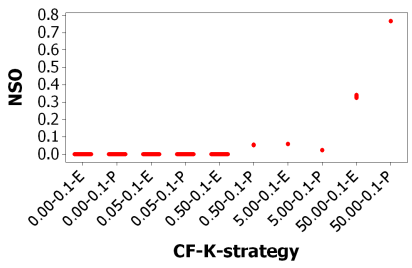
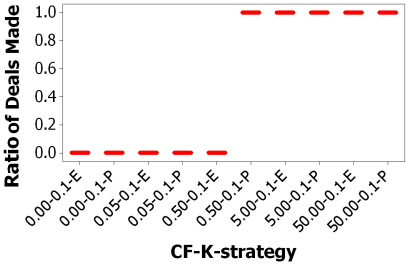


Figure 6.4: Class diagram of negotiation package for CloudSim.

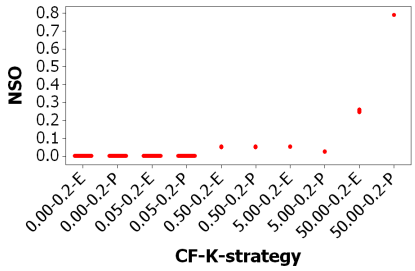
[1] Amazon EC2 Instance Types. `http://aws.amazon.com/ec2/instance-types/`



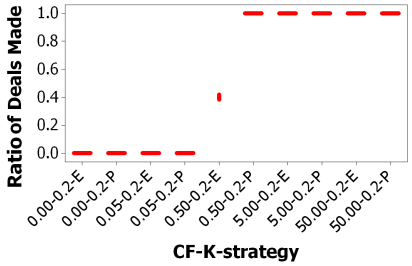
(a) NSO for initial offer=0.1



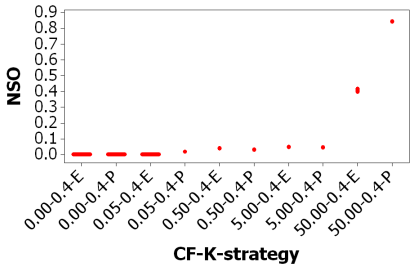
(b) deals made for initial offer=0.1



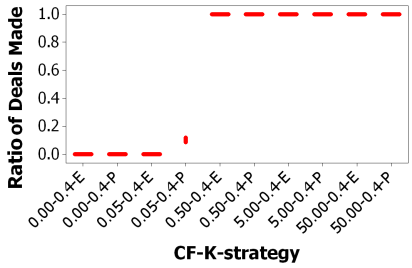
(c) NSO for initial offer=0.2



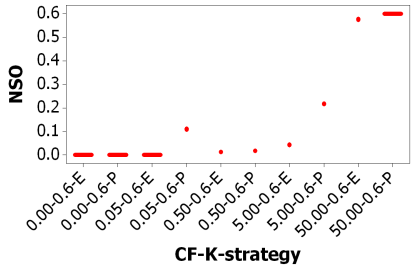
(d) deals made for initial offer=0.2



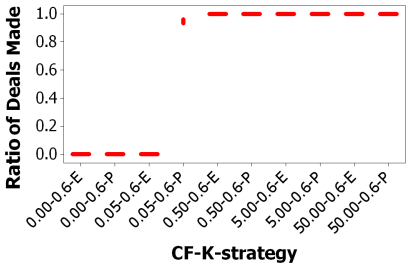
(e) NSO for initial offer=0.4



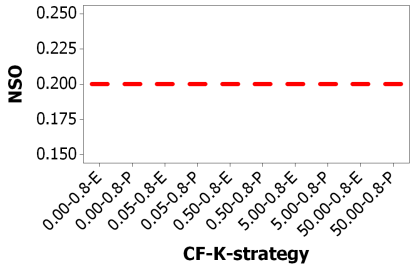
(f) deals made for initial offer=0.4



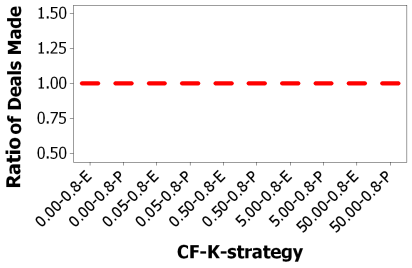
(g) NSO for initial offer=0.6



(h) deals made for initial offer=0.6

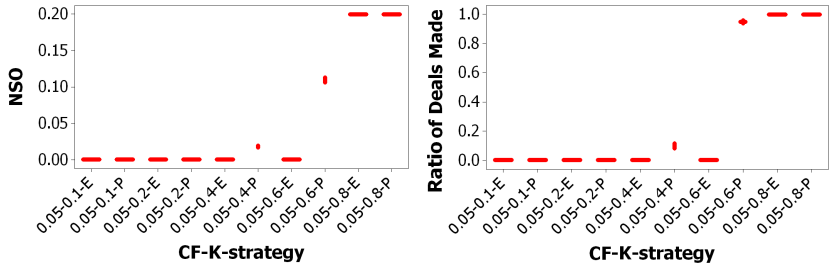


(i) NSO for initial offer=0.8



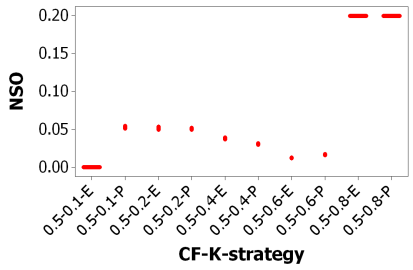
(j) deals made for initial offer=0.8

Figure 6.5: Impact of initial offer on NSO and negotiation success rate.

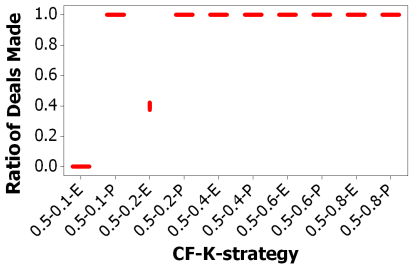


(a) NSO for CF=0.05

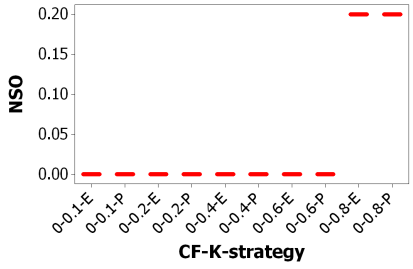
(b) deals made for CF=0.05



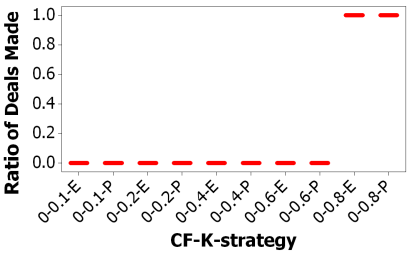
(c) NSO for CF=0.5



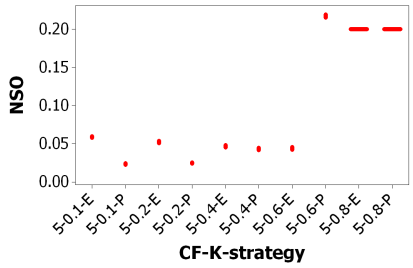
(d) deals made for CF=0.5



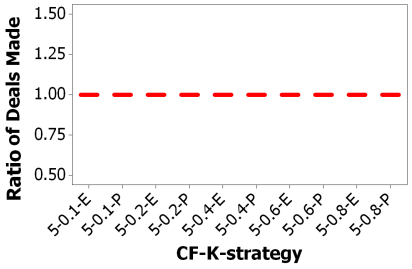
(e) NSO for CF=0



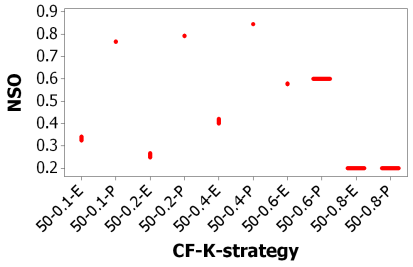
(f) deals made for CF=0



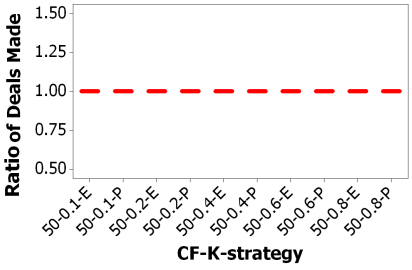
(g) NSO for CF=5



(h) deals made for CF=5



(i) NSO for CF=50



(j) deals made for CF=50

Figure 6.6: Impact of CF on NSO and negotiation success rate.

6.5.1 Effect of Strategies and Negotiation Parameters on Negotiation Outcome

The designed negotiation scenario consists of one broker and one data center, with negotiation parameters (CF and K) equally set for both parties. As shown in Equation (6.19), we use Normalized Social Optimality (NSO) to test the social welfare of negotiation strategies and parameters. The closer the values of NSO are to 0 the higher the social welfares of the strategy. Not surprisingly, when lower values are given to Consensus Factor and initial offer, the ratio of successful negotiation decreases. In contrast, higher values for CF and initial offer increase the chance of reaching an agreement. However, when they are set to extreme values (as shown by Figure 6.5i when the K factor reaches 0.8), the offers received from a data center are accepted in a first round of negotiation and there is no time for a broker to concede. Therefore, the broker has comparatively higher utility value in this case and NSO increases dramatically.

In addition, as illustrated in Figure 6.6, when the polynomial function is used, the chance of reaching an agreement even if the initial offer and CF is set to lower values increases. Nevertheless, as depicted in Figure 6.6i in a majority of the cases, when CF is set to the highest value (50), the exponential function reaches lower NSO, and thus higher social welfare. Moreover, adoption of CF of .5 and 5 as shown in Figures 6.6g and 6.6c and K of .4 and .2 as depicted in Figures 6.5c and 6.5e comparatively result in a higher social welfare. This means that if the objective of the negotiation is to achieve higher social welfare, initial offers should set at maximum below the half of the overall concession that one party is going to make, and then it should not concede either very quickly or too slowly.

$$NSO = \left| \frac{UV_{BR}}{UV_{DC} + UV_{BR}} - \frac{UV_{DC}}{UV_{DC} + UV_{BR}} \right| \tag{6.19}$$

6.5.2 Impact of Change in Deadline on the Ratio of Deals Made

One broker and one data center participate in this negotiation case, and the request generator builds negotiation messages based on a given deadline type probability. For the majority of cases illustrated in the Figure 6.7, when the probability of tight deadline in-

creases, the ratio of agreements decreases until no or a few deals are achieved. However, when CF and K leastwise set to 0.5 and 0.1 for polynomial function and 5 and 0.2 for exponential function, the deadline has no impact on the number of deals made, and both strategies reach 100 % of consensus rate. This shows the dominance of the polynomial function in reaching higher number of deals when the deadline is tight.

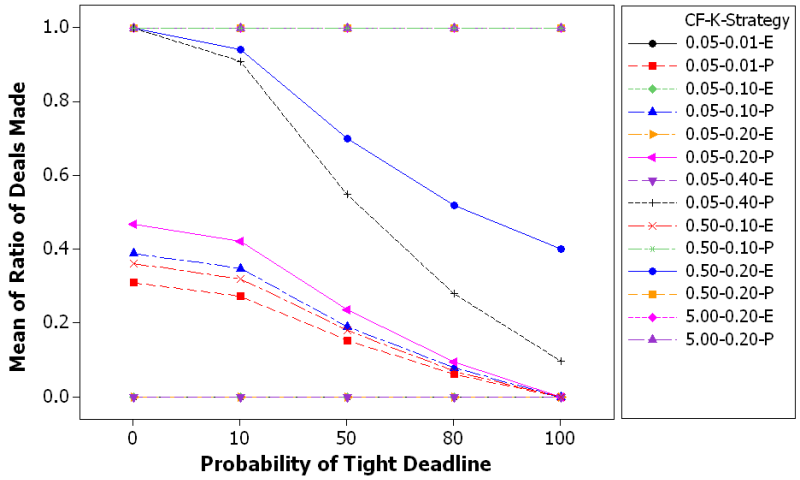


Figure 6.7: Impact of deadline on the success rate of negotiation. By “0.05-0.01-E", we mean that CF, K, and time-dependent functions are set to 0.05, 0.01, and exponential, respectively.

6.5.3 Performance of the Proposed Negotiation Strategy

To demonstrate the efficiency of our negotiation strategy, this experiment was designed with four brokers and a data center. All parties adopted the aforementioned polynomial function. The data center first adopted a pure time-dependent function and concurrently negotiated with four brokers, then we repeated the experiment with the same configuration but this time we replaced the strategy with our negotiation strategy. As Figure 6.8 shows, when the percentage of unbalanced requests increases, the revenue difference between strategies offered in purely time-dependent works [52, 192] and our work grows. The results show that even for the cases where only a small percentage of incoming requests are unbalanced (20 percent), data centers can still increase their profits by almost 10 percent on average. In addition, if the chance of a request to be unbalanced is 50

percent, then the profit growth increases to 20 percent on average. And finally, for the case that Percentage of Unbalanced Requests (PUR) is set to 100, our strategy can dominate previous work's strategy by nearly 27 percent. Therefore, results demonstrates that our negotiation strategy successfully acts as an admission control system that attracts the most profitable requests. In addition, the proposed strategy not only increases the revenue of the data center, but also, as demonstrated in Figure 6.9, increases the combined utility of the whole system. This means the that strategy increases data center profit as well as the whole system's (including brokers) utility.

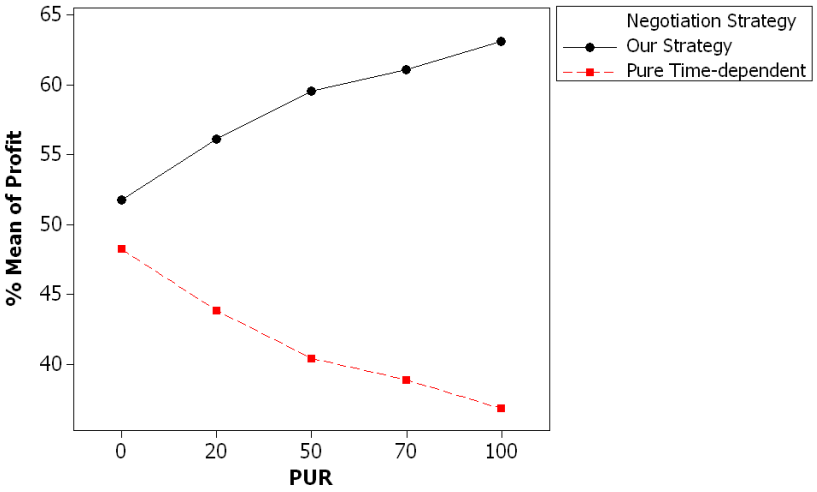


Figure 6.8: Impact of request type on the performance of the strategy. Workloads are built with different Percentage of Unbalanced Requests(PUR).

6.5.4 Effect of Demand to Supply Ratio and Consensus Desirability on Datacenters Revenue.

In order to demonstrate how our proposed strategy can increase its competency when there is another data center in the negotiation scenario, this experiment is designed with four brokers and two data centers. The polynomial function is adopted for the brokers and one of the data centers and our strategy for the second data center. We investigated the performance of the proposed strategy under different market conditions by varying Demand to Supply Ratio.

Demand to Supply Ratio (DSR) is a single numerical measure of the gap in supply and

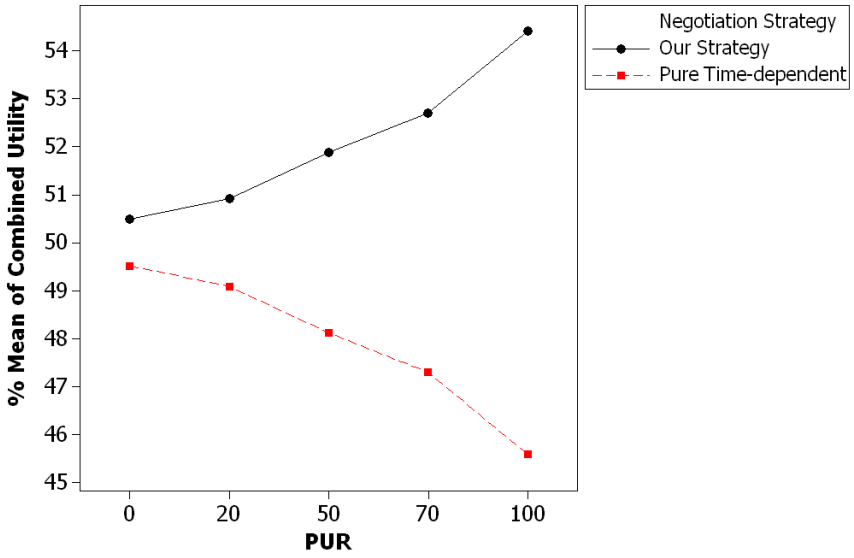


Figure 6.9: Impact of request type on the combined utility of the strategy.

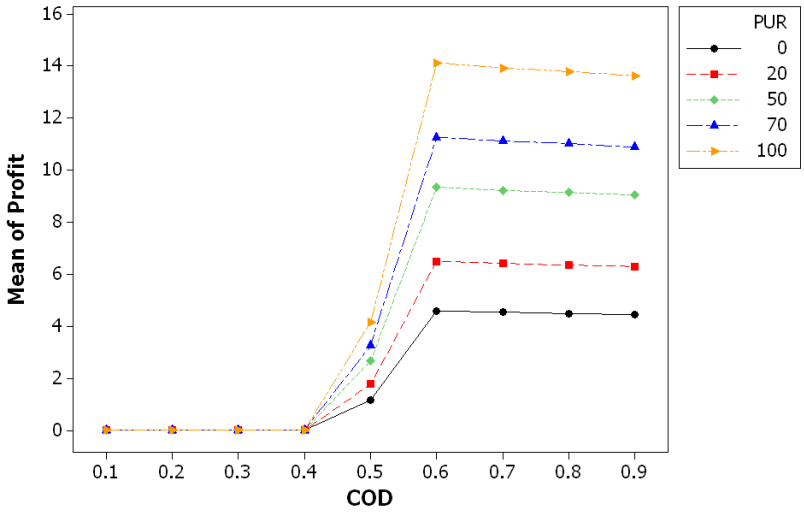


Figure 6.10: Impact of Consensus Desirability (COD) on the data centre profit when DSR is less than one.

demand for resources in the market. Fairly precise estimation of DSR can be calculated by a methodology offered by Macias et al. [115]. When DSR is less than one, competition among providers increases, and they try to win a larger share of markets by attracting as many VM requests regardless the request specification. Therefore, conceding faster by rising Consensus Desirability (COD) increases the chance of attracting more requests, and hence improving the revenue. The experiment shows that (Figure 6.10), when DSR

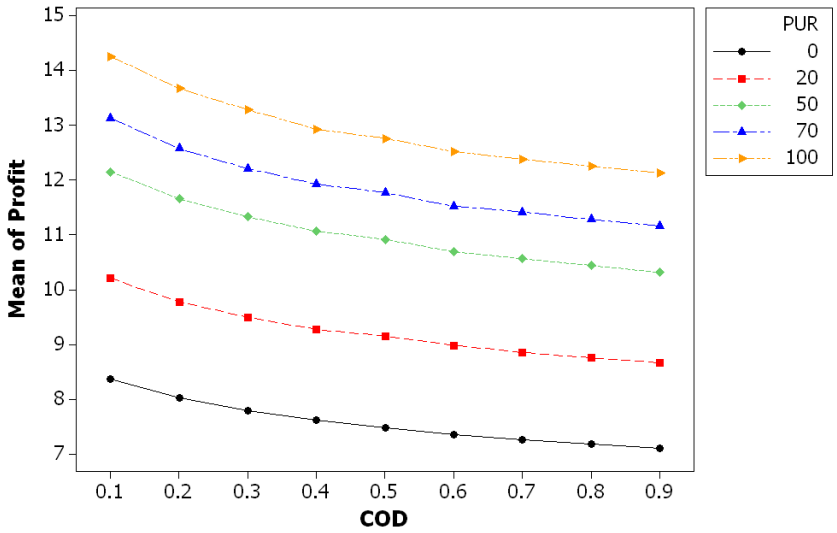


Figure 6.11: Impact of Consensus Desirability (COD) on the data centre profit when DSR is greater than one.

is less than one, data centers have higher revenue if COD is set to higher value. However, after a certain point ($COD = 0.6$), increasing COD would result in no gain, but a slight loss in revenue. In contrast, as illustrated in Figure 6.11, when DSR is greater than one, data centers with lower COD earn higher revenue.

Furthermore, the gap between data center revenue increases when DSR is low, because a failure in reaching an agreement means that those providers have permanently lost a chance of increasing their data center's utilization to other providers. However, when DSR is high, even if providers do not win an agreement at the beginning, their chances increase as other providers' utilization increases and there is no room for new requests.

6.6 Conclusions

In this chapter, we proposed a time-dependent negotiation strategy that is capable of assessing the reliability of offers to increase the dependability of our strategy and fill the gap between decision making and bargaining. To select an appropriate configuration for different negotiation objectives (e.g. maximizing social welfare or number of deals made), we investigated consequence of modification of parameters such as deadline, ini-

tial offer, and type of time-dependent tactic (polynomial or exponential). Although many of the works in the literature apply the same pattern of concession for all clients when negotiating in parallel, we argued that discriminating regarding the pattern of concession helps Cloud providers to accommodate more requests and thus increase their profit. Our approach was tested against purely time-dependent approaches, and it showed its dominance in generating more profit for providers. Furthermore, we showed how providers could dynamically and based on market condition increase or decrease the Consensus Desirability to raise their revenue.

In the next chapter, the monitoring service manager component will be discussed. We will show how the component discovers and selects the required monitoring services based on user preferences (cost and reliability) for each SLA contract in the repository. In addition, the next chapter reveals how we eliminate the impact of the SLA failure cascading on the false positive rate of the monitoring system.

Chapter 7

A Dependency-aware Approach for SLA Management

The major motivations to adopt Cloud services include no upfront investment on infrastructure and transferring responsibility of maintenance, backups, and license management to Cloud Providers. However, one of the key challenges that hold businesses from adopting Cloud computing services is security and performance of Clouds. That is because by migrating to the Cloud, they move some of their information and services out of their direct control. Therefore, their main concern is how well the Cloud providers keep their information (security) and deliver their services (performance). To cope with this challenge, several service level agreement management systems have been proposed. However, monitoring service deployment as a major responsibility of those systems has not been deeply investigated yet. Therefore, this chapter shows how monitoring services have to be described, deployed (discovered and ranked), and then how they have to be executed to enforce accurate penalties by eliminating service level agreement failure cascading effects on violation detection.

7.1 Introduction

MAJOR motivations to adopt Cloud services include reasonable price as they are offered in economy of scale, and transferring responsibility of maintenance, backups, and license management to Cloud service providers. However, one of the key challenges that holds back businesses from adopting Cloud computing services, even if they found it cost effective is that, by migrating to Cloud, they move some of their information and services out of their direct control [88]. The main concern is how confidentially the Cloud providers keep their information (security) and with which quality they deliver their services (performance). To cope with this challenge, service level agreement

(SLA) has been introduced. The SLA contract, which is signed by both parties, includes Quality of Service (QoS) requirements and penalties in case the QoS requirements are not met by providers. Nonetheless, relying only on SLA is not sufficient to ensure Cloud reliability. For example, if a business has a critical Web application deployed on the Cloud and it fails, thousands of dollars might be lost. However, according to most SLA contracts, Cloud providers only give a penalty as much as a portion of the deployment fee. As a result, the responsibility cannot be transferred to the Cloud service providers by SLA. Instead, efficient runtime monitoring services have to be deployed to validate the SLA and enforce penalties. In addition, as noted by Theilmann et al., realization of the vision of dynamic service coordination requires the whole process of monitoring to be automated. Therefore, automating monitoring service deployment is the main objective of this chapter.

We consider a Cloud service chain, which includes services such as virtual unit (infrastructure as a service), virtual appliance, and software as a service. In such an environment where various providers are involved in satisfying user requirements, we face a set of difficulties in SLA monitoring. Firstly, existence of different SLA offers, counter offers, and contract templates makes it difficult to discover necessary monitoring services that have required capabilities to monitor service level objectives in SLAs. Therefore, creating a standard model for describing SLAs in different layers of the Cloud has been considered [102] as a major challenge in this area of research. In this research, this issue has been addressed by a form of semantic SLA, which brings a common language and understanding to all parties involved in service provisioning in the Cloud.

Next, in the Multi-Cloud environment, where services from IaaS , PaaS, IaaS are provisioned and integrated, there are dependencies between performances of services. It means that if one of the lower-layer services (infrastructure layer) is not functioning properly, it can affect performance of higher-layer services. While SLA dependency has been considered by several works [102], no practical approach has been presented to model the dependencies among services. Consequently, this chapter shows how dependency knowledge can be modeled using semantic technology, and how that knowledge can be used in discovery of monitoring services and SLA failure detection. In summary, this

chapter investigates:

- A Semantic- based SLA that can be understood by all parties including providers, requestors, and monitoring services. The Semantic SLA contract is defined in a way, which can be used as a goal for discovery of necessary monitoring services.
- SLA dependency modeling using Web Service Modeling Ontology (WSMO) to build a knowledgebase that can be exploited to eliminate effects of SLA failure cascading on violation detection.
- An algorithms for discovery and ranking of monitoring services that considers user preferences on reliability, budget, and legal constraints.

7.2 Cloud Service and Monitoring Layers

As Emeakaroha et al. [49] and Theilmann et al. [160] cited, Cloud services can be classified in different layers (as depicted in Figure 7.1), which will lead to hierarchical structure of SLA contracts between different supply chain partners. In our scenario, three layers, namely infrastructure, application, and business layers, are considered. They are defined as follows:

1. **Infrastructure layer services:** In this layer Cloud service providers offer virtual units, which are resources that have been virtualized and can be a virtual com-

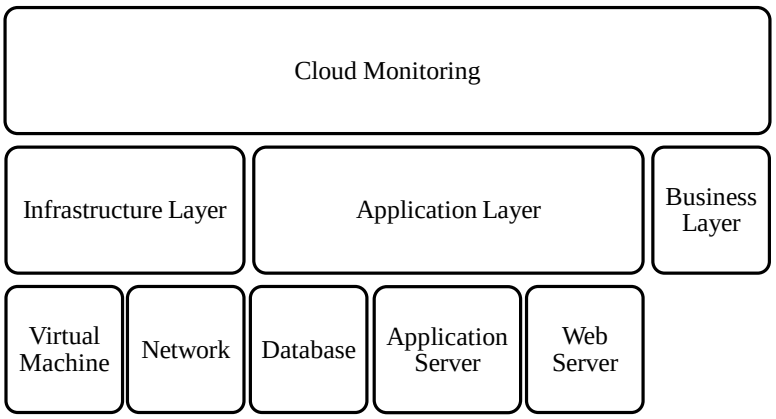


Figure 7.1: Cloud monitoring service layers.

puter, database system, or even a virtual cluster. Monitoring criteria for infrastructure as a service provider typically includes network uptime and server uptime. Other QoS metrics include network (such as bandwidth availability), system performance, support response time, server deployment latency, CPU utilization, and RAM monitoring.

2. **Application layer services:** The role of appliance providers is providing a ready-to-run software package with predictable behavior. Virtual appliances are shown to provide a better service deployment mechanism [158]. Therefore, they are adopted as a major Cloud component functioning in application layer [60].

3. **Business layer service:** This layer answers end-users business requirements by providing services implemented on top of appliance and infrastructure layers. As shown by Comuzzi et al. [34], service level objectives (SLO) in this layer can be modeled by a set of business rules. End-user satisfaction (for example in terms of response time) [88] is another important metric that can be monitored in this layer.

In this chapter, we consider Cloud services, that are located in different layers, with performance dependencies. It means that if one of the lower-layer (infrastructure layer) services in a chain is not functioning properly, it can affect the higher-layer service's performance.

7.3 Motivating Scenario

Amir, the IT administrator of an e-Business website, is required to deploy necessary monitoring services. The e-business services are implemented on top of a LAMP appliance from VMware and hosted on Amazon EC2. In addition, the e-Business application requires access to SalesForce.com as illustrated in Figure 7.2.

In one hand, he has advertised the monitoring services in a repository, and on the other hand he has signed SLAs, which have to be monitored. Therefore, the required monitoring services should be deployed based on the information given in SLA contracts and on Cloud user and provider preferences (regarding cost and reliability of the

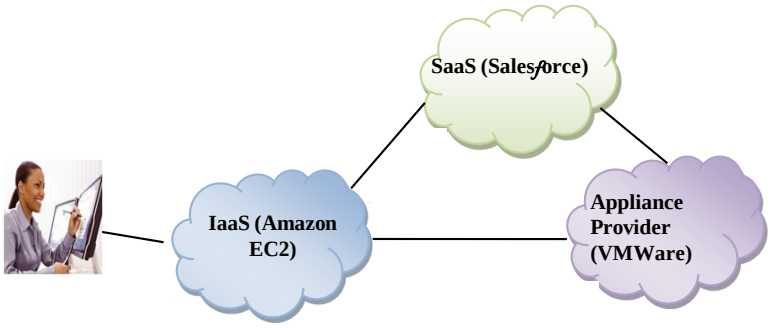


Figure 7.2: Cloud monitoring service layers.

monitoring services). Because his traditional IT monitoring tools simply cannot monitor varied components, he has decided to use third party monitoring services. He prefers to deploy monitoring services with the highest reliability and the least price. As he intends to automate the whole process of monitoring deployment and execution, he faces the following challenges:

1. The first problem is how to discover the required monitoring services using SLA contracts when not all parties (Monitoring Service providers and Cloud service providers) are using the same terminology for describing capabilities of monitoring services and QoS criteria, which has to be monitored in SLA.

2. The second challenge is how he can rank the discovered monitoring services to increase reliability and save on cost. If we assume he can overcome the first two problems, still the following challenge exists:

3. Imagine the monitoring service that is responsible for monitoring Salesforce services, reports a violation of SLA, hence Amir has reported the violation to Salesforce. Salesforce investigated the violation report and replied back stating that their service during that time was fully functional. Therefore, Amir has checked the monitoring service's logs and has found that the root cause of the SLA failure was Amazon EC2 throughput. Now he faces a new challenge:

   - How could he model services performance inter-dependencies in his system?
   - How could he eliminate SLA failure cascading effects on violation detection?

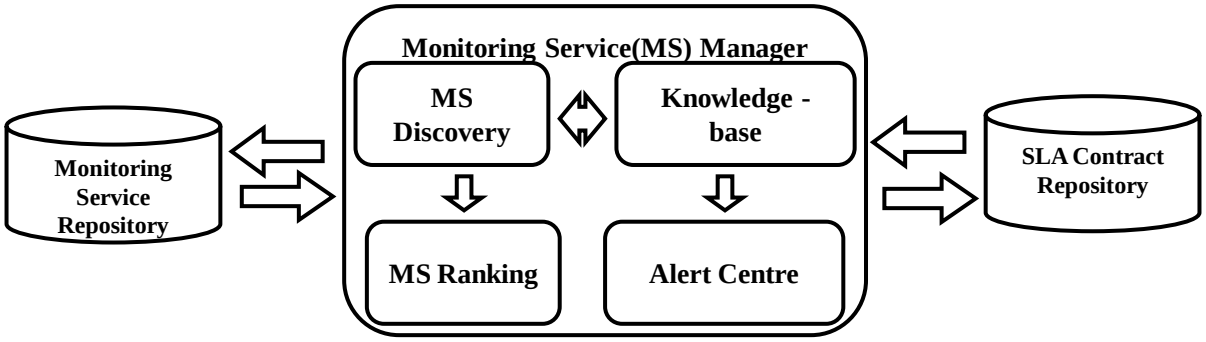


Figure 7.3: Monitoring Service Management architecture.

7.4 Monitoring Architecture

We propose a monitoring service manager as shown in Figure 7.3 to tackle the challenges listed in Section 7.3. The monitoring service manager discovers and selects the required monitoring services based on user preferences (cost and reliability) for each SLA contract in the repository. Moreover, the Alert Center listens to all the incoming alerts from the monitoring services and uses the dependency knowledge to filter alerts caused as a result of SLA failure cascading.

7.4.1 Service Level Agreement Contract Repository

As explained in Chapter 3, in a heterogeneous environment such as the Cloud, it is difficult to enforce syntax and semantics of services, their quality of services, and consequently SLA contracts. Therefore, applying symmetric attribute-based on matching between SLA contracts and monitoring services is impossible. In order to tackle the problems, the first step is creating a QoS ontology for Cloud services that provides common understanding for QoS among all parties involved in service provisioning. For this purpose (as it is shown in Figure 7.4), we extended WSML to support description of Cloud service QoS. Figure 7.4 shows how QoS such as availability, throughput, and their measurement unit ontology can be described using WSML. In the next step, the Cloud QoS ontology is applied to model SLA contracts semantically as illustrated in Figure 7.5. Consequently, monitoring service discovery can perform semantic matching using defined terms in the ontology. The ontology-based discovery increases the level of flexibility and

automation, allowing the two parties to use their own terminology as long as it is related to the commonly understood conceptual model. All the SLAs are stored in the SLA repository to be used as a goal for discovery of required monitoring services. For example, Figures 7.5 and 7.6 show that SLA needs a monitoring service that can monitor the bandwidth, storage, and availability of a Cloud service and, if the set of described obligations is not met by the Cloud service, can raise an alert.

```
namespace { _"http://www.example.org/ontologies/Cloud_Service_QoS#",
wsml _"http://www.wsmo.org/wsml/wsml_syntax#" }

ontology Cloud_Service_QoS

concept  Stockholder
concept  Provider  subConceptOf  Stockholder
concept  Customer  subConceptOf  Stockholder
concept  AplianceProvider  subConceptOf  Provider
concept  VirtualUnitProvider  subConceptOf  Provider
concept  MonitoringService  subConceptOf  Provider

concept  MeasurementUnit
concept  Incoming_Bandwidth  subConceptOf  MeasurementUnit
concept  Incoming_Bandwidth  subConceptOf  MeasurementUnit
concept  Outgoing_Bandwidth  subConceptOf  MeasurementUnit
concept  MemoryUnit  subConceptOf  MeasurementUnit
concept  TimeUnit  subConceptOf  MeasurementUnit
instance  GB  memberOf  MemoryUnit
value  hasValue  "GB"
instance  Kbps  memberOf  Incoming_Bandwidth
value  hasValue  "Kbps"
instance  Kbps  memberOf  Outgoing_Bandwidth
value  hasValue  "Kbps"
instance  Sec  memberOf  TimeUnit
value  hasValue  "Sec"

concept  ApplianceQoS
concept  ApplianceReliability  subConceptOf  ApplianceQoS
concept  ApplianceAvailability  subConceptOf  ApplianceQoS
concept  ResponseTime  subConceptOf  ApplianceQoS
concept  Throughput  subConceptOf  ApplianceQoS

concept  VirtualUnitQoS
concept  StartTime  subConceptOf  VirtualUnitQoS
concept  VirtualUnitAvailability  subConceptOf  VirtualUnitQoS
concept  Latancy  subConceptOf  VirtualUnitQoS
concept  VirtualUnitReliability  subConceptOf  VirtualUnitQoS
concept  RAM  subConceptOf  VirtualUnitQoS
concept  CPU  subConceptOf  VirtualUnitQoS
concept  Storage  subConceptOf  VirtualUnitQoS
concept  Bandwidth  subConceptOf  VirtualUnitQoS
```

Figure 7.4: QoS ontology.

```
namespace { up _"http://www.wsmo.org/ontologies/nfp/upperOnto.wsml#",
dc _"http://purl.org/dc/elements/1.1 #",
reg _"http://www.example.org/contract_goal.wsml#",
oblg  _"http://www.wsmo.org/ontologies/nfp/obligations.wsml#",
CSQ _"http://www.example.org/ontologies/Cloud_Service_QoS.wsml#",
ConOnt  _"http://www.example.org /contract_ontology.wsml#",
}
/************************
* Contract Goal
************************/
goal _"http://www.example.org /contract_goal"

 nonFunctionalProperties
 up#nfp hasValue up#hasObligations
 up#hasObligation hasValue req#DefinitionObligations
 up#hasViolation hasValue oblg#DefinitionViolations
 endNonFunctionalProperties

 importsOntology  {CSQ#Cloud_Service_QoS,
 CloudServicesDependencies}

 capability _"http://www.example.org /contract_goal#capl"
effect
definedBy
 [  [
[   [  POBox hasValue  "P.o.Box  218",
City hasValue "Yorktown, NY 10598, USA"] memberOf Contact and
[   Location hasValue "USA",
TrustedServices hasValue {"CloudHarmony", "CloudStatus", "Nimsoft",
"Monitis", "Hypertic"}
] memberOf  ProviderMonitoringConstraint
] memberOf  ServiceProvider [hasName hasValue "Amazon"] and
[  [  Street hasValue "30Saw Mill River RD",
 City hasValue "Hawthorne", NY 10532, USA"] memberOf Contact and
[   Location hasValue "USA",
TrustedServices hasValue {"Nimsoft", "Monitis", "Hypertic"}
]   memberOf  CustomerMonitoringConstraint
]  memberOf  ServiceCustomer [hasName hasValue "Customer"]
] memberOf Parties and
[ hasInstance hasValue "small",
hasPrice hasValue "$100"
]  memberOf  ServiceDefinition [ hasName hasValue "EC2"]
 ]  memberOf  SLA [ hasName hasValue "ServiceAgreement1"]
```

Figure 7.5: SLA contract goal.

7.4.2 Monitoring Service Repository

There are traditional server monitoring services that can be used to likewise monitor cloud services. There are also vendor specific monitoring services like EC2 CloudWatch[1] . In addition, there are third party independent Cloud monitoring services like Cloud-

[1] Amazon CloudWatch. `http://aws.amazon.com/cloudwatch/`

```
// Contract Ontology
ontology _"http://www.example.org/contract_ontology#"
 instance Cloud_Service_QoS memberOf CSQ#Cloud_Service_QoS
CSQ#MemoryUnit hasValue ?MemoryUnit
CSQ#Incoming_Bandwidth hasValue ?Incoming_Bandwidth
CSQ#Outgoing_Bandwidth hasValue ?Outgoing_Bandwidth
CSQ#TimeUnit hasValue ?TimeUnit
...
 axiom DefinitionObligations
definedBy
hasObligation (CSQ#hasBandwidthËŔ ? BandwidthSupportValue):-
CSQ#hasBandwidth [value hasValue ?BandwidthSupportValue] and
?BandwidthSupportValue  <60  and ?BandwidthSupportValue > 50.
hasStorage hasValue "100".
hasVirtualUnitAvailability hasValue "98".
```

Figure 7.6: SLA contract ontology.

status[2] that advertise their capabilities of Cloud service monitoring in the repository. For the monitoring service to be discovered, they have to expose their monitoring capabilities in a uniform way as discussed earlier. A monitoring service's description contains its monitoring capability [34] and its non-functional properties. We have modeled monitoring services using WSMO in our system. As it is shown in Figure 7.7, the capability of monitoring services identifies the Cloud services and their QoS criteria, which can be monitored. In addition, non-functional properties of monitoring services such as price, reliability, and location have been considered in the modeling. The non-functional properties of monitoring services are used by the ranking component to rank them based on user preferences.

For example, as illustrated in Figure 7.7, the monitoring service is capable of monitoring Cloud virtual unit services that are located in the USA from specific providers (EC2, GoGrid, and RackSpace). Moreover it can only monitor the CPU, bandwidth, memory, availability, and throughput of the services. It is worth mentioning that, semantically described monitoring services use QoS ontology depicted by Figure 7.4. This Ontology provides shared understanding of QoS criteria for all parties in SLA management phases.

[2] Hyperic. http://www.hyperic.com/products/Cloud-monitoring.html

7.4.3 Monitoring Service Manager

The Monitoring Service Manager (MSM) component is responsible for discovery, selection, and coordination of third party monitoring services. The main objective of this component is automating the process of deploying necessary third party monitoring services by applying semantic service technology. It consists of several sub-components such as discovery, ranking, knowledgebase, and alert center, which will be explained in the following sections.

```
//Monitoring Service
webService _"http://www.example.org/VirtualUnitMonitoringService/Nimsoft"

importsOntology {CSQ _"http://www.example.org/ontologies/Cloud_Service_QoS.wsmL#",
VU _"http://www.example.org/ontologies/Virtual_Unit.wsml#"}

capability _"http://www.example.org/VirtualUnitMonitoringService/Nimsoft#Capl"
nonFunctionalProperties
VU#Price hasValue ?price
VU#Location hasValue ?loc
CSQ#Reliability hasValue ?reliability
endNonFunctionalProperties
 sharedVariables {?band, ?CPU, ?memory, ?availability, ?throughput}
 effect
 definedBy
 [
 [  VU#Location hasValue "USA",
  VU#Providers hasValue {"Amazon", "GoGrid", "RackSpace"}
 ]  memberOf ProviderSupported and
 [  CSQ#Bandwidth hasValue ?band,
 CSQ#CPU hasValue ?CPU,
 CSQ#Memory hasValue ?memory,
 CSQ#Availability hasValue ?availability,
 CSQ#Throughput hasValue ?throughput ,
 ]  memberOf QoS_Criteria
  ] memeberOf MonitoringService [ hasname hasValue "Nimsoft"]
```

Figure 7.7: Ontology-based monitoring service modeling.

Knowledgebase: Cloud Service Dependency Knowledge Modeling

In our work, service dependency [176] is defined as a relationship between one service and one or multiple services where if a service A is dependent on service B, performance of a service A in one or multiple QoS criteria can be affected by service B. The QoS dependency of a Cloud service can be defined formally as follows:

let $CS = \{S_1, S_2, \cdots, S_n\}$ be a set of Cloud services. Then service dependency can be

defined as:

$$Cloudservicedependency : \{S_d, S, QC_d, QC, EQC\} \tag{7.1}$$

Where: S_d is a service such that its performance in the QoS criteria QC_d depends on the performance of QoS criteria of QC of service S. In addition, EQC is the expected QoS criteria performance from S. It means the service S_d is able to perform according to its SLA for the QoS criteria QC_d if S can perform as at least equal to EQC for QoS criteria QC.

It is worth mentioning that dependency rules in the knowledgebase are transitive. For example if S_1 performance depends on S_2, and S_2 performance depends on S_3, therefore knowledgebase deduces that S_1 performance also depends on S_3. Knowledge regarding dependencies is particularly significant for:

- Discovery of monitoring services, which will be discussed in the next section, where we explain how does the discovery algorithm differs from what was presented in Chapter 3.
- Prevention of false positives caused by SLA failure cascading, which will be discussed later in this chapter.

We present an approach for modeling dependencies as a foundation for adopting multi-layer Cloud services. Description Logics (DLs) is a known as formalism for representing knowledge. Consequently, DL languages such as WSML-DL are considered the core of the knowledge representation system. Once dependency of services are represented as knowledge, it can be queried by WSML-reasoner for the reasoning purpose. An example of the modeling is shown in Figure 7.8, where the appliance service availability is dependent on virtual unit availability.

Monitoring Service Discovery

As described in privious sections, we have defined all the monitoring services specifications as well as SLA contracts based on WSMO to enable semantic-based discovery of appropriate monitoring services. Figure 7.8 demonstrates the monitoring services modeling based on WSMO. Moreover, an example of how to model SLA contract goals based

```
namespace { _"http://www.example.org/ontologies/CloudServicesDependencies#",
dc _"http://purl.org/dc/elements/1.1#",
up _"http://www.wsmo.org/ontologies/nfp/upperOnto.wsml#",
CSQ _"http://www.example.org/ontologies/Cloud_Service_QoS.wsml#",
WSML _"http://www.wsmo.org/wsml/wsml-syntax#" }
/*************************
Dependencies ontology
*************************/

ontology CloudServicesDependencies

 nonFunctionalProperties
 up#nfp hasValue up#hasDependency
 endNonFunctionalProperties

 importsOntology {CSQ#Cloud_Service_QoS}

//   Dependencies Rules
axiom DefinitionDependencies
 nonFunctionalProperties
 dc#description hasValue
"Note: the appliance service availability is dependent on virtual unit
availability"
 endNonFunctionalProperties
 definedBy
?hasDependency memberOf CSO#ApllianceAvailability
Implies
?hasDependency [up#hasDependency hasValue CSQ#VirtualUnitAvailability].
```

Figure 7.8: Ontology-based dependency modeling.

on WSMO is shown in Figures 7.5 and 7.6. In order to realize the discovery component of our proposed architecture, a monitoring service discovery algorithm (Algorithm 5) has been presented. This algorithm applies five matching operations namely Exact, PlugIn, Subsumption, Intersection, and NonMatch in response to the requested SLA contract goal. The definitions of these matching types can be found in Chapter 1.

As shown in Figure 7.3, the SLA contract repository is responsible to keep all the information regarding the existing SLA contract goals (g_{exist}) together with their matching monitoring services and matching types. In the monitoring service discovery algorithm, if each of the existing SLA contract goals can be matched with the requested SLA contract goal (g_{user}), then the discovery mechanism will not continue anymore, and the already existing goal that consists of the matched monitoring services and the matching type will be sent as the result. In case of no match between g_{user} and g_{exist}, the discovery mechanism is applied on all available monitoring services (w) located in the monitoring service repository $(\mathcal{W})$ to look for the matched monitoring services. In that case, the

Algorithm 5: Monitoring Service Discovery

Input: g_{user} is the SLA contract goal, $\mathcal{G}_{exist}$ is a set of existing SLA contract goals, $\mathcal{W}$ is the set of Cloud services

forall the $g \in \mathcal{G}_{exist}$ **do**
  if $g_{user} \equiv g$ **then**
    /* g is an existing SLA contract in WSMO format */
    return $(g_{user}, matchType(g))$
forall the $w \in \mathcal{W}$ **do**
  /* w is an existing monitoring service */
  if $w \in (\mathcal{T}_p(g_{user}) \sqcap \mathcal{T}_p(g_{user}))$ **then**
    /* $\mathcal{T}_p(g_{user})$ and$\mathcal{T}_p(g_{user})$ are trusted monitoring services of the Cloud and the user */
    $\Omega(g_{user}) = \mathcal{DC}(g_{user}) \sqcup O(g_{user})$ /* $\mathcal{DC}$ is set of QoS Criteria of a described service in SLA that other services performance dependent on */
    /* O is set of obligations in the SLA contract */
    if $\Omega(g_{user}) \equiv \mathcal{QOS}(w)$ ***and*** $\mathcal{N}(g_{user}) \equiv \mathcal{N}_w$ **then**
      return $(w, Exact)$
    else if $\Omega(g_{user}) \sqsubseteq \mathcal{QOS}(w)$ ***and*** $\mathcal{N}(g_{user}) \sqsubseteq \mathcal{N}_w$ **then**
      return $(w, PlugIn)$
    else if $\mathcal{QOS}(w) \sqsubseteq \Omega(g_{user})$ ***and*** $\mathcal{N}_\mathcal{W} \sqsubseteq \mathcal{N}(g_{user})$ **then**
      return $(w, Subsumption)$
    else if $\neg(\Omega(g_{user}) \sqcap \mathcal{QOS}(w) \sqsubseteq \perp)$ ***and*** $\neg(\mathcal{N}(g_{user}) \sqcap \mathcal{N}_w \sqsubseteq \perp)$ **then**
      return $(w, Intersection)$
return $(g_{user}, NonMatch)$

candidate monitoring service must be a member of trusted services, which are supported by both service providers and customers. If the aforementioned conditions are satisfied, the matchmaker process is executed to find out the matching type between monitoring

services and the g_{user}. In that case, two set of elements of g_{user} and (w) are compared with each other, as follows:

1. $\Omega(g_{user})$ with QoS criteria of the monitoring service in which $\Omega(g_{user})$ is a collection of obligations of the g_{user} and QoS criteria of Service_Definition (g_{user}) that performance of other services are dependent on. Here is precisely where we require dependency knowledge for discovery. When a monitoring service is discovered for an SLA contract of a Cloud service, the monitoring service not only has to be capable of monitoring all QoS criteria described in SLA contracts but also has to monitor QoS criteria of the Cloud service (which is defined in the contract) that other services are dependent on. This way, if ascendant services cause an SLA failure, the monitoring service has the record of that, and using the dependency knowledge box, the root cause of the problem can be detected.

2. Non-functional properties (reliability and cost) of the g_{user} and and (w) must be compared with each other.

Monitoring Service Ranking

The Non-functional properties for ranking of monitoring services are cost and reliability, which are defined as follows:

1. Reliability: one important comparison factor for monitoring services is reliability. For example, initially, we have evaluated two products for monitoring Cloud services. However, we quickly realized that these services triggered many false positives because they only do single-node outage verification and, occasionally, those monitoring nodes themselves experienced network issues that caused them to inadvertently trigger outages. Finally, we find a monitoring service that does triple-node outage verification before triggering an outage, which provides a much more reliable result. In order to measure the reliability of monitoring services, a formula is developed as shown in Equation (7.2).

2. Cost: monitoring service cost is a non-functional requirement of a user who wants

to deploy required monitoring services. In our problem, minimization of deployment cost is considered as the objective of users. Besides, there are different kinds of pricing for monitoring services in the market, for example EC2 offers Cloudwatch for free for a 5-min interval monitoring with any EC2 instance. Other services charge per server or per service. For example, Cloudkick charges $99/month for six servers/month. Panopta charges per service monitored at around $1.5/service/month.

In the selection phase, based on user QoS preferences, all discovered services are ranked and the top service in the ranked list is returned. As discussed in the previous section, reliability and cost are considered QoS criteria of monitoring services. We first offer an approach to measure reliability of monitoring services, and then show how monitoring services can be ranked by applying the analytical Hierarchy Process (AHP) approach [148] (more detailed description regarding AHP can be found in Chapter 2). There are several third-party Cloud monitoring services exist in the market, and this allows users some degree of flexibility in choosing the service that best suits their preferences. One very important factor to consider while comparing monitoring services is their reliability, which can be measured based on a monitoring service's false reports generated for each monitored QoS criteria as shown in Equation 7.2. The false positive is referring to the frequency with which the monitoring service reports violation of SLA in error. And the false negative is the frequency with which the monitoring service fails to detect an actual violation of SLA.

$$\begin{aligned} Reliability = \sum_{c=1}^{k} I_c \left(W_{nc} \times \frac{\text{total number of detection}}{\text{false violation detection}} \right. \\ \left. + W_{pc} \times \frac{\text{total number of detection} + \text{number of violation not detected}}{\text{number of violation not detected}} \right) \end{aligned} \tag{7.2}$$

$$Where \begin{cases} I_c \text{ is the relative importance of criterion } c. \\ W_{pc} \text{ is the relative importance of false positive rates compared with false negative rates} \\ W_{nc} \text{ is the relative importance of false negative rates compared with false positive rates} \end{cases}$$

Table 7.1: Major scale of pair-wise comparisons.

Scores	Response to the question
1	Equal importance or preference.
3	Moderate importance or preference of one over another.
5	Strong or essential importance or preference.
7	Very strong or demonstrated importance or preference.
9	Extreme importance or preference.

The reason for introducing weights for false positive and negative rates is to allow users giving higher weights to the rates and criteria that are more important to them. To capture weights in the equation and rank the monitoring services, AHP has been applied, which is one of the most applied methods in the multiple criteria decision making category. The AHP method was suggested by Satty in 1998 [148] and is based on pairwise comparison of criteria to determine their weights in a utility function. The major contribution of AHP is to convert subjective assessments of relative importance to numerical values or weights. The methodology of AHP is based on pairwise comparisons of the criterion by asking the question of "how important is criterion C_i compared with criterion C_j?" The answer (which could be one of those shown in Table 7.1) to the question determines weights for criteria. Figure 7.9 depicts the process of choosing the best monitoring service provider when two criteria (cost and reliability) are considered. After the pairwise comparison, as Figure 7.9 shows, the relative importance was assigned to each criterion. In the next step, similar questions have to be asked to evaluate the performance scores for monitoring services on the subjective criteria. And then, based on the results of this phase, alternatives can be ranked and selected. We have adopted an open decision maker software to our project, and result of the ranking for the discovered services is demonstrated in Figure 7.11.

Violation Reporting

This subcomponent is in charge of violation detection and reporting to enforce accurate penalties by elimination of SLA failure cascading effects on violation detection. In order

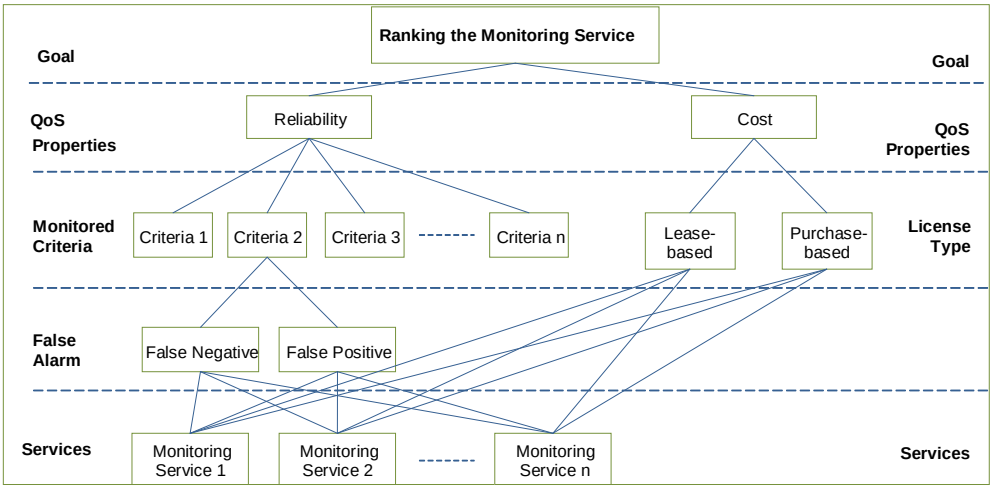


Figure 7.9: Applying AHP for ranking monitoring services.

Algorithm 6: False Alarm Detection

Input: VR

/* VR is a violation report */

1 $FalseAlarm = False$

2 $S_d \leftarrow VR_S$

3 $QC_d \leftarrow VR_Q C$

4 **forall the** $S_a \in DS$ **do**

/* DS is a set of all deployed services */

5 **if** $\exists(S_d, S_a, QC_d, QC_a, EQC)$ **then**

6 $AS \leftarrow S_a$ /* AS is a set which contains all the ascendant services */

7 **forall the** $S_a \in AS$ **do**

8 **if** QC_a *is worse than* EQC **then**

9 $FalseAlarm = True$

to do that, as shown in Algorithm 6, if it receives violation reports(VR) on performance of a service (S_d) on a QoS criteria (QC_d) from the monitoring service, it queries the knowledgebase to check whether the service performance for that criteria depends on other service's performance or not. If it is dependent, then it checks the performance of the

QoS criteria of the ascendant service; if it is not as equal or better than EQC, then it concludes that the reported violation is false, and instead, it reports the root cause of the problem and all the consequent effects.

7.5 Performance Evaluation

Experiments in this section were run on a system with Intel i3-350-M, 2.27-GHz processor and 4 GB of RAM. The main experiments are:

- a feasibility test for the a case study with one SLA contract and five monitoring services in the repository; and
- and Deployment time (discovery and ranking execution time) measurement for different number of monitoring services in the repository.

7.5.1 Monitoring Services Discovery for Case Study

For the case study, the SLA contract in Figures 7.5 and 7.6 is considered to be the goal, and a set of five monitoring services with different QoS parameters and capabilities were imported to the repository. The summary of the capabilities are shown in Table 7.2. Figure 7.10 illustrates the output of discovery algorithm. It shows, three monitoring services, namely monitis, hyperic, and nimsoft, could match (with match type of subsumption) the requirements. The other two monitoring services were not discovered because Cloudharmony does not have the capability to monitor the SLA contract, and Cloudwatch was not in the trusted list of both the Cloud and the user. Then, in the ranking phase, as it is shown in Figure 7.11, monitoring services are ranked using AHP, and nimsoft monitoring service scored higher when reliability criteria is more important to the user.

7.5.2 Deployment Time Measurement

In thus experiment, we increased the number of monitoring services in the repository to investigate the scalability of our approach in terms of execution time. For different

Table 7.2: Monitoring services in the repository for the case study.

Monitoring service	QoS monitoring capabilities	Service monitoring capabilities	Location
Monitis	CPU, memory, storage, load, processes, availability	EC2, GoGrid	USA
Hyperic	CPU, memory, storage, availability, bandwidth	EC2, GoGrid	USA
Nimsoft	CPU, memory, storage, availability, bandwidth	EC2, GoGrid	USA
Cloudharmony	Availability, throughput	EC2, GoGrid, RackSpace, Flexiscale	USA
Cloudwatch	CPU, memory, storage	EC2	USA

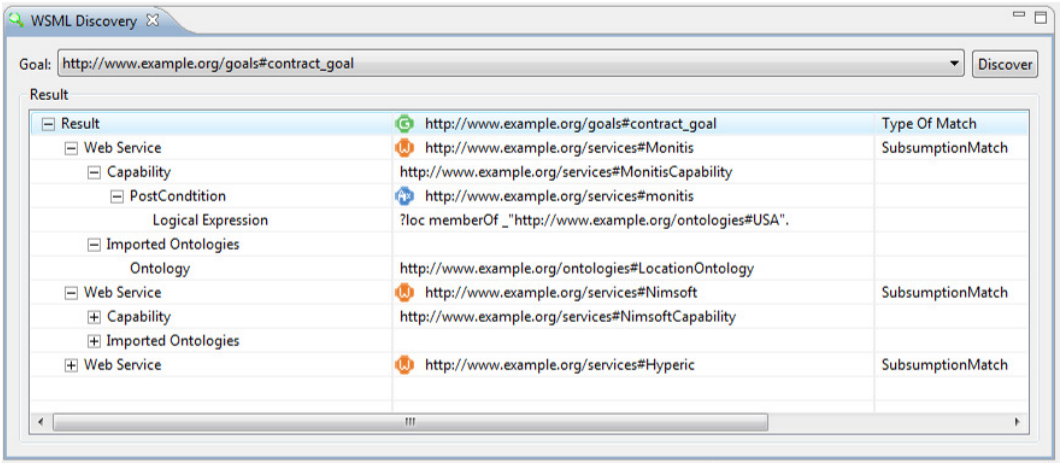


Figure 7.10: Monitoring service discovery algorithm validation for the case study.

numbers of monitoring services, we repeated the experiment 20 times. Each time the SLA contract QoS types were randomly altered. As it is illustrated in Figure 7.12, the mean deployment time, which is the sum of discovery and ranking time, has increased from less than 3 seconds when there are only 30 offers in the repository to almost 10 s when the number of offers increased to 120. Therefore, although our approach causes

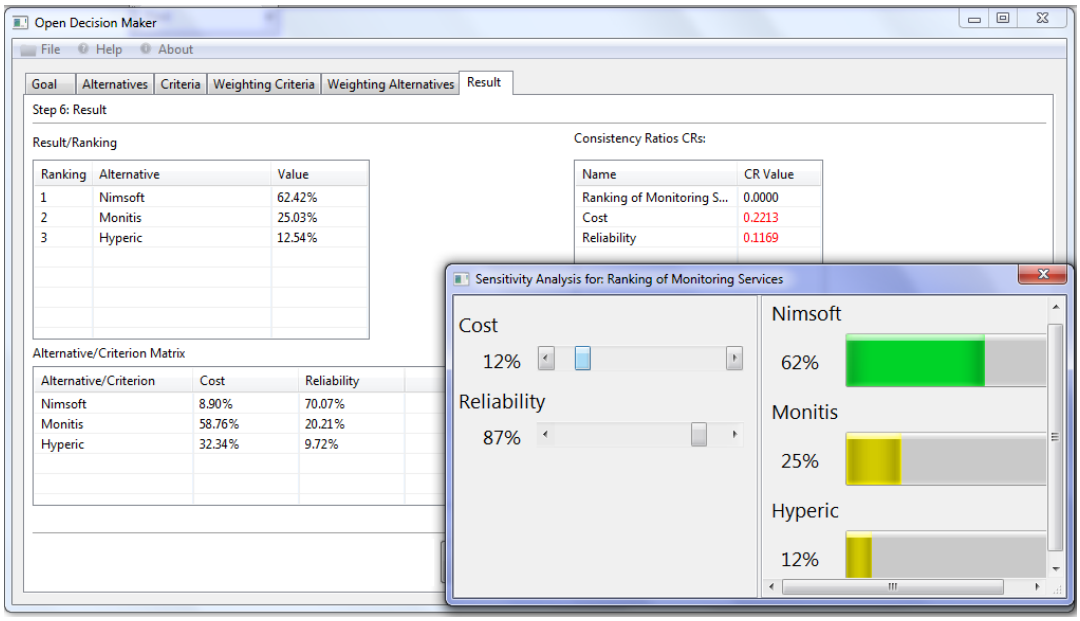


Figure 7.11: Ranking algorithm validation for the case study.

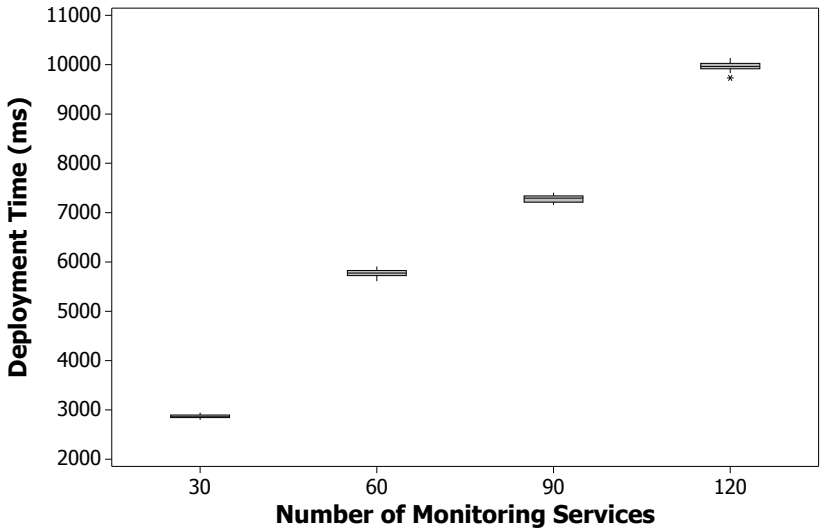


Figure 7.12: Execution time for monitoring service discovery and rankling.

semantic-based matching overhead, still it can discover and rank services in a reasonable time even for large number of services in the repository.

7.6 Conclusions

Automating the process of SLA management in Cloud needs an approach for deployment of required monitoring services. In heterogeneous environment such as Cloud, the discovery of monitoring services cannot be accomplished using syntax-based matching of advertised monitoring services in repository and signed SLAs. Consequently, this work proposed an ontology-based approach for modeling monitoring capabilities and SLA contracts to semantically match them and avoid low recall caused by lack of common QoS understanding. This chapter started with modeling of QoS ontology using WSMO and continued with exploiting the QoS ontology in SLA template and monitoring services description to build an inter-Cloud language for monitoring service deployment. In addition, the chapter discusses the effects of QoS dependencies among services on generation of false SLA violation report. It tackled the problem by deploying fittest monitoring services and filtering violation reports using dependency knowledge. We tested our approach on a case study and the results show the efficiency and effectiveness of the proposed work. In addition, the computational overhead of the semantic discovery and AHP for the repository has been measured for various numbers of monitoring services. The results show the approach is scalable and the deployment can be accomplished in a reasonable time.

Chapter 8
Conclusion and Future Directions

8.1 Discussion

IN order to minimize the risk of failure in a single Cloud computing environment, reduce the cost, and provide better QoS, it is important to exploit the benefit of migrating applications of Cloud users to multiple providers. Although Clouds adopted some common communication protocols such as HTTP and SOAP, the integration and interoperability of all services and finally service deployment and coordination in Multi-Cloud remain the biggest challenges. In this thesis, we addressed the problem of service coordination in Multi-Cloud, the process of making a service ready for use, which includes service discovery, SLA negotiation, service selection, and monitoring.

To tackle challenges involved in Multi-Cloud service coordination, Chapter 1 set objectives pursued by this thesis. In Chapter 2, we investigated in detail the major requirements for each phase of Cloud service coordination and commented on existing works in different contexts to identify gaps in this area of research. Our investigation revealed a lack of architecture that focuses on satisfying user requirements using virtual appliances and machines from various Cloud providers by automating the process of service deployment and coordination.

Therefore, in Chapter 3, an architecture was proposed that captures software, hardware and quality of services requirements. The architecture uses service discovery, SLA negotiation, and service selection techniques to map the user requirements to Cloud offerings (virtual appliances, virtual units or machines, and monitoring services). Furthermore, Chapter 3 explored Cloud service discovery in Multi-Cloud environments. In

a heterogeneous environment such as Multi-Cloud, it is difficult to enforce syntax and semantics of virtual machine descriptions and user QoS requirements among Clouds. Therefore, applying symmetric attribute-based matching between requirements and request is impossible. Therefore, we used ontology-based discovery to semantically match user's requirements to Cloud services. The discovery approach is validated in a case study which shows its effectiveness and applicability.

Currently, there is no integrated repository of semantic-based services for virtual appliances and units. The first step towards describing services and their QoS is to communicate with Clouds and the Cloud monitoring services through their APIs and gather the required meta-data for building the repository. Virtual appliances and units meta-data are defined in the form of Extensible Markup Language (XML). However, to get the advantages of Ontology-based discovery, they have to be described conceptually using Web Service Modeling Ontology (WSMO) ontologies and in the form of Web Service Modeling Language (WSML). The manual translation of Cloud appliance and virtual unit offerings' descriptions is not a feasible approach. Therefore, we offered a translation approach to semantically enrich Cloud offerings that minimizes human intervention. The performance of translation technique was measured for different repository sizes that proves its scalability.

In Chapter 4, we discussed that the problem of migrating multi-tier applications to Multi-Cloud requires a selection approach to satisfy user requirements such as minimizing deployment cost (including data transfer cost) and constraints such as latency between tiers. The selection problem was mapped to the multi-dimensional knapsack problem and tackled by two algorithms, namely Forward-Checking-based-backtracking (FCBB) and the genetic-based approach with necessary fitness and penalty functions. We evaluated the proposed approaches by a real case study using real data collected from 12 Cloud providers, which showed that they deliver near-optimal solutions. Next, we tested Cloud service selection approaches with different types of requests. We noticed that the messages size has a considerable impact on the performance of the algorithms. If the execution time is not the main concern of users, genetic-based selection in most cases achieves better value for the objective function. In contrast, if the massage size between

appliances is small, FCBB can be used to save on execution time while acquiring near-optimal solutions. Furthermore, based on the conducted experiments, we realized that network of applications with higher graph density and data transfer are less likely to be distributed across multiple providers (in contrast to requests with lower data transfer). However, for requests with tight latency requirements, services are still placed across multiple providers to save on deployment cost.

Moreover, as discussed in Chapter 1 we aimed at simplifying the process of service coordination for no-experts users. In Chapter 5, this has been achieved by means of two approaches. The first approach, exploits the benefits of evolutionary algorithms (NSGA-II and SPEA-II) for optimization and fuzzy logic to handle vague preferences of users. Results show that for the proposed case study, we can effectively help unskilled users to identify the service compositions which are closest to their preferences by set of high-level linguistic rules. The second approach uses WSMO to model expert knowledge regarding the compatibility of virtual appliance and infrastructure services. Once the knowledgebase is built, we proposed a compatibility checking algorithm to automatically filter all incompatible services. Results show that the compatibility checking algorithm has acceptable execution time as number of discovered candidates and number of services in composition grow.

Furthermore, Chapter 6 provided the building blocks to evaluate mechanisms for SLA negotiation in Cloud. SLA negotiation strategies can be considered as a means of pricing Cloud services based on the resource availability and utilization. We have restricted our focus specifically to the time-dependent strategy that can handle deadline constraint of users. We then extended it to be capable of assessing the reliability of offers. In addition, a novel approach proposed to dynamically update time-dependent function parameter based on resource utilization to maximize the profit of Cloud providers. In contrast to the majority of the works in the literature that apply the same pattern of concession for all clients when negotiating in parallel, we argued that discriminating regarding the pattern of concession helps Cloud providers to accommodate more requests and thus increase their profit. This way providers offer a more attractive price in earlier stages of negotiation for clients whose requested virtual machines' allocations would

balance resource utilization. This increases the chance of reaching an agreement with the preferred request. Our approach was tested against purely time-dependent strategies, and experimental results shows dominance of our strategy in generating more profit for providers.

Signing SLA contract with Cloud providers is not sufficient to ensure Cloud reliability. For example, if a business has critical web application deployed on Cloud and it fails, it may result in thousands of dollars lost for the business. Nevertheless, according to most SLA contracts, they only give a penalty as much as a portion of the deployment fee. Therefore, the responsibility cannot be transferred to the Cloud service providers by SLA. Instead, an efficient runtime monitoring services have to be deployed to validate the SLA and enforce the penalties. Chapter 7 discusses the effects of QoS dependencies among services on generation of false SLA violation report. Therefore, it tackles the problem by discovering and selecting proper monitoring services and filtering violation reports using dependency knowledge. For the selection of the required monitoring service, a mixture of semantic-based discovery and Analytic Hierarchy Process (AHP) [67] was used. The approach was proven efficient, once tested on a case study, and scalable, once evaluated for service repositories with larger size.

8.2 Future Directions

As shown in Figure 8.1, this thesis reveals six future directions to enhance service coordination in Cloud computing. They are described in detail in the following sections:

8.2.1 Multi-Cloud Auto-scaling and Failure Recovery Optimization

In Chapter 4 we have tackled the service selection problem for migrating multi-tier application to Cloud which is accomplished prior to deployment. However, for the post-deployment phase it would be relevant to study cross-cloud automatic scaling optimization. Automatic scaling is a feature that scales the capacity of the Cloud services up or down automatically based on a set of user-defined conditions. Automatic scaling ensures

Figure 8.1: Future directions.

that the number of Cloud services increases during surge in demand to adhere to service level objectives, and decreases during fall in demand to save cost. Conditions for automatic scaling can be set based on metrics such as average CPU utilization. It is interesting to investigate the problem of automatic scaling in Multi-Cloud environments where users also define a set of constraints and objectives. For example, users can set budget, deployment time, throughput, and latency as constraints and cost minimization as an objective. It is important to investigate the scaling optimization algorithm which select a Cloud service dynamically and on-demand that not only minimizes the cost but also satisfies the other constraints. Another promising research topic is discovering and selecting services for back up, and a deployment configuration which facilitates the recovery in a fast and cost-optimal manner.

8.2.2 Quality of Service Modeling of Cloud Offerings and Dynamic Context-aware Service Selection

Cloud services have specific characteristics and QoS dimensions which have to be identified. After that, it is important to investigate approaches for measuring those QoS criteria.

More specifically, defining criteria which are able to model energy and carbon emission efficiency [65], reliability [86], and trust [135] of a Cloud service are increasingly attractive to users . For example, methods to evaluate reliability and trust of providers from user feedbacks and monitoring services together can be further studied. This consists of collecting required raw data from trusted sources and statistically analyzing and aggregating them.

In addition, user experience is another important benchmark for Cloud service providers. Recently, crowdsourcing is used to create collective knowledge to assess QoS of Cloud services [26]. Discovering the crowd which is able to evaluate a service efficiently, delegating evaluation tasks to crowds, and calculating the accuracy of the aggregated assessments are relevant problems to investigate.

In Cloud computing dynamic modeling of service status and user demand and preferences is an essential task. Researchers can investigate service selection approaches which dynamically maps user requests to services based on the context attributes (users' device characteristics and locations) and status of Cloud services.

8.2.3 Service Selection Where Multiple Spot Markets Exist

Amazon introduced a new type of instances called Spot Instance in 2009 [1] that has not been considered in our service selection problem. The price of each spot instance VM type depends on its demand within each data center. Spot instances offer a low price but less reliable infrastructure service for the public Cloud users. In the Amazon EC2 spot market users bid for infrastructure services and are in charge of balancing between reliability and cost. If we assume there are more than one spot markets with different price histories[87], then it is important to: 1) identify a set of decision making criteria for market selection and 2) select the bid that minimizes the total cost of deployment while meeting Service Level Agreement (SLA) constraints.

[1] Amazon EC2 Spot Instances. `http://aws.amazon.com/ec2/spot-instances/`

8.2.4 Considering Heterogeneous Negotiation Strategies in Multi-Cloud Environments

In this thesis, a time-dependent strategy has been considered for all parties in SLA negotiation. However, it is interesting to study effects of considering heterogeneous strategies in the SLA negotiation problem. Therefore, distinct strategies (e.g. time-dependent, tit-for-tat, etc.) can be assigned to negotiation parties and achieved profits and ratios of deals made can be computed and compared. In addition, as SLA Negotiation can be considered as a means of pricing Cloud services, it is important to compare the pricing methodology and history of all strategies under diverse market conditions (demand to supply ratios).

8.2.5 Combining Fuzzy Similarity and Time-dependent Negotiation Strategies

Applying a combination of fuzzy similarity (see Chapter 2) and time-dependent strategies for the presented negotiation problem in Chapter 6 is another area to be explored. This is particularly applicable for the multi-criteria negotiation scenarios. To apply fuzzy similarity, hill climbing technique can be used to search for feasible contracts that have the highest similarity with the opponent's offer and have the same utility value as the previously offered contract. This strategy can be compared with our strategy to find out whether it can improve the combined utility and social optimality of negotiation parties.

8.2.6 Measuring the Impact of Applying Dependency-aware SLA Violations Detection Approach on Decreasing the Number of False Positives

It is important to investigate the impact of using the proposed dependency-aware SLA filtering (see Chapter 7) on decreasing the number of false positives. More specifically, a simulation environment can be built with infrastructure, and application services with SLA dependencies that is equipped with our proposed SLA violation filtering engine. After that an approach has to be identified to model the failure of Cloud services in the system. By failure model we mean distribution of a service availability and unavailability intervals. For this purpose the Failure Trace Archive (FTA) [99] can be used. Next, in the

experiments failure logs can be collected and analyzed with and without our filtering engine to test its effectiveness on decreasing the number of false positives.

Bibliography

[1] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, "Web services agreement specification (ws-agreement)," in *Global Grid Forum*, vol. 2, 2004.

[2] P. Anedda, M. Gaggero, and S. Manca, "A general service oriented approach for managing virtual machines allocation," in *Proceedings of the 2009 ACM symposium on Applied Computing*. ACM, 2009, pp. 2154–2161.

[3] J. Anselmi, D. Ardagna, and P. Cremonesi, "A qos-based selection approach of autonomic grid services," in *Proceedings of the 2007 workshop on Service-oriented computing performance: aspects, issues, and approaches, High Performance Distributed Computing (HPDC)*, vol. 25, no. 25. Citeseer, 2007, pp. 1–8.

[4] M. Arlitt and T. Jin, "A workload characterization study of the 1998 world cup web site," *IEEE Network*, vol. 14, no. 3, pp. 30–37, 2000.

[5] K. Atanassov and G. Gargov, "Interval valued intuitionistic fuzzy sets," *Fuzzy sets and systems*, vol. 31, no. 3, pp. 343–349, 1989.

[6] A. Averbakh, D. Krause, and D. Skoutas, "Exploiting user feedback to improve semantic web service discovery," *in Proceedings of the 8th International Semantic Web Conference (ISWC)*, pp. 33–48, 2009.

[7] R. Axelrod, *The evolution of cooperation: revised edition*. Basic books, 2006.

[8] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.

[9] S. Bajaj, D. Box, D. Chappell, F. Curbera, G. Daniels, P. Hallam-Baker, M. Hondo, C. Kaler, D. Langworthy, A. Nadalin *et al.*, "Web services policy 1.2-framework (ws-policy)," *W3C Member Submission*, vol. 25, p. 12, 2006.

[10] R. Barth and C. Smith, "International regulation of encryption: technology will drive policy," *Borders in Cyberspace: Information Policy and Global Information Infrastructure*, pp. 283–299, 1999.

[11] A. Bayucan, R. L. Henderson, C. Lesiak, B. Mann, T. Proett, and D. Tweten, "Portable batch system: External reference specification," Technical report, MRJ Technology Solutions, Tech. Rep., 1999.

[12] B. Benatallah, M. Dumas, M.-C. Fauvet, and F. Rabhi, "Towards patterns of web services composition," in *Patterns and Skeletons for Parallel and Distributed Computing*, F. Rabhi and S. Gorlatch, Eds. Springer London, 2003, pp. 265–296.

[13] L. Bodenstaff, A. Wombacher, M. Reichert, and M. Jaeger, "Monitoring dependencies for slas: The mode4sla approach," in *Proceedings of the IEEE International Conference on Services Computing (SCC)*, vol. 1. IEEE, 2008, pp. 21–29.

[14] J. Branke and K. Deb, "Integrating user preferences into evolutionary multi-objective optimization," *Knowledge Incorporation in Evolutionary Computation*, pp. 461–477, 2005.

[15] J. Brans, P. Vincke, and B. Mareschal, "How to select and how to rank projects: The promethee method," *European journal of operational research*, vol. 24, no. 2, pp. 228–238, 1986.

[16] R. Buyya, "Economic-based distributed resource management and scheduling for grid computing," *PhD Thesis, Monash University, Melbourne, Australia*, 2002. [Online]. Available: http://www.buyya.com/thesis/thesis.pdf

[17] R. Buyya and S. Venugopal, "The gridbus toolkit for service oriented grid and utility computing: An overview and status report," in *Proceedings of the 1st IEEE International Workshop on Grid Economics and Business Models (GECON)*. IEEE, 2004, pp. 19–66.

[18] R. Buyya, C. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.

[19] R. Buyya, R. Ranjan, and R. Calheiros, "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services," in *Algorithms and Architectures for Parallel Processing*, ser. Lecture Notes in Computer Science, C.-H. Hsu, L. Yang, J. Park, and S.-S. Yeo, Eds. Springer Berlin Heidelberg, 2010, vol. 6081, pp. 13–31.

[20] R. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[21] E. Carlini, M. Coppola, P. Dazzi, L. Ricci, and G. Righetti, "Cloud federations in contrail," in *Proceedings of the 17th International European Conference on Parallel and Distributed Computing (Euro-Par): Parallel Processing Workshops*. Springer, 2012, pp. 159–168.

[22] C. Carlsson and R. Fullér, "Owa operators for decision support," *in Proceedings of the 3rd European Congress on Intelligent Techniques and Soft Computing (EUFIT)*, vol. 97, pp. 1539–1544, 1997.

[23] E. Cecchet, J. Marguerite, and W. Zwaenepoel, "Performance and scalability of ejb applications," *ACM Sigplan Notices*, vol. 37, no. 11, pp. 246–261, 2002.

[24] Z. Cheng, Z. Du, Y. Chen, and X. Wang, "Soavm: A service-oriented virtualization management system with automated configuration," in *Proceedings of the IEEE In-*

ternational Symposium on Service-Oriented System Engineering (SOSE). IEEE, 2008, pp. 251–256.

[25] M. Chhetri, Q. Vo, and R. Kowalczyk, "Policy-based automation of sla establishment for cloud computing services," in *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 2012, pp. 164–171.

[26] D. Choffnes, F. Bustamante, and Z. Ge, "Using the crowd to monitor the cloud: Network event detection from edge systems," in *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2010.

[27] V. Chvatal, "A greedy heuristic for the set-covering problem," *Mathematics of operations research*, vol. 4, no. 3, pp. 233–235, 1979.

[28] P. Cingolani and J. Alcala-Fdez, "jfuzzylogic: a robust and flexible fuzzy-logic inference system language implementation," in *Proceedings of the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, june 2012, pp. 1 –8.

[29] CloudHarmony, "Benchmarks," http://cloudharmony.com/.

[30] R. Coehoorn and N. Jennings, "Learning on opponent's preferences to make effective multi-issue negotiation trade-offs," in *Proceedings of the 6th international conference on Electronic commerce*. ACM, 2004, pp. 59–68.

[31] C. A. C. Coello *et al.*, "A comprehensive survey of evolutionary-based multiobjective optimization techniques," *Knowledge and Information systems*, vol. 1, no. 3, pp. 129–156, 1999.

[32] C. A. C. Coello, G. B. Lamont, and D. A. Van Veldhuizen, *Evolutionary algorithms for solving multi-objective problems*. Springer, 2007, vol. 5.

[33] C. Coello Coello and M. Lechuga, "Mopso: A proposal for multiple objective particle swarm optimization," in *Proceedings of the 2002 Congress on Evolutionary Computation (CEC)*, vol. 2. IEEE, 2002, pp. 1051–1056.

[34] M. Comuzzi and G. Spanoudakis, "Describing and verifying monitoring capabilities for sla-driven service-based systems," in *Proceedings of the 21st International Conferences on Advanced Information System Engineering (CAiSE)*, 2009.

[35] G. Cretella and B. Di Martino, "Semantic web annotation and representation of cloud apis," in *Proceedings of the third International Conference on Emerging Intelligent Data and Web Technologies (EIDWT)*. IEEE, 2012, pp. 31–37.

[36] G. Cretella and B. Di Martino, "Towards a semantic engine for cloud applications development," in *Proceedings of the 2012 Sixth International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*. IEEE, 2012, pp. 198–203.

[37] G. Cretella, B. Di Martino, and V. Stankovski, "Using the mosaic's semantic engine to design and develop civil engineering cloud applications," in *Proceedings of the 14th International Conference on Information Integration and Web-based Applications & Services*. ACM, 2012, pp. 378–386.

[38] C. Darwin and J. W. Burrow, *The Origin of Species: Or, the Preservation of Favoured Races in the Struggle for Life*. Oxford University Press, 1963.

[39] S. De, R. Biswas, and A. Roy, "An application of intuitionistic fuzzy sets in medical diagnosis," *Fuzzy Sets and Systems*, vol. 117, no. 2, pp. 209–213, 2001.

[40] J. De Bruijn, H. Lausen, A. Polleres, and D. Fensel, "The web service modeling language wsml: An overview," *in Proceedings of the 3rd European conference on The Semantic Web: research and applications (ESWC06)*, pp. 590–604, 2006.

[41] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182–197, 2002.

[42] K. Deb, *Multi-objective optimization*. John Wiley & Sons Hoboken, NJ, 2001.

[43] B. Di Martino, D. Petcu, R. Cossu, P. Goncalves, T. Máhr, and M. Loichate, "Building a mosaic of clouds," in *Proceedings of the 15th International European Confer-*

ence on Parallel and Distributed Computing (Euro-Par): Parallel Processing Workshops. Springer, 2010, pp. 571–578.

[44] M. Dimitrov, A. Simov, V. Momtchev, and M. Konstantinov, "Wsmo studio–a semantic web services modelling environment for wsmo," *in Proceedings of the 4th European conference on The Semantic Web: Research and Applications (ESWC)*, pp. 749–758, 2007.

[45] DMTF, "Open virtualization format," http://www.dmtf.org/standards/ovf.

[46] J. Dujmovic, "Mixed averaging by levels (mal)ï£¡a system and computer evaluation method," in *Proceedings of the Informatica Conf.(in Serbo-Croatian), Bled, Yugoslavia*, 1973.

[47] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry, "The cops (common open policy service) protocol," 2000.

[48] J. Durillo and A. Nebro, "jmetal: A java framework for multi-objective optimization," *Advances in Engineering Software*, vol. 42, no. 10, pp. 760–771, 2011.

[49] V. Emeakaroha, I. Brandic, M. Maurer, and S. Dustdar, "Low level metrics to high level slas-lom2his framework: Bridging the gap between monitored metrics and sla parameters in cloud environments," in *Proceedings of the 2010 International Conference on High Performance Computing and Simulation (HPCS)*. IEEE, 2010, pp. 48–54.

[50] D. Ersoz, M. Yousif, and C. Das, "Characterizing network traffic in a cluster-based, multi-tier data center," in *Proceedings of the 27th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2007, pp. 59–59.

[51] K. Fakhfakh, S. Tazi, K. Drira, T. Chaari, and M. Jmaiel, "Semantic enabled framework for sla monitoring," *International Journal on Advances in Software*, vol. 2, no. 1, pp. 36–46, 2009.

[52] P. Faratin, "Automated service negotiation between autonomous computational agents," Ph.D. dissertation, PhD Thesis, University of London, 2000.

[53] P. Faratin, C. Sierra, and N. Jennings, "Using similarity criteria to make issue tradeoffs in automated negotiations," *artificial Intelligence*, vol. 142, no. 2, pp. 205–237, 2002.

[54] D. Fensel and C. Bussler, "The web service modeling framework wsmf," *Electronic Commerce Research and Applications*, vol. 1, no. 2, pp. 113–137, 2002.

[55] D. Fensel, F. Facca, E. Simperl, and I. Toma, "Web service modeling ontology," in *Semantic Web Services*. Springer Berlin Heidelberg, 2011, pp. 107–129.

[56] M. Ferdinand, C. Zirpins, and D. Trastour, "Lifting xml schema to owl," in *Web Engineering*, ser. Lecture Notes in Computer Science, N. Koch, P. Fraternali, and M. Wirsing, Eds. Springer Berlin Heidelberg, 2004, vol. 3140, pp. 354–358.

[57] J. Fernández Salido and S. Murakami, "Extending yager's orness concept for the owa aggregators to other mean operators," *Fuzzy Sets and Systems*, vol. 139, no. 3, pp. 515–542, 2003.

[58] A. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. Badia, K. Djemame *et al.*, "Optimis: A holistic approach to cloud service provisioning," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 66–77, 2012.

[59] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *International journal of high performance computing applications*, vol. 15, no. 3, pp. 200–222, 2001.

[60] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Proceedings of the Grid Computing Environments Workshop (GCE)*. Ieee, 2008, pp. 1–10.

[61] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, *The Physiology of the Grid*. John Wiley & Sons, Ltd, 2003, pp. 217–249.

[62] J. García, D. Ruiz, A. Ruiz-Cortés, and J. Parejo, "Qos-aware semantic service selection: An optimization problem," in *Proceedings of the IEEE Congress on Services-Part I*. IEEE, 2008, pp. 384–388.

[63] J. Garcıa, I. Toma, D. Ruiz, and A. Ruiz-Cortés, "A service ranker based on logic rules evaluation and constraint programming," in *Proceedings of the 2nd ECOWS Non-Functional Properties and Service Level Agreements in Service Oriented Computing Workshop*, vol. 411. Citeseer, 2008.

[64] J. García, D. Ruiz, and A. Ruiz-Cortés, "On user preferences and utility functions in selection: A semantic approach," in *Proceedings of the Service-Oriented Computing - ICSOC 2007 Workshops*, ser. Lecture Notes in Computer Science, E. Nitto and M. Ripeanu, Eds., vol. 4907. Springer Berlin Heidelberg, pp. 105–114.

[65] S. Garg, C. Yeo, and R. Buyya, "Green cloud framework for improving carbon efficiency of clouds," *in Proceedings of the 17th International European Conference on Parallel and Distributed Computing (Euro-Par)*, pp. 491–502, 2011.

[66] S. Garg, S. Versteeg, and R. Buyya, "Smicloud: A framework for comparing and ranking cloud services," in *Proceedings of the Fourth IEEE International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2011, pp. 210–218.

[67] S. Gass and T. Rapcsák, "Singular value decomposition in ahp," *European Journal of Operational Research*, vol. 154, no. 3, pp. 573–584, 2004.

[68] Í. Goiri, F. Julia, J. Fitó, M. Macías, and J. Guitart, "Resource-level qos metric for cpu-based guarantees in cloud providers," in *Economics of Grids, Clouds, Systems, and Services*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, vol. 6296.

[69] D. E. Goldberg, "Genetic algorithms in search, optimization and machine learning," 1989.

[70] S. Greco, *Multiple criteria decision analysis: state of the art surveys*. Springer, 2004, vol. 78.

[71] R. Grønmo and M. Jaeger, "Model-driven methodology for building qos-optimised web service compositions," in *Distributed Applications and Interoperable Systems*, ser. Lecture Notes in Computer Science, L. Kutvonen and N. Alonistioti, Eds. Springer Berlin Heidelberg, 2005, vol. 3543, pp. 68–82.

[72] W. W. Group *et al.*, "D2v1. 0: Web service modeling ontology (wsmo). wsmo working draft,(2004)," 2004.

[73] T. Guha and S. Ludwig, "Comparison of service selection algorithms for grid services: Multiple objective particle swarm optimization and constraint satisfaction based service selection," in *Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence, 2008. ICTAI'08.*, vol. 1. IEEE, 2008, pp. 172–179.

[74] A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler, "Wsmx-a semantic service-oriented architecture," in *Proceedings of the IEEE International Conference on Web Services (ICWS)*. IEEE, 2005, pp. 321–328.

[75] L. Han and D. Berry, "Semantic-supported and agent-based decentralized grid resource discovery," *Future Generation Computer Systems*, vol. 24, no. 8, pp. 806–812, 2008.

[76] T. Han and K. Sim, "An ontology-enhanced cloud service discovery system," *in Proceedings of the International MultiConference of Engineers and Computer Scientists*, 2010.

[77] I. Horrocks, D. Fensel, J. Broekstra, S. Decker, M. Erdmann, C. Goble, F. van HARMELEN, M. Klein, S. Staab, R. Studer *et al.*, "Oil: The ontology inference layer," Technical Report IR-479, Vrije Universiteit Amsterdam, Faculty of Sciences, Tech. Rep., 2000.

[78] C. Hou, "Predicting agents tactics in automated negotiation," in *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT)*. IEEE, 2004, pp. 127–133.

[79] X. Hu and R. Eberhart, "Multiobjective optimization using dynamic neighborhood particle swarm optimization," in *Proceedings of the Congress on Evolutionary Computation*, vol. 2. IEEE, 2002, pp. 1677–1681.

[80] P. Hung, H. Li, and J. Jeng, "Ws-negotiation: an overview of research issues," in *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*. IEEE, 2004, pp. 10–pp.

[81] C. Hwang, K. Yoon *et al.*, *Multiple attribute decision making: methods and applications: a state-of-the-art survey*. Springer-Verlag New York, 1981, vol. 13.

[82] M. Jaeger and G. Rojec-Goldmann, "Seneca–simulation of algorithms for the selection of web services for compositions," *Technologies for E-Services*, pp. 84–97, 2006.

[83] M. Jaeger, G. Rojec-Goldmann, C. Liebetruth, G. Mühl, and K. Geihs, "Ranked matching for service descriptions using owl-s," in *Kommunikation in Verteilten Systemen (KiVS)*, ser. Informatik aktuell. Springer Berlin Heidelberg, 2005, pp. 91–102.

[84] M. Jaeger, G. Muhl, and S. Golze, "Qos-aware composition of web services: a look at selection algorithms," in *Proceedings of IEEE International Conference onWeb Services (ICIW)*. IEEE, 2005.

[85] M. Jaeger, G. Rojec-Goldmann, and G. Muhl, "Qos aggregation for web service composition using workflow patterns," in *Proceedings.of the Eighth IEEE International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 2004, pp. 149–159.

[86] P. Jaeger, J. Lin, and J. Grimes, "Cloud computing and information policy: Computing in a policy cloud?" *Journal of Information Technology & Politics*, vol. 5, no. 3, pp. 269–283, 2008.

[87] B. Javadi and R. Buyya, "Comprehensive statistical analysis and modeling of spot instances in public cloud environments," *The University of Melbourne, Melbourne, Tech. Rep*, 2011.

[88] D. Jones, *The Defnitive Guide to Monitoring the Data Center, Virtual Environments, and the Cloud*. Nimsoft, Campbell, CA.

[89] A. Jsang and R. Ismail, "The beta reputation system," in *Proceedings of the 15th Bled Electronic Commerce Conference*, 2002, pp. 41–55.

[90] R. Kanagasabai *et al.*, "Owl-s based semantic cloud service broker," in *Proceedings of the 19th International Conference on Web Services (ICWS)*. IEEE, 2012, pp. 560–567.

[91] K. Keahey, I. Foster, T. Freeman, and X. Zhang, "Virtual workspaces: Achieving quality of service and quality of life in the grid," *Scientific Programming*, vol. 13, no. 4, p. 265, 2005.

[92] U. Keller, R. Lara, H. Lausen, A. Polleres, L. Predoiu, and I. Toma, "Semantic web service discovery," *WSMX Working Draft*, 2005.

[93] J. Kephart and W. Walsh, "An artificial intelligence perspective on autonomic computing policies," in *Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, 2004. POLICY 2004.* IEEE, 2004, pp. 3–12.

[94] M. Kerrigan, "D9. 1v0. 2 web service modeling toolkit (wsmt)," *WSMX Deliverable available from http://www. wsmo. org/TR/d9/d9*, vol. 1, p. v0, 2005.

[95] M. Klein, D. Fensel, F. Van Harmelen, and I. Horrocks, "The relation between ontologies and xml schemas," *Electronic Trans. on Artificial Intelligence*, 2001.

[96] M. Klusch, B. Fries, and K. Sycara, "Owls-mx: A hybrid semantic web service matchmaker for owl-s services," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 7, no. 2, pp. 121–133, 2009.

[97] M. Klusch and F. Kaufer, "Wsmo-mx: A hybrid semantic web service matchmaker," *Web Intelligence and Agent Systems*, vol. 7, no. 1, pp. 23–42, 2009.

[98] M. Klusch and M. Klusch, "Semantic web service coordination," pp. 59–104, 2008.

[99] D. Kondo, B. Javadi, A. Iosup, and D. Epema, "The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems," in *Proceedings of*

the10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid). IEEE, 2010, pp. 398–407.

[100] J. Kopeckỳ, M. Moran, D. Roman, and A. Mocan, "Wsmo grounding," *WSMO Working Draft D*, vol. 24, 2005.

[101] J. Kopeckỳ, D. Roman, T. Vitvar, M. Moran, and A. Mocan, "Wsmo grounding. wsmo working draft v0. 1, 2007."

[102] C. Kotsokalis, R. Yahyapour, and M. Gonzalez, "Sami: The sla management instance," in *Proceedings of the Fifth International Conference on Internet and Web Applications and Services (ICIW)*. IEEE, 2010, pp. 303–308.

[103] D. Krafzig, K. Banke, and D. Slama, *Enterprise SOA: service-oriented architecture best practices*. Prentice Hall PTR, 2005.

[104] K. Kritikos and D. Plexousakis, "Semantic qos metric matching," in *Proceedings of the 4th European Conference on Web Services (ECOWS)*. IEEE, 2006, pp. 265–274.

[105] D. D. Lamanna, J. Skene, and W. Emmerich, "Slang: A language for defining service level agreements," ser. FTDCS '03. IEEE Computer Society, 2003, pp. 100–.

[106] S. Lamparter, A. Ankolekar, R. Studer, and S. Grimm, "Preference-based selection of highly configurable web services," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 1013–1022.

[107] S. Lamparter, A. Ankolekar, R. Studer, D. Oberle, and C. Weinhardt, "A policy framework for trading configurable goods and services in open electronic markets," in *Proceedings of the 8th international conference on Electronic commerce: The new e-commerce: innovations for conquering current barriers, obstacles and limitations to conducting successful business on the internet*. ACM, 2006, pp. 162–173.

[108] R. Lawley, M. Luck, K. Decker, T. Payne, and L. Moreau, "Automated negotiation between publishers and consumers of grid notifications," *Parallel Processing Letters*, vol. 13, no. 04, pp. 537–548, 2003.

[109] A. Lawrence, K. Djemame, O. Wäldrich, W. Ziegler, and C. Zsigri, "Using service level agreements for optimising cloud infrastructure services," in *Proceedings of the Towards a Service-Based Internet. ServiceWave 2010 Workshops*. Springer, 2010, pp. 38–49.

[110] J. Li and R. Yahyapour, "Negotiation strategies for grid scheduling," in *Advances in Grid and Pervasive Computing*, ser. Lecture Notes in Computer Science, Y.-C. Chung and J. Moreira, Eds. Springer Berlin Heidelberg, 2006, vol. 3947, pp. 42–52.

[111] M. Litzkow, M. Livny, and M. Mutka, "Condor-a hunter of idle workstations," in *Proceedings of the 8th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1988, pp. 104–111.

[112] Z. Liu, T. Liu, L. Cai, and G. Yang, "Quality evaluation and selection framework of service composition based on distributed agents," in *Proceedings of the Fifth International Conference on Next Generation Web Services Practices (NWESP)*. IEEE, 2009, pp. 68–75.

[113] H. Ludwig, A. Keller, A. Dan, R. King, and R. Franck, "Web service level agreement (wsla) language specification," *IBM Corporation*, pp. 815–824, 2003.

[114] S. Ludwig and S. Reyhani, "Selection algorithm for grid services based on a quality of service metric," in *Proceedings of the 21st International Symposium on High Performance Computing Systems and Applications (HPCS)*. IEEE, 2007, pp. 13–13.

[115] M. Macias and J. Guitart, "Using resource-level information into nonadditive negotiation models for cloud market environments," in *Proceedings of the 2010 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2010, pp. 325–332.

[116] E. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *International Journal of Man-Machine Studies*, vol. 7, no. 1, pp. 1–13, 1975.

[117] K. Mansour, R. Kowalczyk, and M. Wosko, "Concurrent negotiation over quality of service," in *Proceedings of the 2012 IEEE Ninth International Conference on Services Computing (SCC)*. IEEE, 2012, pp. 446–453.

[118] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne *et al.*, "Owl-s: Semantic markup for web services," *W3C Member submission*, vol. 22, pp. 2007–04, 2004.

[119] E. Maximilien and M. Singh, "A framework and ontology for dynamic web services selection," *IEEE Internet Computing*, vol. 8, no. 5, pp. 84–93, 2004.

[120] D. McGuinness, F. Van Harmelen *et al.*, "Owl web ontology language overview," *W3C recommendation*, vol. 10, no. 2004-03, p. 10, 2004.

[121] K. Meffert, N. Rotstan, C. Knowles, and U. Sangiorgi, "Jgap-java genetic algorithms and genetic programming package," *URL: http://jgap. sf. net*, 2008.

[122] P. Mell and T. Grance, "The nist definition of cloud computing (draft)," *NIST special publication*, vol. 800, p. 145, 2011.

[123] D. Menasce, "Qos issues in web services," *IEEE Internet Computing*, vol. 6, no. 6, pp. 72–75, 2002.

[124] D. Menascé, E. Casalicchio, and V. Dubey, "On optimal service selection in service oriented architectures," *Performance Evaluation*, vol. 67, no. 8, pp. 659–675, 2010.

[125] M. Menzel and R. Ranjan, "Cloudgenius: decision support for web server cloud migration," in *Proceedings of the 21st international conference on World Wide Web*. ACM, 2012, pp. 979–988.

[126] J. Morse, "Reducing the size of the nondominated set: Pruning by clustering," *Computers & Operations Research*, vol. 7, no. 1, pp. 55–66, 1980.

[127] H. Muñoz, I. Kotsiopoulos, A. Micsik, B. Koller, and J. Mora, "Flexible sla negotiation using semantic annotations," in *Proceedings of the Service-Oriented Computing (ICSOC), ServiceWave Workshops*. Springer, 2009, pp. 165–175.

[128] B. Narasimhan and R. Nichols, "State of cloud applications and platforms: The cloud adopters' view," *Computer*, vol. 44, no. 3, pp. 24–28, 2011.

[129] T. Nguyen, N. Boukhatem, and G. Puiolle, "Cops-sls usage for dynamic policy-based qos management over heterogeneous ip networks," *Network, IEEE*, vol. 17, no. 3, pp. 44–50, 2003.

[130] M. Paolucci, J. Soudry, N. Srinivasan, K. Sycara, M. Paolucci, J. Soudry, N. Srinivasan, and K. Sycara, "A broker for owl-s web services," in *Extending Web Services Technologies*, ser. Multiagent Systems, Artificial Societies, and Simulated Organizations, L. Cavedon, Z. Maamar, D. Martin, and B. Benatallah, Eds. Springer US, 2004, vol. 13, pp. 79–98.

[131] I. Papaioannou, D. Tsesmetzis, I. Roussaki, and M. Anagnostou, "A qos ontology language for web-services," in *Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA)*, vol. 1. IEEE, 2006, pp. 6–pp.

[132] P. Papakos, L. Capra, and D. Rosenblum, "Volare: context-aware adaptive cloud service discovery for mobile systems," in *Proceedings of the 9th International Workshop on Adaptive and Reflective Middleware*. ACM, 2010, pp. 32–38.

[133] K. Parsopoulos and M. Vrahatis, "Particle swarm optimization method in multiobjective problems," in *Proceedings of the 2002 ACM symposium on Applied computing*. ACM, 2002, pp. 603–607.

[134] P. Pawluk, B. Simmons, M. Smit, M. Litoiu, and S. Mankovski, "Introducing stratos: A cloud broker service," in *Proceedings of the 5th IEEE International Conference on Cloud Computing (IEEE CLOUD)*. IEEE, 2012, pp. 891–898.

[135] S. Pearson and A. Benameur, "Privacy, security and trust issues arising from cloud computing," in *Proceedings of the Second International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2010, pp. 693–702.

[136] D. Petcu, G. Macariu, S. Panica, and C. Crăciun, "Portable cloud applications-from theory to practice," *Future Generation Computer Systems*, 2012.

[137] D. Pisinger, "Algorithms for knapsack problems," Ph.D. dissertation, University of Copenhagen, 1995.

[138] F. Protocol, *Specification, Foundation for Intelligent Physical Agents*.

[139] I. Rahwan, R. Kowalczyk, and H. H. Pham, "Intelligent agents for automated one-to-many e-commerce negotiation," in *Proceedings of the 25th Australasian conference on Computer science*, ser. ACSC '02. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2002, pp. 197–204.

[140] H. Raiffa, *The art and science of negotiation*. Belknap Press, 1982.

[141] M. Rak, L. Liccardo, and R. Aversa, "A sla-based interface for security management in cloud and grid integrations," in *Proceedings of the 7th International Conference on Information Assurance and Security (IAS)*. IEEE, 2011, pp. 378–383.

[142] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed resource management for high throughput computing," in *Proceedings of the Seventh International Symposium on High Performance Distributed Computing (HPDC)*. IEEE, 1998, pp. 140–146.

[143] S. Ran, "A model for web services discovery with qos," *ACM Sigecom exchanges*, vol. 4, no. 1, pp. 1–10, 2003.

[144] R. Ranjan, L. Zhao, X. Wu, A. Liu, A. Quiroz, and M. Parashar, "Peer-to-peer cloud provisioning: Service discovery and load-balancing," *Cloud Computing*, pp. 195–217, 2010.

[145] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, D. Fensel *et al.*, "Web service modeling ontology," *Applied Ontology*, vol. 1, no. 1, pp. 77–106, 2005.

[146] A. RUIZ-CORTÉS, O. MARTÍN-DÍAZ, A. Duran, and M. Toro, "Improving the automatic procurement of web services using constraint programming," *International Journal of Cooperative Information Systems*, vol. 14, no. 04, pp. 439–467, 2005.

[147] T. L. Saaty, "Axiomatic foundation of the analytic hierarchy process," *Management science*, vol. 32, no. 7, pp. 841–855, 1986.

[148] T. Saaty, "Fundamentals of decision making," *Pittsburgh: RWS Publications*, 1994.

[149] A. Sahai, V. Machiraju, M. Sayal, A. P. A. v. Moorsel, and F. Casati, "Automated sla monitoring for web services," in *Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management: Management Technologies for E-Commerce and E-Business Applications*, ser. DSOM '02. London, UK, UK: Springer-Verlag, 2002, pp. 28–41.

[150] C. Sapuntzakis, D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M. Lam, and M. Rosenblum, "Virtual appliances for deploying and maintaining software," in *Proceedings of the 17th USENIX conference on System administration*, 2003, pp. 181–194.

[151] J. Schopf, L. Pearlman, N. Miller, C. Kesselman, I. Foster, M. D'Arcy, and A. Chervenak, "Monitoring the grid with the globus toolkit mds4," in *Journal of Physics: Conference Series*, vol. 46, no. 1. IOP Publishing, 2006, p. 521.

[152] A. ShaikhAli, O. Rana, R. Al-Ali, and D. Walker, "Uddie: An extended registry for web services," in *Proceedings of the 2003 Symposium on Applications and the Internet Workshops.* IEEE, 2003, pp. 85–89.

[153] M. Sierra and C. Coello, "Improving pso-based multi-objective optimization using crowding, mutation and e-dominance," in *Proceedings of the 3rd International Conference on Evolutionary Multi-Criterion Optimization (EMO)*. Springer, 2005, pp. 505–519.

[154] K. Sim, "Grid resource negotiation: survey and new directions," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 40, no. 3, pp. 245–257, 2010.

[155] K. Sim and S. Wang, "Flexible negotiation agent with relaxed decision rules," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 34, no. 3, pp. 1602–1608, 2004.

[156] N. Srinivas and K. Deb, "Muiltiobjective optimization using nondominated sorting in genetic algorithms," *Evolutionary computation*, vol. 2, no. 3, pp. 221–248, 1994.

[157] M. Stollberg, M. Hepp, and J. Hoffmann, "A caching mechanism for semantic web service discovery," in *Proceedings of the 6th international semantic web and 2nd Asian conference on Asian semantic web conference*, ser. ISWC'07/ASWC'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 480–493.

[158] C. Sun, L. He, Q. Wang, and R. Willenborg, "Simplifying service deployment with virtual appliances," in *Proceedings of the IEEE International Conference on Services Computing (SCC)*, vol. 2. IEEE, 2008, pp. 265–272.

[159] E. Szmidt and J. Kacprzyk, "Intuitionistic fuzzy sets in group decision making," *Notes on IFS*, vol. 2, no. 1, pp. 11–14, 1996.

[160] W. Theilmann, U. Winkler, J. Happe, and I. M. de Abril, "Managing on-demand business applications with hierarchical service level agreements," in *Proceedings of the 3rd future internet conference on Future internet*, ser. FIS'10. Springer-Verlag, 2010, pp. 97–106.

[161] I. Toma, D. Foxvog, and M. Jaeger, "Modeling qos characteristics in wsmo," in *Proceedings of the 1st workshop on Middleware for Service Oriented Computing (MW4SOC 2006)*. ACM, 2006, pp. 42–47.

[162] I. Toma, D. Roman, D. Fensel, B. Sapkota, and J. M. Gomez, "A multi-criteria service ranking approach based on non-functional properties rules evaluation," in *Proceedings of the 5th international conference on Service-Oriented Computing (ICSOC)*, ser. ICSOC '07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 435–441.

[163] V. Tosic, K. Patel, and B. Pagurek, "Wsol - web service offerings language," in *Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web*, ser. CAiSE '02/ WES '02. Springer-Verlag, 2002, pp. 57–67.

[164] V. Tran, H. Tsuji, and R. Masuda, "A new qos ontology and its qos-based ranking algorithm for web services," *Simulation Modelling Practice and Theory*, vol. 17, no. 8, pp. 1378–1398, 2009.

[165] D. Tsesmetzis, I. Roussaki, and E. Sykas, "Modeling and simulation of qos-aware web service selection for provider profit maximization," *Simulation*, vol. 83, no. 1, pp. 93–106, 2007.

[166] D. A. Van Veldhuizen, "Multiobjective evolutionary algorithms: classifications, analyses, and new innovations," DTIC Document, Tech. Rep., 1999.

[167] J. Varia, "Best practices in architecting cloud applications in the AWS Cloud," in *Cloud Computing: Principles and Paradigms*, R. Buyya, J. Broberg, and A. Goscinski, Eds. Wiley Press, 2011, ch. 18, pp. 459–490.

[168] M. Vouk, S. Averitt, M. Bugaev, A. Kurth, A. Peeler, A. Rindos, H. Shaffer, E. Sills, S. Stein, and J. Thompson, "Powered by vcl-using virtual computing laboratory (vcl) technology to power cloud computing," in *Proceedings of the 2nd International Conference on Virtual Computing (ICVCI)*, 2008, pp. 15–16.

[169] A. Walsh, *UDDI, SOAP, and WSDL: The Web Services Specification Reference Book*. Prentice Hall Professional Technical Reference, 2002.

[170] N. Wancheng, H. Lingjuan, L. Lianchen, and W. Cheng, "Commodity-market based services selection in dynamic web service composition," in *Proceedings of the of the 2nd IEEE Asia-Pacific Service Computing Conference*. IEEE, 2007, pp. 218–223.

[171] G. Wang, A. Chen, C. Wang, C. Fung, and S. Uczekaj, "Integrated quality of service (qos) management in service-oriented enterprise architectures," in *Proceedings of the 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 2004, pp. 21–32.

[172] P. Wang, "Qos-aware web services selection with intuitionistic fuzzy set under consumers vague perception," *Expert Systems with Applications*, vol. 36, no. 3, pp. 4460–4466, 2009.

[173] X. Wang, T. Vitvar, M. Kerrigan, and I. Toma, "A qos-aware selection model for semantic web services," in *Proceedings of the 4th international conference on Service-Oriented Computing*, ser. ICSOC'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 390–401.

[174] X. Wang, K. Yue, J. Z. Huang, and A. Zhou, "Service selection in dynamic demand-driven web services," in *Proceedings of the IEEE International Conference on Web Services*, ser. ICWS '04. Washington, DC, USA: IEEE Computer Society, 2004.

[175] S. Weibel, J. Kunze, C. Lagoze, and M. Wolf, "Dublin core metadata for resource discovery," *Internet Engineering Task Force RFC*, vol. 2413, p. 222, 1998.

[176] M. Winkler, T. Springer, and A. Schill, "Automating composite sla management tasks by exploiting service dependency information," in *Proceedings of the IEEE 8th European Conference on Web Services (ECOWS)*. IEEE, 2010, pp. 59–66.

[177] Z. Xiao and D. Cao, "A policy-based framework for automated sla negotiation for internet-based virtual computing environment," in *Proceedings of IEEE 16th International Conference on the Parallel and Distributed Systems (ICPADS)*. IEEE, 2010, pp. 694–699.

[178] J. Yan, R. Kowalczyk, J. Lin, M. Chhetri, S. Goh, and J. Zhang, "Autonomous service level agreement negotiation for service composition provision," *Future Generation Computer Systems*, vol. 23, no. 6, pp. 748–759, 2007.

[179] Y. Yao and H. Chen, "Qos-aware service composition using nsga-ii1," in *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, ser. ICIS '09. New York, NY, USA: ACM, 2009, pp. 358–363.

[180] R. Yavatkar, D. Pendarakis, and R. Guerin, "A framework for policy-based admission control," 2000.

[181] H. Yu and S. Reiff-Marganiec, "A method for automated web service selection," in *Proceedings of the IEEE Congress on Services-Part I*. IEEE, 2008, pp. 513–520.

[182] H. Yu and S. Reiff-Marganiec, "Non-functional property based service selection: A survey and classification of approaches," 2008.

[183] L. Zadeh, "Fuzzy logic= computing with words," *IEEE Transactions on Fuzzy Systems*, vol. 4, no. 2, pp. 103–111, 1996.

[184] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 311–327, 2004.

[185] W. Zeng, Y. Zhao, and J. Zeng, "Cloud service and service selection algorithm research," in *Proceedings of the 1st ACM/SIGEVO Summit on Genetic and Evolutionary Computation*. ACM, 2009, pp. 1045–1048.

[186] X. Zheng, P. Martin, and K. Brohman, "Cloud service negotiation: Concession vs. tradeoff approaches," in *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid)*. IEEE Computer Society, 2012, pp. 515–522.

[187] X. Zheng, P. Martin, W. Powley, and K. Brohman, "Applying bargaining game theory to web services negotiation," in *Proceedings of the 2010 IEEE International Conference on Services Computing (SCC)*. IEEE, 2010, pp. 218–225.

[188] C. Zhou, L. Chia, and B. Lee, "Daml-qos ontology for web services," in *Proceedings of the IEEE International Conference on Web Services*. IEEE, 2004, pp. 472–479.

[189] H. Zimmermann, "Fuzzy programming and linear programming with several objective functions," *Fuzzy sets and systems*, vol. 1, no. 1, pp. 45–55, 1978.

[190] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.

[191] F. Zulkernine, P. Martin, C. Craddock, and K. Wilson, "A policy-based middleware for web services sla negotiation," in *Proceedings of the IEEE International Conference on Web Services (ICWS)*. IEEE, 2009, pp. 1043–1050.

[192] F. Zulkernine and P. Martin, "An adaptive and intelligent sla negotiation system for web services," *IEEE Transactions on Services Computing*, vol. 4, no. 1, pp. 31–43, 2011.

Appendix A

Ontologies

A.1 Portion of Developed Ontology

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace { _"http://www.cloudslab.org/CloudProvider#"
 }
ontology CloudProviderOntology
concept VMFormat
hasName ofType _string
concept place
     hasName ofType _string
concept state subConceptOf place
concept country subConceptOf place
concept location
hasState ofType state
hasCountry ofType country

concept cloudService

concept monitoringMetric
hasName ofType _string
instance CPUUtilization memberOf monitoringMetric

concept loadBalancer subConceptOf cloudService
hasPort ofType _integer
hasProtocol ofType _string
isHealthcheckEnabled ofType _boolean
checkInterval ofType _integer
hasTimeOut ofType _integer
hasThreshold ofType _integer

concept virtualAppliance subConceptOf cloudService
 _"http://www.CloudsLab.org/ontologies/VirtualAppliance#imageId" ofType _string

              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#imageLocation"
                  ofType _string

              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#imageState"
                  ofType _string

              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#imageOwnerId"
                  ofType _string

```

```
38              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#isPublic" ofType
                    {_string, _boolean}
39
40              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#architecture"
                    ofType _string
41
42              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#imageType"
                    ofType _string
43
44              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#kernelId" ofType
                     _string
45
46              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#ramdiskId"
                    ofType _string
47
48              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#platform" ofType
                     _string
49
50              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#imageOwnerAlias"
                     ofType _string
51
52              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#name" ofType
                    _string
53
54              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#description"
                    ofType _string
55
56              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#rootDeviceType"
                    ofType _string
57
58              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#rootDeviceName"
                    ofType _string
59
60              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#
                    virtualizationType" ofType _string
61
62              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#hypervisor"
                    ofType _string
63
64              hasProvider ofType cloud
65              isCompatipleWith ofType virtualUnit
66              hasName ofType _string
67              hasFormat ofType VMFormat
68
69  instance  aki00806369  memberOf virtualAppliance
70              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#imageId"
                    hasValue "aki-00806369"
71              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#imageLocation"
                    hasValue "karmic-kernel-zul/ubuntu-kernel-2.6.31-300-ec2-i386
                    -20091001-test-04.manifest.xml"
72              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#imageState"
                    hasValue "available"
73              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#imageOwnerId"
                    hasValue "099720109477"
74              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#isPublic"
                    hasValue "true"
75              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#architecture"
                    hasValue "i386"
76              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#imageType"
                    hasValue "kernel"
```

```
77              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#rootDeviceType"
                    hasValue "instance-store"
78              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#
                    virtualizationType" hasValue "paravirtual"
79              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#hypervisor"
                    hasValue "xen"
80              hasProvider hasValue {AmazonCalifornia}
81              hasName hasValue "aki00806369"
82              hasFormat hasValue AMI
83      instance aki00896a69 memberOf virtualAppliance
84              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#imageId"
                    hasValue "aki-00896a69"
85              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#imageLocation"
                    hasValue "karmic-kernel-zul/ubuntu-kernel-2.6.31-300-ec2-i386
                    -20091002-test-04.manifest.xml"
86              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#imageState"
                    hasValue "available"
87              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#imageOwnerId"
                    hasValue "099720109477"
88              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#isPublic"
                    hasValue "true"
89              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#architecture"
                    hasValue "i386"
90              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#imageType"
                    hasValue "kernel"
91              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#rootDeviceType"
                    hasValue "instance-store"
92              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#
                    virtualizationType" hasValue "paravirtual"
93              _"http://www.CloudsLab.org/ontologies/VirtualAppliance#hypervisor"
                    hasValue "xen"
94              hasProvide hasValue {AmazonVirginia}
95              hasname hasValue    "aki00896a69"
96               hasFormat hasValue AMI
97 concept virtualUnit subConceptOf cloudService
98 hasName ofType _string
99 hasProvider ofType cloud
100
101 instance largeInstance memberOf virtualUnit
102 hasName hasValue "largeInstance"
103 hasProvider hasValue AmazonCalifornia
104
105
106 concept state subConceptOf place
107 concept country subConceptOf place
108
109 instance USA memberOf country
110 hasName hasValue "USA"
111 instance Singapour memberOf country
112 hasName hasValue "Singapour"
113 instance Canada memberOf country
114 hasName hasValue "Canada"
115 instance Ireland memberOf country
116 hasName hasValue "Ireland"
117 instance Brazil memberOf country
118 hasName hasValue "Brazil"
119 instance Japan memberOf country
120 hasName hasValue "Japan"
121 instance China memberOf country
122 hasName hasValue "China"
```

```
instance Australia memberOf country
hasName hasValue "Australia"
instance England memberOf country
hasName hasValue "England"

instance Toronto memberOf state
hasName hasValue "Toronto"
instance Tokyo memberOf state
hasName hasValue "Tokyo"
instance Dublin memberOf state
hasName hasValue "Dublin"
instance Virginia memberOf state
hasName hasValue  "Virginia"
instance Oregon memberOf state
hasName hasValue Oregon
instance California memberOf state
hasName hasValue "California"
instance Hongkong memberOf state
hasName hasValue "Hongkong"
instance Victoria memberOf state
hasName hasValue "Victoria"
instance London memberOf state
hasName hasValue "London"
instance Singapour memberOf state
hasName hasValue "Singapour"

concept cloud
     hasName ofType _string
supportVmFormat ofType VMFormat
hasCountry ofType country
hasState ofType state

instance AMI memberOf VMFormat
hasName hasValue "AMI"

instance OVF memberOf VMFormat
hasName hasValue "OVF"

instance GSI memberOf VMFormat
hasName hasValue "GSI"

instance VMDK memberOf VMFormat
hasName hasValue "VMDK"

instance TerremarkCanada memberOf cloud
hasName hasValue "TerremarkCanada"
supportVmFormat hasValue {OVF, VMDK }
hasCountry hasValue Canada
hasState hasValue Toronto

instance TerremarkEngland memberOf cloud
hasName hasValue "TerremarkEngland"
supportVmFormat hasValue {OVF, VMDK }
hasCountry hasValue England
hasState hasValue London

instance TerremarkChina memberOf cloud
hasName hasValue "TerremarkChina"
supportVmFormat hasValue {OVF, VMDK }
hasCountry hasValue China
```

```
hasState hasValue Hongkong

instance TerremarkAuastralia memberOf cloud
hasName hasValue "TerremarkAustralia"
supportVmFormat hasValue {OVF, VMDK }
hasCountry hasValue Australia
hasState hasValue Victoria

instance AmazonVirginia memberOf cloud
hasName hasValue "AmazonVirginia"
supportVmFormat hasValue {AMI}
hasCountry hasValue USA
hasState hasValue Virginia

instance AmazonSingapour memberOf cloud
hasName hasValue "AmazonSingapour"
supportVmFormat hasValue {AMI}
hasCountry hasValue Singapour
hasState hasValue Singapour

instance AmazonIreland memberOf cloud
hasName hasValue "AmazonIreland"
supportVmFormat hasValue {AMI}
hasCountry hasValue Ireland
hasState hasValue Dublin

instance AmazonCalifornia memberOf cloud
hasName hasValue "AmazonCalifornia"
supportVmFormat hasValue {AMI}
hasCountry hasValue USA
hasState hasValue California

instance AmazonJapan memberOf cloud
hasName hasValue "AmazonJapan"
supportVmFormat hasValue {AMI}
hasCountry hasValue Japan
hasState hasValue Tokyo

relation compatible(ofType virtualAppliance, ofType virtualUnit )



 axiom compatbilewith
     definedBy

?x memberOf virtualAppliance and  ?x[hasProvider hasValue ?p]  and ?y memberOf
    vitrtualUnit and ?y[hasProvider hasValue ?pvu] and
?p[hasCountry hasValue ?capp] and ?pvu [hasCountry hasValue ?cvu] and ?capp[hasName
    hasValue ?cappName] and ?cvu[hasName hasValue ?cvuName] and
stringEqual(?cvuName,?cappName)
implies ?x[isCompatibleWith hasValue ?y].
```

A.2 Deployment Descriptor

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace { _"http://www.cloudslab.org/Deployment#"}
ontology Deployment
 importsOntology  _"http://www.cloudslab.org/CloudProvider#CloudProviderOntology"
```

```
5
6  concept ScalingPolicy
7  hasName ofType _string
8  hasUpperBoundThreshold ofType _integer
9  hasLowerBoundThreshold ofType _integer
10 hasPeriod ofType _integer
11 hasMetric ofType _"http://www.cloudslab.org/CloudProvider#monitoringMetric"
12
13 concept ScalingGroup
14 hasName ofType _string
15 hasVirtualUnit ofType _"http://www.cloudslab.org/CloudProvider#virtualUnit"
16 hasVirtualAppliance ofType _"http://www.cloudslab.org/CloudProvider#virtualAppliance"
17 hasMinSize ofType _integer
18 hasMaxSize ofType _integer
19 hasLocation ofType _"http://www.cloudslab.org/CloudProvider#location"
20 hasLoadBalancer ofType _"http://www.cloudslab.org/CloudProvider#loadBalancer"
21 hasPolicy ofType ScalingPolicy
22 hasSecurityGroup ofType SecurityGroup
23
24 concept SecurityGroupRule
25 hasName ofType _string
26 hasProtocol ofType  _string
27 hasPort ofType  _integer
28 HasSourceIP ofType _string
29 hasAuthorizedSecurityGroup ofType SecurityGroup
30 isAuthorizingSecurityGroup ofType _boolean
31
32 concept SecurityGroup
33 hasName ofType _string
34 hasLocation ofType _"http://www.cloudslab.org/CloudProvider#location"
35 hasRules ofType SecurityGroupRule
36
37 concept DeploymentDescriptor
38 hasScalingGroup ofType ScalingGroup
39 hasSecurityGroup ofType SecurityGroup
40
41 instance WSRule1 memberOf SecurityGroupRule
42 hasName hasValue "WSRule1"
43 hasProtocol hasValue  "TCP"
44 hasPort hasValue  80
45 HasSourceIP hasValue "0.0.0.0/0"
46 isAuthorizingSecurityGroup hasValue false
47
48 instance WSSecurityGroup memberOf SecurityGroup
49 hasName hasValue WSSecurityGroup
50 hasLocation hasValue _"http://www.cloudslab.org/CloudProvider#California"
51 hasRules hasValue WSRule1
52
53
54 instance WSLoadBalancer memberOf _"http://www.cloudslab.org/CloudProvider#
       loadBalancer"
55 hasName hasValue "WShasLoadBalancer"
56
57 instance WSScalingPolicy memberOf ScalingPolicy
58 hasName hasValue "WSScalingPolicy"
59 hasUpperBoundThreshold hasValue 80
60 hasLowerBoundThreshold hasValue 10
61 hasPeriod hasValue 600
62 hasMetric hasValue _"http://www.cloudslab.org/CloudProvider#CPUUtilization"
63
```

```
64 instance webServerScalingGroup memberOf ScalingGroup
65 hasName hasValue "webServerScalingGroup"
66 hasVirtualUnit hasValue _"http://www.cloudslab.org/CloudProvider#largeInstance"
67 hasVirtualAppliance hasValue _"http://www.CloudsLab.org/ontologies/VirtualAppliance#
       aki-00806369"
68 hasMinSize hasValue 1
69 hasMaxSize hasValue 3
70 hasLocation hasValue _"http://www.cloudslab.org/CloudProvider#California"
71 hasLoadBalancer hasValue  WSLoadBalancer
72 hasPolicy hasValue WSScalingPolicy
73 hasSecurityGroup hasValue  WSSecurityGroup
74
75 instance DDsample memberOf DeploymentDescriptor
76 hasScalingGroup hasValue webServerScalingGroup
77 hasSecurityGroup hasValue WSSecurityGroup
```