ADAPTIVE RESOURCE MANAGEMENT IN SLA AWARE ELASTIC CLOUDS

Thesis

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

ANITHAKUMARI S

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

SURATHKAL, MANGALORE - 575025

APRIL, 2019

To my family

DECLARATION

By the Ph.D. Research Scholar

I hereby *declare* that the Research Thesis entitled **ADAPTIVE RESOURCE MANAGEMENT IN SLA AWARE ELASTIC CLOUDS** which is being submitted to the **National Institute of Technology Karnataka, Surathkal** in partial fulfillment of the requirements for the award of the Degree of **Doctor of Philosophy** in **Computer Science and Engineering** is a *bonafide report of the research work carried out by me*. The material contained in this Research Thesis has not been submitted to any University or Institution for the award of any degree.

ANITHAKUMARI S

Reg. No. 121174 CS12F01

Department of Computer Science and Engineering

Place: NITK, Surathkal.

Date: April 19, 2019

CERTIFICATE

This is to *certify* that the Research Thesis entitled **ADAPTIVE RESOURCE MANAGEMENT IN SLA AWARE ELASTIC CLOUDS** submitted by **ANITHAKUMARI S.**, (Reg. No. 121174 CS12F01) as the record of the research work carried out by her, is *accepted as the Research Thesis submission* in partial fulfillment of the requirements for the award of degree of **Doctor of Philosophy**.

(K. Chandrasekaran)
Research Supervisor

Chairman - DRPC
(Signature with Date and Seal)

ACKNOWLEDGEMENT

I would like to take this opportunity to thank those people who have made this thesis possible.

My sincere gratitude to my Research Guide Prof. K. Chandrasekaran for accepting me into the Ph.D. program and for keeping constant faith in me. He is always supportive and remarkably perceptive in advising in the right direction and correcting the vital details. Without his continuous encouragement, I would still be in my Ph.D. dreams. He is a great mentor and I learned so much from him through his long academic journey. I could not have finished my thesis in time without his untiring help in making every sentence concise and correct. I am truly blessed to have come to National Institute of Technology Karnataka, where Prof. K. Chandrasekaran welcomed me with open arms and my life started on a whole new course.

I extend my sincere thanks to Prof. Santhi Thilagam, Department of CSE for her feedback and comments as my Research Progress Assessment Committee (RPAC) member. Special thanks are due to my second RPAC member Prof. K. Vidya Shetty, Department of Chemical Engineering for her constant advise in improvising my work.

I extend my thanks to the Head of the Department, Department of Computer Science and Engineering, for encouraging research related activities in the department and cultivating an open and free research environment. I would like to extend my sincere thanks to the entire faculty and staff of Computer Science department for their untiring support in successful completion of my research work.

My debt to my family is immeasurable. They have offered me their unconditional support in all of my endeavors. Special thanks to my employer LBS Centre for Science and Technology, my colleagues and my friends in NITK for their direct and indirect help and support.

Place: NITK, Surathkal

Date: April 19, 2019 (Anithakumari S)

ABSTRACT

In recent years, there has been an increasing interest in solving the over-provisioning and under-provisioning of elastic cloud resources because of the Service Level Agreement (SLA) violation problem. The recent studies have reported that federated cloud services may serve as a better elastic cloud model over a single provider model. A major problem with the federated cloud is the interoperability between multiple cloud service providers. Therefore in this thesis, a proactive SLA aware adaptive resource management approach is proposed for elastic cloud services. Aim of this thesis is to develop a suitable SLA monitoring framework to predict the SLA violations and adaptively allocate the cloud resources to improve the elasticity. It achieves the mutual benefits for cloud consumers and service providers by means of calculating and reducing penalty cost. Our framework has been implemented and validated on a private cloud using OpenNebula 4.0. The results have shown that the proposed proactive approach has significantly reduced the SLA violations compared to a reactive approach. As an additional contribution, the presented work solves the interoperability issues of the federated cloud using an innovative SLA matching algorithm. The simulation results of this work show that the said approach performs better than its counterparts.

Keywords: Cloud Computing, Service Level Agreement (SLA), SLA negotiation, SLA monitoring, SLA violation, Flexible resource allocation, Interoperability, Inter-Cloud Gateway (ICG).

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

ADS	Adaptive Dimensional Search
API	Application Program Interface
AWRT	Average Weighted Response Time
CBR	Case-Based Reasoning
CIM	Common Information Model
DDM	Dynamic Demand Model
DEFF	Dynamic Evaluation Framework for Fairness
DeSVi	Detecting SLA Violation Infrastructure
DNM	Dynamic Node Model
GUI	Graphical User Interface
HMC	Heterogeneous Mobile Cloud Computing
IaaS	Infrastructure as a Service
JESS	Java Expert System Shell
JVM	Java Virtual Machine
KPI	Key Performance Indicators
KVM	Kernel-based Virtual Machine
LoM2HiS	Low level Metrics to High level SLAs
NS	Negotiation Service
PaaS	Platform as a Service
QoS	Quality of Service
QU4DS	Quality Assurance for Distributed Systems
SaaS	Software as a Service

SDR	Search Dimensionality Ratio
SLA	Service Level Agreements
SLO	Service Level Objective
SOA	Service Oriented Architecture
SVR	Support Vector Regression
TFIDF	Term Frequency Inverse Document Frequency
TNV	Total Negotiation Value
VM	Virtual Machine
WSAG4J	Web Service Agreement Framework for Java
WSLA	Web Service Level Agreement
WSRF	Web Services Resources Framework
XML	eXtensible Markup Language

Chapter 1

INTRODUCTION

Cloud computing provides convenient and on-demand network access to a shared pool of configurable computing resources. The benefits of cloud computing are cost effectiveness, scalability and ease of management. Cloud computing provides the computing utilities to the consumers in a pay-as-you-go manner. On the other hand, Service Level Agreement (SLA) and adaptive resource allocation plays major role to provide cloud services that are beneficial to both service provider and consumer.

Resource Allocation in cloud computing plays a major role in providing cloud services to consumers. The cloud resource allocation becomes more complex when the demand from the consumers changes dynamically. To meet the dynamically changing requirements, there is a need to have a method which will allocate the resources proactively. However, most of the existing approaches are reactive in nature and they allocate the resources only when an SLA violation has occurred.

One of the essential characteristics of cloud computing is elasticity (Coutinho et al., 2015). *Rapid elasticity is where computing resources are provisioned elastically to scale rapidly based on the user's requirement*. Cloud computing becomes more powerful if we can add dynamic elasticity as a feature. Dynamic elasticity (Galante and Bona, 2012) in the context of the cloud is considered as the ability of the cloud to be elastic while the services are online. In the literature, elasticity and scalability are treated with same meaning which creates confusion in design and development. In Herbst et al. (2013), the authors presented a new advanced definition of elasticity by considering its core aspects in a clear and unambiguous manner and thus differentiated elasticity from scalability.

Scalability is a prerequisite for elasticity, but it does not consider temporal aspects of how fast, how often, and at what granularity the scaling actions should be performed. It is the ability of the system to sustain increasing workloads by making use of additional resources, and therefore, in contrast to elasticity, it is not directly related to how well the provisioned resources match the actual resource demands at any point in time.

An SLA is the set of agreements settled among the cloud service consumers and providers. Typically an SLA includes a collection of Service Level Objectives (SLOs) which define Quality of Service (QoS) properties for the agreed upon service. The use of newly improved mechanisms for negotiating and monitoring service level agreements is highly essential because of the dynamic change in service requirements. Online monitoring of SLAs (Buyya et al., 2011) is advantageous to both the involved parties because it detects the possibility of violations in SLA and initiates some actions to correct or compensate. *Therefore, SLA monitoring and negotiation is considered to be important*.

Guaranteed SLA of dynamic clouds mainly focuses on elasticity which aims to meet QoS requirements such as availability and performance while minimizing cloud cost. In Bouchenak (2010), the author described the necessities of an SLA aware elastic cloud.

These requirements are:

- Monitoring and observation of the cloud in an online manner: This is to capture variations periodically in cloud usage and workload to determine violations in SLA and to generate cloud reconfiguration when necessary. QoS measures can be used at various levels to offer low-level metrics for Infrastructure as a Service (IaaS) clouds or higher level metrics for Software as a Service (SaaS) clouds.

Defining a scalable, accurate and non-intrusive distributed algorithm for monitoring cloud services is the crucial issue in this context.

- *Modeling the cloud: Fluctuating and non-linear workloads affect the vibrant performance of cloud computing and this influence the QoS of the service*. A cloud is also classified by its actual configuration (i.e., the number of cloud services, the location of machines hosting cloud services and service parameters of individual clouds) which influences both cloud QoS and cost. Cloud modeling aims to concentrate the effect of cloud workload and configuration on QoS and the cost of the cloud.

The main task to accomplish this is the definition of a model that is precise, capable of accommodating the variation of cloud workload and is easy to use with real-world applications.

- Automated cloud control: Automated control of the cloud targets to build a dynamic elastic cloud that meets QoS requirements as mentioned in the SLA while minimizing cloud cost. The use of a cloud model permits to target the variations of cloud configuration and workload and the corresponding effect on QoS and cost.

In this research work, we aim at satisfying the above mentioned requirements of SLA aware elastic cloud. To satisfy these requirements, suitable SLA management framework (Chapter 3) and Modeling of cloud infrastructure with adaptive control (Chapter 4) are proposed and presented with the suitable experiments.

1.1 Gaps Identified in Existing Literature

The identified research gaps are:

- The lack of dynamic adaptation strategies at run-time to provide elastic services.
- The lack of online SLA monitoring framework for considering dynamically changing SLA parameters.
- The lack of suitable solutions to avoid resource over provisioning to escape from SLA violations.
- The lack of appropriate strategies to consider the adaptive resource allocation across multiple cloud providers where interoperability is a major issue.

To efficiently tackle these research gaps, SLA-based solutions are proposed and design, implementation and verification of the solutions are presented in this thesis.

1.2 Problem Statement and Objectives

Cloud providers should ensure the QoS requirements using a minimal amount of computational resources. SLA management helps customers to validate and supervise the quality of services through scheduled contracts and agreements.

To address the above limitations, the proposed problem solution is to Design and develop an adaptable resource management model for cloud computing environment that is SLA aware.

The aim of this research work is to develop a framework for monitoring and analyzing SLA parameters to determine SLA violations and to develop an adaptive resource allocation system for reducing SLA violations.

Research Objectives

1. To build a flexible, reliable and dynamic SLA management system for monitoring and detecting SLA violations in cloud computing by incorporating some reactive actions to prevent SLA violation by adding autonomic adaptation of SLA parameters and Service Level Objectives.

2. To develop and implement an autonomic resource allocation system for reallocating resources dynamically during execution of services according to variation in workloads and detected SLA violations.

3. To design and develop a framework for performing autonomic SLA matching and clustering according to applications using case based reasoning.

4. To develop and implement inter-dependency and interoperability by providing dynamic elasticity in the context of multiple competitive/cooperative cloud providers in an autonomic manner.

1.3 Research Methodology

A quantitative research methodology has been followed to analyze the results obtained during this research work. In the initial stage, a critical review is conducted on the related research works. As shown in Figure 1.1, the research methodology consists of five phases which are described as follows:

Phase 1: This consists of critical review of all the approaches coming under the purview of the project with a gap analysis at the end. A critical review has been done according to specific areas of research and after analysing each of the individual papers the critical review is written. This phase is required, to know about the existing research

works present in this area and further carry out an accurate gap analysis which facilitates proceeding in the correct direction.

Phase 2: This involves design and development of a model for predicting SLA violation in cloud. The proposed method has been quantitatively analysed by creating a simulation environment using Cloudsim and by additionally using Web Service Agreement for Java (WSAG4J) for verifying the proposed method.

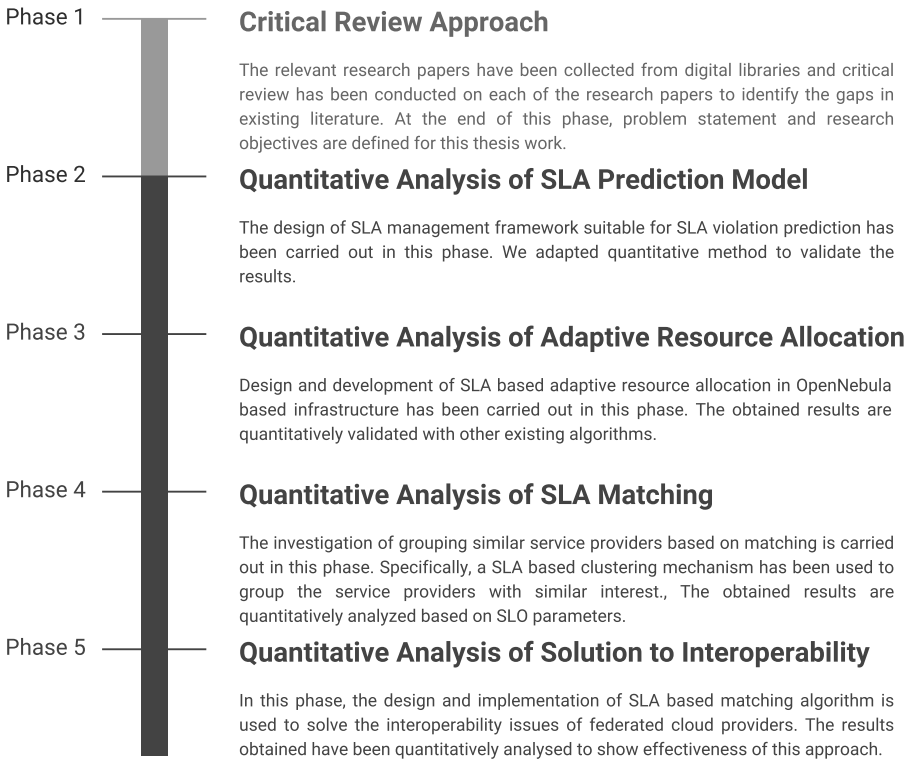

Figure 1.1 Schematic diagram of the research methodology

Phase 3: This involves design and development of an adaptive resource allocation mechanism in a realtime cloud environment, OpenNebula. The results have been quantitatively analysed and the obtained results are compared with existing algorithm for its efficiency in terms of resource allocation. This model is based on the SLA monitoring mechanism developed in the phase 2, over which the designed framework works. The current and previous phases both collectively are designed for a single cloud provider and after this the research proceeds towards a federated cloud model.

Phase 4: This involves design and development of an clustering mechanism with SLOs as parameters. The work is carried out based on the results obtained in simulation environment of Cloudsim and the clustering results obtained are quantitatively analysed based on the SLO parameters. This phase is aimed specifically at federated cloud environment. The output of this phase consist of clustered service providers, according to their SLOs.

Phase 5: A federated cloud environment is simulated based on the SLA aware clusters developed in the previous phase. This again uses cloudsim and involves quantitative analysis which includes response time as its important parameter. This phase primarily involves studying the efficiency of the proposed SLA based mechanism in a federated environment and comparing it with other algorithms performing the same task.

The phases proposed here ensure that the main goal and objectives of the work are achieved without any issue. All the objectives are covered and their inter-dependencies are well established which ensure a proper flow for achieving the desired result.

1.4 Thesis Contributions

This thesis provides a SLA management framework to avoid SLA violation before it occurs. The proposed SLA management approach is used for adaptive resource allocation and to solve the interoperability issues. The key contributions are as follows:

- **SLA Management Framework**: A SLA negotiation and monitoring mechanism that is suitable for predicting the violation before it occurs.
- **Adaptive Resource Allocation**:A SLA based adaptive resource allocation mechanism designed and further tested on the OpenNebula cloud.
- **SLA Matching**: A SLA matching based solution towards solving the problem of interoperability issues. The investigation of grouping similar service providers with the help of matching SLA documents.
- **SLA based Interoperability**: A design and implementation of proposed SLA matching mechanism for interoperable federated cloud. The federated cloud is formed using CloudSim and the proposed approach is verified for its effectiveness on this simulated cloud infrastructure.

1.5 Thesis Structure

The rest of the thesis is organized as follows:

- Chapter 2 presents an overview of recent work in the areas of SLA management, adaptive resource allocation and interoperable federated cloud.
- Chapter 3 presents the proposed SLA management framework for continuously monitoring SLA parameters for detecting SLA violation and avoidance. The design, implementation, experimental environment and results that corresponds to objective 1 are presented in this chapter.
- Chapter 4 describes the proposed adaptive resource allocation in cloud computing. This chapter gives an idea about the generic structure of cloud provider's data center used for the proposed approach. Further, the experimental environment and results that corresponds to Objective 2 are presented in this chapter.
- Chapter 5 explains the techniques used for SLA matching and clustering of multiple cloud providers. The conceptual idea and experimental results specific to Objective 3 are presented in this chapter.
- Chapter 6 introduces the implementation of Inter-Cloud infrastructure with an SLA based clustering approach. The interdependency and interoperability of cloud resources is enhanced using a flexible resource allocation algorithm materialized with an adaptive dimensional search. The CloudSim based simulation and results that corresponds to Objective 4 are also presented in this chapter.
- Chapter 7 presents the thesis summary and conclusions of the research work. This chapter also highlights some of possible future research works.

Chapter 2

LITERATURE REVIEW

This chapter presents an overview of cloud computing along with the importance of SLA and SLA based elastic cloud services. The chapter subsequently presents the review of existing literature on SLA management approaches, resource allocation approaches and federated cloud approaches. The research gaps that were identified as the result of literature review are finally presented.

2.1 Background Information

This section outlines the overview of cloud computing, role of SLA in cloud computing, and importance of considering SLA based elastic cloud services.

Overview of Cloud Computing

According to Mell and Grance (2011) (NIST definition), "*cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction*". Cloud services are established based on five essential characteristics: on-demand self-service, broad network access, resource pooling, rapid elasticity or expansion, and measured service (Mell and Grance, 2011). The first characteristic is the *on demand self-service,* which enables consumers the provisioning of computing power, storage, networks, and software flexible and straightforward. Second is *broad network access* in which cloud service consumers can access available computing resources over

the network. The third is *resource pooling* in which computing resources are served as a shared pool and the customers can access resources from this shared pool. Fourth is *rapid elasticity* where computing resources are provisioned elastically to scale rapidly based on the user requirements and the last one is *measured service* where computing resources' usage is monitored, measured and reported to provide transparency for both cloud service providers and consumers.

Cloud computing model is gaining much attention from both industry and academia in the area of large-scale distributed computing because users can rent virtual machines and storage space instead of buying and owning hardware. In a pure perspective, cloud computing seems similar to computing paradigms like grid computing, and cluster computing, but the fundamental differences of cloud computing are its technical characteristics such as on-demand resource pooling, rapid elasticity, self-service, almost infinite scalability, end-to-end virtualization support, and robust support of resource usage metering and billing. Cloud computing offers following services to consumers (Mell and Grance, 2011):

- Infrastructure-as-a-Service (IaaS) cloud facilitates access to hardware resources (servers and storage devices)
- Platform-as-a-Service (PaaS) cloud facilitates the access to software resources (operating system and software developing environment)
- Software-as-a-Service (SaaS) cloud is a substitute for classical software applications running locally on personal computers.

Role of Service Level Agreements (SLAs)

An SLA is an agreement regarding the promises and assurances of the quality and quantity of the services offered and is established between a service consumer and a service provider (Wu and Buyya, 2012). SLA elaborates general considerations and expectations of delivered service and contain components like **involved participants, activation-time, scope, SLOs, approaches to assess the SLOs, penalty and exclusions** (Dan et al., 2004). SLOs represent the goals of the service provider and assure the promise of the service provider to preserve a predefined level of the service for a predefined time period. Some sample SLOs are the availability of the service, system response time, resolution time for

a service outage, etc. Cloud service providers utilize the concept of Key Performance Indicators (KPIs) to specify the quality of the provided services. These KPIs keeps varying according to cloud providers. The work of Burkon (2013) consolidated quality indicators for an SaaS model and these indicators are Response time, Availability, Throughput, Timeliness, Reliability, Scalability, Security, Uptime History, Granularity, Elasticity, Interoperability, Usability, and Testability.

An SLA serve as the foundation for the expected level of service by mentioning the QoS parameters of the service. In a general perspective, SLAs are used to ensure customers a certain level of quality for the services, and if this level is not achieved, the provider has to give penalties for the violation of the contract (Wu and Buyya, 2012). Generally the SLA contract :

- Specifies the required parameters and possible values for each service elements,
- Confirms the ownership of the data stored on the service provider's system and specifies the rights associated,
- Presents the infrastructure details and security standards to be maintained by the service provider, along with the rights of the consumer and
- Mention the rights and cost to continue or break the usage of the service.

In its basic form, SLA is the set of agreements settled among the cloud service consumers and providers. Typically an SLA includes a collection of SLOs which define QoS properties for the agreed upon service. The usage of newly improved mechanisms for negotiating and monitoring service level agreements are highly essential because of the dynamic change in service requirements. Online monitoring of SLAs (Buyya et al., 2011) is advantageous to both the involved parties because it detects the possibility of violations in SLA and initiates some actions to correct or compensate. *Therefore, SLA monitoring and negotiation is considered as one of the important objective of this research work.*

Elastic Cloud Services

One of the important enabling technology of our research work is elastic cloud services. The other important technical aspects are SLA management, resource allocation and fed-

erated cloud environment. According to Shawky and Ali (2012), elasticity of a cloud computing system is the ability to increase and decrease overtime in response to users' demands. This definition of elasticity represents the basic features of cloud computing and the constraints associated to elasticity. Here they tried to measure the elasticity of a cloud service using representative measures. The elasticity issues of business applications to be deployed and operated on public cloud infrastructure is the major challenge while considering profits in cloud computing. According to Suleiman et al. (2012), different aspects to be considered are modelling and measuring of economics and elasticity, application workload patterns and its impact on achieving elasticity and economics, economics-driven elasticity decisions and policies, and SLA-driven monitoring and elasticity of cloud-based business applications. The elasticity decision policies really help cloud consumers and users to analyze and evaluate elasticity capabilities of various cloud infrastructures and its suitability for meeting their business application requirements.

In Martin et al. (2011), the authors introduced a service management framework to support elastic services. Elastic services are offered in the cloud infrastructure, by employing management of virtual infrastructure to dynamically orchestrate the deployment of virtual machines, management of storage requirements, and organize resources according to the changing needs of the organization. Elastic services generate extra management challenges such as, (i) the involved components, resources and services must be capable of automatically adjusting to the variations in the underlying set of managed resources and (ii) adding or removing resources, such as a virtual machine, to a service can involve significant delays. It is also harder for the service management system to guarantee predictable performance and to support problem determination since the underlying resource configuration changes.

SLA Aware Elastic Clouds

The elastic clouds guarantees to meet the QoS requirements such as availability and performance while minimizing cloud cost. In order to satisfy the QoS requirements, it is essential to satisfy a minimum service level to customers. The usage of computing resources in clouds is efficient only if it satisfies the dynamically changing needs of the customer in a cost effective way and in line with the conditions specified in SLAs. The authors of Wu

et al. (2011) describe resource allocation algorithms for cloud providers by considering infrastructure cost and additional cost due to SLA violations. These algorithms are designed in such a way that the cloud providers can manage the dynamic change of customers and can map the customer requests to infrastructure level parameters. This algorithm tries to reduce SLA violations and increase QoS parameters. Adding SLA awareness to the elastic cloud services is advantageous to both cloud service consumers and service providers as it helps in reducing the number of SLA violations.

In Bouchenak (2010), the authors have described the requirements of an SLA aware elastic cloud. These requirements are: Monitoring and observation of the cloud in an on-line manner, modeling the cloud, and automated cloud control. The first requirement is to monitor the SLA violations periodically and generate necessary re-configurations to avoid further SLA violations where designing a scalable algorithm to monitor the cloud services is crucial issue to meet this requirement. The second requirement is to model the cloud environment that can manage fluctuating and non-linear workloads with minimum SLA violations. As the type of workload affects the cloud system in terms of meeting QoS requirements, it is important to model the cloud precisely which may be easily paired with real-world cloud systems. The third requirement is to monitor SLAs and to perform necessary re-configurations as automated tasks. Here, the objective of the automated controlling mechanism is to reduce the cloud cost while meeting the maximum number of QoS requirements mentioned in SLA. This research work considers the above mentioned requirements for an SLA aware elastic cloud and these requirements have been addressed by a suitable SLA management framework and by an adaptive resource allocation mechanism.

The remaining part of this chapter presents the summary of SLA management approaches, adaptive resource allocation approaches and cloud federation issues.

2.2 Survey of SLA Management Approaches

A key challenge for service providers is the management of SLAs with their customers to satisfy their business intentions. Majority of the current systems fail to consider business intentions and hence fail to deliver a complete SLA management solution. This section briefly summarizes the research works in the area of SLA management in cloud and service

computing.

In Freitas et al. (2011), a generic framework Quality Assurance for Distributed Services (Qu4DS) is presented to support service providers in honoring SLAs while minimizing the costs of infrastructure utilization. This framework incorporates a collection of techniques for QoS management. In addition, it supports the whole SLA life-cycle from creating SLA template to terminating the cloud service. The framework is built on an interface that is compatible with the latest grid and cloud Application Program Interface (API) to manage the infrastructure usage. The changing service loads and random failures are addressed in the framework by including provision for self-adaptation using multiple interacting control loops. The primary objective of Qu4DS is to offer SLA self-management for reducing the cost of service. Cost includes, payments required for of infrastructure resources and the payment of fines due to SLA violations. Based on the cloud architecture layers, Qu4DS seats in the PaaS layer by using resources from an IaaS provider to deliver support to the SaaS layer. The authors failed to address the following key points in this work

- Over provisioning of the requested resources: Cloud providers try to provide an excess number of requested resources to escape from SLA Violations
- Dynamic adaptation strategies for varying workload and infrastructure conditions are not considered.
- Not addressed the autonomic selection/initiation of adaptation strategies

Under SLA management, there are several work concentrated more on SLA negotiation. The important research works on SLA negotiation and the design of machine-readable SLAs for SOA-based computing and cloud computing are summarized below. In Dan et al. (2004), a framework is proposed that is suitable for SOA environment based on SLAs and automatic management. This framework is capable of providing segregated service levels to customers. This framework includes WSLA for the design and negotiation of SLAs, a system based on SLOs for dynamic resource allocation, a method to manage workload which orders requirements dependable with SLAs, and a monitoring system to monitor compliance with the SLA. Chieng et al. (2005) implemented an architecture for service provisioning in an SLA-driven manner which permits flexible and reliable SLA

negotiation of services. The importance is on bandwidth reservation which contributes more to affect QoS. Here, the negotiation is done with high-level service parameters like price, starting time, session length, and guaranteed bandwidth. Di Modica et al. (2009a) presented a protocol based approach for the specification of SLAs which gives provision to renegotiate and alter its terms during the provisioning of the service. Here the renegotiation approach is implemented to reduce the SLA violations which will result in a compromise on the quality of the provided service. This WS-Agreement protocol does not allow the run-time renegotiation but it enables to introduce some renegotiation sessions of between the involved parties.

Silaghi et al. (2012) developed a framework for framing automatic SLA negotiation policies by considering time constraints for the use of computational grids. The framework was built using agent systems and is based on learning the opponent's strategies. The authors proposed the Bayesian learning agent to contract with the build-up period of a bargaining session and proved that opponent learning strategies result in better satisfaction of contracting parties and optimal resource allocation. Resinas et al. (2012) proposed architecture for constructing automated systems to negotiate service agreements using protocol based bargaining approach to work with global environments. This architecture is ready to address many requirements like multi-term negotiation, heterogeneity of the contracting parties, managing incomplete information about the contracting parties, and concurrent negotiations with multiple parties.

Hasselmeyer et al. (2006) presented an approach using agent brokers for SLA negotiation in which the third party agents deal with negotiation on behalf of the participating parties. Theilmann et al. (2010) implemented a reference architecture for multi-level SLA management in the context of the SLA@SOI project. This architecture aims at a standard solution for SLA management that can i) deal with the entire SLA and service life-cycle; ii) support several layers of control in a service-oriented structure, and iii) applicable to various use cases and business domains.

In Dastjerdi and Buyya (2012), the authors have discussed some challenges regarding negotiation of SLAs in cloud computing environments and have described a time-dependent model of negotiation to deal such challenges. Here, the authors have discussed

the challenging issues of SLA negotiation in cloud computing as *Offers reliability* and *Balancing resource utilization*. *Offers reliability* means a mechanism to find out the reliability of offers/counter offers regarding promised QoS values and SLA violations.

Balancing resource utilization implies the use of resources in a balanced and efficient manner. The authors have introduced a sequence of steps for the negotiation of SLA in cloud computing. A service requester identifies the needed requirements such as hardware specifications like CPU, storage, and memory. Also, the requester provides preferences on the QoS and functional requirements as input for the discovery of suitable cloud services. Then the negotiation is started by the Negotiation Service (NS) with discovered service providers on QoS criteria, like availability and price, as per the preferences of requesters.

Client's budget and the deadline for getting cloud resources are used by the client NS to decide on acceptance or rejection of an offer. Client NS uses a time-controlled tactic which reads client's preferences as input and generates offers automatically. Once the Cloud NS receives the offer, it uses functional QoS requirements and cloud resources utilization supplied by the monitoring system to create counter offers. This negotiation strategy failed to consider the benefits of both the involved parties.

In Sahai et al. (2002), the authors have introduced some base components for standard SLA specification which identifies SLA in a definite and accurate manner. In Keller and Ludwig (2002) the authors described a WSLA framework suitable for defining and monitoring SLAs in inter-domain fields. This framework provides a flexible and extensible language based on the eXtensible Markup Language (XML) schema and a runtime architecture for defining and monitoring SLAs in dynamic e-business. This frame work enables service providers and customers to define a variety of SLAs unambiguously but the dynamic nature in resource instrumentation is not taken care. An approach for SLA driven management ideal for distributed systems is introduced in Debusmann and Keller (2003) using Common Information Model (CIM). The implementation of a common information model capable of operating among multiple service providers is not that much successful in this work.

In Barbosa et al. (2006), the authors tried to evaluate multiple architectures which perform SLA auditing by considering the qualitative and quantitative aspects. In He et al.

(2009), an SLA management framework is presented using agent-based systems, in which a service customer (an initiator agent) and a service provider (a responder agent) are there for initiating the process of SLA negotiation. The service provider markets its capabilities in service level, and the service consumer gets the marketed information for initializing the process of SLA negotiation. The responder publicizes the competences of the service, and the initiator receives these announcements before initializing the negotiation process.

Intermittent phases of SLA life cycle such as formation, enforcement, and recovery is achieved through the intelligent communications among the agent modules. The initiator agent also claims for compensation at the time of an SLA violation and think of renegotiation or selection of new service provider. The claim of compensation for an SLA violation is also focused in the work of Rana et al. (2008), where the authors proposed a mechanism of compensation like the regulations related to cases of conflict and the influence of the penalty clauses on the choice of SLOs.

In addition to previously discussed SLA negotiation approaches, there are several research articles in literature specifically discussing on proactive and dynamic adaptation methods of service-based operations. The important proactive and adaptive SLA management approaches are summarized here. In Mahbub and Spanoudakis (2011), the authors have presented a framework for integrating service discovery with a proactive approach of SLA negotiation, named as PROSDIN. The identification of services in PROSDIN is purely based on several properties of services such as, behavioral, structural and QoS features. PROSDIN also negotiates an SLA based on QoS values with the services identified alternatively during the discovery process by a rule-based approach using the Jess rule engine Shell (2016).

The works explained in Wieder et al. (2008) describe runtime SLA re-negotiation for managing SLA violations. In Di Modica et al. (2009b), SLOs are modified and renegotiated at run-time, and online services are adjusted to dynamically agreed SLOs. In Di Modica et al. (2007) the authors have suggested a re-negotiation of SLA to deal with SLA violations where SLOs are reviewed and renegotiated during runtime and the already positioned services are dynamically modified to go with the re-negotiated SLOs. Another approach presented in Sakellariou and Yarmolenko (2005) permits the modification

and change of SLOs along with the existing negotiated SLA. A renegotiation protocol is presented in Hasselmeyer et. al. (2008) which lets the customer or provider to start a renegotiation when a change occurs in the business needs of negotiated parties.

In Redl et al. (2012) the authors introduced a mechanism for assessing the cost of SLA matching and provider selection with the SLA mapping technique and discuss the methods for reducing the cost in Grid and Cloud marketplaces. They presented an approach for automatic discovery of semantically similar SLA elements and creation of SLA mappings that map the differences in their syntax specification. Machine learning and automatic reasoning methods are used to analyze the data. Automatic matching of SLA elements and generation of mappings facilitates automatic recommendations of trading partners with reduced cost of joining the market. The drawbacks identified in this research paper are:

- Provision to use multiple similarity metrics for SLA matching is not taken into consideration

- Application domain specific customer requirements are not considered for evaluating the properties of SLA elements.

Boniface et al. (2009) described the dynamic provisioning of services with the help of SLAs. Here they have discussed the provisioning of services according to negotiated SLAs and the monitoring of SLAs for reducing violations by assuming Grid environment as the computing environment. Koller and Schubert (2007) explained the management of QoS values in an autonomic way with the help of a proxy-like approach based on a WS-Agreement implementation. For advanced SLA management, BREIN applies SLA management to Grids, whereas our research work focusses SLA management in clouds. Dobson and Sanchez-Macian (2006) present a unified QoS ontology applicable to QoS-based web services selection, QoS monitoring, and QoS adjustment. However, they do not consider application provisioning and deployment strategies. Research works like QoS-MONaaS (Cicotti et al., 2013), and SLAMonADA (Muller et al., 2012) presented monitoring systems to detect the SLA violations, but no primary focus is given towards the reactive actions. These proposed platforms measured the QoS value at run-time to release a report about SLA validation. Kertesz et al. (2011) and Varalakshmi et al. (2011) pro-

posed a system to detect SLA violations by monitoring SLA for computing the penalty cost service provider.

Table 2.1 Summary of significant research works on SLA management

Publication	Framework	Principle	Contribution	Approach	Domain
Keller and Ludwig (2002)	SLA Compliance monitor	Dynamic allocation/deallocation of resources	SLA negotiaton and monitoring	Language based	Web service
Debusmann and Keller (2003)	CIM based monitoring framework	Common information model and interfaces	SLA monitoring	XML-based prototype	Web service
Chieng et al. (2005)	SLA driven service provisionoing architecture	SLA negotiation for provisioning network bandwidth	SLA negotiaton	Agent based	Network bandwidth
Sakellariou and Yarmolenko (2005)	NA	Guarentee terms are defined with functions rther than fixed values	SLA renegotiaton	XML-based language	Web service
Koller and Schubert (2007)	SLA mangement framework	Plug and play based SLA mangement	SLA management	Proxy based	Web service
Di Modica et al. (2009a)	NA	A run-time support to meet QoS levels	SLA renegotiaton	XML-based language	Web service
Di Modica et al. (2009b),	SLA management framework	Dynamic renegotiation of agreeements	SLA renegotiaton	XML-based language	Web service
Maurer et al. (2011)	FOSII framework with MAPE-K	CBR approach based on rule base	SLA renegotiaton	Case based reasoning approach based on rule base	Cloud service
Mahbub and Spanoudakis (2011),	PROSDIN	Dynamic service discovery to provide runtime support	SLA negotiaton	Rule based	Web service
Freitas et al. (2011)	Q U 4DS	Multiple configurable control loops and automatic service configurations	SLA management	Framework and interfaces	Cloud service
Emeakaroha et al. (2012)	DeSVi architecture	Knowledge databases to manage and prevent SLA violations	SLA monitoring	VM based	Cloud Service
Silaghi et al. (2012)	AgentFSEGA	A time-constrained SLA negotiation	SLA negotiaton	Opponent modeling-agent based	Grid Computing

Research contributions explained in Emeakaroha et al. (2012) described an architecture Detecting SLA Violation infrastructure (DeSVi) for monitoring and detecting SLA violations in Cloud computing. The major components in this architecture are the automatic VM deployer, which is responsible for the allocation of resources , the application deployer which executes the user applications. This DeSVi architecture is good in managing SLAs related to a single Cloud data center but it is not addressed the monitoring of SLAs in cloud environment with multiple data centers. LoM2HiS (Emeakaroha et al., 2010) gives details about SLA violation detection in cloud computing and corresponding reactive actions. These frameworks do monitoring, analysis, planning, and execution

(MAPE loop) Maurer et al. (2011) to detect and react against violations. In Maurer et al. (2011), the authors proposed the concept of autonomic management of Cloud infrastructures by including a Knowledge Management (KM) phase. In KM phase the observed monitoring information is fed to the KM system and its reactive action prevents further SLA violations. The efficient management of SLAs together with resource management is again needs further explorations.

The summary of significant works related to SLA negotiation, SLA monitoring, and detection of SLA violation is given in Table 2.1. From this Table summary, it is evident that most of the approaches are addressed towards the monitoring of SLA parameters. However, the reasons for violation of SLA is not analyzed and not attempted to reduce the real causes of violation. In this research work, we aim to analyze the reasons for SLA violation and inturn our major objective is to reduce the SLA violation.

2.3 Survey of Resource Allocation Approaches

Recent research works that are more closely related to our proposed approach in terms of adaptive resource allocation are summarized in this section. The resource allocation mechanisms discussed in Stillwell et al. (2009), and Li et al. (2010) focused on job scheduling / task scheduling techniques for allocation of resources. They considered the execution time of the tasks and allocation is done according to the priority value. the authors discussed the allocation of cluster resources among competing jobs by using scheduling algorithms based on linear programming approach. Here they have considered the static workload and so adaptive resource allocation according dynamic workload is not a concern in this work. Berenbrink et al. (2007) proposed a mechanism using game-theoretic approaches to determine perfectly suiting task allocations. Here they described an agent based approach in which individual task is linked with a "selfish agent", and each single agent is supposed to select a resource. The cost of the resource is counted as the number of agents involved in selection. The functioning of this system is based on the principle that- ‘migrate to less loaded resources from overloaded resources till the distribution becomes balanced’.

In Jung and Sim (2011) the resource allocation is organized by considering criteria like geographical distance and workload of the datacenter. In this work they are getting an

improved response time because the job requests are allocating to the VMs in the nearest data center. The adaptive resource allocation according dynamic workload is not considered here. In Bonvin et al. (2011) the authors proposed an autonomic and cost-efficient scheme of resource allocation. This scheme adaptively satisfy SLAs for resource availability and guarantees performance against the variations in load and software/hardware failures. Here, the authors used a middleware consisting of Scattered Autonomic Resources referred to as ***Scarce*** which does flexible distribution to prevent stranded and underutilized computational resources and make dynamic adaptations to varying environments, such as load variations, infrastructure failures, or unsteady servers (or virtual machines).

Huang et al. (2013) proposed an adaptive resource management system in a cloud computing environment. The authors have employed Support Vector Regression (SVR) to determine the number of resource utilization by prediction concerning the SLA of each process. Based on this, the resources were redistributed according to the current status of all of the VMs installed in the physical machine. To find the reallocation of resources they used a Genetic algorithm for optimal allocation of resources. At the end, they reached an agreement between physical machines resource utilization which was monitored by the physical machine monitor as well as SLA between virtual machine operators and cloud service providers. But in this prediction based approach the time needed for calculation and the genetic algorithm processing is more and it affects the performance of the resource management in cloud environment.

Addis et al. (2013) proposed a scalable distributed hierarchical framework based on a mixed-integer non-linear optimization of resource management acting at multiple timescales in a very large cloud platform. The run time management framework of their proposed method assumed that the PaaS provider supported multiple transactional services execution with a set of heterogeneous servers. They mainly focused on the CPU and RAM as representative resources among many other physical resources for the resource allocation problem. The main objective of their resource allocation problem is to maximize the profit such that the difference between revenues from SLA contracts and costs associated with servers switching and VM migrations. The real time work load and large time scales are the scope for further expansion since they implemented the resource allocation

by considering fine-grained time scales.

Shen and Liu (2014) proposed a resource sharing platform for the Collaborative Cloud Computing (CCC) called harmony which integrates the resource management and reputation management in a balanced manner. Their work incorporates three innovations: integrated multi-faceted resource/reputation management; multi-QoS-oriented resource selection; and price-assisted resource/reputation control. Lu et al. (2015) proposed a fairness evaluation framework for the resource allocation scheme in cloud computing. In this paper they proposed a Dynamic Evaluation Framework for Fairness (DEFF) to evaluate the fairness on allocating the resources using a resource allocation algorithm.

Table 2.2 Summary of significant research works on adaptive resource allocation

Publication	**Resources organized as**	**Principle**	**Static/ Dynamic allocation**	**Allocation policy**
Wood et. al. (2009)	Pool of resources	VM provisioning, VM migration, VM resizing	Dynamic resource allocation	Time-series prediction techniques
Stillwell et al. (2009)	Clusters of resources	Allocating cluster resources among competing jobs	Static allocation	Linear programming based scheduling
Li et al. (2010)	Resources arranged in the form of a DAG	Resource allocation with preemptable task scheduling	Dynamic resource allocation	Preemptable task scheduling
Jung and Sim (2011)	Pool of resources	Adaptive resource allocation based on geographical distance and workload variations	Dynamic resource allocation	Allocation of job requests to nearest data center, better response time.
Bonvin et al. (2011)	Agent Based prototype SCARCE	Dynamic resource allocation to adapt to varying loads ,failures and SLA requirements	Dynamic resource allocation	-
Huang et al. (2013)	Pool of resources	Optimum resource allocation as per SLA	Algorithm based resource management	Support vector regression (SVR) and genetic algorithm to prediction and estimation
Addis et al. (2013)	Multi tier clouds	Resource allocation policy for multi tier virtualized cloud systems	Static allocation	Time scaled resource prediction
Shen and Liu (2014)	Neural network model for resources	Integrated resource/reputation management platform-Harmony	Static allocation	Neural network model for resource selection

They proposed two sub models to describe the resource demand with dynamic characteristics. These models are Dynamic Demand Model (DDM) and Dynamic Node Model (DNM). They employed two typical resource allocation algorithms such as utility based algorithm and fairness algorithm in order to show the effectiveness of their proposed fairness

evaluation framework.

The Table 2.2 summarizes the research works that are more close to our proposed work in terms of adaptive resource allocation. From the summary of existing works, we found that no major focus is given to controlling resource management in cloud computing majorly based on SLA. The SLA violations effects can be minimized by adjusting the measure of resource provisioning and this should be in a balanced manner since the over-provisioning of resources will again create adverse effect to cloud based business economy. Therefore, in our research work we majorly aim at allocating the resources towards reducing the SLA violations.

2.4 Survey of Federated Cloud Approaches

This section presents the summary of potential research works that motivate us to solve the interoperability issues of cloud computing. The survey presented in this section helped us to identify and address popular challenges in implementing federated cloud. According to Buyya et al. (2010c), the vision of interoperability can be materialized by maintaining a federated cloud environment which supports dynamic expansion and contraction of resources for managing sudden variations in resource demands. Here, the authors suggested that for the development of an inter-cloud environment following essential elements are required:

- Architecture framework for maintaining utility oriented clouds and their federation
- A cloud coordinator for exporting Cloud services
- A cloud broker for mediation
- A cloud exchange as a market maker and
- A software platform for controlling the entire federation.

Toosi et al. (2014) investigated and presented the significance of motivating cloud interoperability. They have identified the list of challenges and complications for materializing Inter-cloud environments. In Paraiso et al. (2012), the authors presented a federated multi-cloud PaaS infrastructure containing two parts, an open service model and a generic

kernel infrastructure. The open service model is used both for the PaaS infrastructure and the SaaS applications hosted on top of it. But the interoperability between the services provided by multiple environments is not much successful in this architecture. Also they haven't experimented about the cloud properties like intercloud elasticity. In Gomes et al. (2012), the authors describe the application of mechanisms based on the General Equilibrium Theory for coordinating the sharing of resources among clouds in the Federated Cloud. A Federated Cloud helps individual clouds to adapt with variations in demands. clouds enable users to cope with unexpected demand loads. That is, individual clouds can obtain resources from other clouds during overload and can offer resources to others during less load. Here an exchange market is proposed for the trade of cloud resources between clouds by considering a less number of clouds and a minimum sample of of resources. A full-fledged federated cloud is the scope for their future work. Goiri et al. (2012) devised a model for characterizing situations in a federated cloud such as: when to go for resource outsourcing to other providers, when to go for admitting resource requests from other providers, what/how to contribute to the federation etc.

The capability of VM to move from one host to another when it is operational is termed as VM Mobility (Dowell et al., 2011) and is very much required in an Inter-cloud environment. Furthermore, it should not disturb the independence of the individual clouds corresponding to privacy, security and autonomy. The significance and need for inter-cloud VM mobility under physical and administrative constraints are discussed in Nagin et al. (2011). Here the authors presented the design and implementation of a technology for enabling live mobility of virtual machines between clouds, while enforcing the cloud insularity requirements of autonomy, privacy, and security. But this design is based on the concept of a common storage space and common network.

In Reich et al. (2007) the authors proposed an SLA-oriented management architecture for providing integration of interoperability among loosely coupled web services. In their work they have described the extension of the autonomic management of Web services associated with the Web Services Resource Framework (WSRF) connected by a structured P2P network. They have provided isolated service level domains in the SLAs, and implemented a regionalized migration algorithm for reallocating services between users to

ensure the decided QoS.

In Amin et al. (2012), the authors present a Publish-Subscribe based middleware for Intercloud Message Exchange and is implemented through Data Distribution Service (DDS). Here an OWL based cloud resource description ontology is utilized by cloud environments for resource cataloging and possible matchmaking prior to workload migration between heterogeneous clouds. The real time VM migration is not a motive of this work and so interoperability through VM mobility is not addressed here.

The CONTRAIL Project developed a negotiation model for cloud federation framework where the SLAs Coppola et al. (2012) deal with the organization and deployment of applications on multiple clouds. Each application could then go with several cloud providers, and the user negotiates with the apt cloud provider. Here in this work, the authors failed to adjust the resource provisioning mechanisms.

Table 2.3 Summary of important research work on federated cloud

Publication	**Principle/ policy**	**Infrastructure**	**Model / Framework**	**Parameters**
Buyya et al. (2010c).	VM migration and scheduling	Cloud coordinator, cloud broker and cloud exchange	Intercloud architecture	Availability and price
Nagin et al. (2011),	VM mobility by VM state and memory transfer	Intercloud proxies, shared storage and virtual network migration	Framework prototype	Autonomy, privacy, and security
Paraiso et al. (2012)	Configurable kernel for Multi-PaaS Infrastructure	Cloud node provisioning together with PaaS and SaaS deployment	FRASCATI platform	Portability, interoperability and heterogeneity
Gomes et al. (2012)	Cloud exchange market controlled by trading strategies	Federated datacenter with a cloud coordinator	Framework prototype	Budget constraints and demand constraints
Goiri et al.(2012)	Insourcing and outsourcing of resources	Federated datacenter with scheduler, resource manager and resource fabrics	Framework prototype	-

The summary of important research works on federated cloud is presented in Table 2.3. As shown in the Table 2.3, the existing research majorly uses availability, price, autonomy, privacy, security, budget constraints and demand constraints as a way to solve interoperability issues. To the best of our knowledge, there are no research work addressing the interoperability issues with SLA based approach. Therefore, in our research work we aim at proposing a holistic solution using SLA templates for solving interoperability issues of federated cloud environment.

2.5 Gap Analysis

The literature survey on the related research works in existing literature shows the importance of combing SLA management and resource management in the field of cloud computing. From the literature survey, we have identified the following major aspects that are to be addressed.

- **RG1**: Dynamic adaptation strategies for varying workload and infrastructure conditions were not considered.
- **RG2**: Predefined SLA parameters only are considered for the calculation of threshold limit which creates difficulties with dynamically changing applications. Therefore, there is need for considering the dynamically changing SLA parameters also for efficient allocation of resources.
- **RG3**: Cloud providers try to provide excess number of requested resources in order to escape from SLA Violations and this leads to wastage of resources due to over provisioning. This is not an efficient method to control SLA violation.
- **RG4**: The dynamic allocation of resources to cloud applications in the context of multiple cloud providers is not taken into consideration.
- **RG5**: Dynamic change in resource requirement and the reduced number of traders in cloud market lead to market instability crashes. This market instability affects the deployment of cloud services and generates costly consumption of hardware infrastructure.

With the preliminary understanding of elastic cloud and it's challenges, this research work aims at providing a suitable SLA management framework, adaptive resource allocation approach and SLA based cloud federation. From the above list of research gaps, we attempt to address the first four research gaps (RG1, RG2, RG3, and RG4) in this thesis work.

2.6 Summary

Here we have done an extensive study of the works related to cloud computing, service computing and grid computing related to the problem of SLA negotiation and resource

management. The negotiation and monitoring of SLAs can contribute more to the business outcome of the service providers since for each violation of the negotiated agreement, the provider has to pay penalty to the consumer. This motivated us to start with SLA monitoring and has proposed an adaptable resource management framework by incorporating SLA monitoring as a preprocessing unit. The next chapter of the thesis gives a description of our proposed SLA management system by covering the negotiation and monitoring of SLAs. The SLA monitoring is for detecting SLA violation and avoiding violations to the possible extent.

Chapter 3

SLA MANAGEMENT FRAMEWORK

An Efficient Resource allocation mechanism should be able to handle dynamically varying requirements effectively. The main aim of this objective is to design and implement an SLA negotiation and monitoring framework which can be used for Efficient Resource Allocation. This chapter specifically demonstrates objective one. The SLA negotiation and monitoring framework has been developed here to predict and avoid SLA violation. This framework is developed using Web Service Agreement for Java (WSAG4J) Framework. Additionally a Dynamic SLA scheme for proactive prediction of SLA violation is developed and implemented. The developed frameworks perform better than the other contemporary approaches.

3.1 Introduction

Service Level Agreement (SLA) identifies an association between the service provider and consumer with mutually accepted terms and conditions for a particular service. SLA allows both the involved parties to establish consent on their roles, rights, and obligations. Consequently, a third party agent could develop these mutually accepted SLAs in the name of the cloud service provider or service consumer. The consumers specify their requirements, and the third party agent starts the negotiation with the available service providers. After receiving the service request, the service provider provides the time slot for accessing the requested services if sufficient resources are available. In addition to the time slot, QoS parameters are also mentioned in the SLAs. The whole process of connecting cloud service provider with the cloud service consumer using SLAs is referred to as SLA Management.

3.1.1 Overview of SLA Management

SLA management in cloud computing has two different perspectives: Service provider's perspective and Service consumer's perspective. According to service providers, it is a problem of satisfying QoS parameters that are feasible, competitive and yield maximum profit. According to service consumers, it is a problem of deciding QoS parameters that satisfy resource requirements, business demands, and available budget. The role of service providers is to assign, release and modify available resources and associated services. On the other hand, service consumers role is to request, access or terminate services. The significant role of SLA Management is to monitor the performance of the cloud service provider. The performance will be calculated in the form of quality of service requirements that are mentioned in SLA documents. Similarly, there will be a set of QoS parameters that are mentioned in SLA documents with respect to cloud service consumers. These QoS parameters play a major role in price calculation and billing.

The job of the SLA management system is to ensure the agreement terms mentioned in the SLA are met. The SLA management lifecycle has following phases: The first step is the SLA establishment phase where the cloud customer and cloud provider negotiate significant SLA values for offering a specified service by considering needed measures or values. That is a customer could get the necessary measures such as response time provided by the service provider, consider them as SLA parameters and get the approval from the service provider. Alternatively, the negotiation of SLA has to be done as per the defined SLA template, using a negotiation system. Once the negotiation of SLA is initialized, the SLA deployment agent has to take care of checking its validity and distribute it into appropriate components and involved parties.

3.1.2 Role of SLOs in SLA

Usually, SLAs are viewed as a collection of multiple SLOs. The quality parameter of an SLA is measured as an SLO, and the SLA violation means non-attainment of pre-fixed SLOs. The service provider has to pay the penalty this SLO non-attainment.

In general, the SLO values will be monitored by SLA monitoring engine. As shown in Figure 3.1 there are two phases called as negotiation and execution. During the negotiation

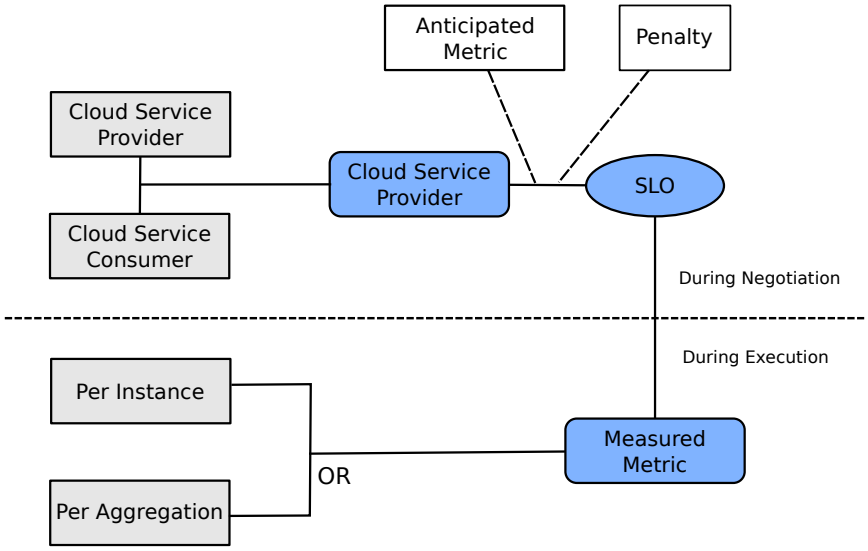

Figure 3.1 Domain model of SLA

phase, the cloud consumer and the cloud service provider will negotiate with each other. Specifically, anticipated metrics namely time to order, time to deliver, Delay in the end to end transaction, Quality of item, Failed service request rate, Availability of service rate and the penalty for each of attribute (in case of SLA violation) will be mutually negotiated by the cloud consumer and cloud provider. At the end of this negotiation phase SLOs, anticipated metric, penalties would have been decided. In the execution phase, the negotiated SLOs will be measured. The SLO metrics for a real-time shopping service are illustrated in Table 3.1. Time to order, time to deliver, delay in the end to end transaction are considered to be instance level SLOs which are measured over an interval of time. Quality of good, failed service request rate, availability of service rates are considered to be aggregated level SLOs and are measured over instances. During execution, the service provider will be penalized if there is an SLO violation. For example, as shown in Table 3.1, if time to order takes more than 36 hours, the service consumer will be given 3 % discount by the service provider as the penalty. Similarly, for each SLO violation, the service provider will be penalized and the service consumer will receive the benefit as per the negotiated SLOs compensation plan.

The monitoring engine of our SLA management framework monitors the SLO values which is measured in a per-interval/ per-instance manner. Figure 3.1 illustrates the diagrammatic representation of measuring SLO metric in a per-interval/per-instance manner

Table 3.1 Sample SLO

Type of SLO	SLO	Anticipated metric	Penalty	Interval
Instance level	Time to order	$\leq$ 36Hrs	3% discount	Not applicable
	Time to delivery	$\leq$ 4 days	5% discount	Not Applicable
	Delay in end to end transaction	$\leq$7days	10% discount	Not Applicable
Aggregated	Quality of good	High	80% discount	Not Applicable
	Failed service request rate	$\leq$1%	20% discount on next purchase	Bi-weekly
	Availability of service rate	$\geq$ 99%	20% discount on next purchase	Monthly

during SLA negotiation and after negotiation. The penalty values will also be taken care of while measuring SLO metrics. Example of a set of SLO metrics is shown in Table 3.1 by assuming a real-time shopping service.

3.2 SLA Negotiation Framework

Negotiation means a process of resolving conflicts among multiple parties involved in a business scenario which is intended to reach a common agreement mutually favorable to each other. Different forms of negotiations are possible according to the multiplicity of the involving parties (Badidi, 2016). The multiple possibilities are:

1. *One-to-one negotiation:* It is the kind of negotiation in which a single customer bargains with a single provider for the attainment of an agreement for the delivery of an item or service.

2. *One-to-many negotiation:* It is the kind of negotiation in which a customer bargains with several providers where the providers compete or collaborate to provide the item or service.

3. *Many-to-one:* It is the kind of negotiation in which several customers bargain with a single provider and the customers compete or collaborate to reach an agreement with the provider or cooperate to share the item or service offered by the provider.

4. *Many-to-many:* This form of negotiation involves multiple consumers and multiple providers and where several customers bargain with a multiple provider independently and the customers compete or collaborate to reach an agreement with the provider or cooperate to share the item or service offered by the different providers.

The one-to-one negotiation is the simplest negotiation in which the two negotiating parties bargain on a single issue (Lee and Ferguson, 2010) but in reality both the sides need to negotiate on multiple criteria. As an example, if a customer wants to buy a car, he needs to negotiate with the car dealer on criteria like - the price, the color, working conditions, the warranty, etc. The standard way to manage negotiation is: Characterize the choices of each party using a utility function as a first step and mark the conclusions based on the values of corresponding utility functions (Lai et al., 2006).

The process of negotiation includes the following major components:

Objects of Negotiation: These are the set of items or properties the negotiating parties try to negotiate for reaching a joint agreement. Quality of service attributes like availability, reliability, response time, price, etc. can be taken as objects for a service negotiation.

Negotiation protocol: This defines the significance of the involving parties which states the roles and rules for governing the dealings between the parties. Negotiation protocol identifies the negotiation states and the changing actions in each state, i.e., the set of valid actions possible to each participating parties in individual negotiation states. Two classes of negotiation protocols are there: bilateral negotiations and auctions. Mutual negotiations comprise of two parties such as a provider, a consumer and a negotiating protocol for submitting proposals and offers.

Decision model: These are the set of tools and models used to compute negotiation decisions.

Cloud computing environment with multiple service providers employs a third party agent called a cloud broker to resolve the complexity of SLA negotiation. The role of the cloud broker is to act as an intermediary to service providers and consumers and to settle

with the best SLA. It accomplishes the following tasks on user behalf:

- search for all possible cloud services according to the requirements of the user;
- verify the truthfulness of service providers;
- confirm the party to negotiate by compiling previous experiences, user requirements and other needed measures;
- select the best possible price among available providers; and
- negotiation among different SLAs/providers.

Enough communication interfaces are provided in the proposed architecture to implement interaction between the service consumer and the broker, between the broker and the service provider and between the service provider and the service consumer. Specific policies are there to govern the SLA negotiation, and the negotiator determines the price of the service, the penalty for violation, etc. as per the negotiation policies.

The consumers' requirements are varying dynamically, and this points to think about dynamic re-negotiation of SLA. The re-negotiation permits to have multiple negotiations in cases of lost, delayed or re-ordered messages. The re-negotiation starts with a re-negotiating state, the agreement contract goes into a re-negotiating state which is viewed as part of the present agreement, and after the finishing of re-negotiation, the present state is viewed as a superseded state. Issues considered in dynamic SLA re-negotiation are:

- Delete an existing SLA and negotiate a new one;
- Remove or add an SLA objective; and
- Redefine the parameters of SLA.

The proposed SLA negotiation involves an iterative process between the involved parties and proceeds through three major phases as follows:

- Initiate the negotiation **(prepareAgreement)**
- Negotiate the values **(negotiate)**

- Create new SLA **(commit)**

The bilateral negotiation of SLA uses an offer and counter-offer style for proceeding with the negotiation. During the *prepareAgreement* phase, the negotiation unit initiates the process of negotiation by giving one requirement as the input offer and collects multiple counter offers. During the *negotiate* phase, the negotiation initiator and the negotiation responder will have interactions to fix the values of the SLO parameters and will come up with the negotiated SLOs. During the *commit* phase, the negotiated SLO values are stored as a document and this document will be shared among the involved parties.

3.2.1 Architecture of SLA Negotiation Framework

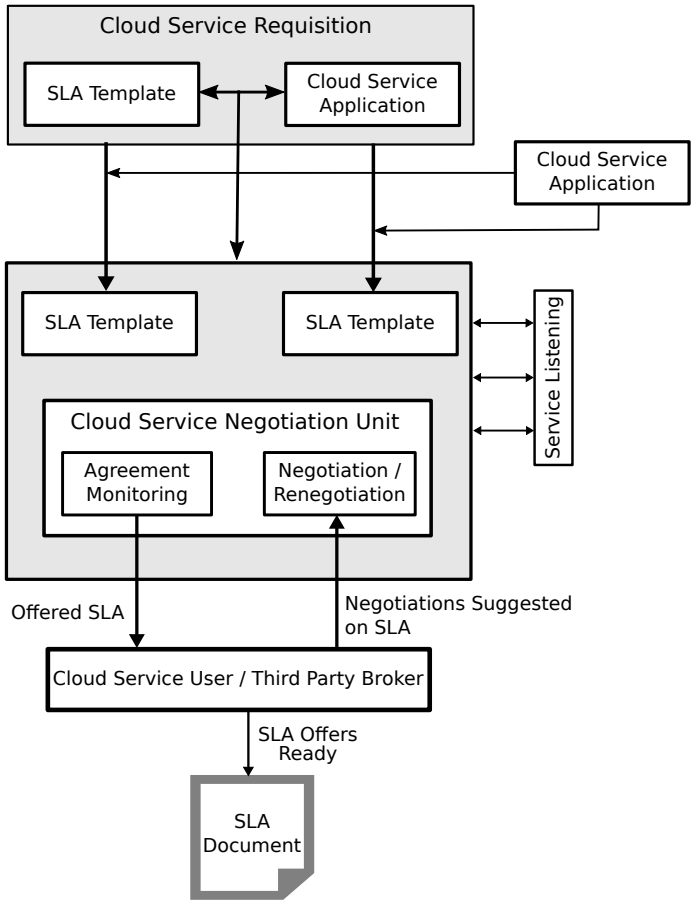

Figure 3.2 Architecture of SLA negotiation

The SLA negotiation architecture (Figure 3.2) contains the following main units such as: (i) cloud service requisition unit, (ii) service listening unit, (iii) cloud service negotiation unit and (iv) agreed SLA document. Cloud service requisition unit is to identify the

apt cloud service and service provider for the cloud consumer's application. The detection and selection of the service is done after considering several criteria such as: QoS of the expected cloud service, nature and behaviour of the service, provisioning interface and the integrity and reliability of the cloud service provider. Service listening unit checks external service registry periodically for the changes and updates of the service. Cloud service negotiation unit is responsible for the negotiation of SLA which goes through possible negotiation specifications and finalizes the suitable interface for interacting with the service. The interface gives the option for initiating the cloud service provider or cloud service broker, initiating the process of negotiation and informing the cloud service provider that an SLA has generated as a result of the negotiation. The last unit, agreed SLA document contains all details about the agreed upon SLA. Different SLOs and their permissible values, the penalty for violation of SLO values, etc. are examples. Almost all of the research works in this area have used XML language for identifying the service agreement between the service consumer and service provider. In this XML based SLA negotiation message, negotiation rules and negotiation decisions are focused more.

3.2.2 SLA Negotiation Rules

The process of SLA negotiation is done according to the rules defined for negotiation using XML schema. These rules are framed in the form of conditional statements such as:
if (condition) *then* **perform action** *else* **perform action**
The 'condition' deals with QoS values and the '**perform action**' decides whether to change the QoS value or not. A negotiation rule is given in Figure 3.3 as an example. The negotiation unit acts according to the action part specified in the negotiation rule. Different rule actions can be: (i) accept actions, (ii) reject actions and (iii) set actions. Accept actions are for accepting multiple SLO values (QoS attributes) in an SLA. Reject actions are for rejecting the values of QoS attributes and set actions are for proposing new set of values for QoS attributes. The negotiation rule is used by the cloud broker which mentions that the service consumer has made an offer (or counter-offer) of the requested service. The negotiation rules expressed in regular XML schema have converted to the particular format of the negotiation engine provided with the broker. We have designed a negotiation engine using rule-based concept with JESS (Java Expert System Shell (Friedman-Hill et al.,

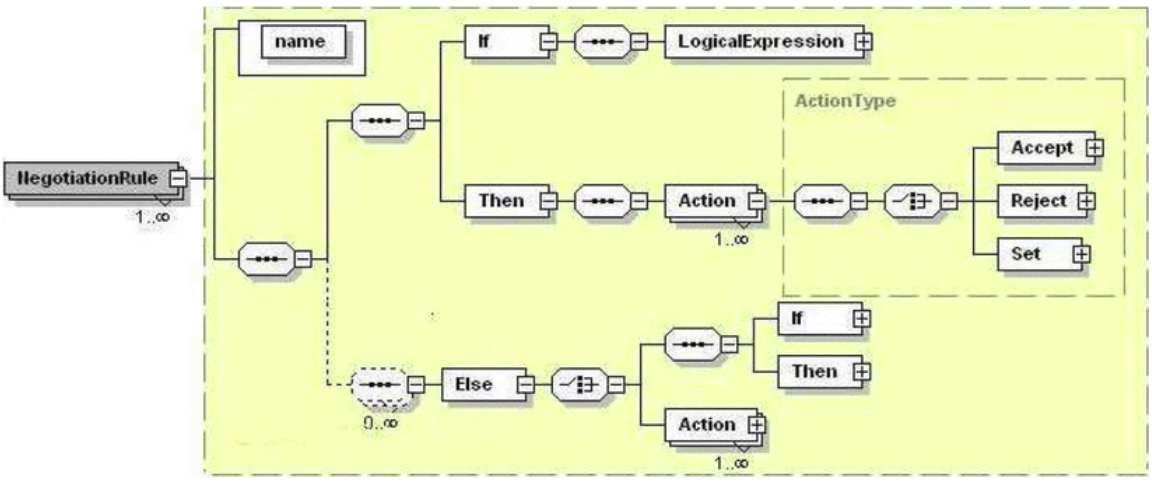

Figure 3.3 Sample of SLA negotiation rule

1997)) for the current negotiation framework. The general form of a JESS rule is:

$(defrulerule-name(logical-operator(cond-1...cond-n)) \rightarrow action)$

where,

$cond-i$ defines the logical conditions over the known facts.

The negotiation rules are get translated into JESS rules inside the broker.

3.2.3 SLA Negotiation Decision Model

Negotiation decisions are taken by using time-based decision functions where the decisions are finalized by considering many factors. Some factors which influence the decision making are cloud customers' requirements, cloud providers' preferences, environmental conditions, etc. The beauty of the time-based functions are, it can accommodate all ranges of all possible criteria for making negotiations. In a negotiation environment, multiple attribute values are there to negotiate, and for each argument, there are multiple possibilities to consider. The best possibility is selected depending on the exactness of the argument value. So an entire SLA negotiation policy can be represented in mathematical form as follows.

Let 'i' be the attribute considering for negotiation at a particular point and 'x_i' be its attribute value, in the current multi-attribute negotiation model. The possible value of 'i' varies from 1 to n and the possible value of 'x_i' can be in between 'max_x_i' and 'min_x_i'. Let 'w_i' be the assumed weight of attribute 'i' which represents the importance of attribute

'i' in entire SLA negotiation.

$$\sum_{1 \leq i \leq n} w_i = 1,$$

since we assume normalized weights for all attributes.

Then the Total Negotiation Value (TNV) is,

$$TNV = \sum_{1 \leq i \leq n} w_i N_V(x_i) \tag{3.1}$$

$$N_V(x_i) = \begin{cases} \frac{max\_x_i - x_i}{max\_x_i - min\_x_i}, & if \quad x_i \leq max\_x_i \\ \frac{x_i - min\_x_i}{max\_x_i - min\_x_i}, & otherwise \end{cases} \tag{3.2}$$

The negotiation decisions are finalized after analyzing the offer and counter-offer values. Here service customer (C) provides an offer to the service provider P and P provides counter offer back to C. The time measure is assumed as the difference in the time the offer is initiated (unit 't') and its corresponding counter-offer at t^C. Let $O^t{}_C$ be the offer value from cloud consumer to the provider and $O^{t^C}_{P \to C}$ be the counter offer from the cloud provider. Then the negotiation decision is formally fixed after evaluating the TNV.

The negotiation decision is formally represented as:

$$D_{(t \leftrightarrow t^c)} = \begin{cases} \text{accept, if } N_V(O^t_{C \to P}) \geq N_V(O^{t^C}_{P \to C}), \\ \text{reject, if } t^C > t_{max}, \\ O^{t^C}_{p \to C} \text{ otherwise} \end{cases} \tag{3.3}$$

where,

t_{max} is the maximum time for negotiation.

In this negotiation environment, the counter offers and hence the negotiation decisions may vary as per the variations in time, resource availability and behavioral changes.

SLA negotiation Process

The sequence diagram given in Figure 3.4 shows the steps involved in the process of SLA negotiation. A set of message interactions are necessary, to negotiate SLA, among the service consumer and service provider. The detailed picture of all message transactions occurred at the time of SLA negotiation is shown in Figure 3.4 and explained as follows:

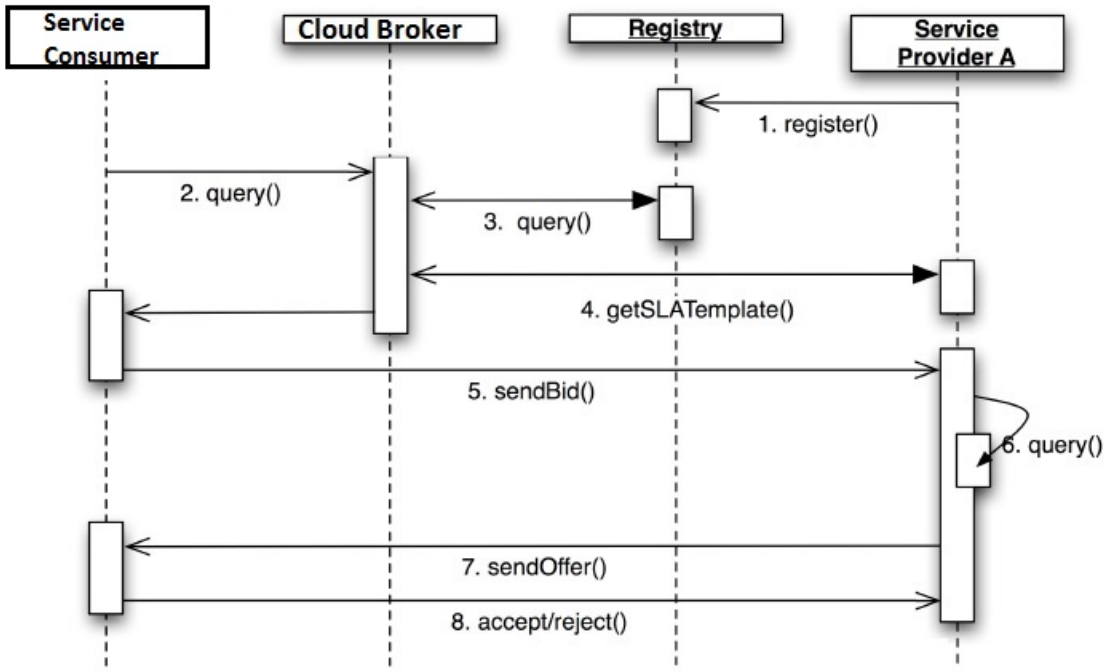

Figure 3.4 Message interactions during SLA negotiation

1. The *Service Provider A* starts the registration using the **1.register()** message to the *Registry*. This message consists the details of services that can be provided by the *Service Provider A*. The *Registry* collects the details from various cloud providers and stores the details.

2. The *Service Consumer* sends a **2. query()** message to *Cloud Broker*. This message consists of the specification /requirements of requested cloud services.

3. The *Cloud Broker* analyzes the cloud *Service Consumer's* requirements and forms a new query **3.query()** that contains the request details to be sent to *Registry*. Further, the *Registry* checks for the available services from the list of registered *Cloud Service Providers*. After identifying the available *Cloud Service Providers*, the *Registry* sends the response message to *Cloud Broker*.

4. The *Cloud Broker*, requests the SLA templates of available *Cloud Service Providers* through the message **4.getSLATemplate()**. The received SLA templates will be analyzed and sent to *Service Consumers* as a response to resource requests.

5. After receiving the SLA template, the *Service Consumer* will communicate with selected *Cloud Service Provider* with **5.sendBid()** message. This message consists of an offer for initiating the SLA negotiation.

6. The *Cloud Service Provider* analyze the incoming request and send the **6.query()** to check whether the requested amount of services available or not.

7. If the requested services are available, the *Cloud Service Provider* responds to the *Service Consumer* with the message **7. sendOffer()**. This message consists of the counter offer and this will be analyzed at the *Service Consumer* side.

8. Finally, the *Service Consumer* decides whether it can accept or reject the negotiation and the decision can be communicated through **8. accept/reject** message.

The negotiation unit begins the first phase of negotiation by collecting all available SLA templates from all service providers. The negotiation initiator selects the apt template from the collected set as an initial point, which describes the background for all subsequent iterations. It is required that all successive offers should follow this initial template, which helps the agreement provider to evaluate the constraints of the original template, at the time of negotiation. In the second phase, the initiator generates a new SLA template as per the selected format. The agreement initiator is free to modify the contents of the created template by adding service descriptions, service properties, and the guarantee terms according to the original template. This new template is forwarded to agreement responder through a message, and the responder checks the credibility of the input document. The agreement provider, then checks whether the defined service can be provided or not. If it is possible, the agreement provider sends back the template of the agreement to the client, which indicates that this particular offer based on this template will be admitted. In the other case, the provider initiates some mechanism to generate counter offers. In the third phase, the negotiation initiator checks whether the received counter offer meets the requirements and stops the negotiation process, if it is not satisfying, and starts all steps from first phase.

To establish an SLA negotiation, the system has to go through a sequence of steps. The proposed Algorithm (1) for SLA negotiation contains the following steps.

Algorithm 1 SLA Negotiation

1. SLA negotiation at the user requirement level.
2. SLA negotiation at the service provider level.
3. SLA negotiation at the job execution level.
4. Confirmation of negotiated SLA.

In the first step the user requirements concerning service descriptions, particularly meta data of the service, is communicated to the cloud broker in the system.

The meta data include:

- Information about the computing service
- Estimated time for completing the service
- Estimated service cost, etc.

The broker then checks for a cloud service provider, who can provide the service as per the user requirements. This checking is done in step 2 and on completion of step 2 the service broker comes up with a particular service provider, who is capable of providing the needed cloud service. In the third step, the actual SLA negotiation has been executed. As part of the execution, both the involved parties institute a consent on the QoS parameters of the provided service and the agreement terms are finalized. This finalized SLA document is then communicated to the involved parties as a confirmation.

The complete process of SLA negotiation is depicted in Algorithm 2.

The process of SLA negotiation proceeds through the following activities. As per the activity diagram shown in Figure 3.5, the negotiation process begins with the *Selection of service*. The initial selection is done from a comparative study among the service details, collected from all sources such as the previous users of the cloud service, grading/rating of the providers, etc. and the requirements of the service collected from the cloud service user. From this preliminary comparison, a particular service is selected, and then the steps for negotiation are initiated. Service selection process also depends on the service registry which contains the details of all existing services together with the description of a new modified service. The negotiation framework selects the most suitable cloud service that does not have a negotiated user or consumer and proceeds with the negotiation steps. The QoS traits of the selected service and corresponding SLO values are negotiated in this

Algorithm 2 SLA Negotiation Process

Input: Set of cloud service requests.
Output: Mutually agreed SLA document.

Consider a particular service request,
Initialize the minimum count of service providers available to satisfy the service request.

```
limit ←count of resources in the cloud
count ← 0
for each cloud resource ← 1 to limit do
   take communication message from requester
   switch (communication)
   case: 'sendsignal'
      communicate message 'empty-template'
      set flag←1
      break
   case: 'receivesignal'
      communicate message 'filled-template'
      break
   case: 'sendsignal && flag==1'
      communicate message 'tentative-SLA'
      break
   case: 'acceptsignal'
      count++
      if (count==limit)
         communicate message 'commit'
      else continue
      break
   case: 'rejectsignal'
      break
   default:
      communicate error message
end;
```

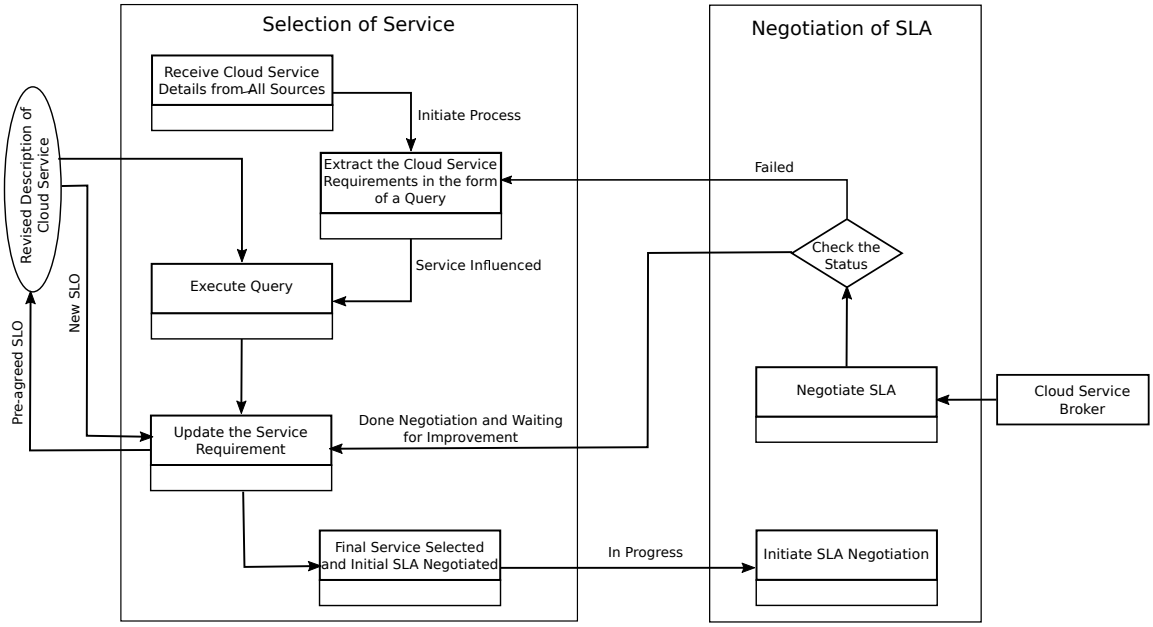

Figure 3.5 Process of SLA negotiation

phase to generate the most suitable SLA for both the parties. If the negotiation with the selected service is not going smoothly, then a second service is selected and it proceeds with the same negotiation steps.

3.2.4 Experimental Results

Here in this section, we have evaluated the performance of our SLA negotiation prototype based on the existing implementation of WSAG4J (Waldrich, 2016; Wäldrich, 2011). The experimental evaluation for the monitoring of SLA to detect SLA violation and the associated analysis is included in the next chapter together with the SLA driven resource allocation. To evaluate the prototype, the WSAG4J (version 2.0) framework was used together with a real-life use case scenario of automated SLA delivery in the field of cloud computing. A clear use case for service negotiation is already used by several researchers in the field of automated SLA processing (Waldrich, 2016) (Comuzzi and Spanoudakis, 2010). Web services meant for handling SLAs are defined as neutral mediators and must be reachable to both the parties, guaranteeing established agreements. So the automated SLA negotiation should ensure availability, scalability and performance requirements. To evaluate the use case of automated SLA delivery, three negotiation scenarios such as *Garage, Designs, and Proposals* were considered to replicate the cloud computing environment.

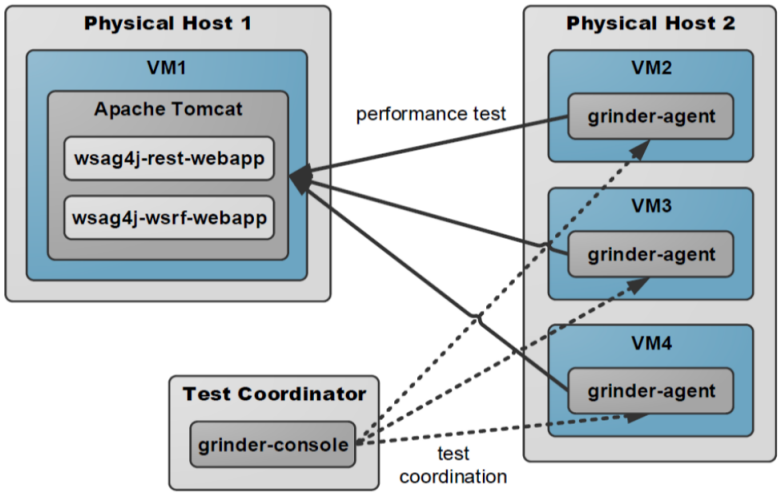

Figure 3.6 Testing architecture for SLA negotiation

That is every Service Negotiation has to deliver at least one *Garage (Garage of Agreements)* having multiple *Designs (Agreement Designs)* which gives a description of the services provided and act like a model for coming *Proposals (Agreement proposals)*. Here the framework deals with one garage containing three designs.

Find Garage: This is the first scenario in which the initiator of the agreement requests for garage given by the provider. In this first step, the initiator has to search for services of an unfamiliar provider.

Find Designs: This is the next scenario in which the service discovery will be proceeded. In this service discovery step, the initiator collects information about available agreement designs for the garage.

Find Negotiation: This is the third scenario in which the negotiation process between the initiator and the responder will occur. The responder is responsible for the service provider side, and the initiator takes care of the service consumer side.

The architecture for testing SLA negotiation (shown in Figure 3.6) is taken from Wäldrich (2011) which was set up using multiple physical host machines (servers) networked using Ethernet links. Each physical host machine is equipped with multiple Kernel-based Virtual Machine (KVM) containing Ubuntu 13.04, Java and JDK 1.8. The load tests have been conducted on these two physical server machines by providing four virtual machines, and

the tests have coordinated with the help of Grinder 3.11 (Grinder, 2016), a load testing framework based on Java.

Host 1 runs VM1 which processes the WSA4J web apps using Apache Tomcat, and Host 2 contains VM2, VM3, and VM4 which is dedicated for processing load test components (called grinder agents) of the Grinder framework. A grinder-console running monitors all these agents on a third host machine. The grinder-console is responsible for all coordination activities such as code distribution, synchronization of the tests and measurement of test results.

The load tests have been conducted for each test scenario several times, where each test scenario is executed with a varying number of clients operating simultaneously for evaluating scalability. The tests initiated using a single client, and the count increased one by one till 4 and the Java Virtual Machine (JVM) and Apache Tomcat have restarted in between to get more accurate results. The test results have been measured for 100 test runs.

The response times of all the three test scenarios considered are shown in Figures 3.7, 3.8 and 3.9. The scale value of each response time is fixed according to the measurements, and these values have been taken after assessing the test run measures. From the results, it is visible that for the second scenario the response time is less and the percentage failure of negotiation is least.

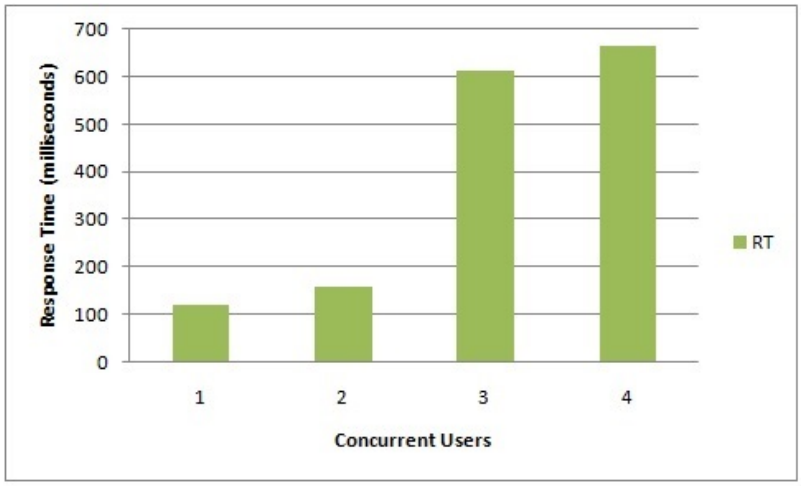

Figure 3.7 Response time for Find Garage

Figure 3.10 gives the response time comparison for all the scenarios and shows that the

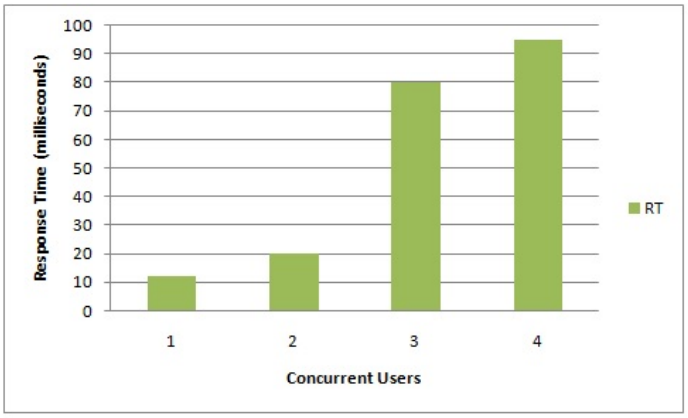

Figure 3.8 Response time for Find Design

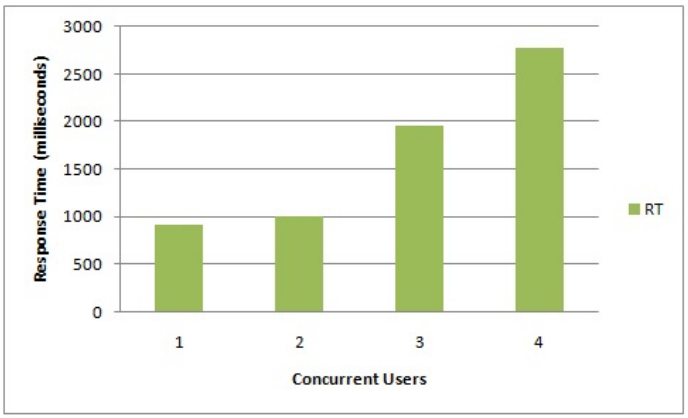

Figure 3.9 Response time for Find Negotiation

response time is much higher for the third scenario. This shows that the negotiation process is more complex in the third scenario and consumes more time to complete. It must be emphasized that web services providing WS-Agreement and WS-Agreement Negotiation act as neutral notaries which must by definition always be reachable to both agreement parties, enabling 24/7 verification of SLAs.

From Figure 3.13 and Figure 3.11, it is clear that out of 15 and 25 arbitrarily generated requests, 22% and 34% of the requests are serviced by the proposed method of SLA negotiation and the rate of sanctioning the resources is more when compared with the resource allocation without negotiation.

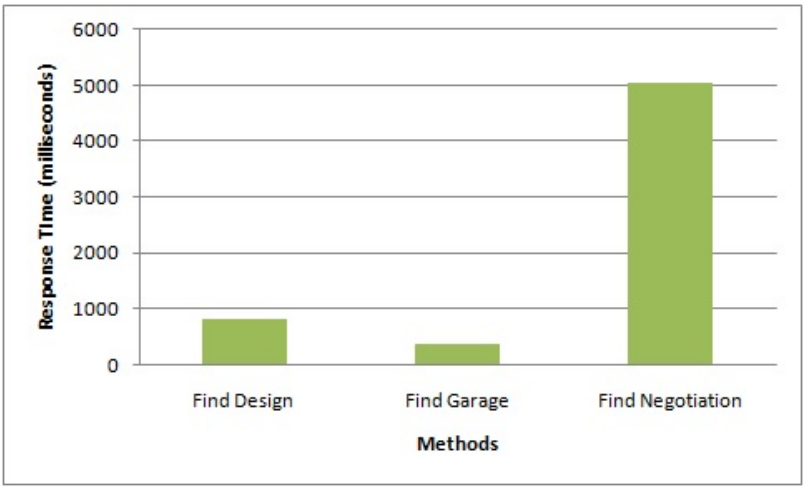

Figure 3.10 Response time comparison for all scenarios

Figure 3.11 Rate of serviced requests

Figure 3.13 Rate of serviced requests

Figure 3.12 Percentage of negotiated SLAs

In our experimental studies, we have considered a list of computing resources which are stored in a database within the system. Some examples of computing resources are RAM, number of CPU cores, memory, network bandwidth, processor availability, etc. Here we are assuming a service requesting environment where every service has its executable code. These services have a set of limitation on QoS measures. For each service 10 resources are randomly selected from the database by assuming these resources as available for the selected request. These are given as input to the negotiation. The random process model that we have chosen generates 50 requests and chooses the resources randomly. The SLA negotiation results for the described experiment are plotted, which are shown in Figure 3.11, Figure 3.12 and Figure 3.13. Figure 3.11 and Figure 3.13 show the rate of sanctioning of the service requests with and without the newly devised SLA negotiation.

These results prove the efficiency of our proposed method, and it is confirmed that the throughput of the scheduler gets improved with the usage of our approach. This performance improvement shows that with the use of our proposed negotiation method the available resources/services in the cloud environment can be allocated to the needy service consumers efficiently. This efficient allocation enhances the utilization of resources and improves the efficiency and rating of the provided cloud service, which in turn increases the business volume of the cloud provider. As a whole, our newly proposed SLA negotiation method will improve the business revenue of the cloud provider and customer satisfaction of the cloud service user.

3.3 SLA Monitoring Framework

Monitoring an SLA means periodic testing of the SLO values to determine whether the involved parties are following the agreed terms or not. SLA monitoring is done in three different forms such as online monitoring, proactive monitoring, and reactive monitoring. Approach refers to the continuous periodic monitoring of SLO values. In proactive monitoring approach corrective actions are done before the detection of SLA violations. Here, the SLA negotiation is done immediately after service discovery, which ensures the smooth and continuous availability of the cloud service, according to the contracted terms of the pre-agreed SLA. Reactive monitoring refers to a different approach where one of the involved parties starts complaining to the monitor that some violation of the agreement is taking place. This reactive monitoring is advantageous because it gives an immediate response to SLA violations and also there is no overhead of continuous monitoring.

The problem of SLA violation is crucial in cloud computing because the cloud providers have to pay the penalty to their customers upon each violation and so it is necessary to have a provision for monitoring and detecting SLA violations. In this section we have presented our proposed SLA management system which monitors the SLA and detects SLA violation. In our management architecture, we have attached a monitoring module and have presented a proactive runtime approach for detecting SLA violation based on event monitoring. Runtime detection and prediction are appreciative because it yields high effect in adaptive resource allocation in autonomic resource management(Anithakumari and Chandrasekaran, 2015). The detection of SLA violation has been implemented through monitoring and analysis of runtime data is performed. The provision to perform monitoring, analysis and verification have been included in the prototype implementation of our architecture.

3.3.1 Overview

The overall architecture of our SLA management system is given in Figure 3.14. The measurement subsystem measures the SLA parameters and SLOs are extracted from this parameters. The measurement sub-system components can be part of the service provider or the service user, but the party providing measurement should be trusted by a other

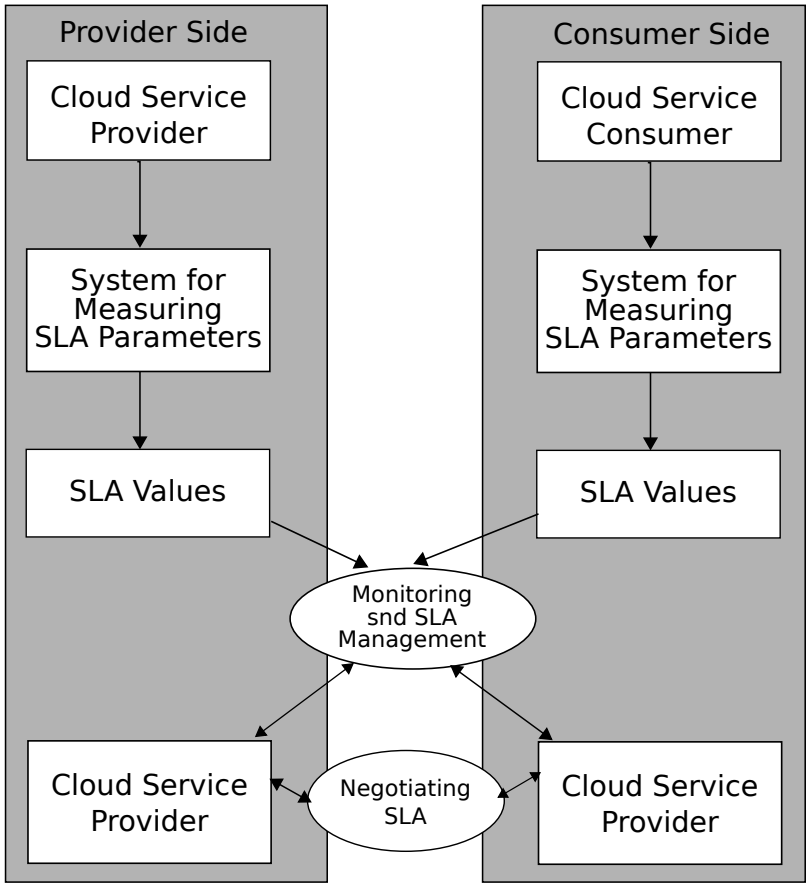

Figure 3.14 SLA management system

party. The measurement could also be partially or fully implemented by the third-party component running on a separate machine. The values of the SLA parameters are given as input for the evaluation procedure, which can run on either the service user or service provider or both on the service user and service provider. The evaluation procedure checks the values against the guaranteed conditions of the SLA. If any value violates a condition in the SLA, predefined actions are invoked.

3.3.2 SLA Monitoring Engine

The detailed view of monitoring engine is shown in Figure 3.15 (Anithakumari and Chandrasekaran, 2015). The monitoring engine mainly contains two different phases: the runtime monitoring of SLA parameters and the analysis and verification of SLOs. The runtime monitoring is co-ordinated by the monitoring manager who continuously monitors the application performance. The service management unit extracted the parameters correspond-

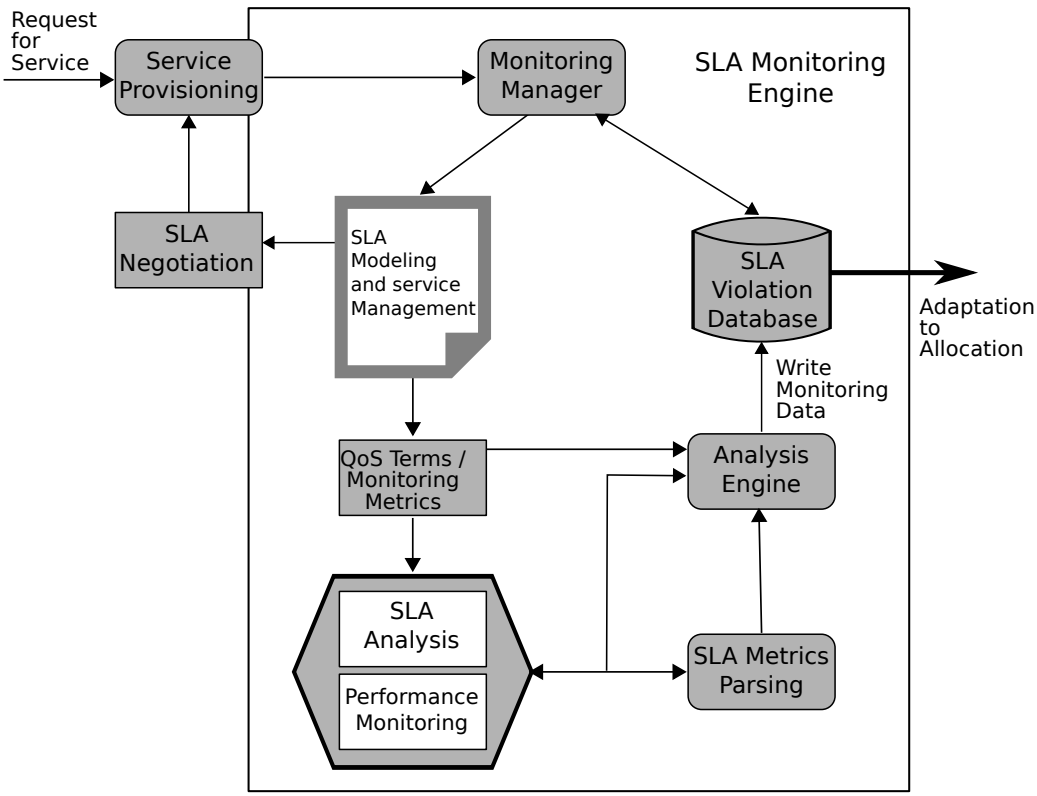

Figure 3.15 SLA monitoring engine

ing to QoS terms specified in the SLA and made it available to the next unit, the analysis engine. Some metrics require further refinement, and so they are forwarded to decoder unit, then to the parsing unit and then to the analysis engine. The analysis engine verifies the values through a matching between the needed value, i.e., the threshold limit, and the measured values. If the measured value exceeds the threshold limit, then it is identified as a violation, and the corresponding data values are passed to the violation database for next level processing. The data values contained in the violation database are decoded properly for making adaptive decisions at the time of adaptive resource management(Anithakumari and Chandrasekaran, 2015). This stored database is also useful for recalculating the value of SLA violation threshold at next iterations.

3.3.3 Detection and Avoidance of SLA Violation

The algorithm used for detecting SLA violations is given in Algorithm 3.

In Algorithm 3, **input** is the measured SLA values obtained from the monitoring tools and **output** is a *flag value* showing the status of SLA violation. The trigger signal used

Algorithm 3 Detection of SLA Violation

1. Let *response time* and *job execution time* be two SLO values.
Let *flag value* be a boolean value for controlling the trigger signal in adaptive resource allocation.
2.
   a. Initialize SLA violation limit and SLA threat limit.
   b. Reset the *flag value* to zero.
3.
   a. Monitor SLA parameters using the monitoring tool.
   b. Measure SLO values like *response time*, *job execution time* and *number of resources*.
4. Compare the measured SLO values with SLA threat limit.
5. If the SLO value $\geq$ threat limit then
   a. Detect it as a possibility of SLA violation.
   b. Set the value of *flag* to 1.
   c. Give trigger signal to dynamic resource allocation.

in Algorithm 5 for controlling adaptive resource allocation is operated by this *flag value*. The *flag value* represents the possibility of SLA violation. If the *flag* is set, the violation possibility is detected, and if the flag value is not changed, there is no possibility of a violation. The measured SLO values like 'response time' and 'job execution time' are compared with *threat threshold limits* of violation and if the SLO values are going beyond the threshold limits, it is identified as a possibility of a violation.

3.4 Dynamic SLAs

Dynamic SLA is an alternate form of SLA where the agreements are not static, don't have predefined bonds and can be modified before signing, negotiated on its content, and renegotiated if the customer and the provider want to modify it. (Comuzzi et al., 2009). The involved parties can concentrate on the customization of SLA template and agreement terms and can do negotiation and renegotiation with the use of dynamic SLAs. Negotiation and renegotiation is the stage where both provider and consumer try to settle with an agreement through structured message exchange. In this stage, both the parties try to apply their knowledge, expectations and economic axioms to maximize some utility functions (Comuzzi et al., 2009).

Agreement Terms Agreement terms gives a formal definition of the QoS properties of the agreed service. A small sample of the available QoS is considered in our context and major

terms include: Availability, Accessibility and Response time.

Availability: The service S is monitoring for a time t and during t, S is unavailable for a small break duration b then availability(A) of service S is measured as

$A = \frac{(t-d)}{t}$

Accessibility: The accessibility of a service operation can be measured after monitoring the number of all invocations to that operation and the number of dropped invocations. Let O be the operation of the service S monitoring for a time t, I_a be the number of all invocations to O during time t and I_d be the number of dropped invocations (means invocations that were not serviced) then accessibility for the service operation O represented as Ac is

$Ac = \frac{I_a - I_d}{I_a}$

Response Time: The Response Time of an operation can be measured by monitoring request message and response message or service message. Let O be the monitored operation of the service S monitoring for a time t and R_M, S_M be the request message and service message of operation O. To measure the response time we are assuming the request message R_M to be fully received at the providers end at time unit t_I and the provider placing the service message S_M fully on the path at time unit t_O. Then the response time t_R for the specified operation O is the difference between these two time units and the response time T_R of the entire service is the average of all individual operation's response time.

$t_R = t_O - t_I$

and $T_R = \frac{\Sigma t_R}{n}$

where n is number of operations in service S

3.4.1 Negotiation and Renegotiation

The SLA life cycle begins with the negotiation phase, where the service consumer and provider do message exchanges to agree on a well-defined set of guarantees. These guarantees denote the obligations between involved parties. The notion of dynamic SLA is possible through the multi-round negotiation process. The process will start with existing knowledge source, collected as part of the earlier consumption of the same service. The working conditions in the service-based systems change dynamically such as, the resources available at the time of negotiation may become unavailable or simultaneous usage of hardware or virtual resources generates dependencies among multiple SLAs of the provider. So

it becomes necessary to regulate, re-provision or renegotiate SLAs in due course.

3.4.2 Results and Analysis

Results are derived from a private cloud environment setup using opennebula where the monitoring is implemented by an open source monitoring tool Ganglia (Massie et al., 2004). The private cloud setup comprise of multiple host machines which are controlled by a front-end controller where each host is capable of generating multiple VM images. The SLA parameter considered here for detection of SLA violations is response time. The experiments are conducted on a private cloud set up using opennebula 4.6. The proposed system is being studied for appropriate implementation by deploying in Esper engine, Gmond module from Ganglia open source project. The processing and manipulation of SLA judgments documents. The management of SLA document is done using Domain Specific Language as the parameter extraction is done using XML parser. To implement the parameter mapping, Java Methods are used and the generated outputs will be forwarded to initiate the detection and violation. In the experimental set up we have created three vir-

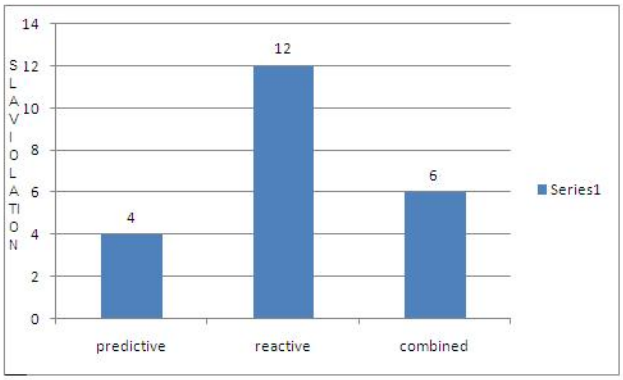

Figure 3.16 Detection and avoidance of SLA violation

tual machines in two different physical machines and uploaded web applications on it. We have measured the quantity of predictive, reactive and combined modes of SLA violations over a fixed time span of 2 hours. Predictive mode means detection of the SLA violation, reactive mode means avoidance of SLA violation and combined mode means the combination of both SLA detection and SLA avoidance. The obtained results are shown in Figure 3.16. which clearly shows that proposed approach is better.

3.5 Summary

From the review of the existing literature and the investigation of previous research works, we have realized the true significance for a clear and formal policy for handling SLAs in the context of cloud computing. In this work, we have proposed a reliable and flexible architecture for SLA management. In our negotiation environment, the negotiation is initiated by the third-party support and is implemented as a host support service. The general environment implements this with a third party agent, and some cloud providers itself have the provision to do this.

The key observation we have made in this situation is that there is a scarcity of standardized protocols and templates for cloud providers or services. This is very much essential when we think of monitoring and analyzing the cloud service provisioning. In our SLA management architecture, we have invoked a middleware interface to cater to the need of the lack of standard set of metrics which influence the monitoring of SLAs across cloud providers. Some efforts to standardize the services of the cloud have started already, and still, it is in the infant stage. We also emphasize the importance of similar attempts in the context of SLA monitoring and management. In the next chapter, we have explored the possibility of implementing dynamic resource allocation which is the primary focus of cloud computing. In our thesis work, it is attempted to approach resource management differently by utilizing the effects of SLA monitoring.

Chapter 4

ADAPTIVE RESOURCE ALLOCATION

The work in this chapter corresponds to the work for Objective two. The aim of this objective is to design and implement an Adaptive Resource Management framework in a real-time cloud. This objective works over the framework developed in objective one. The adaptive resource management framework manages the resource in a dynamic manner and is realized using Open Nebula Private Cloud. The results show this approach to be better than other approaches.

4.1 Introduction to Cloud Providers' Data Center

Cloud computing environment is generally viewed as a multi-layer arrangement containing different layers such as cloud provider layer, cloud user layer and end user layer (as shown in Figure 4.1). Cloud provider layer describes the infrastructure arrangement and server organization at the cloud providers' data center. The end-user layer includes the end users, and the cloud user layer describes the interface between cloud provider layer and end user layer. For dynamically addressing the resource allocation problem we make use of the infrastructure organization at the cloud provider layer. The cloud provider layer contains cloud providers who provide multiple computing resources as services through a shared data center. These computing resources can provide performance isolation and efficient resource sharing through virtualization technology, as per the basic feature of cloud computing. Virtualization technology helps to improve the efficiency of allocating physical resources to processes since it allocates resources to applications without considering other applications' workloads. So the individual application gets the resource capacity in a maximum possible manner. Virtualization also helps to provide the maximum elasticity to computing resources by implementing flexibility of virtually growing or shrinking the quantum of resources. Here we have proposed criteria to find out the quantum of resource capacity each VM is getting from the corresponding physical machine. This decision algorithm is explained in Section 4.2. An intermediate virtualization layer does the creation of multiple virtual images from the available physical resources and the maintenance of

these virtual images. This layer generates multiple virtual machines, on top of the physical infrastructure layer, which is isolated from one another and is capable of serving individual applications. So these applications are also isolated like running on a dedicated machine, and it uses a fraction of the entire resource capacity.

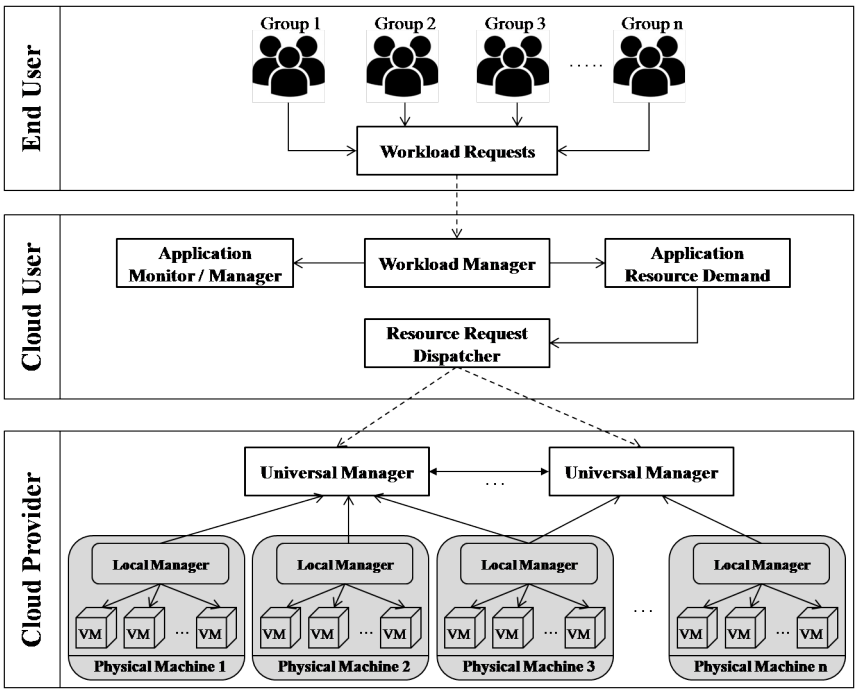

Figure 4.1 Over all architecture of a data centre

In order to proceed with the mathematical calculations, we assume a standard structure for a cloud provider's data center such as: The data centre contains a total of P physical servers and the maximum possible VMs that can be created by all servers is taken as V. The set of applications processed by i^{th} server is taken as S_i, and the number of VMs created on i^{th} server is taken as n_i. That is $\sum_{i=1}^{P} n_i = V$. The system model for dynamic resource allocation and VM management within a single server machine is discussed in section 4.2 and the model for global resource management, by considering all the servers in the data centre, is explored in section 4.3.1.

4.2 Modeling of Private Cloud

This section discusses the dynamic VM management in a single server machine by considering the server machines in a cloud provider's data center where each VM is viewed as a single computing machine and makes use of an admission control policy for allotting or dropping incoming requests. As per the admission control policy, the VMs may lose

some of the incoming requests, because of the limitations in capacity and excess count in incoming requests. This control policy helps to address the remaining requests without affecting the assured QoS guarantees.

We have assumed a system model for adaptive VM management in a single server machine as explained below. The primary component in this model is the *allocation management* module which is to take care of all incoming workload requests and to service these requests by considering system characteristics, application's properties, VM availability and SLA contracts for maximizing the revenue of the service provider. The *allocation management* module has configured with quantitative measures of the application and SLA metrics. These measures have updated according to application change or SLA change.

The *requested workload* module is for monitoring and reading the workload requirement of each application. It contains the provision to keep track of all processing applications and accordingly predicts the workload required for the current scenario. This input is forwarded to the *allocation management*. The *allocation management* decides on the allocation decision and in consultation with the *middleware control*, initiate the virtual resource mappings and generate VMs in the *virtualization layer*. The admission control policies are also taken care before the allocation of the VM images. The *allocation management* decision is based on an optimized performance model which considers SLA parameters and workload conditions of the processing applications.

The adaptive resource allocation has been implemented by making many resource allocation decisions. The interval between two adjacent decision values is viewed as *decision interval* and this can be taken as a constant value or variable according to the characteristics of the system. By choosing a smaller value for decision interval, we can make a more accurate resource allocation in a single server system. The primary component in the system model is *allocation management* module because this is the module responsible for making resource allocation decision. The resource allocation decision is determined based on an optimization model by considering performance and efficiency values.

Resource Allocation Decision

In our analytical system model, the resource allocation is controlled by SLA parameters and some system parameters. The estimated measures calculated from SLA parameters and applications' workload also play a role in this decision making. The service level efficiency in cloud computing is very much dependent on SLA parameters, and so the SLA performance which is related to the VM's ability to service applications by satisfying the application's response time specified in the agreement. The SLA parameters that

we are mainly focussing are throughput(T^{TH}), response time threshold (R^{TH}), probability values(P()). The system parameters are the total numbers of VMs(V) created by the virtualization layer, the utilization value(u) of each VM and the service time average(S) of the application on a physical server. Among these values, the utilization value we are seeing as the maximum possible value provided by the service provider and the throughput value is the maximum throughput limit or throughput threshold(T^{TH}).

The *allocation management* component gets an estimate a_i (arrival rate of requests) from the *requested workload* component for each application during the next controller interval. If there occur some deviations in arrival rate from the estimated value, then a load optimizer unit is initiated, and the deviations have optimized with the optimizer unit. Here a_i is the rate of arrival requests over the considered controller interval. From these arrived requests some may be rejected because of resource limitations, and so the actual arrival rate becomes lesser than a_i. Among the processed set of requests, some may violate the agreed response time values, and so they are not taken for the calculation of actual throughput, T_i.

In case of fixed controller intervals the events which are considerably smaller than the controller interval could mislead the allocation manager such as bulk quantum of requests (with less duration). These requests coming from some applications can stop the allocation of resources to some other class of applications because of deficiency in resource availability and this will lead to substantial penalties for the provider. For minimizing this undesired effect, the *requested workload component* provides the estimated probability (P_i) of a class of requests having higher arrival rate for the next controller interval. The parameter P_i is to represent the certainty level to assure maximum profit to the provider for VM_i, and it can be bypassed, to consider workload changes, by assigning a value 1.

To proceed with the analytical model, we assume that the application coming from a user is a unique entity and is submitted to particular VM. This application coming from customer i is submitted to VM_i which is serviced in a mean service time S_i and the utilization value (upper limit) of VM_i is u_i. In this model each VM is eligible for a guaranteed fraction of available physical server and so we estimate the average service time by taking f_i, the fraction of service time given to VM_i, as S_i/f_i. Correspondingly, resource allocation decision is the decision making of allocation fraction f_i (i=1,2...V) given to each VM_i. So f_i is viewed as the important decision variable in our resource allocation problem and analytical system model.

Model for Optimization

The optimization model estimates the resource allocation decision for maximizing cloud provider's profit by utilizing predicted workload in each controller interval. The model is expressed as an objective function which calculates the overall sum on all expected applications. The model is,

$$maximize(\sum_{i=1}^{V} c_i T_i - d_i X_i P(X_i)) \tag{4.1}$$

where,
V - total number of VMs allocated,
c_i - cost of resource the consumer has to pay for a unit of throughput
d_i - penalty cost the provider has to pay for a unit of throughput on SLA violation
T_i - throughput limit as per SLA
X_i - throughput penalty due to SLA violation which can be calculated as,

$$X_i = \begin{cases} T_i - T_i^F, \text{ if there is SLA violation} \\ 0, \text{ otherwise} \end{cases} \tag{4.2}$$

where,
T_i^F is the upper limit of throughput at the time of SLA violation and is always lesser than $T_i, ie., T_i^F < T_i$.
$P(X_i)$- Probability of throughput penalty on SLA violation
The model for optimization is designed by considering the following approximations and constraints:

- Predicted arrival rate of requests, a_i, includes both serviced requests and rejected requests. As per admission control policies (Perros and Elsayed, 1996), some of the resource requests get rejected and the remaining only get serviced.
  ie, $a_i = a_i^{adm} + a_i^{disc}$ where, a_i^{adm} is the share of admitted requests and a_i^{disc} is the share of discarded requests.

- Probability of getting more response time is within the limit of an assumed probability value α_i
  i.e., $P(R_i{}^F > R_i) \leq \alpha_i$ where, $R_i{}^F$ is the response time on the occurrence of an SLA violation and is always greater than response time as per the negotiated SLA. α_i is the assumed limit of probability of being $R_i{}^F > R_i$.

- The VM utilization λ_i is the ratio of arrival rates admitted, a_i^{adm} and a_i^{sat} (a small part of arrival rate for which i^{th} VM is saturated). VM utilization can be represented

as a function of rate of acceptance of arrived requests and the fraction of resource capacity VM gets.
ie., $\lambda_i = g(a_i^{adm}, f_i)$ and $\lambda_i = a_i^{adm}/a_i^{sat} \leq u_i$. This constraint assures a guaranteed stability condition by limiting λ_i to u_ix100%.

- The resource allocation capacity can be formally represented as $\sum_{i=1}^{V} f_i = 1$. That is the total sum of the assigned allocation percentage is limited by 100.

The rate of accepted arrival requests is related by expected arrival rate, satisfied response time and the maximum permitted utilization rate. The set of all considered domain variables in this model is restricted with normal boundary conditions such as: (i) $0 \leq f_i \leq 1$ (ii) $\lambda_i \geq 0$ (iii) $X_i > 0$ (iv) $a_i^{adm} \geq 0$ (v) $a_i^{disc} \geq 0$ and (vi) $a_i^{sat} \geq 0$. The evaluation of the above described optimization model is done with the help of queuing techniques.

Performance Prediction for the model

Determining estimated probability distribution of response time is the most critical task in performance prediction because the actual probability distribution can only be calculated for some particular types of queuing models and these model do not always fit for considered system constraints. So we assume the following set of approximations for the purpose of analysis.

1. Resource requests from applications are in Poisson rate of arrival and the service time of applications are exponentially distributed.

2. Two types of queues for modeling the system such as M/M/1 queue (works with FCFS scheduling) and M/G/1 queue (works with round robin scheduling and process sharing).

The saturating throughput T_i of VM_i is

$$T_i = \frac{\lambda_i f_i}{E[S_i]} \leq \frac{f_i}{E[S_i]} = a_i^{sat}. \tag{4.3}$$

where,
$E[S_i]$- expected service time
f_i - fraction of service time given to VM_i
λ_i - ratio of arrival rates
a_i^{sat} - part of arrival rate for which i^{th} VM is saturated

This saturating throughput is determined by applying Utilization Law (Menasce et al., 2004) in the system model. The concept of saturating throughput gives some restrictions

to the rate of acceptance of resource requests such as,
$a_i^{adm} = \frac{u_i f_i}{E[S_i]}$ and is mentioned as a constraint in our optimization model.

The tail probability distribution of the response time is calculated using Markov's Inequality Papoulis and Pillai (2002) as the first approximation. Expected response time of the applications is determined as $E[R_i] = \frac{E[S_i]/f_i}{(1-\lambda_i)}$ using the principles discussed in Kleinrock (1975). So for assuring the stability condition, the tail distribution response time is approximated as:

$$P({R_i}^F > R_i) \leq \frac{E[R_i]}{R_i} = \frac{1}{R_i} \frac{E[S_i]}{f_i - a_i^{adm} E[S_i]} \leq \alpha_i \tag{4.4}$$

where,
R_i^F - response time with SLA violation
R_i - response time without SLA violation
α_i - upper limit probability for getting more response time
f_i - fraction of service time given to vm_i
$E[S_i]$- expected service time
$E[R_i]$- expected response time
a_i^{adm} - share of admitted request

This tail distribution equation is acceptable to both types of queues (M/M/1 and M/G/1(PS)) and it gives a loose bound on the values. Certain improvements on values by using some tighter bounds is possible with another approximation, Chebyshev's Inequality Papoulis and Pillai (2002) where the result is influenced by both variance and average of response time. Here variance of response time depends on type of queue. In M/M/1 Kleinrock (1975),

$$Var[R_i] = [\frac{E[S_i]/f_i}{(1-\lambda_i)}]^2 \tag{4.5}$$

and in M/G/1(PS)Yashkov (1987)

$$Var[R_i] \leq \frac{\lambda_i (E[{S_i}^2]/{f_i}^2)}{(1-\lambda_i)^3} \tag{4.6}$$

For this equation it requires both first and second moment of the service times and here we assume like, this can be measured through a pre-production execution. With this average and variance of the response time, Chebyshev's Inequality limits the probability as:

$$P({R_i}^F \geq R_i) \leq \frac{Var[R_i]}{(R_i^F - E[R_i])^2} \leq \alpha_i \tag{4.7}$$

Now we get more precise bound than the previous case and require additional information like the queue type and second moment of the service time.

The next approximation is by considering the percentile case for calculation in M/M/1 queue. The tail distribution response time requirement can be calculated as per Kleinrock (1975) as:

$$P({R_i}^F > R_i) = e^{-R_i(f_i/E[S_i])(1-\lambda_i)} \leq \alpha_i \tag{4.8}$$

The values generated by this equation is accurate only when there is no rejection of application requests. But here we are preferring a general admission control policy and so this generates some approximations. The accuracy of each of these approximations (Markov, Chebyshev, Percentile) is studied in the experimental analysis.

4.3 SLA Aware Resource Allocation

This section presents the core concept behind the proposed adaptive resource management / allocation system based on SLA.

4.3.1 Global Resource Management

In this section we address the resource allocation problem with an eye towards maximizing the global utilization. The concept of global utilization is the overall utilization of all physical servers in the data centre environment. Our global utilization approach has built on utility function by using a vector based approach. For defining the global utilization value, we assume a general structure for the cloud provider's data centre as explained in section 4.1.

Here each server allots its resources for processing two types of application workloads such as: (i) addressing the incoming resource requests from different applications and (ii) executing received applications. So the workload in each server is measured as a consolidation of incoming requests and executing jobs or applications and the performance metrics are measured as response time R (by considering incoming requests) and throughput T (by considering executing applications). The workload intensity (I) on a server k is defined as,

$$I_k = \begin{cases} (a_{k,1}, a_{k,2} \ldots a_{k,S_k}) \text{and} \\ (j_{k,1}, j_{k,2} \ldots j_{k,S_k}) \end{cases} \tag{4.9}$$

where,

a_k - the average arrival rate of application S on the server k

j_k - number of applications run concurrently on the server k.

In a similar way, response time and throughput represented in vector form as,
$R_k = (R_{k,1}, R_{k,2} ... R_{k,S_k})$ and
$T_k = (T_{k,1}, T_{k,2} ... T_{k,S_k}$. These performance values for a single server can be measured in terms of workload on that server and the number of VMs generated on that server.
i.e., $R_k = f(I_k, n_k)$ and $T_k = g(I_k, n_k)$.
Now we can define the utility function F_k for the performance of a single server k on the basis of response time and throughput. The function is

$$F_k = \begin{cases} f(R_k) \\ g(T_k) \end{cases} \tag{4.10}$$

which can be re-written as

$$F_k = \begin{cases} f(f(I_k, n_k)) \\ g(g(I_k, n_k)) \end{cases} \tag{4.11}$$

With this utility function F_k we can compute the global utilization of the cloud provider's data centre, by combining the effects of individual utility functions and is represented as $G = h(F_1, F_2, ... F_P)$ where, P is the total count of servers in the environment. The notations f,g and h are used to represent functions. The following discussions give a clear idea about these functions and their computation.

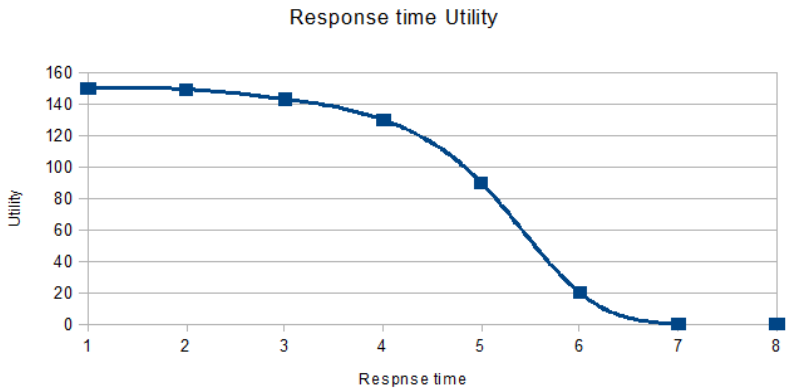

Figure 4.2 Response time utility value

Let us start with defining the different utility functions involved in this analytic model. The nature of response time utility function can be fixed as, the function should represent a decreasing utility with the increase in response time. Similarly the function for throughput utility should represent an increase in value with the increasing throughput. The response

time for application S on server k is set as $\alpha_{k,S}$ as per the SLA and so the utility should be sharper, near to this value. The utility function is represented as,

$$f(R_{k,S}) = \frac{L_{k,S}e^{-R_{k,S}+\alpha_{k,S}}}{1+e^{-R_{k,S}+\alpha_{k,S}}} \tag{4.12}$$

where,
$L_{k,S}$ - the scaling factor.
$R_{k,S}$ - response time for application S on server k.
This function has a variation point at $\alpha_{k,S}$ and decreases quickly after a particular response time value. Here we have shown a response time utility function (Figure 4.2) where the value $L_{k,S}$=150 and $\alpha_{k,S}$ is 4. The overall response time utility on a server k is calculated as the weighted sum of all the values.
i.e.,

$$f(R_k) = \sum_{j=1}^{S} f(R_{k,S})w_{k,S} \tag{4.13}$$

where,
$w_{k,j}$ - The weight of j^{th} application in server k, such that, $0 < w_{k,j} < 1$ and $\sum_{j=1}^{S} w_{k,j} = 1$.
S - Total number of applications.
Correspondingly we define the utility function by considering the throughput and the function is,

$$g(T_{k,S}) = L_{k,S}(\frac{1}{1+e^{-T_{k,S}+\beta_{k,S}}} - \frac{1}{1+e^{\beta_{k,S}}}) \tag{4.14}$$

where,
$L_{k,S}$ - the scaling factor
$T_{k,S}$ - Throughput for application S on server k
$\beta_{k.S}$ - Preset value of throughput

Similar to the response time case, here the utility function decreases to zero after a fixed minimum throughput. In the Figure 4.3 $\alpha_{k,S} = 150$ and $L_{k,S}$=3. As per this function the utility value is zero for zero throughput. The overall throughput utility on a server is now calculated similar to the response time utility.
i.e.,

$$g(T_k) = \sum_{j=1}^{S} g(T_{k,j})w_{k,j} \tag{4.15}$$

where,

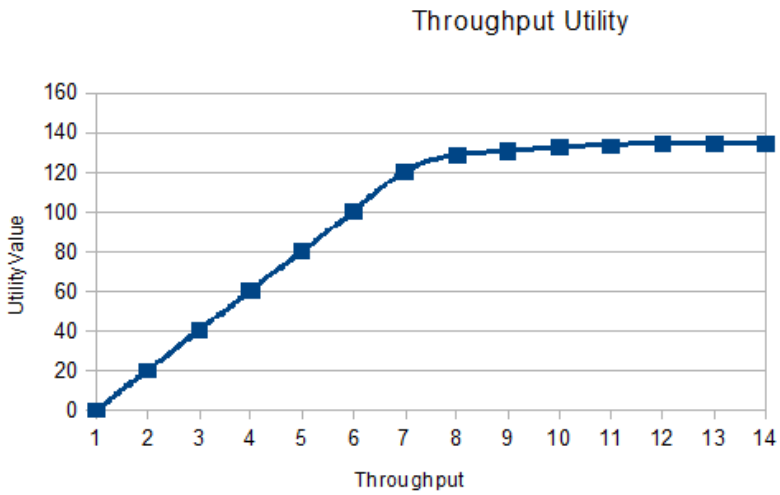

Figure 4.3 Throughput utility value

$0 < w_{k,j} < 1$ and $\sum_{j=1}^{S} w_{k,j}$=1.
In this adaptive resource allocation our focus is to dynamically allocate and reallocate VMs generated on different servers in the cloud provider's data centre in a more efficient way such that the global utilization of data centre is maximum.

In the next subsection we explain about the proposed mechanism of resource allocation for getting the optimized global utilization of servers.

Allocation Management: Global Model

We have already discussed about the resource allocation/management of a single server in section 3. Here we have to globalize the same mechanism for all the available servers in the environment and so we generalize the scenario as follows. The local manager (allocation management local to a single server) is responsible for collecting all measurements local to a server, computing local utility functions and then imparting these data to the global manager. The global manager is for coordinating all the local managers and is responsible for VM deployment and allocations.

Local manager: The detailed view of a local manager associated with a single server is shown in Figure 4.4. The workload predictor is to collect data about coming requests and predict workload intensity. The predictor take necessary data from workload predictions unit which is responsible for estimating the timely resource requirements in a single server. The workload predictions unit completes its job with the help of another unit named as resource request handler. The resource request handler is to monitor the current status of running applications and resource requirements in the local server. Now the

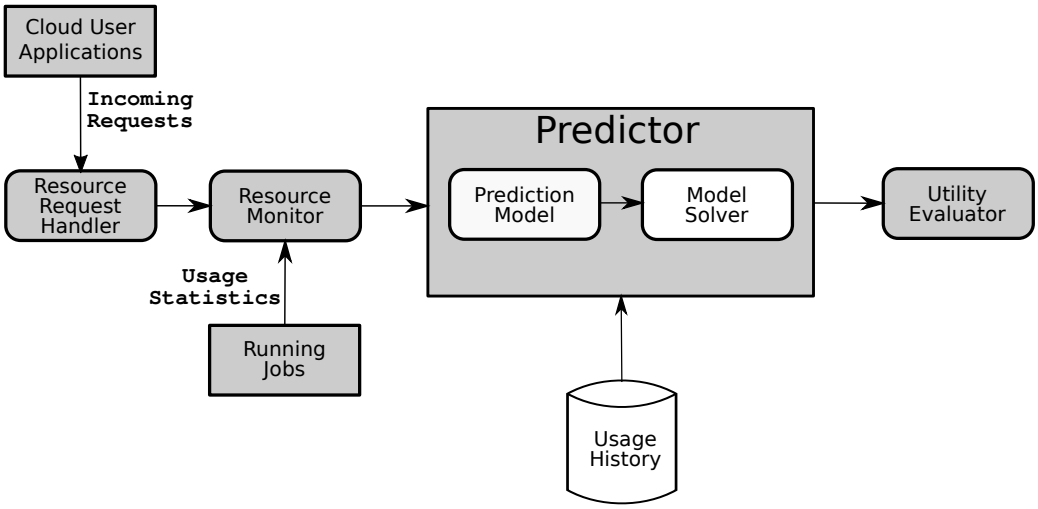

Figure 4.4 Local manager structure

resource monitor unit forwards the information about the estimated workload to the predictor unit. The predictor model, by using standard prediction techniques such as regression, predicts/forecasts about the expected workload.

Now the workload prediction unit forwards the information about the estimated workload to the predictor unit. The prediction model associated with the predictor unit now starts iterations on workload forecasting which takes the observed workload parameters and previous prediction effects. These predictions are processed in utility evaluator together with negotiated SLA values and the utility values are computed for a single server.

The detailed view of a local manager associated with a single server is shown in Figure 4.4. The workload predictor is to collect data about coming resource requests and to predict workload intensity. The predictor takes necessary data from resource monitor which is responsible for monitoring the resource requirement in a single server. The workload predictions unit completes its job with the help of another unit called the resource request handler. The resource request handler is to monitor the current level of resource requests, from user applications, within the current local system.

The prediction model in the predictor block will create some mathematical model for forecasting the future workload and forward it to the model evaluator. The evaluator now computes the workload intensity by using the measurements from currently running applications, coming resource requests and forecasted values. These workload intensities and SLA values are processed in utility evaluator and the utility function is computed as per Equation 4.11. For performance evaluation, the performance metrics such as throughput, response time etc. are taken instead of workload predictions. That is the local manager is responsible for reading the workload predictions and evaluating the performance condi-

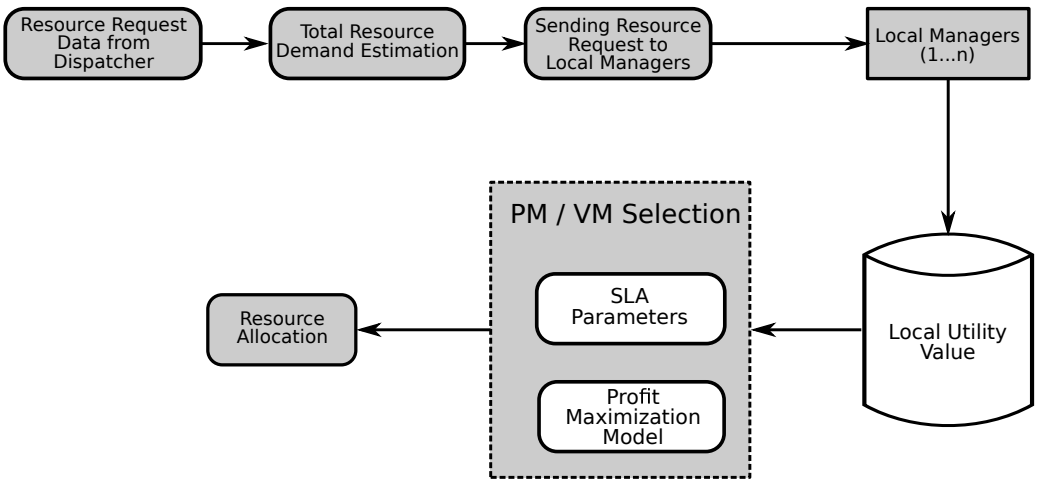

Figure 4.5 Global manager structure

tions purely local to a single server. The processing of a local manager is given in Algorithm 1. In a similar way the global manager takes care of the overall performance metrics and overall utility of the datacentre.

Algorithm 4 -Workload Prediction in a Local Manager

Input : Workload parameters and SLA parameters,
Output: Utility function values

Let n be the number of coming applications
for i = 1 to n do
begin
Monitor and Read the service requests coming form applications
Store workload parameters in the knowledge base
Predict the arrival rate of service requests from applications
(i) Use monitored workload values together with stored values
(ii) Apply statistical forecasting (regression) technique on stored values to get a prediction
Compute performance predictions with workload values and current VM configuration
Compute the utility function using performance predictions, SLA values and current performance parameters throughput and response time
end

The structural description of global manager is depicted in Figure 4.5. In the global manager there is control unit called global compiler which is to determine the control interval of the global manager. That is, it takes care of the time interval in which the universal manager algorithm should be initiated. In the case of fixed control interval, the algorithm should be executed in fixed intervals and in other cases it should be possible to act according to the changes/variations in utility function. The global manager algorithm

operates a combinatorial search on the configuration vector $n = (n_1, n_2, ...n_P)$ to determine the number of VMs to be created on each server. These numbers are communicated to each server machine k and in turn each server will communicate back the corresponding utility function U_k.

4.3.2 SLA Driven Resource Allocation

Adaptive resource management in cloud computing is strongly associated with SLAs. Monitoring and analysis of SLA parameters gives a clear picture of the timely requirements of the service consumer and if the resource allocation system is capable of accommodating the changes dynamically according to the changing expectations of the user, then the resource management system becomes cent percent adaptive to the environment. With the effective utilization of SLA for adaptive resource management, cloud computing community is getting several benefits:

- Active involvement of service users in service management **(user driven management)**
- Assurance of Quality of Service in service provisioning and utilization **(Assured QoS)**
- Self configuring and self organizing of computing resources and requests **(Autonomic management of resources)**
- Dynamic allocation of virtual machines to changing requests by using different partitions of same physical resources **(Sharing of resources through virtualization)**

For configuring the resources in an efficient and sensible manner, a service provider has to consider several facts into account such as: SLAs, the Business Level Objectives of the service provider, the current status of the system and the job complexity of the system. In Hasselmeyer et al. (2007) the authors described the details about how far SLAs control the resource configurations and cloud services. Usually the terms in SLAs can not be utilized directly for configuring the affected resources. The Service Level Agreement may contain a collection of abstract terms which mean different concepts to different providers. As an example, for the term "performance" different parties give different measures and so calculated and given in different ways according to the corresponding infrastructure. So for going towards the infrastructure layer, a mapping of high level terms to low level is necessary which creates another agreement called Operational Level Agreement (OLA). A simple sample of SLA objectives (Table 4.1) and corresponding mapping rules (Table 4.2) are shown below for getting a clear idea of these parameters. Such a mapping is essential

for a service provider because he has to be aware of what he wants to give to the service requesters to satisfy like processing time, processing power etc. of an SLA.

Table 4.1 SLA parameters

SLA Parameter	Possible value
Incoming bandwidth(bandwidthin)	>10 Mb/s
Outgoing bandwidth(bandwidthout)	>12 Mb/s
Storage(S)	1024 GB
Availability(A)	>99.2%
Response Time(R)	0.01ms

Table 4.2 Mapping of resource metrics to SLA parameters

Resource Metrics	SLA parameter	Mapping Rules
Downtime,Uptime	Availability(A)	A=(1-(Downtime/Uptime))*100
Inbytes, outbytes, packetsize, bandwidthin, bandwidthout	Response Time(R)	R=Rin+Rout

The authors of Hasselmeyer et al. (2007) used an approach which integrates infrastructure and business-specific knowledge to do autonomic translation of SLA parameters into configuring information. Such an autonomic adaptation process may perform the infrastructure management during runtime. SLAs have very high effect on configuring systems, particularly in case of dynamic on-demand service provisioning. This simplifies selection and formation of SLAs, by using a protocol and SLA representations and makes the usage of database to accommodate configuration information. But, studies of real business use cases revealed the fact that pre-defined configurations can only be used in a restricted category of scenarios, because the service provider's system configuration does not only depend on the SLAs, but also on the job types and the current workload.

Business Level Objectives such as *utilization of resources to a hundred percent* or *get the maximum profit while spending as little money as possible*, are not provided by available resource management systems. Additionally, a certain degree of knowledge of the

service provider's infrastructure is also required for the better usage of resource management. As a consolidation, a system that builds on a "base" configuration represented by BLOs and the complexity analysis of a job seems like a promising approach towards SLA-supported resource management.

The cloud service provider has to assure the availability of cloud service in line with the corresponding SLO value mentioned in the SLA for avoiding potential penalties owing to SLA violations. This work (Anithakumari and Chandrasekaran, 2015) focusses on the management of resource allocation by providing additional images to executing jobs, which reduces the number of SLA violations in an opennebula cloud environment. The algorithm we developed for implementing adaptive resource management in cloud computing is depicted here as Algorithm 5. Algorithm 3, described in previous chapter, discusses the monitoring and analysis of SLA values and Algorithm 5 concentrates on the adaptive resource management. In this adaptive resource management SLA monitoring and analysis are done as a first step, already explained in Section 3.3 and as per the results of the analysis the next steps are done for resource allocation and management.

Algorithm 5 -Adaptive Resource Allocation

1. Read the availability of resources like CPU, RAM, Memory, VM image, Network etc.
2. Allocate normal resources to the job.
3. Check whether any trigger signal at adaptive resource allocation.
4. If yes
   a. submit resource request to resource provisioning unit.
   b. Release and allocate extra resources by providing VM images.
   c. Update estimation of job execution time, response time and resource availability.
   d. Convey back to SLA negotiator and resource provider about the resource allocation/re-allocation.

Algorithm 5 takes the available resources like CPU, RAM, memory, VM images etc. as the input and gives the same set of resources (after utilization) as the output. This algorithm is operated with the control of a trigger signal. If the trigger is there then a re-allocation of resources is required and there is no re-allocation in the absence of a trigger. For allocating additional resources to jobs a request is given to the resource provisioning unit, which is in charge of resource allocation and re-allocation. After allocation the resource availability is re-estimated and conveyed back to SLA negotiator and Resource provider units. Time complexity of both algorithms is O(n) where 'n' is the number of jobs executed .

4.4 Experimental Results

This section presents the results of SLA monitoring together with the results of SLA driven resource management. The private cloud setup comprise of multiple host machines which

are controlled by a front-end controller where each host is capable of generating multiple VM images. The SLA parameter considered here for detection of SLA violations is response time. The experiments are conducted on a private cloud set up using opennebula 4.6. Computing services are hosted in virtual machines of multiple host machines and are submitted, monitored and controlled by using the GUI package Sunstone. Runtime monitoring and measurement have done by Gmond module provided by Ganglia monitoring tool. The Ganglia tool is installed on the front end controller which serves as cloud provider for the setup. The measured metrics are forwarded to event stream processing where Esper engine is utilized for event processing. The metrics and events are passed among different units using Java Messaging Service (JMS). SLA parameters and corresponding SLO values are extracted from the document containing negotiated SLA using an XML parser. Database management is established using MySQL database. We fixed the threshold limit, for the SLA parameter (response time) as 2 sec, for SLA violation and if the measured parameter values are going beyond this violation threshold limit, it is detected as a violation. Similarly for prediction we made the threat threshold limit as 1.7 sec.

The preliminary results obtained for detection and prediction of SLA violation is shown in Figure 4.6. The detection module counted the number of violations and predictions for several days and all the three parameters are depicted in the graph. The obtained results and the analysis on these results are as follows: If there is no SLA violation identified means all hosts have enough computing resources to process their applications without compromising the quality of service. But if there is any violation or possibility of violation it is purely an indication of the compromise on quality of service. For our proposed module, number of SLA violations obtained is between 20 to 25 and the number of predictions is 45 to 50 per month on an average. From this results, it is clear that our detection module has succeeded in predicting majority of SLA violations.

The results obtained for our adaptive resource management system is shown in Figure 4.7. The job execution time of 4 different jobs, allocated to different VMs, with adaptive resource allocation and without adaptive resource allocation are shown. Our adaptive resource allocation works well by taking less time for job completion in three cases and in one case it is taking more time than the normal approach. That is if we generalize the criteria, our algorithm is giving better performance for around 75 % of the total cases.

The overall results show that our SLA associated adaptive resource management system works well in private cloud environment and generates more efficiency to resource allocation and processing. So it can be applied to online resource allocation for any cloud environment.

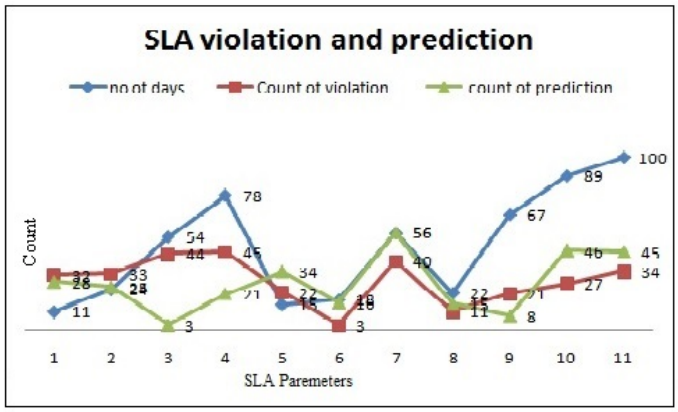

Figure 4.6 SLA violation and prediction

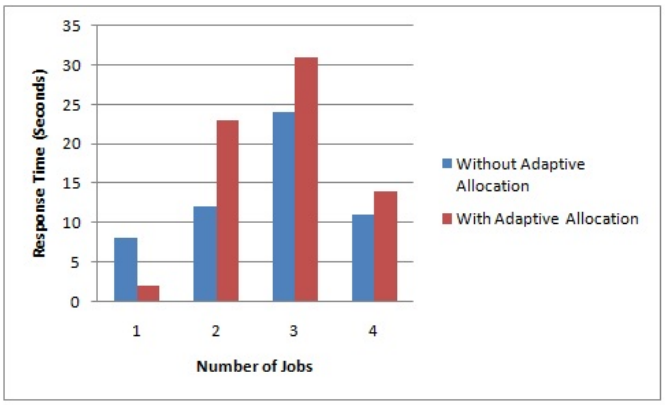

Figure 4.7 Adaptive resource allocation

4.5 Summary

We have introduced a resource allocation system that is aware of the varying resource requirement in each application and adapt the system in a dynamic way. The proposed system is ready to self-organize resource provisioning according to the changing demand. We take into account resource management issue of the hosted applications and objectives related to the provisioning of virtual machines to achieve separation of concerns on the basis of a distributed approach. An efficient way to express and quantify application satisfaction in resource management with regard to SLA is provided by using utility functions and is seen as an unbiased trade-off between multiple objectives like SLA violation and excess resource allocation which might conflict with each other. The next chapter of the thesis introduces the structure of a cloud SLA and describes the clustering of multiple cloud service providers by implementing an SLA matching algorithm.

Chapter 5

SLA MATCHING AND CLUSTERING

The first two objectives were collectively concerned about SLA operations on a single cloud service provider. Practically, a federated cloud is more prevalent than the private cloud and hence the objective 3 and 4 tries to design and implement the resource allocation mechanism in a federated environment. The objective three mainly concentrates on matching the available cloud service providers based on their SLA's. Based on the matching results these service providers are grouped together using a clustering algorithm. These clustered service providers are considered to be a group of service providers who share similar SLA's thus showing the way to interoperability. The implementation is done using CloudSim simulator and Python is used for clustering..

The matching of the SLA templates is performed with the use of a string similarity distance called Soft Term Frequency Inverse Document Frequency (TFIDF). After that clustering of different cloud service providers is done based on the mapped SLAs in the second phase with the help of k-means clustering technique. Finally, to optimally allocate the resources between different cloud computing providers in the same cluster we design a flexible resource management system with the use of population-based meta-heuristic approach like Adaptive Dimensional Search (ADS) in the last stage.

5.1 Matching of SLA Templates

Based on the SLA templates of different cloud providers such as public and private cloud automatic SLA mapping is performed with the help of the string similarity metric called TFIDF distance metric. From that mapped SLAs different clusters of cloud providers is formed. In a single cluster when the shortage of resources occurs with the private cloud, the resources of another public cloud present is allocated optimally through the ADS algorithm.

Normally the cloud service providers will provide the IT services to the users present

in the cloud based on the contract duly signed between them called as Service level agreements (SLAs) which includes the type of service provided by the service provider to the user. While accessing the services from the intended service provider such as a private cloud shortage of resources may occur, and this can be tackled with the help of an interoperable cloud computing environment. For that purpose, the SLAs of different clouds are mapped to form the clusters of clouds, and from that, the resources can be accessed by the one who met the resource shortage problem from the public cloud. The process of SLA mapping is done here with the help of Case-based reasoning (CBR) approach and the steps are explained below.

SLA templates of public and private cloud

In general any SLA template contains three parts, they are: i) SLO, ii) SLA metric and iii) SLA parameters. The SLO defines the type of the service requested from the user and the quantity can be measured by the specification from SLA metric. The SLA parameters represents the name or unit used to denote the SLA metric. The SLA templates of private cloud will be differ from that of public clouds in terms of either the SLA metric defined for the measurement of SLOs or the SLO itself. The general layout of the SLA template is given in the following Figure 5.1.

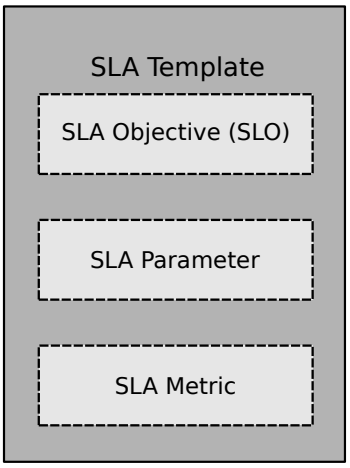

Figure 5.1 General layout of an SLA template

Mathematical Representation of SLA elements: The elements of the SLA templates can be represented mathematically from the representation given in Leymann et.al. (2008) as follows.

DEFINITION A Service Level Agreement can be represented as a tuple $SLA = (\Phi, SLO)$, where Φ - is the set of Service level parameters the SLA provides and SLO is a single objective.

The set of service level parameters can be given as in Equation 5.1 below:

$$\Phi = \phi_1, \phi_2, ..\phi_p \tag{5.1}$$

where p is the total number of service level parameters present in the available SLA templates.

The parameters of SLA are two types such as numerical value and the enumeration of parameter values. The numerical SLA parameter defines for example the availability or response time of the service and the enumeration gives the list of values the parameters used. Thus the types of SLA parameter can be represented as in Equation 5.2.

$$P \in \{Numerical, Enumeration\} \text{ with } m : \Phi \rightarrow P \tag{5.2}$$

where $m : \Phi \rightarrow P$ is the mapping which gives every parameter with the type.

DEFINITION Service Level Parameter (SLP) $\phi \in \Phi$ represents a solitary QoS property like availability. Every single parameter has a related description set D_ϕ which signifies all values ϕ can yield.

For example the description set for response time R_t is $D_{R_t} = [0, \infty]$. This represents that the response time for the given service may range from zero to infinity.

DEFINITION Service level objective (SLO) is a logical formulation comprises of logical operators and a set of guarantees. The operator set allowed is $\{ bigvee, \bigwedge, \rceil, \rightarrow, \leftrightarrow, (,) \}$. The SLA is satisfied, if and only if the SLO evaluates to true.

The formulation of a possible SLO is represented as in Equation 5.3.

$$SLO = (\nabla_{\phi_1} \bigwedge \nabla_{\phi_2}) \rightarrow \nabla_{\phi_3} \tag{5.3}$$

where ∇_ϕ is the guarantee of a service level parameter and is represented as in Equation 5.4 below:

$$\nabla_\phi = (g_\phi \in G_\phi) \tag{5.4}$$

If the service level parameter is of numerical type then the term G_ϕ can be represented as an interval given in Equation 5.5

$$G_\phi = [a, b] \tag{5.5}$$

After analyzing the templates of the public and private SLAs the mapping can be done by the finding the similarity between the templates. This is done with the help of a string similarity metric called as Soft TFIDF. The similarity metric is generally used to calculate

the distance between the strings in the templates by that the similarity between the templates can be found. The Soft TFIDF metric between strings S_X and S_Y in template X and Y is calculated using the below Equation 5.6 (Cohen et al., 2003).

$$SoftTFIDF = \Sigma_{w \in CLOSE(\Theta, S_X, S_Y)} V(w, S_X).V(w, S_Y.D(w, S_Y)) \tag{5.6}$$

where,
w - the string/word in the template whose similarity has to be found.
The best similarity distance with word w and S_Y is,

$$D(w, S_Y) = max(\lfloor sim(w, S_y)) \rfloor) \mid S_y \in S_Y \tag{5.7}$$

where,
$sim(w, S_y)$ - the secondary similarity distance metric.
The secondary similarity metric is calculated using the Jaro similarity distance. This distance for the strings S_X and S_Y is given in the Equation 5.8.

$$Jaro(S_X, S_Y) = \frac{1}{3}\left(\frac{|S_{X'}|}{|s_X|} + \frac{|S_{Y'}|}{|s_Y|} + \frac{|S_{X'}| - T_{r_{S_{X'}S_{Y'}}}}{|s_X|}\right) \tag{5.8}$$

where,
$S_{X'}$ - characters of S_X which are common in S_Y
$S_{Y'}$ - characters of S_Y which are common in S_X
$T_{r_{S_{x'}S_{Y'}}}$ - Half of the number of transpositions for $S_{X'}$ and $S_{Y'}$
The term V in Equation 5.6 is determined using Equation 5.9 given below:

$$V(w, T) = log(TF_{w,T} + 1).log(IDF_w) \tag{5.9}$$

where,
$TF_{w,T}$ - frequency of word w in string T
IDF_w - inverse of fraction of strings in the template that has w
Then the mapping is performed by defining the translation function in which the SLA metric in private template is translated into that of the public template. For example, if the SLA metric of private template X is represented in Mega Bytes and for public template Y is represented in Giga Bytes means then the translation function from template X to Y will be of the form,
Y=X*1000
For the large number of public and private templates this type of translation function is defined and the case will be used for the new mapping of private template and this is

discussed detail in the following section.

SLA matching by Case Based Reasoning Approach

In a cloud computing environment number of public and private clouds will be present and so the matching of SLA templates has to be done in an automatic manner. To accomplish this CBR methodology is used. The CBR approach will work on the principle of remembering the past identical situation and the reuse of that knowledge and information whenever a new problem arrives.

CBR: The process involved in the CBR approach contains four phases. They are: Retrieve, Reuse, Revise and Retain. In the retrieval phase the mechanism will retrieve the previous mapping cases and that cases will be used again, that is, the same solution used before for the templates of similar cases is reused in the reusing stage. Then the solution is revised by means of the feedback from the user. And based on the feedback the knowledge about the mapping is stored in the retain phase. The process of CBR approach in mapping the SLA templates is given in the following Figure 5.2

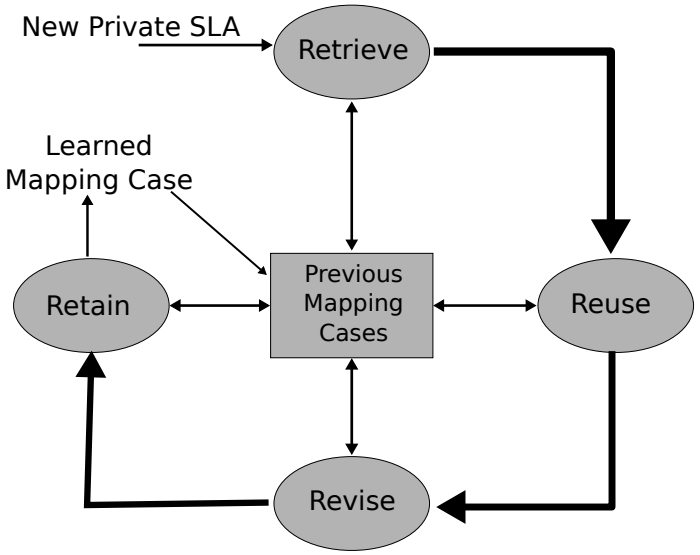

Figure 5.2 CBR approach for automatic SLA matching

As shown in the Figure 5.2 in the retrieve phase when a new private SLA template arrives the most similar SLA is retrieved from the previous mapping cases by finding the similarity metric between them through the Soft TFIDF metric. In which the template which is having highest similarity will be selected. And in the reusing phase, the translation function defined for that corresponding template is used again. Any feedbacks regarding the translation function definition is given in the revise stage and based on the feedback the experience is stored and used for learning process. Thus the SLAs are mapped in this approach automatically for different incoming newly arrived private SLA templates. Based

on the mapped SLAs clusters are formed in the next section.

5.2 Clustering of CSPs Based on Mapped SLAs

After mapping the SLA templates, based on the knowledge available from the CBR approach clustering of the cloud service providers is performed with the use of K means algorithm. Through this clustering method, finally the clusters of cloud providers is formed that are having the most similar SLA templates.

K means Clustering algorithm

K means algorithm is the generally used clustering technique to form different clusters with elements of similar nature and we used this algorithm here to form the clusters of different service providers as this is the simplest algorithm to produce the clusters with minimum overhead. The steps of the K means clustering algorithm is given in Algorithm 6.

Algorithm 6 K-means Clustering Algorithm

step 1: Select K number of initial centroids
step 2: Calculate the squared distance from each centroid to all other points
step 3: Assign the point to the centroid which is closest to them to form clusters
step 4: Update the centroids in each cluster if the centroids formed are similar to previous ones

Step 1. Centroid Selection: Here, based on the SLA mapping K number of centroids are initially considered and the centroids may refer to cloud service providers.

Step 2. Distance Measurement: In this stage the closeness of the centroid with that of other cloud providers is measured using Equation 5.6.

Step 3. Cluster Formation: Assign other cloud providers to the centroids whose SLAs are more similar with that of the centroid.

Step 4. Centroid Updation: After the cluster formation the centroids are again updated and it is represented mathematically as in Equation 5.10.

$$C_{new} = \frac{1}{N_c} \left(\Sigma_{X_i \in C_M} (X_i) \right) \tag{5.10}$$

where,
N_c - Total number of elements in cluster C_M and
$X_i - i^{th}$ element in the dimension D.

Step 5. Termination: Terminate the clustering if $C_{new} = C_M$, otherwise repeat the previous step.

Thus at the end of this, different clusters of cloud providers will be formed with similar SLA templates. Once the clusters are formed the flexible resource allocation between the clouds can be made by means of optimum resource allocation.

5.3 Experimental Results

The proposed methodology is implemented using the language of Java in the CloudSim tool of Netbeans IDE, Version 7.4, and using Intel i5 under a Personal Computer with 2.99 GHz CPU, 8G RAM and Windows 8 system. Here we are considering six private clouds and eight public clouds and with three SLA templates for each public cloud and two SLA templates for each private cloud for the mapping phase. The number of virtual machines considered are 500 to provide the resources to the requested cloud provider. Initially we perform the mapping of SLA templates of public and private clouds. Each of the new incoming SLA template is mapped automatically based on the available existing mappings. After that to enable the interoperability different clusters of cloud service providers are formed based on the SLA mappings obtained with the help of K-means clustering technique in which a single cluster contains number of cloud providers with similar SLA templates.

The performance of the proposed methodology is validated with the help of a number of evaluation metrics correspondingly to each of the processes involved. The effectiveness SLA mapping is validated with the help of overall net utility and overall cost (Breskovic et al., 2011b). Similarly the performance of the optimization algorithm we used here is validated by varying the SDR parameter in the algorithm. The effectiveness in the process of resource requests is analyzed with the help of Average Weighted Response Time (AWRT) and Slowdown (Javadi et al., 2012).

Overall net utility:

The utility function defines the utility of the user in utilizing the public SLA templates in mapping per single SLA parameter and the overall net utility can be calculated by summing the utility values for all the users for all SLA parameters. The utility function for a single user i for the single SLA parameter is calculated using the Equation 5.11.

$$f_i(U) = \begin{cases} 0, \text{if the parameter not exists in any SLA template} \\ 1, \text{if both parameters exist but differs in the SLA template} \\ 2, \text{if only one parameters differs in the SLA template} \\ 3, \text{if both parameters are same in the SLA template} \end{cases} \tag{5.11}$$

And the overall net utility is,

$$F(U) = \Sigma_{i=1}^{n} \Sigma_{k=1}^{m} f_{i,k}(U) \tag{5.12}$$

where,
i=1,2,..,n is the number of users and k=1,2,..,m is the number of SLA parameters.

Overall cost:

The overall cost defines the cost incurred in mapping single SLA parameter of a single user to that of the public SLA templates and is calculated by summing the cost value of all users for every SLA parameters. The cost function for a single user and the overall cost is given by the Equations 5.13 and 5.14.

$$f_i(C) = \begin{cases} 0, \text{ no new SLA mappings necessary} \\ 1, \text{ user create new SLA mapping for one or two parameters} \\ 2, \text{ user create new SLA mapping for the missing parameter} \\ 3, \text{ user modifies the SLA mapping} \end{cases} \tag{5.13}$$

Overall cost function is,

$$F(C) = \Sigma_{i=1}^{n} \Sigma_{k=1}^{m} f_{i,k}(C) \tag{5.14}$$

where,
i=1,2,..,n is the number of users and k=1,2,..,m is the number of SLA parameters.

The utility and cost values for the proposed SLA mapping technology with N (N=10) number of SLA mappings is given by the following Table 5.1 and represented graphically in the Figure 5.3 and Figure 5.4.

SLA mapping through Soft TFIDF metric with other SLA mapping techniques

Here the efficiency of SLA mapping by our proposed methodology with other existing works namely maximum method, threshold method and significant-change method (Breskovic et al., 2011b) in terms of the parameters such as overall net utility and overall cost. The

Table 5.1 Utility and cost values for SLA mappings (N=10)

User request	Utility Value	Cost Value
1	2074.02	2.5253
2	2404.02	4.5282
3	2854.02	6.4971
4	3840.10	1.5734
5	4052.94	4.8280
6	4500.91	4.1557
7	5161.02	3.1325
8	6039.91	1.1582
9	6750.10	2.0069
10	7627.94	4.2929

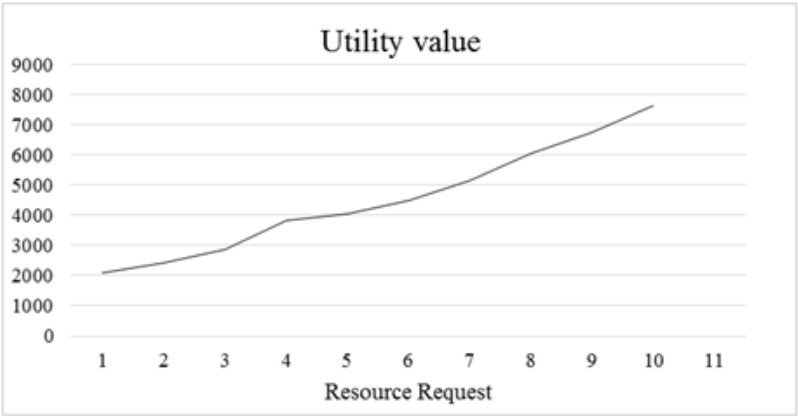

Figure 5.3 Utility values for N SLA mappings (N=10)

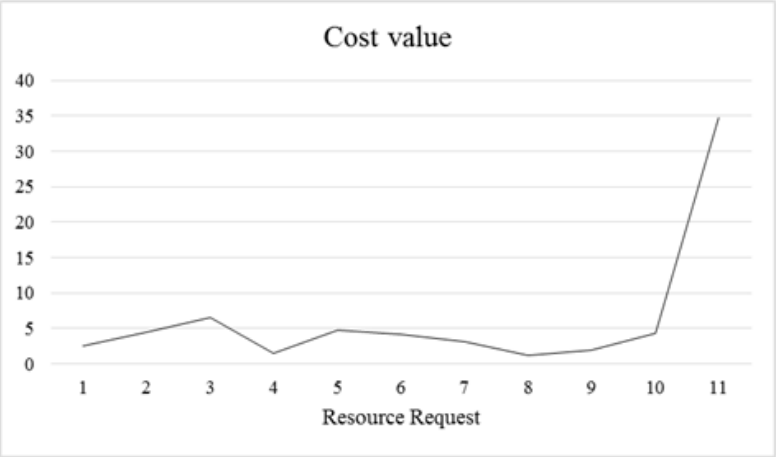

Figure 5.4 Cost values for N SLA mappings (N=10)

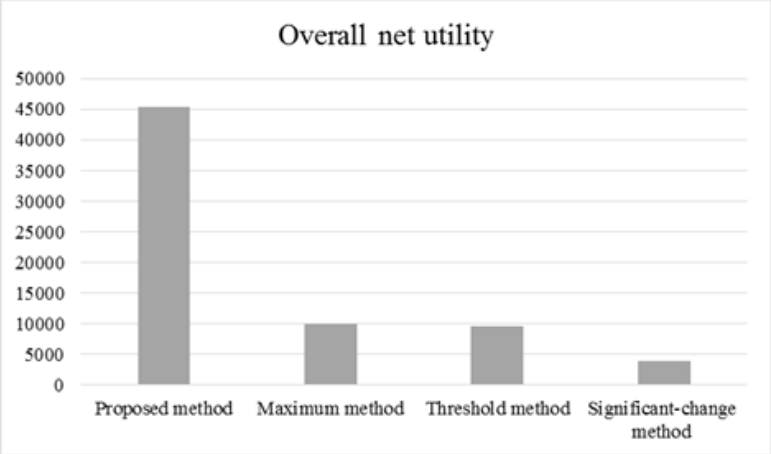

Figure 5.5 Overall net utility of proposed methodology with existing works

results of comparison are given in the Table 5.2 as well as in the Figures 5.5 and 5.6.

As seen from the Table 5.2, Figures 5.5 and 5.6 it is evident that the overall net utility of our proposed methodology is maximum when compared with other existing works and it ensures the better utilization of SLA mappings and also the overall cost is minimum. This shows the betterment of our proposed methodology in mapping the SLA templates.

Table 5.2 Comparison of SLA mapping

Methodology	Utility Value	Cost Value
Proposed	45304.98	34.6982
Maximum	9999.5265	100.56
Threshold	9548.456	238.145
Significant change	3846.15	345.45

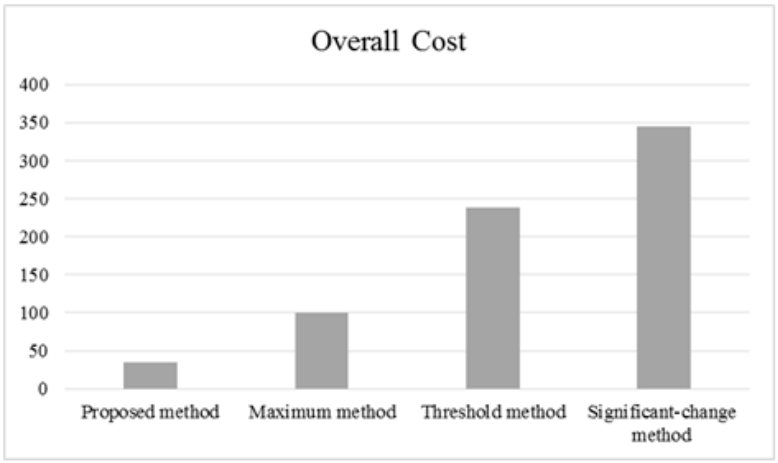

Figure 5.6 Overall cost of proposed methodology with existing works

Results for Clustering:

After mapping the SLAs we get a set of six values, i.e. properties, and the values for these properties are assigned on the scale of 5, where 5 is the best and 1 is worst. Once these values are obtained, clustering operation is done. The output of K-means clustering are clusters of cloud service providers who can be grouped together. The clusters are formed based on six values, i.e. six SLOs have been separately considered. Clustering for all the service providers has been done based on single objective at a time. Thus for each value the clustering mean distance will be different i.e. the average distance between clusters center point and clusters is different. The mean distance among each of the cluster will determine the best service attribute that can be used for selection of services. The six attributes that have been given are as follows:

- Availability
- Scale up
- Scale down
- VM Size
- VM Speed
- Response Time

The Table 5.3 mentioned here is obtained after the SLA mapping operations.The resultant is a value ranging from 1-5, where 1 is least preferred and 5 is the most preferred according to the SLOs. Based on the SLA mapping results the SLOs have been given values and consecutively clusters have been formed. The cluster formation is done according

Table 5.3 SLO range values

Sl No	**VM size**	**VM speed**	**Scale UP**	**Scale Down**	**Avail-ability**	**Response time**
1	5	4	4	3	4	3
2	3	5	3	3	3	5
3	4	3	4	5	5	5
4	2	5	5	5	5	5
5	1	4	3	3	5	4
6	5	3	4	4	5	5
7	4	2	5	2	4	3
8	3	2	5	1	4	4
9	2	2	4	5	4	5
10	3	3	3	4	2	3
11	4	5	5	3	2	4
12	5	5	5	2	2	2
13	4	2	5	4	4	2
14	4	4	4	5	4	5

to each and every attribute separately thereby having six cluster formation results. After this the best clusters have been chosen. Here we have kept a standard K value as 3. The value 3 is considering the size of the cloud (Number of VM's). Anything beyond this 3 size might have very less number of VM's in each cluster. The diagram depicted in the Figure 5.7, Figure 5.8, Figure 5.9, Figure 5.10, Figure 5.11 and Figure 5.12 show the clustering done for the service providers for each of SLO values viz. Availability, Scale up, Scale Down, Response Time, VM Size and VM respectively. Here each cluster is shown using different colour and the cluster head is shown using a black coloured star. The actual results obtained are in 1 dimension but these Figures are represented as 2 dimension for the ease of understanding. The Y axis of the graph represents the mapping values obtained for SLOs mentioned and X axis is marked according to the serial number of the values available as additionally this avoids overlapping of same values. Again, as mentioned the values for all SLOs range from 1-5, where 5 denotes the most preferred service provider and 1 denotes the least preferred service provider for that particular SLO.

To determine the efficiency of cluster we have calculated the average distance of the centroids from the mean point. The more the average distance we are considering it be a better cluster.

Cluster according to Availability:

Here the clustering is done according to single SLO of availability, shown in Figure 5.7.
The cluster heads are formed at the following points: 1, 5 1, 2.25 1, 4
Here the average distance is 1.00.

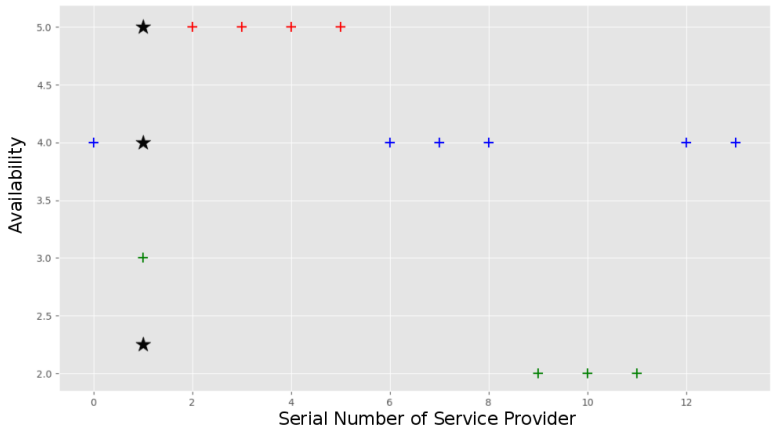

Figure 5.7 Clusters according to Availability

Clustering according to Scale up:

Here the clustering is done according to single SLO of scale up, shown in Figure 5.8.

The cluster heads are formed at the following points: 1, 5 1, 3 1, 4
Here the average distance is 0.667.

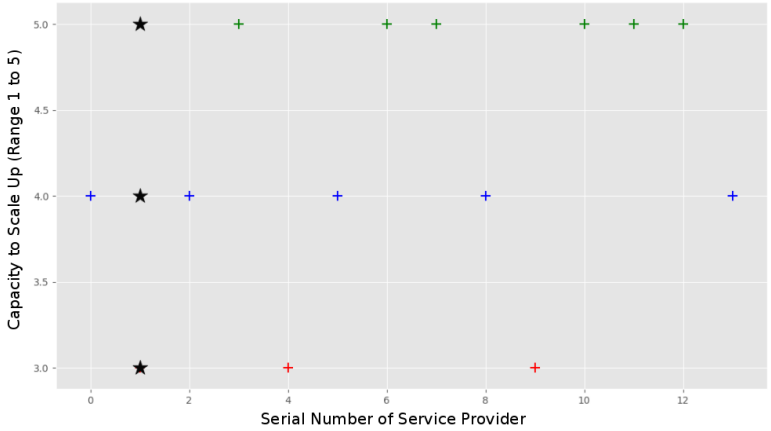

Figure 5.8 Clusters according to Scale Up Capacity

Clustering according to Scale Down:

Here the clustering is done according to single SLO of Scale Down, shown in Figure 5.9.
The cluster heads are formed at the following points: 1, 3.428 1, 5 1, 1.667.
Here the average distance is 1.132.

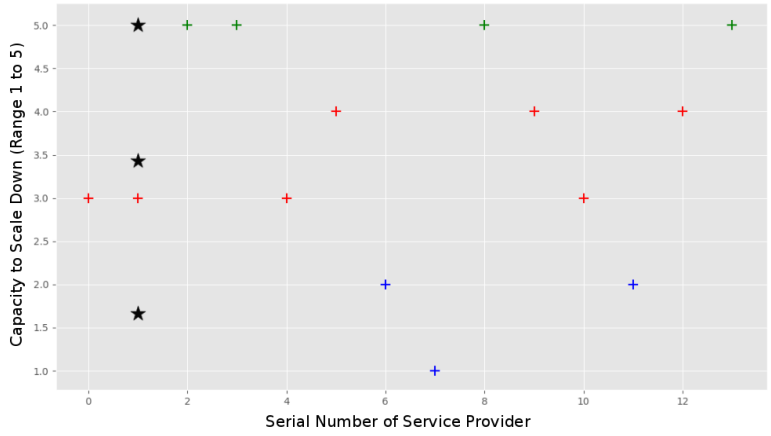

Figure 5.9 Clusters according to Scale Down Capacity

Clustering according to Response Time:

Here the clustering is done according to single SLO of Response Time (Figure 5.10).
The cluster heads are formed at the following points: 1, 5 1, 2.6 1, 4
Here the average distance is 0.844.

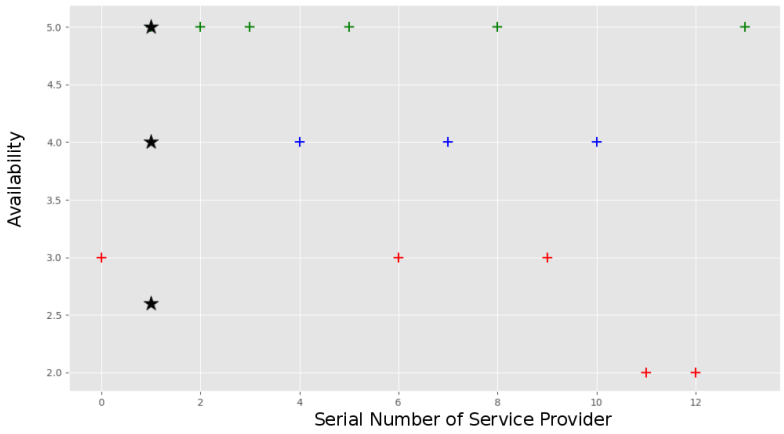

Figure 5.10 Clusters according to Response Time

Clustering according to VM Size:

Here the clustering is done according to single SLO of VM Size, shown in Figure 5.11.

The cluster heads are formed at the following points: 1, 1.667 1, 4.375 1, 3

Here the average distance is 0.907.

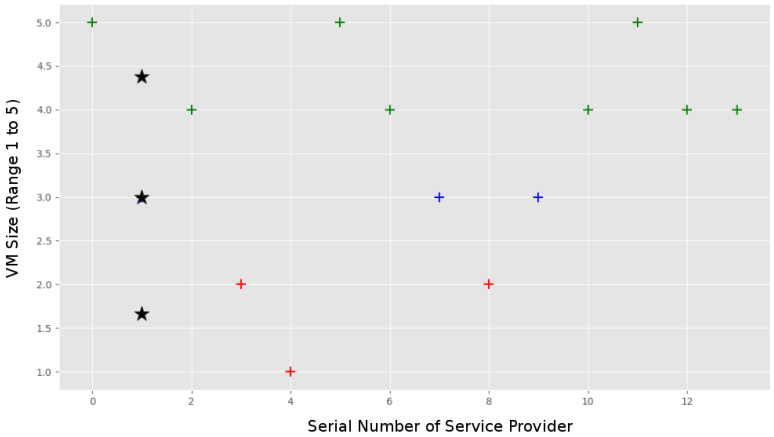

Figure 5.11 Clusters according to VM Size

Clustering according to VM Speed:

Here the clustering is done according to single SLO of VM Size, shown in Figure 5.12.

The cluster heads are formed at the following points: 1, 3.5 1, 5 1, 2

Here the average distance is 1.00.

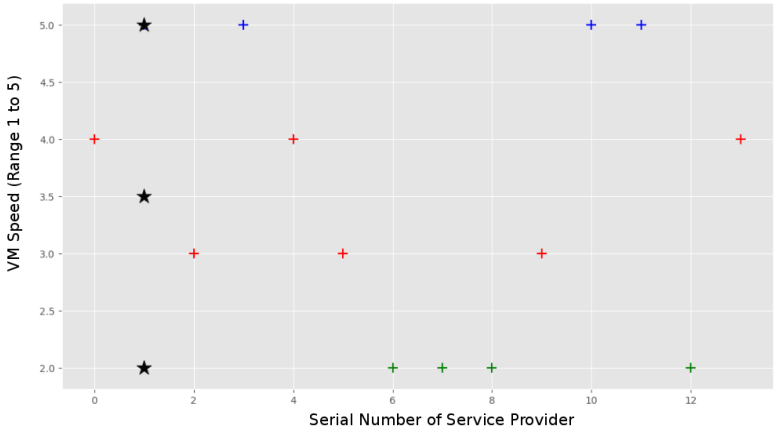

Figure 5.12 Clusters according to VM Speed

5.4 Summary

Here we have proposed an interoperable cloud computing environment based on SLA mappings of public and private clouds through CBR approach to achieve a flexible resource allocation mechanism. First we have mapped the public and private SLA templates with the help of a string similarity metric called Soft TFIDF an effective metric used to find the similarity between set of strings. And this knowledge is used in the automatic mapping of SLA templates by CBR approach. After creating the mapping the clouds that are having similar SLA templates are grouped into same cluster with the use of K-means clustering technique.The performance evaluation of the proposed methodology is individually as well as comparatively shown in the results section with other existing works available for the SLA mapping. The parameters used to show the effectiveness of our proposed methodology are Overall net utility, overall cost in mapping the SLA templates. The next chapter of the thesis introduces the basic concepts of inter-operable cloud computing and the implementation of flexible resource allocation in an inter-cloud environment.

Chapter 6

SLA BASED CLOUD FEDERATION

The fourth objective is an extension of objective three. Here the grouped service providers are shown to be interoperable by Designing and Implementing a framework which demonstrates interoperability between them. Here Adaptive Dimensional Search (ADS) with Inter-Cloud Gateway is used for showing interoperabilty, deployed over Java based CloudSim. The results prove that this method is better than other approaches.

The proposed solution for interoperability for federated cloud contains three phases: First, an automatic SLA matching is performed to form an interoperable cloud environment by considering the SLA templates of different cloud service providers. The matching of the SLA templates is performed with the use of a string similarity distance called Soft TFIDF. After that clustering of different cloud service providers is done based on the mapped SLAs in the second phase with the help of k-means clustering technique. Finally, to optimally allocate the resources between different cloud computing providers in the same cluster we design a flexible resource management system with the use of population based meta-heuristic approach like ADS in the last stage. The first two phases such as: (i) matching of SLA templates and (ii) clustering of cloud service providers are explained in the previous chapter and the work regarding the the third phase named flexible resource allocation is described here in this chapter.

This chapter is a logical extension of the work done in chapter 5. In chapter 5, SLA mapping of all Cloud Service Providers has been done and based on the results the clusters have been created. Here the work is done on the created cluster and a method is proposed to make it interoperable. Here a method is proposed to share the resources in case of urgent requirement i.e. the cloud clusters that have been considered are clustered based on the similarity between them, thus these service providers are the most suitable group of service providers who can share there resources.

This chapter describes the concept of proposed adaptive resource allocation in an interoperable way. Before going to the details of adaptive resource allocation, the concept of

interoperability and interdependency is explained in initial sections. In cloud computing model multiple service providers are there to provide services and similarly multiple service users are there to use. Because of the presence of multiple cloud service providers the users should have the flexibility to change between different providers. This is based on their requirements, the inter-dependence (interoperability) between the parties and the service conditions specified as part of the SLAs. A cloud computing system mainly includes infrastructure, platform, application and data as components, where:

- Infrastructure is a set of resources used for communication, storage and computation.
- Platforms is the set of software programs which helps the applications to process.
- Application is various types of application programs for performing business problems.
- Data is an illustration of machine-process-able of information stored in computer.

Word interoperability conveys different meaning to different people in cloud scenario. In some context it means that some applications are able to move from one environment to the other, and these applications works exactly same in both the environments. Another might mean applications are running in various clouds and they can share information, which might require a common set of interface cloud interoperability focuses on the scenario where for variety of cloud computing platforms and providers, each customers can use the same management tools, server images and other software. Every cloud environment has one or more operating systems and databases. Every cloud may contain processes, hypervisors, security, a storage model, a cloud API, a networking model, licensing models and more. Rarely, do two providers implement their clouds in exactly the same way, with all the same moving pieces. This introduces the significance and need for interoperability.

The infrastructure, application and platform components can be as in traditional computing enterprises, or they can be cloud resources that are (respectively) software application platforms (PaaS), software application programs (SaaS), and virtual processors and data stores (IaaS). Data components can be interoperable via application components rather than directly. There are no data interoperability interfaces. The two important kinds of cloud computing interoperability under consideration are platform interoperability between PaaS services and platforms and application interoperability between SaaS services and applications. Programs concerned with the deployment, configuration, provisioning, and operation of cloud resources are comes under application. Management interoperability is defined as Interoperability between these programs and the cloud resource environments. It is very important and it also can define as the interoperability between

cloud services (SaaS, PaaS, or IaaS) and programs concerned with the implementation of on-demand self-service.

Here we have established a mechanism for flexible resource allocation between the cloud service providers based on SLA mapping and clustering techniques in the interoperable cloud computing environment through the use of ADS algorithm. The proposed methodology and the steps are detailed in the following subsections which contains the design and implementation of an adaptable resource management system with dynamic elasticity and automatic SLA mapping.

The resource allocation process we have discussed here is occurring at the time when one of the cloud provider in the cluster faces the problem of shortage of resources and this can be tackled with the help of Inter Cloud Gateway (ICG) present at each of the cloud which employs the optimum resource allocation mechanism by which the service provider will provide the resources.

6.1 Interdependency and Interoperability

In the standard cloud computing model, where the client only utilizes a single cloud data centre, it creates several challenges. The unavailability of cloud service can create many problems. It leaves customers relying solely on it and forbids access to essential resources.

The term Inter-Cloud has been described as a cloud of clouds (Cases, 2010), and a formal definition is provided in the following paragraph. Essentially, an Inter-Cloud allows for the dynamic coordination and distribution of load among a set of cloud data centres (as shown in Figure 6.1).

According to Cases (2010), Inter-Cloud computing has been formally defined as:
"A cloud model that, for the purpose of guaranteeing service quality, such as the performance and availability of each service, allows on-demand reassignment of resources and transfer of workload through a inter-networking of cloud systems of different cloud providers based on coordination of each consumers requirements for service quality with each providers SLA and use of standard interfaces."

This generic definition does neither specify who initiates the inter-cloud endeavour-the cloud providers or the clients nor does it specify the willingness of cloud providers to collaborate to form an inter-cloud. Cloud federation and multi-cloud are two more similar terms we are going to familiarize within this context. A cloud federation is formed when a set of cloud providers are interconnected and they permit to share their infrastructures and resources among each other (Ferrer et al., 2012). The multi-cloud refers to the handling of multiple, independent cloud services. They differ from federation in a way that, a multi-cloud atmosphere does not mean simply sharing of provider's infrastructures and

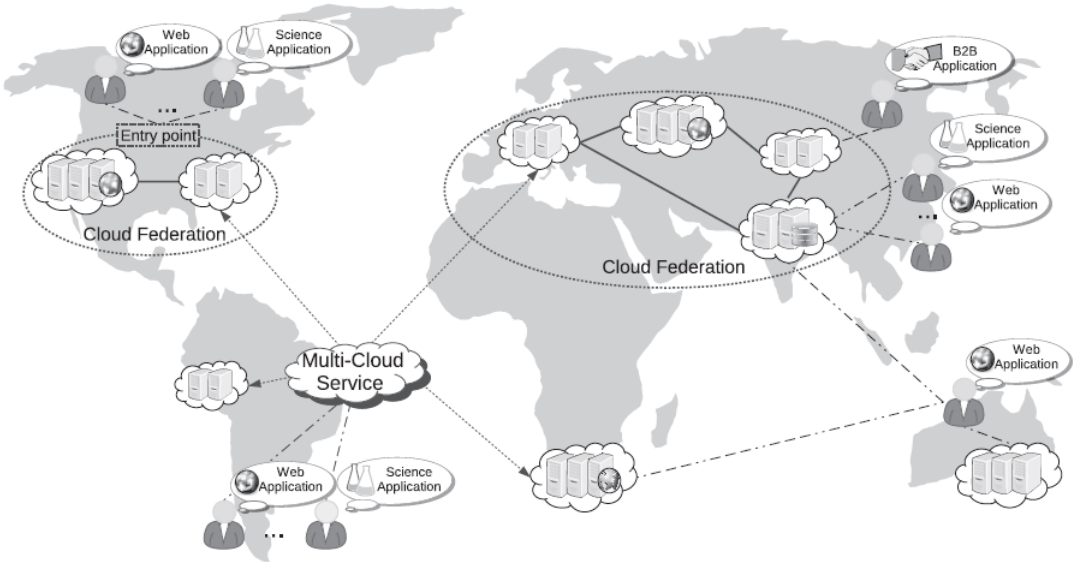

Figure 6.1 Inter-cloud coordination: schematic diagram 1

volunteer interconnection. Managing resource provisioning and scheduling are the direct responsibilities of clients or their representatives (Ferrer et al., 2012). Cloud federations and multi-clouds comes under the classification of Inter-Cloud.

The benefits for cloud providers which are provided by an Inter-Cloud environment are:

- *Geographical Location Diversity:* Many data centers are established by the leading cloud service providers world wide. However, it is unlikely that any provider will be able to establish data centres in every country and administrative region (Buyya et al., 2010c). Many applications have legislative requirements as to where data are stored. Thus, a Data centre within a region or a country may not be enough, and application developers will need fine-grained control (specific country or state) as to where resources are positioned. Only by utilizing multiple clouds one can gain access to so widely distributed resources and provide well-performing and legislation-compliant services to clients.

- *Noble Application Flexibility:* During the past several years, there have been several cases of cloud service outages, including ones of major vendors. The implications from one of Amazon's data centres failure were very serious for customers who relied on that location only. In a post-mortem analysis, Amazon advised their clients to design their applications to use multiple data centres for fault tolerance Amazon (2015). Furthermore, in Berkeley's report on cloud computing, Armbrust et al. (2009) emphasize that potential unavailability of service is the number one inhibitor

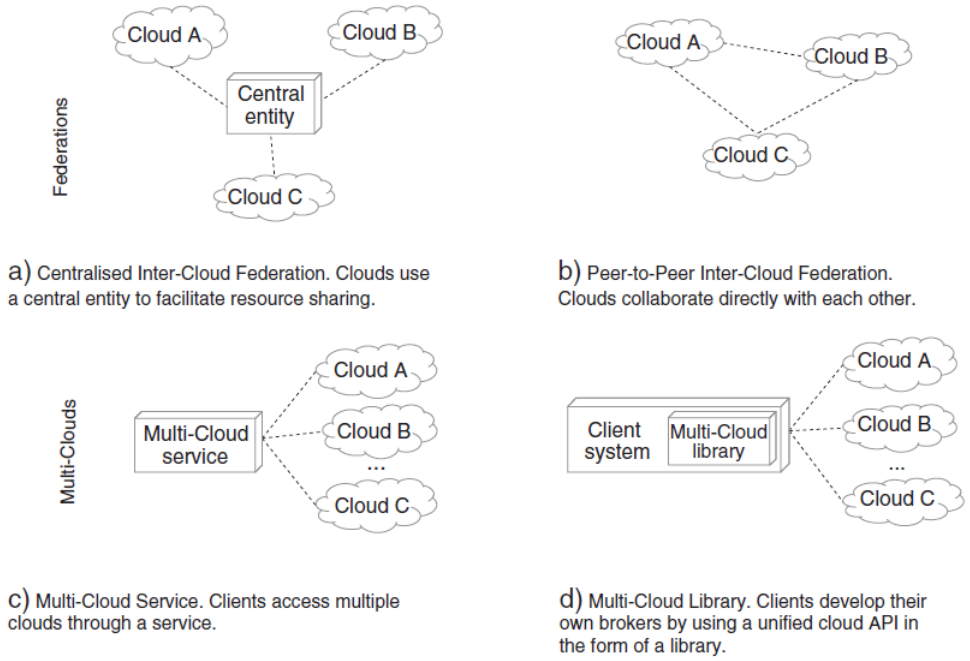

Figure 6.2 Inter-cloud coordination: schematic diagram 2

to adopting cloud computing. Thus, they advise the use of multiple providers. Besides fault tolerance, using resources from different providers acts as an insurance against a provider being stopped because of regulatory or legal reasons as well.

- *No vendor lock-in:* By using multiple clouds and being able to freely transit workload among them, a cloud client can easily avoid vendor lock-in. In case a provider changes a policy or pricing and it might impact its client negatively and could lead their client to migrate elsewhere.

The cloud customers can diversify their infrastructure portfolio in terms of both vendors and location and it is viewed as a big benefit of an inter-cloud environment. Consequently, they can possibly craft their businesses more adjustable to vendors' policy and availability changes and easily expand in new governmental regions. Cloud service providers may also have significant incentives from participating into an Inter-Cloud initiative. A paramount idea of cloud computing is that a cloud service should deliver constant availability, elasticity and scalability to meet the agreed customers' requirements (Lewis, 2010). A cloud provider should ensure enough resources at all times. Workload spikes can come unexpectedly, and thus, cloud providers need to over provision resources to meet them. Another issue is the huge amount of data centre power consumption (Beloglazov et al., 2011). Keeping an excess of resources in a ready to use state at all times for coping with unexpected load spikes leads to increased power consumption and cost of operation.

Thus, cloud providers' benefits can be summarized as follows:

- *On-demand Expansion:* Being able to offload to other clouds, a provider can scale in terms of resources like cloud-hosted applications do within a cloud. A cloud should maintain in a ready to use state and should have enough resources to meet its expected load and a buffer for typical load deviations. If the workload increases beyond these limits, resources from other clouds can be leased (Buyya et al., 2010c).

- *Enhanced SLAs to customers:* Knowing that even in a worst-case scenario of data centre outage or resource shortage the incoming workload can be moved to another cloud, a cloud provider can provide better SLAs to customers.

However, achieving all these benefits for both cloud providers and clients should be done without violating applications' requirements. Appropriate application provisioning and scheduling should honour the requirements in terms of performance, responsiveness and legal considerations. Existing approaches to achieve this vary in terms of architecture, mechanisms and flexibility. In this work, we explore cloud environments having Inter-Cloud resource scheduling and application scheduling mechanisms. We focus on identifying and analyzing the coarse-grained requirements in Inter-Cloud resource scheduling for distributed applications.

Inter-Cloud computing defines that, it is an endeavor that interconnecting the infrastructures of multiple cloud providers. The voluntarily lending of the infrastructures of the cloud providers with in an Inter-Cloud depends on the political and financial incentives. Inter-Cloud Gateway is a broker unit which implements the interconnection between multiple cloud providers. Generally, it is a service which acts for the provider to deploy application components and provision resources. In our work, we agree to this general idea and elucidate an Inter-Cloud Gateway as an automated entity with the following responsibilities:

- For a specific application, automatic resource provisioning and management across multiple clouds. It means that , this includes allocation and reallocation of resources (e.g. VMs and storage).

- Automatic distribution of application components in the provisioned resources.

- Load balancing and scheduling of the incoming requests to the allocated resources.

The two mechanisms through which the Inter-Cloud Gateway can be materialized are:
SLA based: The brokering requirements in SLA specified by the application developers is in the form of objectives and constraints. Either the cloud provider or the Inter-Cloud

service works for the client determines on brokering approach honoring the specified SLA. **Trigger-Action:** Application developers define a set of triggers and incorporate one or more actions to each of them. A trigger becomes active when a predefined condition, considering the externally visible application performance indicators becomes true. An action is executed when a correspondent trigger becomes active.

Provisioning activities and scale up /down of resources usually includes trigger actions. For example, if an allocated VM consumes more than 512 MB of memory or if the number of active TCP/IP sessions becomes more than 50, a trigger becomes active. An associated action may be the allocation of a new VM. The key feature of the SLA-based approaches is that they are fully visible to the application developers. The service or the service provider takes care of the provisioning activities. However, the providers do not have any idea or direct control in SLA-based approach, and doesn't know about how their own applications are provisioned across clouds. Thus, a certain level of trust between the two parties is needed here. Also, to explain the provisioning requirements of an application, one needs mature SLA specification and services and formalism.

The Cloud Standards Customer Council report shows that it was highlighted that SLA contracts put forward by the current cloud providers are immature (Council, 2012). Generally, clients are offered only inviolable standard SLA contracts, thus limiting their capability to define application-specific clauses. Most contracts like above only specify performance related clauses and do not confess for provisioning restrictions, for example, which area to be use for data storage. Thus, the acceptance of SLA-based approaches depends on the mediators' SLA offerings and advancements of providers.

The *Trigger-Action* approach is less transparent than the SLA-based one, because application developers need to specify the exact scalability and provisioning rules. This gives a finer grained control about how the application behaves. The *directly managed* brokering mechanisms are mostly used when there is no mediator between the application and the set of utilized clouds. Directly managed brokers are hosted separately and need to keep track of the performance characteristics of the application themselves. It is the responsibility of the application developers to develop such brokers in a way that meets the availability and dependability requirements.

6.2 SLA Based Dynamic Elasticity

In our research, we have proposed a system to elastically extend cloud services by integrating remote cloud resources on demand. Our system is using SLA based approaches to extend application specific cloud resources. We have proposed an autonomic Inter-Cloud Gateway capable of monitoring the demand of applications and responding by acquiring

or releasing cloud nodes.

The elastic cloud implementation relies on IaaS technologies such as EC2 (Services, 2015) and the Nimbus Workspace Service (Keahey et al., 2005) deployed on Science Clouds (Clouds, 2015). The Nimbus Workspace Service is responsible for deploying nodes on the Nimbus cloud, as requested by the cloud client. The Workspace Service initiates the transfer of the VM image from the storage pool to the nodes. Once the VM image has been deployed, the workspace service begins the boot process. As the nodes begin to boot they enter the contextualization phase. We rely on the Nimbus Context Broker (Keahey and Freeman, 2008) to provide contextualization of the nodes. The Nimbus Context Broker provides a secure mechanism for dynamically contextualizing a set of virtual appliances. We use the Context Broker to create a trusted environment between the newly deployed cloud worker nodes and the cluster head node.

For dynamic elasticity in the cloud it uses the Nimbus cloud client to launch or terminate nodes on Nimbus-based clouds. As this is an initial prototype, elastic cloud also uses Torque command line programs to monitor the queue and available nodes and to dynamically add and remove nodes from Torque. Before we add a cloud resource, we first create the VM image and save it in the different clouds we will be utilizing. We install all of the necessary software and libraries as well as the Torque client software. Torque is free open source software, so there is no need to acquire licenses for the cloud nodes. For simplicity, the Torque client is pre-configured to join the cluster head node on boot. The dynamic elasticity is controlled by iterative processing which periodically examine the job queue, executing a policy, and performing cluster management functions, such as terminating nodes that have been flagged for or shut down by the policy. Due to the time required to launch a VM, we have created a thread pool to launch machines in parallel. The threads (default is 5) work from a single "deploy node" queue to launch machines. The short and consistent time required to terminate a machine allows the main elastic site thread to terminate nodes serially, when needed.

We have also created a Python cluster object, which stores all of the relevant information about the cluster, such as the number of running cloud nodes and the number of cloud nodes that are available for work. The cluster object also contains methods for manipulating the cluster like launching or terminating VMs. Policies are implemented as individual Python modules. Policies have access to the cluster and queue objects and can use this representation of the system in order to determine if additional nodes are needed or if nodes may be terminated. The policies themselves are responsible for directly manipulating the cluster object by calling methods to either launch VMs or schedule them for termination.

The policy framework of clouds' dynamic elasticity is meant to be extensible, allowing administrators to customize or define their own policies to fit the needs of their users. The

primary reason we choose to implement policies as Python modules instead of creating our own policy definition language was to minimize the learning curve for administrators adopting our solution. We find that very detailed and customized, and these kinds of languages are often too complicated and frustrating for users to learn for a single purpose, diminishing the desire of users wishing to adopt our implementation of elastic site. Python is a widely used language that is easy to learn and robust enough to define any policy, no matter how simple or complicated.

6.3 Flexible Resource Allocation

The resource allocation process we have discussed here is occurring at the time when one of the cloud provider in the cluster faces the problem of shortage of resources and this can be tackled with the help of Inter Cloud Gateway (ICG) present at each of the cloud which employs the optimum resource allocation mechanism by which the resource provider will provide the resources. To optimally allocate the resources between different cloud computing providers in the same cluster we design a flexible resource management system with the use of population based meta-heuristic approach like ADS in the last stage.

The resource allocation is performed based on Hypervisor standardization. This can lead to an enhanced adaptive resource management system in cloud computing with different cloud providers. The proposed methodology can be implemented using cloudsim simulator and the performance can be compared and analyzed with the conventional methods for adaptive resource allocation and management. The proposed methodology can be depicted in the form of a block diagram as shown in Figure 6.3.

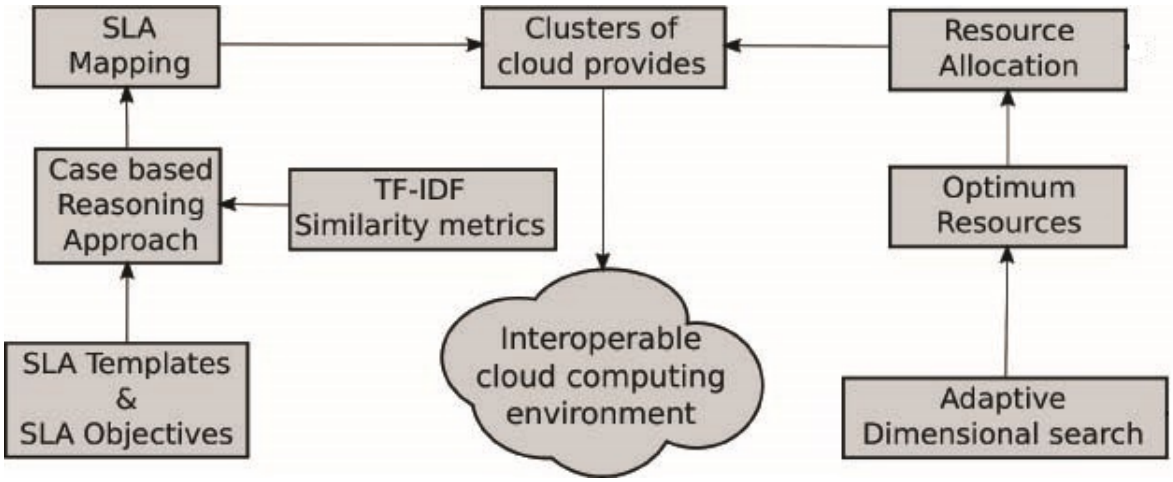

Figure 6.3 Schematic diagram of the proposed methodology

As seen from the Figure 6.3 initially based on the SLA templates of different cloud providers such as public and private cloud automatic SLA mapping is performed with the

help of the string similarity metric called TFIDF distance metric. From that mapped SLAs different clusters of cloud providers is formed. In a single cluster when the shortage of resources occurs with the private cloud the resources of another private cloud present is allocated optimally through the ADS algorithm when the request for the resources arrived at the Inter-Cloud Gateway present in each cloud. Through this an interoperable cloud computing is constructed with dynamic elasticity in cloud resources allocation.

Resource Allocation Through Inter-Cloud Gateway

In this work the resource allocation between different cloud providers with in the same cluster is done through the use of optimal resource allocation mechanism. This resource allocation mechanism is used here to provide the fair amount of resources to the intended cloud provider when it faces the problem of resource scarcity and it is accomplished by means of the resource request sent by them. The optimal resource allocation mechanism used here is equipped with the new Meta heuristic approach called ADS algorithm. Before discussing about this algorithm for resource provisioning first we analyze about the operation of ICG in the clouds.

Resource allocation through Inter-Cloud Gateway (ICG)

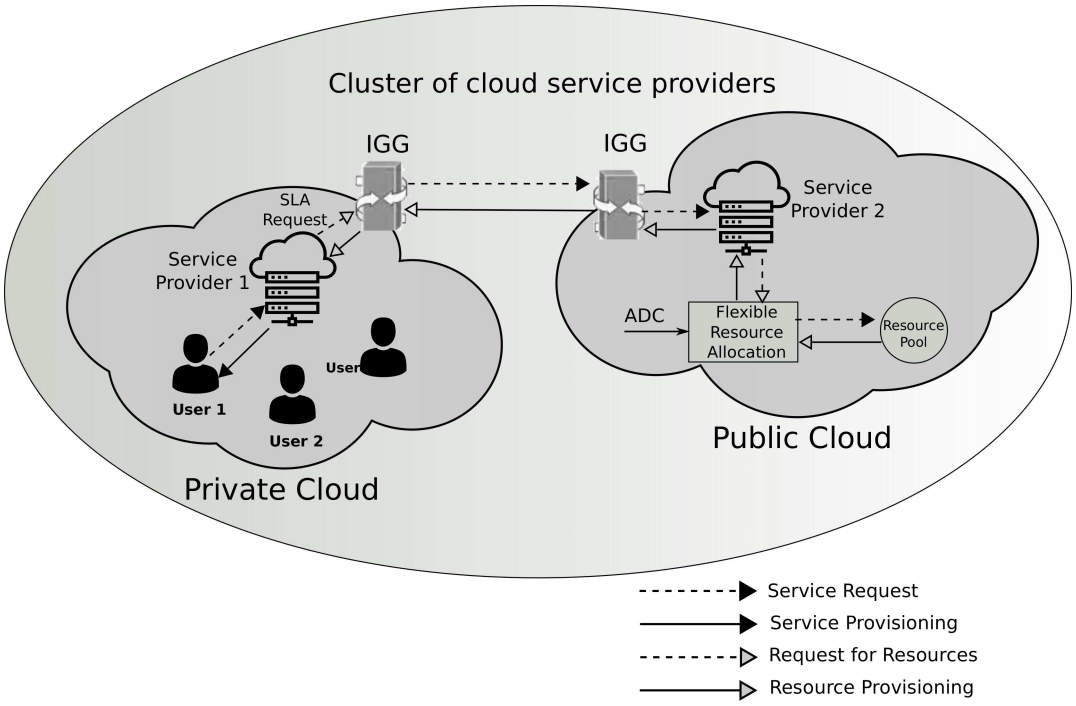

Figure 6.4 Flexible resource allocation through ICG and ADS algorithm

In the above Figure 6.4 the way the resources are optimally allocated when the request for resources arrived at the ICG is clearly mentioned. At first, while providing a service

to the intended user, shortage of resources for providing the service may occur. On that particular moment the service provider 1 in the private cloud will send the request to the public cloud present in that same cluster through the ICG and is forwarded to the service provider 2 in the public cloud. The request for resources may be of the form in Equation 6.1.

$$Req = \{R_T, R_n, T_u\} \tag{6.1}$$

where,

R_T - Type of resources requested (e.g. Bandwidth, Memory)

R_n - Total number of resources requested

T_u - Resource Utilization time.

Based on the request as given in Equation 6.1 the service provider 2 will employ the resource allocation mechanism to optimally allocate the resources in the form of virtual machines (VMs) to the service provider 1 from the resource pool. Note that, the resources will be allocated only when the requested resources are available otherwise, the service provider 1 has to wait for a certain period of time until the resources used in another location becomes free. To achieve the optimal allocation of resources to the requesting provider the Adaptive Dimensional Search algorithm is used here and is explained in the following section.

Adaptive Dimensional Search algorithm

ADS algorithm (Hasançebi and Azad, 2015) is a new Meta heuristic approach but it differs from nature inspired Meta heuristic approaches in the sense that it does not employ any metaphor for its implementation. The candidate solutions for the current population in this algorithm can be generated from the best solutions obtained for previous population. In Hasançebi and Azad (2015) the algorithm is used for the discrete truss optimization and we modified this algorithm in our approach for the flexible resource allocation between the clouds based on the resources requested from one cloud to another. To provide the optimal resource allocation with the fairness utilization of resources and this can be achieved by minimizing the Skewness and this is taken as the objective function of the resource allocation algorithm.

Skewness:

Skewness (Xiao et al., 2013) is the parameter which is used to enumerate the inequality in the utilization of resources. Consider that the number of resources as m, and the utilization of j^{th} resource as u_j then the Skewness of the n^{th} service provider can be calculated as,

$$skewness(n) = \sqrt{\Sigma_{j=1}^{m} \left(\frac{u_j}{\overline{u}} - 1\right)^2} \tag{6.2}$$

where,

$\overline{u}$ - Average resource utilization for n^{th} service provider.

The steps involved in the ADS algorithm is explained as follows and the flow of the algorithm is depicted in the Figure 6.5.

Steps in ADS algorithm

Step 1. Population initialization: Initialize ρ number of resources as the input for the ADS algorithm.

Step 2. Fitness Calculation: The fitness for the algorithm we used here is the minimization of Skewness and is calculated using the Equation 6.2.

Step 3. Search Dimensionality Ratio (SDR) calculation: SDR is the main parameter in ADS algorithm and this is updated in each iteration to produce the optimum result. It is defined as the ratio of number of allocated resources to the total number of resources and it is given by Equation 6.3.

$$SDR = \frac{N_A}{N_T} \tag{6.3}$$

where,

N_A- Number of resources allocated to the specified service provider

N_T- Total number of resources in the intended public cloud

At first iteration SDR can be calculated by Equation 6.3 and in the succeeding iterations SDR is updated by means of the Equation 6.4.

$$SDR^t = \begin{cases} \frac{SDR^{(t-1)}}{\lambda} \text{ if improvement in best solution} \\ \lambda . SDR^{(t-1)}, \text{ if no improvement in best solution} \end{cases} \tag{6.4}$$

where,

λ - the factor and usually taken to be less than one and t, t-1 represents the current and previous iteration respectively.

Step 4. Generation of new population: Based on the SDR value from Equation 6.3 and 6.4 the new populations for the next iteration can be generated using the following Equation 6.5.

$$P_i^{new} = \begin{cases} P_i^c, \text{if } r_j > SDR \\ P_i^c + round\left[N_j(0,1) \times \left(\sqrt{(P_i^{max} - P_i^{min})} \times \right. \right. \\ \left. \left. \left(\sqrt{(P_i^{max} - P_i^{min})} - 1 \right) \times \frac{t}{max - t} \right) \right] \text{ otherwise} \end{cases} \tag{6.5}$$

where,

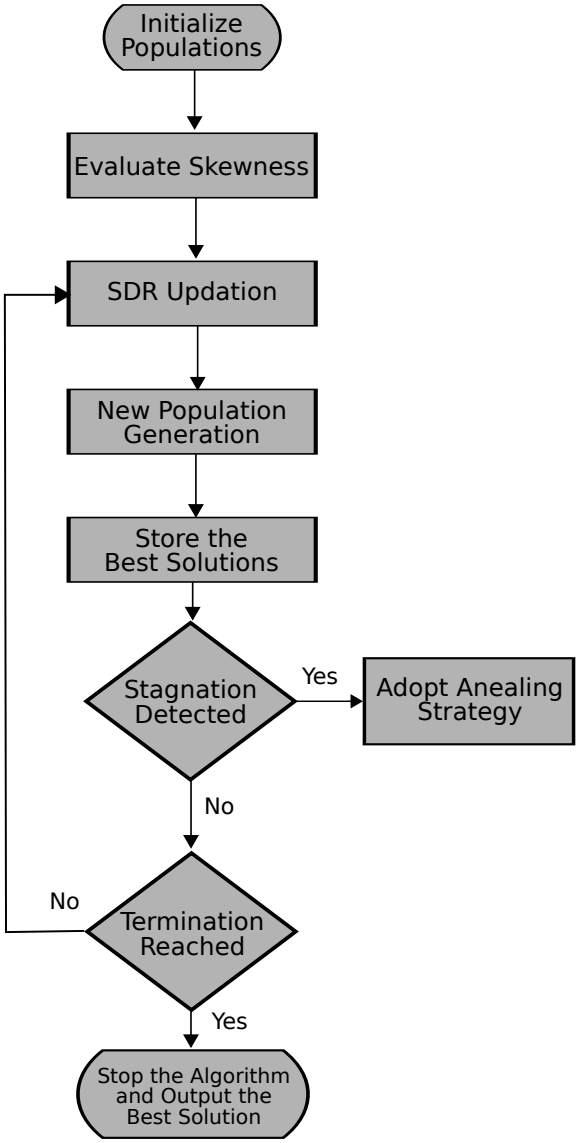

Figure 6.5 Flowchart of adaptive dimensional search

P_i^c - i^{th} resource in the best minimization of Equation 6.2 obtained so far

P_i^{new} - new population generation

r_j - random number sampled between [0, 1]

$N_j(0,1)$ - number of resources allotted to that sample.

t - current iteration number

max-t - maximum number of iterations to be performed in the optimization

Step 5. Elitism: In this stage the best solutions obtained are stored in a separate place and used in succeeding populations if the candidate solutions in that population does not outperform it.

Step 6. Stagnation control: In order to avoid the algorithm to be optimized towards a local optimum, several stagnation control strategies have to be adopted and in this algorithm

annealing strategy is used for the stagnation control.

Step 7. Stopping criteria: The algorithm is terminated if the maximum number of iterations are reached or the best solution is not obtained i.e. no improvement for certain number of iterations. Otherwise, the algorithm will continue to perform the Step 2.

The input for the algorithm is the random number of resources and based on the Skewness minimization optimal resources to be allocated for the service provider will be obtained as the output. This ADS algorithm is used to successfully allocate the optimum number of resources to the intended service provider with the consideration of Skewness minimization. The proposed methodology is implemented using the CloudSim tool and the experimental results are given in the next section.

6.4 Experimental Results

The proposed methodology is implemented using CloudSim. Here we are considering six private clouds and eight public clouds and with three SLA templates for each public cloud and two SLA templates for each private cloud for the mapping phase. The number of virtual machines considered are 500 to provide the resources to the requested cloud provider. Each of the new incoming SLA template is mapped automatically based on the available existing mappings. After that to enable the interoperability different clusters of cloud service providers are formed based on the SLA mappings with the help of K-means clustering technique in which a single cluster contains number of cloud providers with similar SLA templates.

Table 6.1 AWRT and Slowdown Values

Methods	AWRT (sec)	Slowdown (seconds)
Proposed methodology	488.91	383.8881
Conservative backfilling	1000	6500
Selective backfilling	750	4000
Easy backfilling	1500	$5x10^5$

Then in each cluster when one of the cloud provider is in the need for more resources to process certain user request, the resources can be requested from other cloud provider within the same cluster by sending request through the ICG for the resources to be utilized for a particular amount of time. When the ICG at other cloud receives this request it will analyze the situation i.e. the current availability of resources. If it is available, it will allocate the optimum resources in the form of VMs with the help of resource allocation

mechanism equipped with ADS algorithm otherwise the requested cloud provider has to wait until the resources are released which may be currently used in some other location. The performance achieved by our proposed method is depicted here in this section.

In this section the performance of our proposed resource allocation scheme in an interoperable cloud computing based on SLA mapping is compared with other existing works for SLA mapping (Breskovic et al., 2011b) and resource allocation (Javadi et al., 2012) in cloud. From the extensive literature review it has realized that only these above cited works tried the possibility of SLA mapping and so we used these results to compare the efficiency of our proposed algorithm.

Performance comparison of the proposed resource allocation mechanism with existing works

The Average Weighted Response Time (AWRT) and slowdown are the important factors in analyzing the performance of the resource allocation mechanism. Here we are evaluating our proposed method in terms of these parameters with other existing works (Javadi et al., 2012) and the results are given in the Table 6.1 and in the Figure 6.6 and Figure 6.7.

The AWRT and slowdown factor of the proposed methodology while responding to the request for resources by the resource allocation mechanism is compared with other existing techniques such as Conservative Backfilling, Selective Backfilling and Easy Backfilling which are the techniques used for the provisioning of cloud resources in Javadi et al. (2012) and the comparison is given in the form of graphs in Figures 6.6 and 6.7 respectively.

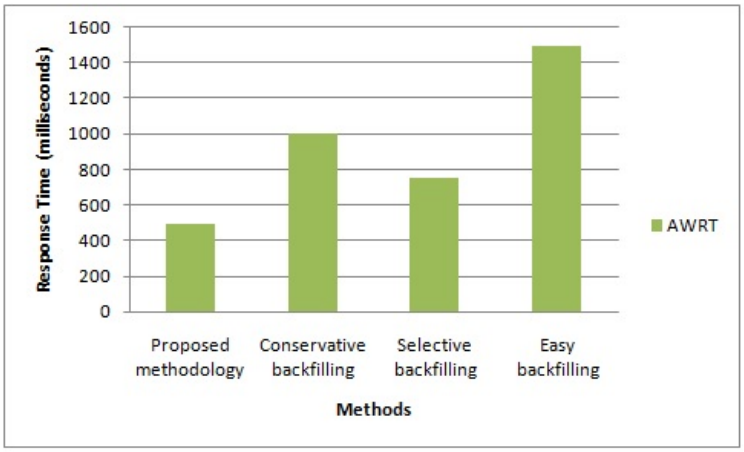

Figure 6.6 Performance comparison of proposed method (AWRT)

The Figure 6.6 represents that the AWRT of our proposed methodology yields minimum result than the selective backfilling technique which was minimum previously. Similarly, the slowdown incurred during our proposed resource allocation scheme is also minimum compared to the selective backfilling method and move towards zero. Thus from

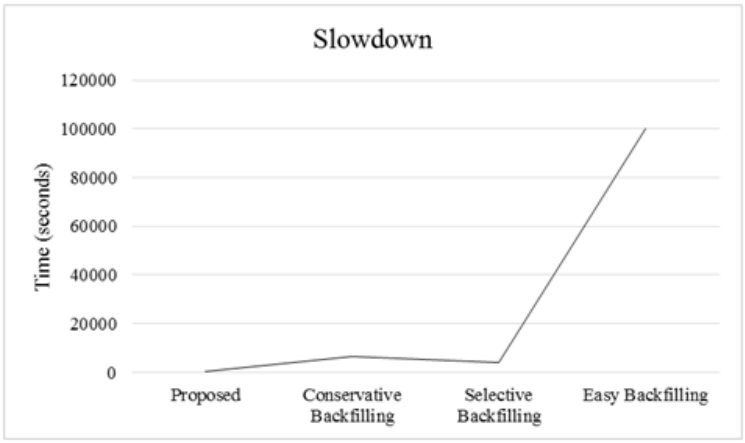

Figure 6.7 Performance comparison of proposed method (Slowdown)

the Table 6.1, Figures 6.6 and 6.7 it is proved that both the AWRT and Slowdown factor of our proposed methodology has achieved better results compared with existing works to allocate the resources in cloud computing.

6.5 Summary

The methodology we are proposed is to provide optimal resource allocation between different cloud service providers when one of them faces the problem of shortage of resources. The performance evaluation of the proposed methodology is individually as well as comparatively shown in the results section with other existing works available for the SLA mapping to provide the required services from other clouds and resource provisioning techniques. The parameters used to show the effectiveness of our proposed methodology are Overall net utility, overall cost in mapping the SLA templates, variation of the SDR parameter of the ADS algorithm in achieving the optimum results with in minimum iterations and AWRT, slowdown to measure the effectiveness of the resource allocation mechanism. And the experimental results shows that our proposed methodology achieves good results in terms of these parameters, such as, the overall net utility get maximized compared to the existing methods and overall cost, AWRT and slowdown factor get minimized. Another important factor we are considering here is the Skewness which measures the unevenness in the utilization of resources and with the help of the ADS algorithm it also get minimized to certain amount. Thus our proposed methodology finally constructs a better interoperable cloud computing environment with the awareness of SLAs of different clouds with flexible resource allocation.

Chapter 7

CONCLUSION & FUTURE WORK

The rapid increase of corporate businesses using cloud resources has increased the complexity of SLA management in cloud computing. The primary objective of this research work is to explore the significance of SLAs in cloud computing and the dominance of autonomic resource management systems for addressing the dynamically changing cloud resource requirements. In this dissertation we have presented an approach for SLA controlled adaptive resource management in cloud computing. This chapter of the thesis concludes our work by giving a summary of the preceding chapters and describing some of the future work directions.

7.1 Thesis Summary

The Thesis began with an introduction to cloud computing with a brief introduction to SLA in Cloud. It was followed by literature review where Research Gaps in SLA Management, Resource Allocation in Cloud and Federated Cloud Approaches are explored. Based on the investigation it is found that there is a need to have adaptive resource allocation in cloud using SLA and it is further extended to federated cloud environment. There are four objectives that have been considered here:

1. To build a flexible, reliable and dynamic SLA management system for monitoring and detecting SLA violations in cloud computing and incorporate some reactive actions to prevent SLA violation by adding autonomic adaptation of SLA parameters and SLOs.

2. To develop and implement an autonomic resource allocation system for reallocating resources dynamically during execution of services according to variation in workloads and detected SLA violations.

3. To design and develop a framework for performing autonomic SLA matching and

clustering according to applications using case based reasoning.

4. To develop and implement inter-dependency and interoperability by providing dynamic elasticity in the context of multiple competitive/cooperative cloud providers in an autonomic manner.

The First objective has been achieved in Chapter 3, where a SLA Negotiation and Monitoring Framework has been successfully implemented and tested. Detection and avoidance of SLA violation is also described briefly in this chapter. The concept of dynamic SLAs together with renegotiation strategy is included in this chapter along with the experimental evaluation of all proposed frameworks and algorithms. Further in Chapter 4, an adaptive resource allocation mechanism over the SLA monitoring framework has been developed. This developed work is dynamic in nature and has been successfully implemented in a real time cloud based on Open Nebula. The first two objectives concentrate towards the a single service providers but the next two concentrate on a federated cloud environment. Thus, Chapter 5 explores the objective 3, where SLAs for a federated cloud are matched and clusters of like behaved cloud service providers are formed. This is implemented using cloudsim. Further based on the clusters, an interoperable cloud has been designed and implemented successfully which is part of final objective 4. Thus as proposed, an interoperable federated cloud environment has been successfully implemented using SLAs. Chapter 7 presents the summary and conclusions of the research work. This chapter also introduces some future expansions possible to this work.

7.2 Conclusion

SLAs are important for any cloud service model, forming the basis of interaction and trust between the user and service provider. In this research SLAs have been considered as the basis of research and consequently used for resource management in cloud and further have been extended to federated cloud. The first part of the work deals with development of a SLA based cloud monitoring framework which has been designed and implemented. Results have been compared with other methods and our work proves to be better than them and correctly captures SLA violations. The second part of the work deals with resource allocation in cloud environment by designing an adaptive resource allocator. This cautiously allocates the resources based on SLAs and results show it to be efficient as compared to other similar non-SLA based methods. The next portion of the work is primarily aimed at federated cloud. Here the SLA based resource allocation mechanism for federated cloud has been designed and implemented. Our method allows SLAs to be infused into federated cloud environment. The overall function of this federated model is completely SLA based,

supporting interoperability without any hidden intent, as SLAs have been agreed upon by users and service providers as well.

7.3 Future Work

We envision a set of future extensions identified during the course of this research work.

- A future enhancement to this research work is possible in the field of service life cycle management in SLA negotiation. The life cycle of SLA negotiation can be expanded to address the re-negotiation of negotiated SLA after the detection of a limited number of SLA violations. The life cycle representations has to be standardized after accommodating this re-negotiation phase.
- Another functional enhancement related to this work is in the area of adaptive resource allocation in interoperable clouds by considering more SLA parameters to address a general cloud application. In this presented work we have considered a small subset of SLA parameters. This can be expanded by taking more number of SLA parameters to use in a broad cloud computing environment.
- One more future expansion possible to this work which is in the field of SLA matching and clustering. The SLA matching can be improved to accommodate any general SLA template.

Bibliography

Abrahao, B., Almeida, V., Almeida, J., Zhang, A., Beyer, D., and Safai, F. (2006). Self-adaptive sla-driven capacity management for internet services. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, 557–568. IEEE.

Abu Sharkh, M., Jammal, M., Shami, A., and Ouda, A. (2013). Resource allocation in a network-based cloud computing environment: design challenges. *Communications Magazine, IEEE*, 51(11), 46–52.

Addis, B., Ardagna, D., Panicucci, B., Squillante, M. S., and Zhang, L. (2013). A hierarchical approach for the resource management of very large cloud platforms. *Dependable and Secure Computing, IEEE Transactions on*, 10(5), 253–272.

Alhamazani, K., Ranjan, R., Mitra, K., Rabhi, F., Jayaraman, P. P., Khan, S. U., Guabtni, A., and Bhatnagar, V. (2015). An overview of the commercial cloud monitoring tools: research dimensions, design issues, and state-of-the-art. *Computing*, 97(4), 357–377.

Ali, S., Jing, S.-Y., and Kun, S. (2013). Profit-aware dvfs enabled resource management of iaas cloud. *Int J Comput Sci Issues (IJCSI)*, 10.

Almeida, J., Almeida, V., Ardagna, D., Francalanci, C., and Trubian, M. (2006). Resource management in the autonomic service-oriented architecture. In *Autonomic Computing, 2006. ICAC'06. IEEE International Conference on*, 84–92. IEEE.

Amazon (2015). Summary of the amazon ec2 and amazon rds service disruption.

Amin, M. B., Khan, W. A., Awan, A. A., and Lee, S. (2012). Intercloud message exchange middleware. In *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, page 79. ACM.

Andrew, A. M. (1998). Modern heuristic search methods. *Kybernetes*, 27(5), 582–585.

Anithakumari, S. and Chandrasekaran, K. (2015). Monitoring and management of service level agreements in cloud computing. In *Cloud and Autonomic Computing (ICCAC), 2015 International Conference on*, 204–207. IEEE.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., et al. (2009). Above the clouds: A berkeley view of cloud computing. Technical report, Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley.

Badidi, E. (2016). A broker-based framework for integrated sla-aware saas provisioning. *arXiv preprint arXiv:1605.02432*.

Barbosa, A. C., Sauvé, J., Cirne, W., and Carelli, M. (2006). Evaluating architectures for independently auditing service level agreements. *Future Generation Computer Systems*, 22(7), 721–731.

Beam, C. and Segev, A. (1997). Automated negotiations: A survey of the state of the art. *Wirtschaftsinformatik*, 39(3), 263–268.

Beloglazov, A., Abawajy, J., and Buyya, R. (2012). Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems*, 28(5), 755–768.

Beloglazov, A., Buyya, R., Lee, Y. C., Zomaya, A., et al. (2011). A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in computers*, 82(2), 47–111.

Bennani, M. N., Menasce, D., et al. (2005). Resource allocation for autonomic data centers using analytic performance models. In *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, 229–240. IEEE.

Berenbrink, P., Friedetzky, T., Goldberg, L. A., Goldberg, P. W., Hu, Z., and Martin, R. (2007). Distributed selfish load balancing. *SIAM Journal on Computing*, 37(4), 1163–1181.

Berman, F., Chien, A., Cooper, K., Dongarra, J., Foster, I., Gannon, D., Johnsson, L., Kennedy, K., Kesselman, C., Mellor-Crumme, J., et al. (2001). The grads project: Software support for high-level grid application development. *International Journal of High Performance Computing Applications*, 15(4), 327–344.

Bernhardt, T. and Vasseur, A. (2007). Esper: Event stream processing and correlation. *ONJava, in http://www. onjava. com/lpt/a/6955, O'Reilly*.

Blair, G. and Grace, P. (2012). Emergent middleware: Tackling the interoperability problem. *IEEE Internet Computing*, (1), 78–82.

Boniface, M., Phillips, S. C., Sanchez-Macian, A., and Surridge, M. (2009). Dynamic service provisioning using gria slas. In *Service-Oriented Computing-ICSOC 2007 Workshops*, 56–67. Springer.

Bonvin, N., Papaioannou, T. G., and Aberer, K. (2011). Autonomic sla-driven provisioning for cloud applications. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, 434–443. IEEE.

Bouchenak, S. (2010). Automated control for sla-aware elastic clouds. In *Proceedings of the Fifth International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, 27–28. ACM.

Brandic, I. (2009). Towards self-manageable cloud services. In *Computer Software and Applications Conference, 2009. COMPSAC'09. 33rd Annual IEEE International*, 2, 128–133. IEEE.

Brandic, I., Music, D., and Dustdar, S. (2009). Service mediation and negotiation bootstrapping as first achievements towards self-adaptable grid and cloud services. In *Proceedings of the 6th international conference industry session on Grids meets autonomic computing*, 1–8. ACM.

Breskovic, I., Haas, C., Caton, S., and Brandic, I. (2011a). Towards self-awareness in cloud markets: A monitoring methodology. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, 81–88. IEEE.

Breskovic, I., Maurer, M., Emeakaroha, V. C., Brandic, I., and Dustdar, S. (2011b). Cost-efficient utilization of public sla templates in autonomic cloud markets. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, 229–236. IEEE.

Burkon, L. (2013). Quality of service attributes for software as a service. *Journal of Systems Integration*, 4(3), 38.

Butt, A. R., Zhang, R., and Hu, Y. C. (2006). A self-organizing flock of condors. *Journal of parallel and distributed computing*, 66(1), 145–161.

Buyya, R., Beloglazov, A., and Abawajy, J. (2010a). Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges. *arXiv preprint arXiv:1006.0308*.

Buyya, R., Garg, S. K., and Calheiros, R. N. (2011). Sla-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions. In *Cloud and Service Computing (CSC), 2011 International Conference on*, 1–10. IEEE.

Buyya, R., Pandey, S., and Vecchiola, C. (2010b). Market-oriented cloud computing and the cloudbus toolkit. *Large Scale Network-Centric Distributed Systems*, 319–358.

Buyya, R., Ranjan, R., and Calheiros, R. N. (2010c). Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *International Conference on Algorithms and Architectures for Parallel Processing*, 13–31. Springer.

Buyya, R., Yeo, C. S., and Venugopal, S. (2008). Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*, 5–13. Ieee.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6), 599–616.

Cases, U. (2010). Functional requirements for inter-cloud computing. In *Global Inter-Cloud Technology Forum, GICTF White Paper*.

Chieng, D., Marshall, A., and Parr, G. (2005). Sla brokering and bandwidth reservation negotiation schemes for qos-aware internet. *IEEE Transactions on Network and Service Management*, 2(1), 39–49.

Choi, T., Kodirov, N., Lee, T.-H., Kim, D., and Lee, J. (2011). Autonomic management framework for cloud-based virtual networks. In *Network Operations and Management Symposium (APNOMS), 2011 13th Asia-Pacific*, 1–7. IEEE.

Cicotti, G., DâĂŹAntonio, S., Cristaldi, R., and Sergio, A. (2013). How to monitor qos in cloud infrastructures: The qosmonaas approach. In *Intelligent Distributed Computing VI*, 253–262. Springer.

Clouds, S. (2015). Summary of the amazon ec2 and amazon rds service disruption.

Cohen, W. W., Ravikumar, P. D., Fienberg, S. E., et al. (2003). A comparison of string distance metrics for name-matching tasks. In *IIWeb*, 2003, 73–78.

Comuzzi, M., Kotsokalis, C., Spanoudakis, G., and Yahyapour, R. (2009). Establishing and monitoring slas in complex service based systems. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, 783–790. IEEE.

Comuzzi, M. and Spanoudakis, G. (2010). Dynamic set-up of monitoring infrastructures for service based systems. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, 2414–2421. ACM.

Coppola, M., Dazzi, P., Lazouski, A., Martinelli, F., Mori, P., Jensen, J., Johnson, I., and Kershaw, P. (2012). The contrail approach to cloud federations. In *Proceedings of the International Symposium on Grids and Clouds (ISGC 2012)*, 2, page 1.

Costache, S. V., Parlavantzas, N., Morin, C., Kortas, S., et al. (2012). Themis: A spot-market based automatic resource scaling framework. In *HPDC 2012 Poster Session*.

Council, C. (2012). Practical guide to cloud service level agreements version 1.0.

Coutinho, E. F., de Carvalho Sousa, F. R., Rego, P. A. L., Gomes, D. G., and de Souza, J. N. (2015). Elasticity in cloud computing: a survey. *annals of telecommunications-annales des télécommunications*, 70(7-8), 289–309.

Cuomo, A., Rak, M., Venticinque, S., and Villano, U. (2012). Enhancing an autonomic cloud architecture with mobile agents. In *Euro-Par 2011: Parallel Processing Workshops*, 94–103. Springer.

Dai, Y., Xiang, Y., and Zhang, G. (2009). Self-healing and hybrid diagnosis in cloud computing. In *Cloud computing*, 45–56. Springer.

Dan, A., Davis, D., Kearney, R., Keller, A., King, R., Kuebler, D., Ludwig, H., Polan, M., Spreitzer, M., and Youssef, A. (2004). Web services on demand: Wsla-driven automated management. *IBM systems journal*, 43(1), 136–158.

Dastjerdi, A. V. and Buyya, R. (2012). An autonomous reliability-aware negotiation strategy for cloud computing environments. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, 284–291. IEEE.

Debusmann, M. and Keller, A. (2003). Sla-driven management of distributed systems using the common information model. In *Integrated Network Management VIII*, 563–576. Springer.

Di Modica, G., Tomarchio, O., and Vita, L. (2007). A framework for the management of dynamic slas in composite service scenarios. In *International Conference on Service-Oriented Computing*, 139–150. Springer.

Di Modica, G., Tomarchio, O., and Vita, L. (2009a). Dynamic slas management in service oriented environments. *Journal of Systems and Software*, 82(5), 759–771.

Di Modica, G., Tomarchio, O., and Vita, L. (2009b). A framework for the management of dynamic slas in composite service scenarios. In *Service-Oriented Computing-ICSOC 2007 Workshops*, 139–150. Springer.

Dikaiakos, M. D., Katsaros, D., Mehra, P., Pallis, G., and Vakali, A. (2009). Cloud computing: Distributed internet computing for it and scientific research. *Internet Computing, IEEE*, 13(5), 10–13.

Dobson, G. and Sanchez-Macian, A. (2006). Towards unified qos/sla ontologies. In *Services Computing Workshops, 2006. SCW'06. IEEE*, 169–174. IEEE.

Doorenbos, R. B. (1995). *Production matching for large learning systems*. PhD thesis, University of Southern California.

Dowell, S., Barreto, A., Michael, J. B., and Shing, M.-T. (2011). Cloud to cloud interoperability. In *System of Systems Engineering (SoSE), 2011 6th International Conference on*, 258–263. IEEE.

Dowlatshahi, M., MacLarty, G., and Fry, M. (2003). A scalable and efficient architecture for service discovery. In *Networks, 2003. ICON2003. The 11th IEEE International Conference on*, 51–56. IEEE.

El-Darieby, M. and Krishnamurthy, D. (2006). A scalable wide-area grid resource management framework. In *Networking and Services, 2006. ICNS'06. International conference on*, 76–76. IEEE.

Ellens, W., Zivkovic, M., Akkerboom, J., Litjens, R., and van den Berg, H. (2012). Performance of cloud computing centers with multiple priority classes. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, 245–252. IEEE.

Emeakaroha, V. C., Brandic, I., Maurer, M., and Dustdar, S. (2010). Low level metrics to high level slas-lom2his framework: Bridging the gap between monitored metrics and sla parameters in cloud environments. In *HPCS*, 48–54.

Emeakaroha, V. C., Netto, M. A., Calheiros, R. N., Brandic, I., Buyya, R., and De Rose, C. A. (2012). Towards autonomic detection of sla violations in cloud infrastructures. *Future Generation Computer Systems*, 28(7), 1017–1029.

Endo, P. T., Sadok, D., and Kelner, J. (2011). Autonomic cloud computing: giving intelligence to simpleton nodes. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, 502–505. IEEE.

Erdil, D. C. (2011). Dependable autonomic cloud computing with information proxies. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, 1518–1524. IEEE.

Erdil, D. C. (2012). Autonomic cloud resource sharing for intercloud federations. *Future Generation Computer Systems*.

Faratin, P., Sierra, C., and Jennings, N. R. (1998). Negotiation decision functions for autonomous agents. *Robotics and Autonomous Systems*, 24(3), 159–182.

Fatima, S. S., Wooldridge, M., and Jennings, N. R. (2005). A comparative study of game theoretic and evolutionary models of bargaining for software agents. *Artificial Intelligence Review*, 23(2), 187–205.

Feller, E. and Morin, C. (2012). Autonomous and energy-aware management of large-scale cloud infrastructures. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, 2542–2545. IEEE.

Feller, E., Rohr, C., Margery, D., and Morin, C. (2012). Energy management in iaas clouds: A holistic approach. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, 204–212. IEEE.

Ferrer, A. J., HernáNdez, F., Tordsson, J., Elmroth, E., Ali-Eldin, A., Zsigri, C., Sirvent, R., Guitart, J., Badia, R. M., Djemame, K., et al. (2012). Optimis: A holistic approach to cloud service provisioning. *Future Generation Computer Systems*, 28(1), 66–77.

Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., and Stoica, I. (2009). Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28.

Freitas, A. L., Parlavantzas, N., and Pazat, J. (2011). Cost reduction through sla-driven self-management. In *Web Services (ECOWS), 2011 Ninth IEEE European Conference on*, 117–124. IEEE.

Freitas, A. L., Parlavantzas, N., and Pazat, J.-L. (2010). A qos assurance framework for distributed infrastructures. In *Proceedings of the 3rd International Workshop on Monitoring, Adaptation and Beyond*, 1–8. ACM.

Friedman, E. (2003). Jess in action: rule-based systems in java.

Friedman-Hill, E. et al. (2008). Jess, the rule engine for the java platform.

Friedman-Hill, E. J. et al. (1997). Jess, the java expert system shell. *Distributed Computing Systems, Sandia National Laboratories, USA*.

Frutos, H. M. and Kotsiopoulos, I. (2009). Brein: Business objective driven reliable and intelligent grids for real business. *IBIS*, 8, 39–41.

Galante, G. and Bona, L. C. E. d. (2012). A survey on cloud computing elasticity. In *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, 263–270. IEEE Computer Society.

Goiri, Í., Guitart, J., and Torres, J. (2012). Economic model of a cloud provider operating in a federated cloud. *Information Systems Frontiers*, 14(4), 827–843.

Gomes, E. R., Vo, Q. B., and Kowalczyk, R. (2012). Pure exchange markets for resource sharing in federated clouds. *Concurrency and Computation: Practice and Experience*, 24(9), 977–991.

Goscinski, A. and Brock, M. (2010). Toward dynamic and attribute based publication, discovery and selection for cloud computing. *Future Generation Computer Systems*, 26(7), 947–970.

Goudarzi, H. and Pedram, M. (2011). Multi-dimensional sla-based resource allocation for multi-tier cloud computing systems. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, 324–331. IEEE.

Grinder (2016). The grinder, a java load testing framework.

Hallsteinsen, S., Geihs, K., Paspallis, N., Eliassen, F., Horn, G., Lorenzo, J., Mamelli, A., and Papadopoulos, G. A. (2012). A development framework and methodology for self-adapting applications in ubiquitous computing environments. *Journal of Systems and Software*.

Hao, W., Gao, T., Yen, I.-L., Chen, Y., and Paul, R. (2006). An infrastructure for web services migration for real-time applications. In *Service-Oriented System Engineering, 2006. SOSE'06. Second IEEE International Workshop*, 41–48. IEEE.

Hasançebi, O. and Azad, S. K. (2015). Adaptive dimensional search: a new metaheuristic algorithm for discrete truss sizing optimization. *Computers & Structures*, 154, 1–16.

Hasselmeyer, P., Koller, B., Parkin, M., and Wieder, P. (2008). An sla renegotiation protocol. In *Proceeding of the Second Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop*.

Hasselmeyer, P., Koller, B., Schubert, L., and Wieder, P. (2006). Towards sla-supported resource management. In *High Performance Computing and Communications*, 743–752. Springer.

Hasselmeyer, P., Mersch, H., Koller, B., Quyen, H., Schubert, L., and Wieder, P. (2007). Implementing an sla negotiation framework. In *Proceedings of the eChallenges Conference (e-2007)*, 4, 154–161.

He, Q., Yan, J., Kowalczyk, R., Jin, H., and Yang, Y. (2009). Lifetime service level agreement management with autonomous agents for services provision. *Information Sciences*, 179(15), 2591–2605.

Herbst, N. R., Kounev, S., and Reussner, R. Elasticity in cloud computing: What it is, and what it is not.

Herbst, N. R., Kounev, S., and Reussner, R. H. (2013). Elasticity in cloud computing: What it is, and what it is not. In *ICAC*, 13, 23–27.

Hielscher, J., Kazhamiakin, R., Metzger, A., and Pistore, M. (2008). A framework for proactive self-adaptation of service-based applications based on online testing. *Towards a Service-Based Internet*, 122–133.

Hofmann, P. and Woods, D. (2010). Cloud computing: the limits of public clouds for business applications. *Internet Computing, IEEE*, 14(6), 90–93.

Holze, M. and Ritter, N. (2011). System models for goal-driven self-management in autonomic databases. *Data & Knowledge Engineering*, 70(8), 685–701.

Huang, C.-J., Guan, C.-T., Chen, H.-M., Wang, Y.-W., Chang, S.-C., Li, C.-Y., and Weng, C.-H. (2013). An adaptive resource management scheme in cloud computing. *Engineering Applications of Artificial Intelligence*, 26(1), 382–389.

Hussain, O., Hussain, F., Hammadi, A., and Dillon, T. (2012). A framework for sla management in cloud computing for informed decision making.

Javadi, B., Thulasiraman, P., and Buyya, R. (2012). Cloud resource provisioning to extend the capacity of local resources in the presence of failures. In *High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, 311–319. IEEE.

Jrad, F., Tao, J., and Streit, A. (2012). Sla based service brokering in intercloud environments. In *CLOSER*, 76–81.

Jung, G. and Sim, K. M. (2011). Agent-based adaptive resource allocation on the cloud computing environment. In *Parallel Processing Workshops (ICPPW), 2011 40th International Conference on*, 345–351. IEEE.

Kang, C., Park, K., and Kim, S. (2006). A differentiated service mechanism considering sla for heterogeneous cluster web systems. In *Software Technologies for Future Embedded and Ubiquitous Systems, 2006 and the 2006 Second International Workshop on Collaborative Computing, Integration, and Assurance. SEUS 2006/WCCIA 2006. The Fourth IEEE Workshop on*, 6–pp. IEEE.

Katsaros, G., Kousiouris, G., Gogouvitis, S. V., Kyriazis, D., Menychtas, A., and Varvarigou, T. (2012). A self-adaptive hierarchical monitoring mechanism for clouds. *Journal of Systems and Software*, 85(5), 1029–1041.

Kaufman, L. M. (2009). Data security in the world of cloud computing. *Security & Privacy, IEEE*, 7(4), 61–64.

Keahey, K., Foster, I., Freeman, T., and Zhang, X. (2005). Virtual workspaces: Achieving quality of service and quality of life in the grid. *Scientific programming*, 13(4), 265–275.

Keahey, K. and Freeman, T. (2008). Contextualization: Providing one-click virtual clusters. In *eScience, 2008. eScience'08. IEEE Fourth International Conference on*, 301–308. IEEE.

Keller, A. and Ludwig, H. (2002). Defining and monitoring service-level agreements for dynamic e-business. In *Lisa*, 2, 189–204.

Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *Computer*, 36(1), 41–50.

Kertesz, A., Kecskemeti, G., and Brandic, I. (2011). Autonomic sla-aware service virtualization for distributed systems. In *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*, 503–510. IEEE.

Khader, D., Padget, J., and Warnier, M. (2010). Reactive monitoring of service level agreements. In *Grids and Service-Oriented Architectures for Service Level Agreements*, 13–22. Springer.

Kim, H., El-Khamra, Y., Rodero, I., Jha, S., and Parashar, M. (2011). Autonomic management of application workflows on hybrid computing infrastructure. *Scientific Programming*, 19(2), 75–89.

Kim, H. and Parashar, M. (2011). Cometcloud: An autonomic cloud engine. *Cloud Computing: Principles and Paradigms*, 275–297.

King, T. M. and Ganti, A. S. (2010). Migrating autonomic self-testing to the cloud. In *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on*, 438–443. IEEE.

King, T. M., Ramirez, A. E., Cruz, R., and Clarke, P. J. (2007). An integrated self-testing framework for autonomic computing systems. *Journal of Computers*, 2(9), 37–49.

Kleinrock, L. (1975). *Queuing systems*. Wiley.

Koller, B. and Schubert, L. (2007). Towards autonomous sla management using a proxy-like approach. *Multiagent and Grid Systems*, 3(3), 313–325.

Lai, G., Li, C., and Sycara, K. (2006). Efficient multi-attribute negotiation with incomplete information. *Group Decision and Negotiation*, 15(5), 511–528.

Lee, B.-Y. and Lee, G.-H. (2007). Service oriented architecture for sla management system. In *Advanced Communication Technology, The 9th International Conference on*, 2, 1415–1418. IEEE.

Lee, C. C. and Ferguson, M. J. (2010). To reveal or not to reveal? strategic disclosure of private information in negotiation. *European Journal of Operational Research*, 207(1), 380–390.

Leitner, P., Ferner, J., Hummer, W., and Dustdar, S. (2013). Data-driven and automated prediction of service level agreement violations in service compositions. *Distributed and Parallel Databases*, 31(3), 447–470.

Leitner, P., Michlmayr, A., Rosenberg, F., and Dustdar, S. (2010). Monitoring, prediction and prevention of sla violations in composite services. In *Web Services (ICWS), 2010 IEEE International Conference on*, 369–376. IEEE.

Lewis, G. (2010). Basics about cloud computing. *Software engineering institute carniege mellon university, Pittsburgh*.

Li, J., Qiu, M., Niu, J.-W., Chen, Y., and Ming, Z. (2010). Adaptive resource allocation for preemptable jobs in cloud systems. In *Intelligent Systems Design and Applications (ISDA), 2010 10th International Conference on*, 31–36. IEEE.

Li, Y., Sun, K., Qiu, J., and Chen, Y. (2005). Self-reconfiguration of service-based systems: A case study for service level agreements and resource optimization. In *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, 266–273. IEEE.

Liu, H., Bhat, V., Parashar, M., and Klasky, S. (2005). An autonomic service architecture for self-managing grid applications. In *Grid Computing, 2005. The 6th IEEE/ACM International Workshop on*, 8–pp. IEEE.

Lu, D., Ma, J., and Xi, N. (2015). A universal fairness evaluation framework for resource allocation in cloud computing. *Communications, China*, 12(5), 113–122.

Ludwig, A. and Franczyk, B. (2008). Cosma–an approach for managing slas in composite services. In *Service-Oriented Computing–ICSOC 2008*, 626–632. Springer.

Ludwig, H., Keller, A., Dan, A., King, R., and Franck, R. (2003). A service level agreement language for dynamic electronic services. *Electronic Commerce Research*, 3(1-2), 43–59.

Ludwig, S. A., Kersten, G. E., and Huang, X. (2006). Towards a behavioural agent-based assistant for e-negotiations. In *In Proc. of Montreal Conf. on E-Technologies (MCETECH), Montreal*. Citeseer.

Mahbub, K. and Spanoudakis, G. (2011). Proactive sla negotiation for service based systems: Initial implementation and evaluation experience. In *Services Computing (SCC), 2011 IEEE International Conference on*, 16–23. IEEE.

Manzalini, A. and Moiso, C. (2011). Self-optimization of resource allocation in decentralised server farms. In *Intelligence in Next Generation Networks (ICIN), 2011 15th International Conference on*, 219–224. IEEE.

Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., and Ghalsasi, A. (2011). Cloud computingâĂŤthe business perspective. *Decision support systems*, 51(1), 176–189.

Martin, P., Brown, A., Powley, W., and Vazquez-Poletti, J. L. (2011). Autonomic management of elastic services in the cloud. In *Computers and Communications (ISCC), 2011 IEEE Symposium on*, 135–140. IEEE.

Massie, M. L., Chun, B. N., and Culler, D. E. (2004). The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7), 817–840.

Maurer, M., Brandic, I., Emeakaroha, V. C., and Dustdar, S. (2010). Towards knowledge management in self-adaptable clouds. In *Services (SERVICES-1), 2010 6th World Congress on*, 527–534. IEEE.

Maurer, M., Breskovic, I., Emeakaroha, V. C., and Brandic, I. (2011). Revealing the mape loop for the autonomic management of cloud infrastructures. In *Computers and Communications (ISCC), 2011 IEEE Symposium on*, 147–152. IEEE.

Mearns, H., Leaney, J., Parakhine, A., Debenham, J., and Verchere, D. (2011). An autonomic open marketplace for inter-cloud service management. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, 186–193. IEEE.

Mell, P. M. and Grance, T. (2011). Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, United States.

Menasce, D. A., Almeida, V. A., Dowdy, L. W., and Dowdy, L. (2004). *Performance by design: computer capacity planning by example*. Prentice Hall Professional.

Mikic-Rakic, M. and Medvidovic, N. (2004). Support for disconnected operation via architectural self-reconfiguration. In *Autonomic Computing, 2004. Proceedings. International Conference on*, 114–121. IEEE.

Mobach, D. G., Overeinder, B. J., and Brazier, F. M. (2001). A ws-agreement based resource negotiation framework for mobile agents. *Scalable Computing: Practice and Experience*, 7(1).

Moreno-Vozmediano, R., Montero, R. S., and Llorente, I. M. (2009). Elastic management of cluster-based services in the cloud. In *Proceedings of the 1st workshop on Automated control for datacenters and clouds*, 19–24. ACM.

Morgan, G., Parkin, S., Molina-Jimenez, C., and Skene, J. (2005). Monitoring middleware for service level agreements in heterogeneous environments. In *Challenges of Expanding Internet: E-Commerce, E-Business, and E-Government*, 79–93. Springer.

Muller, C., Oriol, M., Rodríguez, M., Franch, X., Marco, J., Resinas, M., and Ruiz-Cortes, A. (2012). Salmonada: A platform for monitoring and explaining violations of ws-agreement-compliant documents. In *Principles of Engineering Service Oriented Systems (PESOS), 2012 ICSE Workshop on*, 43–49. IEEE.

Nagin, K., Hadas, D., Dubitzky, Z., Glikson, A., Loy, I., Rochwerger, B., and Schour, L. (2011). Inter-cloud mobility of virtual machines. In *Proceedings of the 4th Annual International Conference on Systems and Storage*, page 3. ACM.

Narayanan, V. and Jennings, N. R. (2006). Learning to negotiate optimally in non-stationary environments. In *Cooperative Information Agents X*, 288–300. Springer.

Papazoglou, M. P., Traverso, P., Dustdar, S., and Leymann, F. (2007). Service-oriented computing: State of the art and research challenges. *Computer*, 40(11), 38–45.

Papoulis, A. and Pillai, S. U. (2002). *Probability, random variables, and stochastic processes*. Tata McGraw-Hill Education.

Papuzzo, G. and Spezzano, G. (2011). Autonomic management of workflows on hybrid grid-cloud infrastructure. In *Proceedings of the 7th International Conference on Network and Services Management*, 230–233. International Federation for Information Processing.

Paraiso, F., Haderer, N., Merle, P., Rouvoy, R., and Seinturier, L. (2012). A federated multi-cloud paas infrastructure. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, 392–399. IEEE.

Paschke, A. and Bichler, M. (2008). Knowledge representation concepts for automated sla management. *Decision Support Systems*, 46(1), 187–205.

Pengbo, S., Qian, Z., Yu, F. R., and Yanhua, Z. (2014). Qos-aware dynamic resource management in heterogeneous mobile cloud computing networks. *Communications, China*, 11(5), 144–159.

Perera, S. and Gannon, D. (2009). Enforcing user-defined management logic in large scale systems. In *Services-I, 2009 World Conference on*, 243–250. IEEE.

Perros, H. G. and Elsayed, K. M. (1996). Call admission control schemes: a review. *IEEE Communications Magazine*, 34(11), 82–91.

Petcu, D., Macariu, G., Panica, S., and Crăciun, C. (2013). Portable cloud applicationsâĂŤfrom theory to practice. *Future Generation Computer Systems*, 29(6), 1417–1430.

Quan, D. M. and Kao, O. (2005). Sla negotiation protocol for grid-based workflows. In *High Performance Computing and Communications*, 505–510. Springer.

Rak, M., Cuomo, A., and Villano, U. (2011). Chase: an autonomic service engine for cloud environments. In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2011 20th IEEE International Workshops on*, 116–121. IEEE.

Rana, O. F., Warnier, M., Quillinan, T. B., Brazier, F., and Cojocarasu, D. (2008). Managing violations in service level agreements. *Grid Middleware and Services*, 349–358.

Ranjan, R. (2014). The cloud interoperability challenge. *Cloud Computing, IEEE*, 1(2), 20–24.

Ranjan, R., Wang, L., Zomaya, A. Y., Georgakopoulos, D., Sun, X.-H., and Wang, G. (2015). Recent advances in autonomic provisioning of big data applications on clouds. *IEEE Transactions on Cloud Computing*, 3(2), 101–104.

Redl, C., Breskovic, I., Brandic, I., and Dustdar, S. (2012). Automatic sla matching and provider selection in grid and cloud computing markets. In *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing*, 85–94. IEEE Computer Society.

Reich, C., Bubendorfer, K., Banholzer, M., and Buyya, R. (2007). A sla-oriented management of containers for hosting stateful web services. In *e-Science and Grid Computing, IEEE International Conference on*, 85–92. IEEE.

Resinas, M., Fernández, P., and Corchuelo, R. (2012). A bargaining-specific architecture for supporting automated service agreement negotiation systems. *Science of Computer Programming*, 77(1), 4–28.

Rochwerger, B., Breitgand, D., Levy, E., Galis, A., Nagin, K., Llorente, I. M., Montero, R., Wolfsthal, Y., Elmroth, E., Caceres, J., et al. (2009). The reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4), 4–1.

Sahai, A., Durante, A., and Machiraju, V. (2002). Towards automated sla management for web services. *Hewlett-Packard Research Report HPL-2001-310 (R. 1)*.

Sakellariou, R. and Yarmolenko, V. (2005). On the flexibility of ws-agreement for job submission. In *Proceedings of the 3rd international workshop on Middleware for grid computing*, 1–6. ACM.

Sánchez, A., Montes, J., Pérez, M. S., and Cortes, T. (2012). An autonomic framework for enhancing the quality of data grid services. *Future Generation Computer Systems*, 28(7), 1005–1016.

Services, A. W. (2015). Summary of the amazon ec2 and amazon rds service disruption.

Shawky, D. M. and Ali, A. F. (2012). Defining a measure of cloud computing elasticity. In *Systems and Computer Science (ICSCS), 2012 1st International Conference on*, 1–5. IEEE.

Shell, J. (2016). Jess: Java expert system shell.

Shen, H. and Liu, G. (2014). An efficient and trustworthy resource sharing platform for collaborative cloud computing. *Parallel and Distributed Systems, IEEE Transactions on*, 25(4), 862–875.

Silaghi, G. C., Şerban, L. D., and Litan, C. M. (2012). A time-constrained sla negotiation strategy in competitive computational grids. *Future Generation Computer Systems*, 28(8), 1303–1315.

Singh, S. and Chana, I. (2016a). Qos-aware autonomic resource management in cloud computing: a systematic review. *ACM Computing Surveys (CSUR)*, 48(3), 42.

Singh, S. and Chana, I. (2016b). A survey on resource scheduling in cloud computing: Issues and challenges. *Journal of grid computing*, 14(2), 217–264.

Skene, J. and Emmerich, W. Engineering an sla checker using mda technologies.

Solomon, B., Ionescu, D., Litoiu, M., and Iszlai, G. (2010). Designing autonomic management systems for cloud computing. In *Computational Cybernetics and Technical Informatics (ICCC-CONTI), 2010 International Joint Conference on*, 631–636. IEEE.

Sriram, I. and Khajeh-Hosseini, A. (2010). Research agenda in cloud technologies. *arXiv preprint arXiv:1001.3259*.

Stillwell, M., Schanzenbach, D., Vivien, F., and Casanova, H. (2009). Resource allocation using virtual clusters. In *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*, 260–267. IEEE.

Streitberger, W. and Eymann, T. (2009). A simulation of an economic, self-organising resource allocation approach for application layer networks. *Computer Networks*, 53(10), 1760–1770.

Suleiman, B., Sakr, S., Jeffery, R., and Liu, A. (2012). On understanding the economics and elasticity challenges of deploying business applications on public cloud infrastructure. *Journal of Internet Services and Applications*, 3(2), 173–193.

Technium, T. (2016). The technium: A cloudbook for the cloud.

Thain, D., Tannenbaum, T., and Livny, M. (2005). Distributed computing in practice: the condor experience. *Concurrency and computation: practice and experience*, 17(2-4), 323–356.

Theilmann, W., Happe, J., Kotsokalis, C., Edmonds, A., Kearney, K., and Lambea, J. (2010). A reference architecture for multi-level sla management. *Journal of Internet Engineering*, 4(1).

Toosi, A. N., Calheiros, R. N., and Buyya, R. (2014). Interconnected cloud computing environments: Challenges, taxonomy, and survey. *ACM Computing Surveys (CSUR)*, 47(1), 7.

Unger, T., Leymann, F., Mauchart, S., and Scheibler, T. (2008). Aggregation of service level agreements in the context of business processes. In *Enterprise Distributed Object Computing Conference, 2008. EDOC'08. 12th International IEEE*, 43–52. IEEE.

Van, H. N., Tran, F. D., and Menaud, J.-M. (2009). Sla-aware virtual resource management for cloud infrastructures. In *Computer and Information Technology, 2009. CIT'09. Ninth IEEE International Conference on*, 1, 357–362. IEEE.

Varalakshmi, P., Priya, K., Pradeepa, J., and Perumal, V. (2011). Sla with dual party beneficiality in distributed cloud. In *Advances in Computing and Communications*, 471–479. Springer.

Verma, D. (1999). *Supporting service level agreements on IP networks*. Sams Publishing.

Villela, D., Pradhan, P., and Rubenstein, D. (2007). Provisioning servers in the application tier for e-commerce systems. *ACM Transactions on Internet Technology (TOIT)*, 7(1), 7.

Wailly, A., Lacoste, M., and Debar, H. (2011). Towards multi-layer autonomic isolation of cloud computing and networking resources. In *Network and Information Systems Security (SAR-SSI), 2011 Conference on*, 1–9. IEEE.

Wäldrich, O. (2011). Orchestration of resources in distributed, heterogeneous grid environments using dynamic service level agreements.

Waldrich, O. (2016). Wsag4j: Web service agreement for java. version 2.0.

Weiss, A. (2007). Computing in the clouds. *networker*, 11(4).

Weyns, D., Schmerl, B., Grassi, V., Malek, S., Mirandola, R., Prehofer, C., Wuttke, J., Andersson, J., Giese, H., and Göschka, K. M. (2013). On patterns for decentralized control in self-adaptive systems. In *Software Engineering for Self-Adaptive Systems II*, 76–107. Springer.

Wieder, P., Seidel, J., Wäldrich, O., Ziegler, W., and Yahyapour, R. (2008). Using sla for resource management and scheduling-a survey. In *Grid Middleware and Services*, 335–347. Springer.

Wood, T., Shenoy, P., Venkataramani, A., and Yousif, M. (2009). Sandpiper: Black-box and gray-box resource management for virtual machines. *Computer Networks*, 53(17), 2923–2938.

Wu, L. and Buyya, R. (2010). Service level agreement (sla) in utility computing systems. *Performance and dependability in service computing: Concepts, techniques and research directions*, 1, 1–25.

Wu, L. and Buyya, R. (2012). Service level agreement (sla) in utility computing systems. In *Grid and Cloud Computing: Concepts, Methodologies, Tools and Applications*, 286–310. IGI Global.

Wu, L., Garg, S. K., and Buyya, R. (2011). Sla-based resource allocation for software as a service provider (saas) in cloud computing environments. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 195–204. IEEE Computer Society.

Xiao, Z., Song, W., and Chen, Q. (2013). Dynamic resource allocation using virtual machines for cloud computing environment. *Parallel and Distributed Systems, IEEE Transactions on*, 24(6), 1107–1117.

Xiong, K. and Suh, S. (2010). Resource provisioning in sla-based cluster computing. In *Job Scheduling Strategies for Parallel Processing*, 1–15. Springer.

Yan, J., Kowalczyk, R., Lin, J., Chhetri, M. B., Goh, S. K., and Zhang, J. (2007). Autonomous service level agreement negotiation for service composition provision. *Future Generation Computer Systems*, 23(6), 748–759.

Yashkov, S. (1987). Processor-sharing queues: Some progress in analysis. *Queueing Systems*, 2(1), 1–17.

Yazir, Y. O., Matthews, C., Farahbod, R., Neville, S., Guitouni, A., Ganti, S., and Coady, Y. (2010). Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, 91–98. Ieee.

Zang, C., Fan, Y., and Liu, R. (2008). Architecture, implementation and application of complex event processing in enterprise information systems based on rfid. *Information Systems Frontiers*, 10(5), 543–553.

Zhang, L. and Ardagna, D. (2004). Sla based profit optimization in web systems. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, 462–463. ACM.

Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1), 7–18.

Ziegler, W., Wäldrich, O., Wieder, P., Nakata, T., and Parkin, M. (2008). Considerations for negotiation and monitoring of service level agreements. Technical report, Technical Report TR-0167, CoreGRID.

Zulkernine, F. H. and Martin, P. (2011). An adaptive and intelligent sla negotiation system for web services. *Services Computing, IEEE Transactions on*, 4(1), 31–43.

LIST OF PUBLICATIONS/ CONFERENCE PAPERS

Journal Publications

[1] S. Anithakumari, K. Chandrasekaran (2014). *"Autonomic Cloud Computing: Self Management in Cloud Computing"*. ICIC Express Letters: An International Journal of Research and Surveys vol. 8, pp. 2959-2964, Publisher: ICIC International, Tokai University, Japan.

[2] S. Anithakumari, K. Chandrasekaran (2015). *"Autonomic Cloud Computing: Autonomic Properties Embedded in Cloud Computing"*. International Journal of Advanced Research in Computer Science and Software Engineering, Volume 5, Issue 4, 2015. 979-991.

Conference Proceedings

[1] Anithakumari, S., and K. Chandrasekaran. "Allocation of Cloud Resources in a Dynamic Way Using an SLA-Driven Approach." In Proceedings of the 2nd International Conference on Data Engineering and Communication Technology (ICEDECT), pp. 415-422. Springer, Singapore, 2019.

[2] Anithakumari, S., and K. Chandrasekaran. "Adaptive Resource Allocation in Interoperable Cloud Services." In Advances in Computer Communication and Computational Sciences (CSADC), pp. 229-240. Springer, Singapore, 2019.

[3] Anithakumari, S., and K. Chandrasekaran. "Negotiation and monitoring of service level agreements in cloud computing services." In Proceedings of the International Conference on Data Engineering and Communication Technology, pp. 651-659. Springer, Singapore, 2017.

[4] Anithakumari, S., and K. Chandrasekaran. "Interoperability Based Resource Management in Cloud Computing by Adaptive Dimensional Search." In 2017 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), pp. 77-84. IEEE, 2017.

[5] Anithakumari S., K. Chandrasekaran (2015). *"Monitoring and Management of Service Level Agreements in Cloud Computing"* IEEE International Conference on Cloud and Autonomic Computing (ICCAC) 2015, USA pp. 204-207, doi: 10.1109/ICCAC.2015.28

[6] Anithakumari S., K. Chandrasekaran (2014). *"Autonomic SLA Management in Cloud Computing Services."*Recent Trends in Computer Networks and Distributed Systems Security. Springer Berlin Heidelberg, 2014. 151-159

BIO-DATA

Name : Anithakumari S

Email Id : lekshmi03@gmail.com

Date of Birth : 24-05-1974

Address : Lekshmi Vihar, TC 52/1118(3),

Anugraha Nagar, Poozhikkunnu

Estate P O, Pappanamcode,

Thiruvananthapuram.

Educational Qualifications:

Degree	Year of Passing	University
B.Tech.	1995	Mahatma Gandhi University, Kottayam.
M.Tech	2006	National Institute of Technology Calicut.