Quality of Service Based Cloud Computing Framework for Resource Management

PhD Thesis

Submitted By:

Saeed Ullah
Reg. No: 1094-212004

Supervised By:

Professor Meritorious Dr. Mohammad Daud Awan

Dean Faculty of Computer Science, Preston University, Kohat, Islamabad Campus

Co-Supervised By:

Professor Dr. Malik Sikandar Hayat Khiyal

Faculty of Computer Science, Preston University, Kohat, Islamabad Campus

Faculty of Computer Science

Preston University, Kohat

Islamabad Campus

December, 2017

IN THE NAME OF ALLAH, THE MOST BENEFICENT, THE MOST MERCIFUL.

PLAGIARISM UNDERTAKING

I solemnly declare that research work presented in the thesis titled "Quality of Service Based Cloud Computing Framework for Resource Management" is solely my research work with no significant contribution from any other person. Small contribution/help wherever taken has been duly acknowledged and that complete thesis has been written by me.

I understand the zero tolerance policy of the HEC and Preston University, Kohat, towards plagiarism. Therefore I as an Author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of PhD degree, the University reserves the rights to withdraw/revoke my PhD degree and that HEC and the University has the right to publish my name on the HEC/University Website on which names of students are placed who submitted plagiarized thesis.

Saeed Ullah

Reg. No: 1094-212004

PhD Scholar

ACKNOWLEDGEMENTS

Alhamdulillah, all praises to Allah for the strengths and His blessing in completing this thesis. First and foremost, I offer my profoundest gratitude to my supervisor, Dr. Prof. Merit. Dr. Mohammad Daud Awan and co-supervisor Prof. Dr. Malik Sikandar Hayat Khiyal. I would like to thank them for continuous guidance, support, and encouragement.

I also acknowledged the guidance of my committee members. Their suggestions were very viable in completion of this research.

I would like to thank my parents for their unconditional support throughout my degree. Besides, a special thanks to my friends for their support.

I would like to give heartfelt thanks to my wife for his endless help in supporting and motivating me. I express my deepest appreciation for her care.

Finally, to my beloved daughter Haniya Saeed, I would like to express my thanks for being such a good girl and always cheering me up.

This thesis is dedicated to

my respected supervisors,

whose passion for knowledge and learning was inspiring and contagious

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	**Illustration**
BLAST	Basic Local Alignment Search Tool
AES	Advanced Encryption Standard
AWS	Amazon Web Services
API	Application Program Interface
B2B	Business to Business
C^3	Cloud Computing Commodities
CP	Cloud Provider
CSP	Cloud Service Provider
EC2	Elastic Compute Cloud
FASTA	FAST-All
FTP	File Transfer Protocol
GNU	GNU's Not Unix
HDFS	Hadoop Distributed File System
HPC	High Performance Computing
IaaS	Infrastructure as a Service
IP	Internet Protocol
iSCSI	Internet Small Computer Systems Interface
JVM	Java Virtual Machine
NIST	National Institute of Standards and Technology
ORM	Object Relational Mapping
P2P	Peer to Peer
PaaS	Platform as a Service
PC	Personal Computer
SDK	Software Development Kit
SOA	Service Oriented Architecture
SaaS	Software as a Service
SLA	Service Level Agreement
SLO	Service Level Objectives

QoE	Quality of Experience
QoS	Quality of Service
VM	Virtual Machine
VPC	Virtual Private Cloud
VPN	Virtual Private Network
VPS	Virtual Private Servers

ABSTRACT

Cloud Computing is an evolving information technology development, deployment and delivery model consisting of a collection of interconnected and virtualized computers enabling real time delivery of services and solutions over the Internet. One of the critical concerns in this environment is the provisioning of optimal software and hardware resources to ensure a better quality of service (QoS). The classic cloud computing model where services are provided by a single vendor introduces numerous challenges. Cloud services may be interrupted due to unavailability, natural disaster or abrupt increase of the load and hence the system may not be able to provide services to thousands of customers who solely rely and pay for resources. One of the recently emerging areas in cloud computing is deployment of virtual machines across multiple clouds based on providers' ranking. This involves benchmarking of different cloud providers, development of different techniques for selection of candidate providers, frameworks for ranking cloud providers and monitoring service level agreement (SLA) violations. Most of the existing literature is focused on employing centralized approaches for overall system ranking and monitoring, however, these approaches are not efficient for an environment where job migration and auto-scaling of virtual machines take place across cloud boundaries. The main objective of this research work is the development and evaluation of a QoS based ranking framework for IaaS computing resources across multiple clouds for resource negotiation, provisioning of physical resources, monitoring and ranking, based on job execution experience. We propose a broker enabled QoS ranking, negotiation and monitoring framework based on user level QoS requirements that determine users' needs and utility for choosing a best-fit cloud provider among a list of candidate cloud providers. Simulation and real test-bed experimentation results suggest that our proposed framework not only gained higher profit margin but also attained more user satisfaction in terms of lower job rejection and failure rate.

Chapter 1: Introduction

This thesis proposes a novel QoS ranking, negotiation and monitoring framework based on user level QoS requirements to find best-fit IaaS cloud provider across multiple clouds through the matchmaking process of cloud broker. A set of new metrics, algorithms and policies are designed and developed to evaluate study results on simulated as well as real cloud infrastructure. This chapter presents background, motivation and methodology of this research work.

1.1 Overview of Cloud Computing

Cloud computing is an evolving platform that has grasped the attention of scientific community and business industry towards the provisioning of computing resources as a utility and software as a service over a network. The platform of cloud computing shares many similar characteristics of grids and clusters but it has its own unique attributes and capabilities such as dynamic services that can be composed with web interfaces, virtualization and value added support from third party services to build cloud compute, storage and application services for end users. Thus, clouds are promising to provide services while abstracting the underlying hardware infrastructure on which these services are hosted. This trend of computing commodities provisioning is perceived as a fifth utility after electricity, gas, water and telephone as the services/resources are available whenever and wherever with just pay as you go policy without incurring the capital investments on the infrastructure [1].

Cloud computing is a new computational paradigm that relies on formulating the economies of scale by sharing the resources and in return maximizing the utility of shared resources. The maximum utilization of the resources is the key design goal underlying cloud computing systems. Cloud users can access available computing, platform and software resources as a single centralized view of the system and thus the existence of the underlying network and details of installation, maintenance and licensing of the applications are mostly transparent to the cloud users [2].

As huge investment in infrastructure, maintenance and up-gradation is required for business and scientific applications, cloud computing benefits in a way that it cuts down the capital investment cost required to purchase and install computing devices and provisions the capacities and services ‘as and when’ required in just *pay as you go fashion*. Business owners can simply acquire or rent-out the hardware, software or any other service according to business needs and can just pay for the use of the service, hardware or software to the provider without being worried for configuration, maintenance and other associated issues. It is just like renting a certain amount of server resources and not to purchase the whole physical or part of infrastructure. Cloud computing relies on the existing technologies being deployed as 'ready to use', so there is no need for the consumers to be expert of each and every kind of technology being used in the cloud, rather resource execution is the only concern left at consumers end.

1.1.1 Definition of Cloud Computing

Many definitions of cloud computing exist. NIST defined cloud computing as [3]:

"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (for example, networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models."

Buyya et al. definition of cloud computing is as under [4] :

"A Cloud is a type of parallel and distributed system consisting of a collection of interconnected and virtualised computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers"

As shown in figure 1.1, cloud computing provides access to software and hardware resources as a service that is seamless to consumers in a way that provider of the service could be anywhere in the world by providing interfaces through internet for desktop, web and application APIs/ libraries.

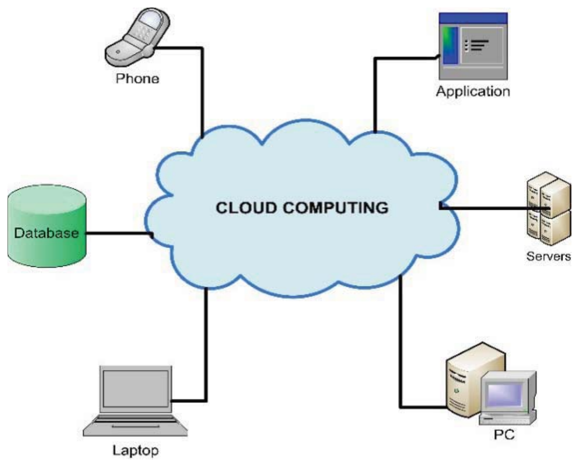


Fig. 1.1: Cloud Computing Environment, adapted from [5]

Cloud computing, as described in definition, is a group of resources interconnected virtually, located in a container that can be migrated from one physical place to another without service interpretation. These containers are elastic and scalable enough to add more and more resources capacity available for the consumers with the concept of virtualization. A typical cloud container is depicted in figure 1.2:

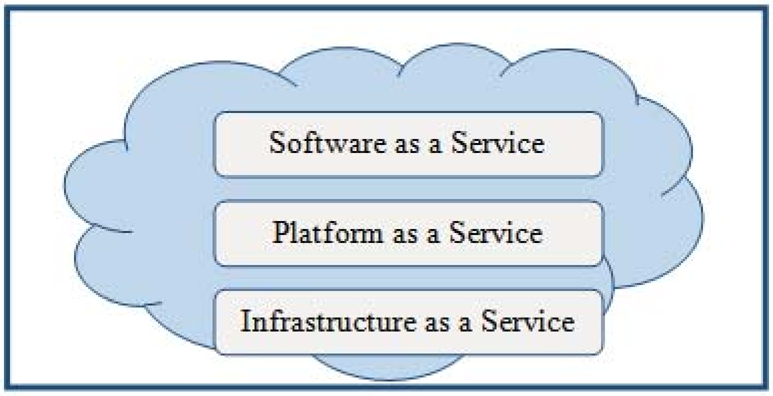


Fig. 1.2: Cloud Service Stack, adapted from [6]

The cloud container can be mapped to a web server for resource provisioning across the cloud. In this model, Infrastructure is the resource fabric that holds the real devices or set of devices in the data center, platform provisions like an operating system of the devices in the infrastructure and software/service can be mapped to the web application being deployed on the web server [6].

1.1.2 Background

In the early history of computing, when some complex and large computations had to be performed, services as well as expensive software and hardware infrastructure had to be purchased by the company or personnel to perform necessary computation and calculations. With the evolutions of computing platform, the new trend of personalized, powerful and to-do-yourself personal computers has taken place over the use of large room based, centralized mainframe systems. A significant portion of the current generation of platforms consists of inexpensive off-the-shelf software solutions installed in user devices. Thus, the idea of buying services in order to orchestrate business or scientific tasks emerged and this trend evolved to cloud computing [7].

The underlying cloud concept was originated from the mainframes used in 1950s and later for organizational and academic needs of computation. The large mainframes were utilized using the time sharing CPU model among multiple terminals during their mode of inactivity which, in turn, resulted in the increase of return on investment. With the passage of time, many optimized algorithms were proposed in order to efficiently utilize the idle cycles of computing resources, where the computing capacities were made available to more number of users. In 2008, Eucalyptus AWS API[1] provisioned the deployment of first open source private and hybrid cloud. The dramatic growth of this concept took place as soon as the quality of service and issues related to the providers and consumer were taken care by different projects of IT. Followed by Eucalyptus, many other cloud platforms came into practice like Open Stack, IBM Smart Cloud and Oracle Cloud.

Just like business entities, research community is also getting benefits from cloud services by adopting the utility of cloud computing innovations at relatively lower costs. As vast amounts of resources are required for high performance computing (HPC) applications to accommodate their processing requirements, the advent of cloud computing has shifted such computation from the dedicated clusters to the widely available clouds in a pay as you go fashion.

1.1.3 Technologies in Cloud Computing

Cloud computing is evolved from cluster and grid computing where clusters and network of clusters are interconnected in parallel to serve the dedicated tasks over networks. In case of

[1] http://aws.amazon.com/about-aws/

clouds, it is all done over the internet with an agreed Quality of Service (QoS) level to end users. The success behind the cloud lies in the concept of visualization where physical resources are further divided into hundreds and even thousands of virtualized devices. Thus, serving more and more consumers the capacity, computation and storage without any conflict unlike the grid. The concept of services in cloud has been taken from the service oriented architecture (SOA), where resources are provided in form of services to provide efficient solutions for the organizations using them for their routine tasks [5]. The five main cloud computing characteristics as mentioned by NIST are [3]:

- *On-demand Self Service*: Access to the service and resources unilaterally
- Broad Network Access: Resources and services can be accessed through heterogeneous client platforms
- *Resource Pooling:* Service provisioning is coped with pools of services and resources thus accommodating multiple consumers at a single instant of time
- *Rapid Elasticity:* With the increase or decrease of demand, resources/services can be provisioned or released at runtime
- *Measured Service:* Like any other utility, service can be measured in terms of its usage and control

1.2 Cloud Computing Service Models

In the world of information technology (IT), three general service models are defined, followed by other emerging models of cloud as stated by research community:

1.2.1 Infrastructure as a Service (IaaS)

Consumers utilizing IaaS model have full access to resource fabric which includes virtual machine (VM) instance, its operating system image, storage and configuration. However, consumer does not have control over the underlying infrastructure of the cloud as the physical details of the VMs are taken care by the provider of the service. Web hosting companies are example of such model that charge for their services to manage files and web pages on their servers. Virtual Private Servers (VPS) provide access to entire server as a dedicated server solution including operating system (OS), hosted web applications and configuration while hardware dependant details are usually provided at an abstract level to users. This mechanism also allows the provision to subdivide the physical machines further into logical servers for the scalability purpose. IaaS resource requirements can be configured, installed and monitored using vendor specific APIs or web based control panels. Amazon, CloudNine, Microsoft, Rackspace and WebFusion are some of the examples of IaaS providers.

1.2.3 Software as a Service (SaaS)

As the name suggests, consumers utilizing SaaS model have provision to execute user applications, being hosted on the provider's cloud infrastructure. Through a thin cloud interface, user applications such as Google Docs, online games and web based email servers can be accessed and executed at any part of the world. In this model, the burden of configuration or other operational activities of the servers, hardware, network and storage are not the responsibility of users. However, SaaS is different from the ordinary shared server in a sense that it holds the concept of container where the website is in a virtual container that can be physically moved from one place to another (may be on a better hardware whenever and howsoever needed) without interrupting the service [8]. RackSpace Mosso and WebFusion are well known SaaS providers in cloud computing.

1.2.4 Platform as a Service (PaaS)

Google Apps and Microsoft Azure are examples of such model. In this model, users can develop and execute applications hosted on the cloud. Application, web development and configuration tools are available so users can manage their work conveniently. However, they acquire a portion of control over the configuration of the pre-configured cloud environment. Clients can invoke services that are implemented over the cloud using REST and SOAP based protocols. Application scalability, availability and security issues can easily be managed by the consumer of the service through pre-installed rich platform services and scripts [8].

1.3 Deployment Models

1.3.1 Public Cloud

Cloud vendors offer their cloud resources as public to their end user based on pay as you go pricing policy. The services provided by such clouds may be free for end users or based on subscription model. Such clouds can be managed by institutions from different areas such as business, organizations and research community. Organizations having spike period of workload may use this model for their resource provisioning, for instance an online game may face availability issue in case of gaming events and hence spare capacity is required. Since procumbent of additional resources is not a feasible option as these resources may be underutilized during valleys, hence this model provides a better alternative to reduce additional investment costs. Google cloud services, Amazon web service, online games, web based email servers are best examples of public cloud.

1.3.2 Private Cloud

These clouds provision their resources to specific communities or organizations in an isolated environment. Such a model is motivated to be used when jurisdictional or regulatory laws govern data privacy and security issues. Amazon provides private clouds as Virtual Private Cloud (VPC) and Virtual Private Network (VPN) [8]. By utilizing a private cloud, an organization is lowering the expenditures that are required to keep up the resources and the virtualized environment to meet the business needs. Such clouds are hosted and managed either internally by the organization or externally or by any third party.

1.3.3 Community Cloud

This model is developed and deployed for specific collaborative organizations with common concerns to share infrastructure and available resources. An example is different government departments sharing IT resources [8]. The costs are spread over few members, just like a public cloud, so it better accommodates the budget of the community for investing in the establishment of its own infrastructure.

1.3.4 Hybrid Cloud

A cloud infrastructure with combination of two or more above stated deployment models is a hybrid cloud infrastructure. The services are composed of multiple cloud services, thus increasing the capacity of the service being provided. The deployed model may be unique in its properties but may rely on the standardized technology for data and application portability like cloud bursting for load balancing [1]. As an example, if a project is not getting sufficient services from its private cloud, it may connect to the public cloud, rendering services whenever needed. Such composition and integration of service would be able to manage better workload during spike period.

1.4 Commercial Clouds

1.4.1 Amazon [9]

Amazon Web Service (AWS), the pioneer of cloud infrastructure services, was launched in 2006. Amazon, being ranked 35 among Fortune 500 companies, has an estimated US$ 2.1 billion earning in 2012. AWS provides computational, storage, networking, deployment, management application services, database and much more. Amazon products are offered through control panel as well as Amazon product API which is a well defined set of APIs adopted as open standard by several other cloud architectures such as Enomalism [10], Eucalyptus [11] and OpenNebula [12]. Amazon, with 7000 consulting partners and more than 3000 technology partners, offers cloud services in ten different regions.

1.4.2 Google Cloud Computing [13]

Google entered in cloud computing market with the launch of Google App Engine in April 2008. Google released Object Storage Service in 2010 and Compute Engine in 2012. Google offers services like storage, big data, Apache Hadoop, mobile applications, gaming solutions and many more.

1.4.3 Rackspace [14]

Rackspace entered in the cloud market in 2008 following the company's acquisition of Slicehost [15]. The company has 6 data centers in different parts of the world. The company, with over 4800 employees, has revenue of US$ 1.3 billion in 2012. It offers services like Windows and Linux instances, storage, CDN and many more. Rackspace has also strong integration support with open source cloud platform OpenStack [16].

1.5 Inter-Cloud

In cloud, provisioning of computing resources is offered in the form of Virtual Machines (VM), being deployed on physical computing nodes. Cloud data center needs to be efficient and scalable to connect thousands and even thousands of thousands of such physical machines. However, installation, configuration and management of these hardware resources pose an important problem: Time-varying patterns of cloud load over different data centers. Usually, maximum utilization of cloud resources is observed during daytime while cloud usage remains downward during nights or weekends [17]. Data center capacity needs to be increased if a cloud provider (CP) is aiming to provide its services to all users, resulting in huge capital investment along with the maintenance cost incurred during the service lifecycle. This over-provisioning of resources can result in under utilization of computing nodes during valleys resulting in overall increase in service cost and price. On the contrary, if maximum utilization is the primary objective, data center capacity needs to be scale down to support average number of user requests. This forces the provider to reject user requests during peaks. This will result in the loss of revenue as well as loss of reputation/ trust of future cloud consumers [18].

The classic cloud computing model where services are provided by a single vendor introduces numerous challenges. Cloud services may be interrupted due to unavailability, natural disaster or abrupt increase of the load. Although one of the key features of cloud computing is the illusion of infinite resources, capacity in cloud provider's data centers is limited and eventually can be fully utilized [18]. Under such circumstances, the system may not be able to provide services to thousands of customers who solely rely and pay for resources.

There have been several cases of service outage during the past few years. A classic example is Amazon's data centers (northern Virginia) server failure whose implications were very serious for the customers who relied on those data centers. Some data from customers was irretrievably lost [19]. An important lesson learnt from this failure, as suggested by Daryl Plummer a cloud expert and Gartner research fellow, not to put everything at risk with just one data service provider [20]. Following the incident, many companies reverted to on-site servers to cope service downtime. However, some large service consumers, including Priceline, Netflix, SmugMug and Zynga, suffered little outage as their services were spread across different data centers.

Furthermore, data protection regulations/ legislation may vary in different countries. For example, customer information according to Swiss law has to physically reside within Switzerland. As another example, German citizen employee information cannot be processed by IT systems outside Germany, without written approval from the employee [21]. Since a medium-sized cloud vendor cannot operate in all regions across the globe, a mechanism is required by means of 'network of clouds' (Inter-Cloud) where different cloud networks may mutually cooperate to provide efficient, flexible and scalable service quality through inter-cloud systems and compete against bigger cloud providers in the market [22]. Inter-Cloud network model, being presented in figure 1.3, can be 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 [sic] interworking 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" [23].

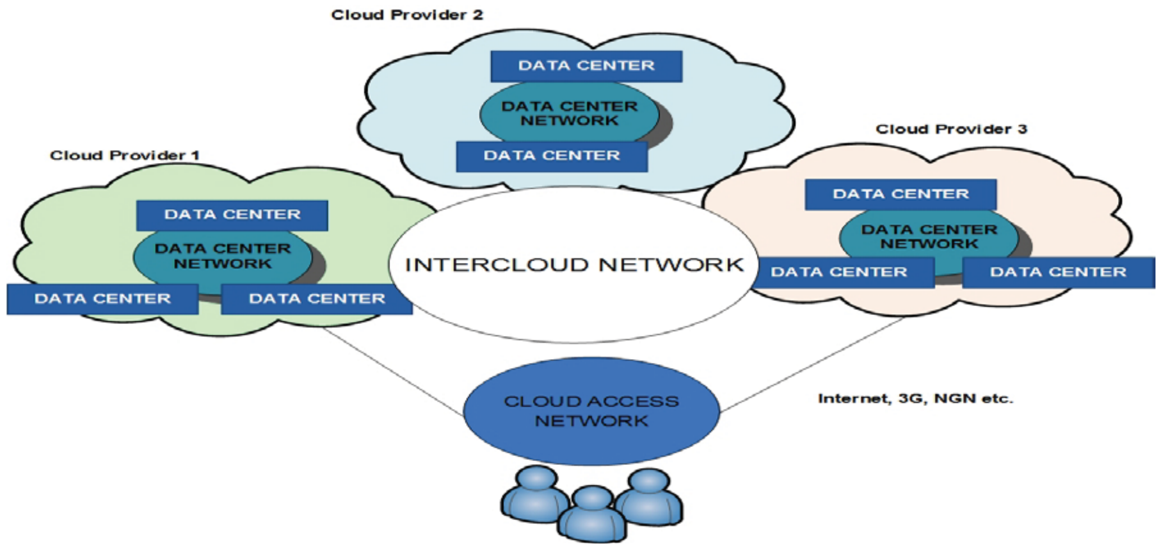


Fig. 1.3: Inter-cloud Network Model

1.6 Research Motivation

Much research effort has been made in different areas of cluster, grid and cloud computing focusing on the addressing research issues like negotiation [24], [25], provisioning of resources [26], [27] and resource monitoring [28]. One of the recently emerging areas in cloud computing is deployment of virtual machines across multiple clouds based on providers' ranking. This involves benchmarking of different cloud providers [29], [30], [31], [32], development of different techniques for selection of candidate providers [33], [34], frameworks for ranking cloud providers [35], [36] and monitoring SLA violations [37].

However, most of the researches, cited above, deal with cloud providers operating in isolation. In cloud federation, a broker performs or facilitates multiple clouds to share resources. Cloud broker acts as an intermediary between resource consumers and producers [38]. Cloud consumers can find best provider and service through the matchmaking process of cloud broker

[39], [40], [41]. One of the critical concerns in this environment is the provisioning of optimal software and hardware resources to ensure a better QoS [22], [42]. However, considering the nature of cloud federation where resource demands and load spikes are unknown in advance, the task of mapping incoming job requests to cloud resources becomes more challenging. The situation becomes more complicated with the growth of cloud providers in a federation as the selection of optimal clouds, meeting QoS requirements of user jobs become increasing difficult. Often one cloud provider may provide cost effective services for computational intensive jobs, they may be expensive for data storage services. Since power and other incurring charges may vary over different regions, a single cloud provider may also offer same infrastructure services but with different pricing schemes. Considering the diversity of such an environment, it is a major challenge for cloud consumers to select the right 'cloud provider' that may meet their requirements. It also involves trade-offs between critical and non-critical functional and non-functional requirements to select a best match. Hence, evaluation of different cloud providers in an objective way is necessary to satisfy user quality requirements which may change constantly in such a dynamic environment. It is not just sufficient to discover and coordinate multiple cloud providers, but it is also important to evaluate which is the most suitable cloud provider [43]. QoS Manager is a crucial component in the broker operating in federated environment as it is responsible of acquiring the virtual resources from the providers and ensuring that the negotiated QoS is being delivered [22], [44].

To satisfy the aforementioned requirements, cloud broker needs to be more SLA aware to identify the candidate cloud providers based on QoS requirements. Since these requirements change may vary on individual job to job basis, ranking and monitoring cloud providers' performance is an important area of research for dynamic allocation of resources under dynamic environment of 'network of clouds'. Most of the existing literature is focused on employing centralized approaches for overall system ranking and monitoring, however, these approaches are not efficient for an environment where job migration and auto-scaling of virtual machines take place across cloud boundaries [42].

1.7 Research Problems and Objective

This study deals with the issues involved while selecting a best-fit IaaS candidate among a set of cloud providers for a particular set of quality attributes. During this study, following issues arise which are categorized as under:

1.7.1 Benchmarking

- How can different IaaS cloud providers be benchmarked for a particular set of quality attributes?

  This involves investigation of different tools, techniques and methodologies for benchmarking IaaS cloud service providers. A final comparison table is generated mapping QoS with associated ranked values that can be used as an input for selecting candidate cloud providers based on particular QoS requirements for a job at run-time.

1.7.2 Provisioning of Resources based on QoS Ranking

- How can a generic broker based QoS ranking and resource selection framework be designed for discovery of best-fit candidates?

  The design and development of resource selection framework based on QoS ranking algorithm for selection of candidates cloud providers is proposed and the model is validated through necessary experiments.

1.7.3 Monitoring SLA Violations

- How can a monitoring component be deployed to monitor SLA violations and identify necessary migration strategies across multiple clouds?

  This phase involves development of monitoring components along with risk alerts to detect any SLA violation during job execution. Necessary migration strategies are developed to enforce job execution with SLA constraints.

1.7.4 Deployment and Testing

- How can the proposed framework and resource provisioning policies be evaluated?

  There were two possible directions to address this issue: using cloud simulators or evaluating usage scenarios on real cloud test-beds. Initially, the proposed model was evaluated through CloudSim simulator. However, simulators allow limited set of capabilities and these results cannot be generalized, so the proposed framework was evaluated on heterogeneous IaaS cloud providers where performance of the proposed model was validated at run-time.

To tackle the above mentioned challenges, the aim and objectives of the study are summarized below:

- Examine benchmarking techniques and associated tools that may help finding necessary statistics about cloud service providers. Cloud performance based on QoS parameters is assessed so that income job requests may be mapped to best cloud providers.
- Look into necessary tools and techniques to generate Service Level Objectives (SLOs) to model monitoring services necessary to enforce SLA biding.
- Evaluate proposed framework based on QoS preferences using real cloud testbeds such as Amazon AWS, Google Compute Engine and Rackspace Cloud. Since in this study, intended cloud platform is IaaS, VM instances from the popular cloud vendors were leased for performance evaluation as well as benchmarking and ranking based on quality attributes. This resulted in productivity of real world statistics as well experiment repeatability for other researches in the similar directions.
- Implement a broker prototype to evaluate study results on real cloud infrastructure
- Implement negotiation strategies in broker based multi-cloud environment for better resource utilization among cloud providers.

1.8 Methodology

The first objective of this research work was the development and evaluation of a QoS based ranking framework for IaaS computing resources across multiple clouds for resource negotiation, provisioning of physical resources, monitoring and ranking, based on job execution experience. A significant part of the effort was put on benchmarking cloud providers for the services they offer, integration of multiple cloud providers through a common API and management of job life cycle: job submission, run-time execution and performance evaluation for benchmarking a cloud provider for a particular set of quality attributes.

The research methodology consisted of the following steps:

- **Experimentation**: Different benchmarking tools such as Bonnie++, IOZone, IPerf, SPECjvm, Phoronix Test Suite Suites are used to gather necessary statistics about IaaS metrics (Processor, Memory, Storage and Networking capabilities).
- **Mathematical Modeling**: A ranking based algorithm was developed and validated to prioritize cloud vendors based on quality attributes involved during SLA agreement.
- **Prototyping**: Initially, the proposed model was evaluated through CloudSim simulator. Based on the results, a broker based federation of multi-cloud environment, using JCloud API of different cloud vendors, was developed to validate this proposed model.

1.9 Thesis Organization

The outline of thesis is presented in figure 1.4 which shows the structure of thesis chapters. The remainder of thesis is as under:

Chapter 2 describes a taxonomy and survey of cloud service selection, ranking and benchmarking taxonomy. The proposed broker based resource provisioning framework, negotiation mechanism and QoS based Ranking algorithm is investigated in chapter 3. Chapter 4 presents the methodology, workload model, benchmarking and experimentation design. Simulation and real work experimentation results are discussed in chapter 5. Finally, the thesis concludes in Chapter 6 by discussing conclusions and exploring future research directions.

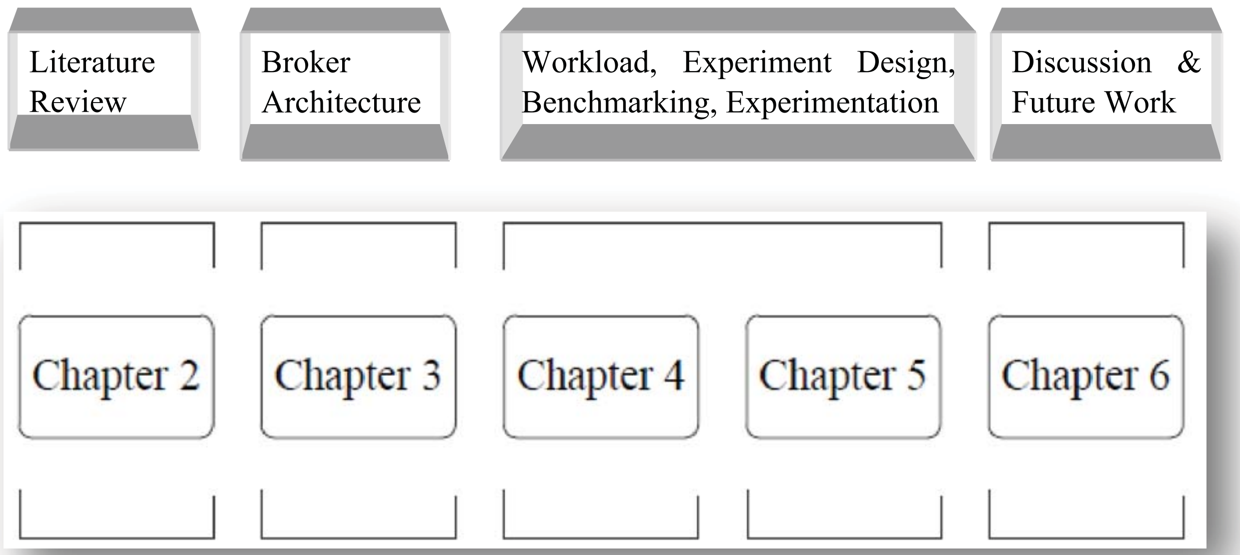


Fig. 1.4: Structure of Thesis

Chapter 2: Related Work

Quality of Service (QoS) is a broad topic in the field of computer networks and distributed systems. It refers to guarantee a certain level of conformance for the resources under provision [45]. Although various QoS standards are available for other fields of computer networks, much research is still needed to define QoS standards and metrics in the field of cloud computing [46], [47]. However, some proposed methodologies for QoS metrics classification are as under:

- Qualitative vs. Quantitative: Quantitative QoS metrics refer to the features that can be expressed and measured in units such as reliability, availability and latency, while qualitative metrics are based on user's experience such as trust and reputation.
- Deterministic vs. Non-deterministic: Deterministic QoS metrics include attributes that are known before a job is submitted such as price of service. On the other hand, non-deterministic are uncertain at the time of invocation such as response time [48].
- Runtime related: This group includes metrics that can be calculated dynamically such as scalability, elasticity, fault tolerance and latency [49].

QoS details are mostly provided in Service Level Agreement (SLA) document, a legal binding between the service consumer and the cloud provider [50].

2.1 Service Selection Taxonomy

Service selection based on some given preferences has remained a topic of interest for cluster, grid and service oriented research community. In such an environment, resources are represented as components or web services and researchers have proposed different techniques to find the optimal solution by filtering and selecting the best matches among them. Since this scenario also applies in cloud computing where different models are presented as services, cloud computing researchers can be benefited by the techniques proposed for other distributed paradigm. The aim of this section is to explore different research techniques directed to address service selection issues.

2.1.1 Process of Service Selection

Under such selection process, after the user job is submitted to job admission controller, several steps take place before a particular service is invoked. User requirements are modeled and formulated with necessary objectives and constraints. As an example, the authors in [51] addressed the issue of bandwidth as service selection criteria. Then different decision making strategies are applied to nominate the best fit candidate among a list of services.

The authors in [52] proposed an algorithm to filter a set of services from a service pool based on user preferences. Their work, based on QoS modeling, enables user to query and invoke services from a pool of services. A four-stage process is applied to retrieve the best fit service according to the user requirement. Initially, user submits QoS criteria along with relevant priorities to the search process. Search process matches all the services that meet the description requirement of the user. The filter step removes the services that do not meet the QoS constraints of the job and the rank process sorts the list according to the QoS priorities of the user job. Users can then select the best-matched service among the list of services. Finally, the update search module saves these heuristics so that decision about subsequent requests can be made based on user's experience.

A QoS based service composition algorithm (LOEM) is presented in [53]. The study addressed the issue of maximizing utility function based on user QoS criteria. Multiple service composition requests can be submitted by users in a single job. To model this architecture, a chain of services is grouped together to meet the user QoS preferences. Since the problem of finding optimal combination of services from a pool of services is NP hard, efficient estimation for efficiency is achieved through pareto-optimal solutions. The proposed algorithm first filters out services that do not match or provide poor utility then they introduced two variables: h as number of services and n as total number of services where h/n services are selected as candidate services. Afterwards, mixed integer programming is used to select pareto-optimal solutions.

The authors in [54] investigated QoS criteria for provisioning cloud services and analyzed different issues associated with cloud platform. Firstly, they used SLA as driving force for meeting QoS requirements. SLAs in their approach specify requirements for every actor of the system and provide a mechanism for QoS requirements, obligations and penalties in case of violations. Secondly, a model is proposed involving cloud vendors, users and consumers where users are service providers and end-users are application consumers. Services providers play a key role in their proposed approach as they have to create two SLAs: one for cloud provider and the other for cloud users. The authors viewed cloud model through business centric approach where profit maximization was the key goal while considering user QoS requirements. Four different scheduling schemes are introduced in their approach. First is the 'acquisition of VM' where an incoming request is assigned a new VM. Second is the 'wait' where a request is put in the queue until resources are available. Third is 'insert' where a request from the queue is inserted as soon as resources got available under the constraint that SLA should not be violated and finally 'penalty delay' where SLAs can be violated in favor of job execution. These four schemes provide a balance between QoS requirements satisfaction by users and profit at providers' end. These policies are orchestrated in a way to achieve maximum profit. If the utility of these policies is poor, incoming user request is rejected.

A prototype model was proposed for QoS based workflow execution systems in [55]. The system aimed to capture and publish QoS related requirements from the running modules in a service index. The proposed design is composed of three different modules. Coordination between different components of the system is handled by workflow manager. At each cloud platform, workflow enactor component is deployed which is capable for workflow execution. This component is also responsible for comparing and monitoring processes according to QoS requirements defined in service level objectives. If any violation is found, it triggers an alarm which is handled by which is responsible for workflow execution.

The authors in [56], [57] proposed ontology based framework for modeling user preferences, being considered as an optimization problem. They presented service discovery and composition process focused on semantic web services. Their work is based on their previous research [58] to utilize semantically described utility functions for defining user preferences. The proposed optimization problem can be solved using dynamic or constraint programming. In their work, QoS details are extracted from semantic description of web services. These values are then linked to user requirements which generate utility function based on QoS criteria. To retrieve the specifications of required optimization, XSL transformation is applied to the utility functions. Although, constraint satisfaction optimization problem is selected for optimization problem but various other techniques can also be applied using XSL meta-data [59].

Cloud services can be diverse in nature and may depend on multiple criteria (pricing policy, cost performance and so on). Hence, a methodology is required to select cloud services made up of different criteria to map user requirements. The complexity of services and number of available options have complicated the process of service selection. A multi-criteria cloud service selection methodology has been proposed in [60] to select services based on multiple requirements. The selection process is based on the best matched vector table for a user requirement criteria vector against all service descriptor vectors. In the first step of the process, a weighted difference is calculated by subtracting user requirement vector from a list of decision matrix. To compute the conformity of each service to the user requirements, the product of matrix and transpose of user requirements is calculated. An element with the minimum value corresponds to the best matched service for the given set of requirements. In the second step, the effect of mutual cancellation between criteria exceeding above or below is restricted by multiplying the matrix by scalar *-1* followed by replacing each element by $e^{\text{power of respective element}}$. The novelty of the proposed approach is the ability to diminish the influence of any criteria that exceeds user requirements.

The authors in [61] proposed a two-step algorithm for cloud service selection. In the first step, services in a hierarchical structure are probed to filter a list of available service. In the second step, an optimized service selection process is performed to select services with maximum gain and minimum cost being incurred. Clients can access services using centralized or distributed service proxies. The locality and replications issues are addressed so that users can invoke cloud services from multiple proxies based on service types, geographic area and preference setup.

A cloud service provider indexing structure has been proposed in [62] to capture similarity among various properties of service providers. In the designed structure, different properties such as service type, quality of service, measurement and pricing units have specific locations to be stored and retrieved in an index, developed in B+ tree. Based on this index, a service selection algorithm is proposed to aggregate list of services and to provide ranking of potential service providers. A refinement process is used to further reduce the number of service providers that need to be fully examined. The authors proposed a design structure based query algorithm to find k best vendors close to optimal for each desirable service, started from a best cloud provider with an ascending order to search for a list of best providers. The proposed algorithm is compared with a baseline brute-force algorithm which shows that the performance of proposed algorithm is 100 times faster for 10,000 service providers.

2.2 Taxonomy of Service Ranking

One positive sign for cloud community market is the great deal of competition being faced by major cloud vendors [63]. This may result beneficial in terms of better QoS, available choices for low price resource selection and introduction of interactive and creative services. However, it may pose the issue of vendor selection as cloud market is growing at rapid pace. In this section, different techniques proposed in the literature to address this issue are reviewed.

In cloud computing marketplace, multiple cloud providers are available with the same functions but with different QoS attributes. The authors in [35] opined to evaluate cloud providers in an objective way to find the most suitable ones. The proposed framework SMICloud, based on ISO SMI [64], is a cloud services evaluation framework that addresses the issue of cloud provider selection based on benchmarking of services and feedback from user experiences. ISO SMI KPIs are used as an assessment tool to compare and evaluate cloud services. The novelty of their work is the classification of requirements where users can specify requirements as essential or non-essential. By using different QoS metrics, they addressed the issues of measurement of SMI attributes and ranking of cloud providers based on these attributes. As a case study, different cloud vendors were compared using Analytic Hierarchy Process (AHP) by investigating performance heuristics from past researches. Based on these measures, cloud vendors were ranked according to services they provide.

The authors in [65] argued that quality of services of external services in SaaS is uncertain till these are not rendered. Normally, QoS attributes show randomness during different time intervals. For this, quality of services that is addressed by service provider, and QoS preferences that are aimed by user should be addressed in a systematic and coordinated manner. Normally users cannot describe QoS values at a precise but may express the values in a range. So, the authors proposed SaaS service selection for group user with interval numbers and worked on for three quality attributes reliability, reputation along with price to obtain a group of optimal services. The result of experiments for QoS values of four alternatives demonstrated that the proposed approach has linear or polynomial time complexity, so it is an effective and fast approach.

A service ranking system (SRS) was proposed in [66] that considers both static and dynamic aspects of cloud systems. User feedback is not considered while ranking cloud providers in static approach while dynamic ranking takes into account the user preferences. Seven attributes: throughput, availability, reliability, cost, response time, security and user feedback are considered for services selection and ranking. During the first stage of proposed algorithm, user is prompted to prioritize most important attributes. Then qualitative values of these attributes are calculated by service monitoring. Afterwards, user is asked to weight these attributes where cumulative weight of these attributes should be equal to one. Finally, value of every attribute is multiplied by weight and these values are sorted as ranking result. Since, in static ranking, user intervention is not required so the system assigns weights based on stage 2 and 4. A repository of cloud services along with performance information can be an add-on for better result formulation.

The authors in [67] proposed a quality of experience (QoE) aware cloud service ranking approach based on Markov chain for integration of QoE metrics for optimization of ranking results. QoE is a term used to describe overall system performance from user perspective and it is widely adopted for evaluation of multimedia services. The proposed approach not only addresses the ranking of individual services but a prediction model is also applied to assure the reliability of ranking. The process starts with user preference ranking of services which are dispatched to user evaluation component. This component employs clustering algorithm to users group based on preferences. The cloud service monitoring component collects historic data from services at runtime. For user incoming request, cloud service selection component invokes cloud ranking component to take necessary decisions based on user group level.

The authors in [68] argued to measure qualitative values of services before any comparison is being made. They ranked different services based on prediction of qualitative values. In traditional model of evaluation of service components on stand-alone systems, different service calls can be invoked to predict the overall performance; however, this may not be a case with cloud computing which involves different layers of abstraction between a service call and the response. They argued that measuring performance at server end could be a reasonable solution to calculate performance values for different parameters which may be close to what providers normally claim. However, the case may vary with qualitative values as these values are based on user preferences and evaluation of such parameters can be tested from client side. The proposed approach, CloudRank, applies qualitative values of service prediction from client side. There are two types of users in the proposed design; normal users with different requirement levels and active users who rank the system based on certain parameters. A data repository is used to store user information along with suggested ranking of services. The system comprises of three components; a similarity computation measures similarity values for all active users based on comparison from data repository. The classification component clusters users based on obtained similarity values. Finally, the ranking component executes ranking algorithm and shows the final results to the user. To evaluate the QoS ranking prediction accuracy, a large-scale real-world web service evaluation data including 300 distributed users and 500 real- world web services, all over the world, was collected. Normalized discounted cumulative gain metric was used to evaluate ranking results. For these 500 real-world web services, each user invoked each web service for one time, a total of 150,000 web service invocations were conducted. The response-time and throughput values of each invocation were recorded. Experimental results demonstrated that the proposed approach outperformed other rating-based approaches and the traditional greedy methods.

The 'aggregate approach' proposed in [69] is based on different performance benchmarking as well as input from users about service ranking. User feedback is aggregated in the benchmarking results for comparison and ranking of services. Performance results from benchmarking and user feedback are considered as objective and subjective assessment respectively. The proposed system comprises of four components. The first component, cloud selection service, aims to filter services that satisfies user's requirement without involving their qualitative value. This information is forwarded to other components of system for further evaluation. The benchmarking component is responsible for performance analysis and testing. The user feedback management component is assigned the responsibility of fuzzification of qualitative measures and it produces a series of fuzzy numbers which can be used by rules based engine for comparison of data. The final component, assessment aggregation component, returns candidate services through performed assessment by other components of the system. For the experimental purposes, two cluster systems were evaluated to test effects of job submission intervals, scheduling policies, and different categories of workload on the reported metrics of the systems. Three sets of experiments were performed to evaluate and rank two cluster systems. During the first experiment, five jobs were submitted at the same time, each job was assigned to process 8 GB data. Each job had 160 map tasks and one reduce task. In the second experiment, only one job was executed to process 40 GB data. According to the Hadoop configuration, the job had 720 map tasks while the configuration of *mapred.reduce.tasks* was set to five to match with first experiment. Experiment three was repeated with the same configuration with a minor adjustment in map tasks which were reduced to 360. Based on the experimental results, it was concluded that different categories of workloads may affect the system under test for different evaluation metrics and hence necessary customization is somehow required for benchmarking suits, according to the user requirements.

The authors in [70] proposed SLA matching approach to define cloud provider capabilities for a given quality requirement through matching SLA parameters. A four step process is involved while assessing cloud provider. The first step is to create cloud model (a RDF file) *CloudC* that lists cloud resources and their quantities, properties and a requirement model (RDF file) *CloudR* that mentions job required resources and its quantities. Then these models are transformed into graph structure using Jena API. Next step is to find pair-wise connectivity graph and induced propagation graph. Finally mapping between two models is performed using RDF Schema to determine they are equal, sub-class or super-class equivalent.

Table 2.1 presents a brief comparison of different schemes based on some commonalties, while considering the above discussion. It can be inferred from the table that mostly QoS attributes are treated as general requirements while requirements from users are mostly considered as equally important in most of the frameworks. In case of CloudRank, necessary training data in the framework is stored which can be retrieved by querying the QoS values provided by other users or by the QoS values collected through monitoring cloud services, while this concept is not addressed by other researchers.

Table 2.1: Comparison of Different Cloud Ranking Frameworks

Framework	**Technique**	**QoS attributes**	**Classification of requirements**	**Knowledgebase**
SMICloud [43]	AHP	SMI (ISO)	Essential/ Non-Essential	None
SRS [66]	Weighting	Pre-defined	None	None
CloudRank [68]	Prediction	Generic	None	Yes
Aggregation [69]	Fuzzy logic	Generic	None	None
SLA Matching [70]	None	SLA	None	None

2.3 Benchmarking Taxonomy

Benchmarking is the process of measuring services and products based on performance metrics to industry best practices and standards [71], [72]. It is based on certain indicators resulting in a metric form performance that can be comparable to other processes and products of similar nature. Cloud vendors offer heterogeneous types of resources such as computational, storage and network services with different level of quality of service. Before ranking a particular cloud provider, it is necessary to benchmark its performance based on industry best standards to compare the actual performance in contrast to the stated QoS which may vary over time.

Different providers offer different resource configurations and use different pricing and provisioning models, and, while information about pricing levels and specifications are publicly available, there is limited information about the resource performance levels. This is important for organizations looking for first opportunities to migrate their in-house IT systems to the cloud. They would like to obtain a quick assessment of the price/performance levels of different IaaS providers to match their specific business needs. However, no two vendors offer the same resource configurations, pricing and resource configuration, making the task of selecting appropriate computing resources complex, expensive and time-consuming. By combining the benchmark results with pricing information, enterprises can better identify the most appropriate cloud providers and offerings based on their specific business needs. Benchmarking as a Service (BaaS), the process of performance benchmarking of cloud infrastructure, may provide a future extension to existing cloud services.

From 2010, more and more cloud benchmarks are being developed. Smart CloudBench [73] is a platform that automates the performance benchmarking of cloud infrastructure by incorporate price as a metric. It collects a number of performance metrics for TPC-W benchmark on twenty different cloud server types under variable load conditions, including average response time, maximum response time, total number of successful interactions and total number of timeouts. It also calculates the standard deviation of average response time to determine the consistency. Then, the benchmark maps the performance metrics with the cost of the system configuration. Therefore, if users have a budget and performance requirements, they can easily shortlist the candidates from a list of different resource configurations. Even though simple load tests were performed during tested experimentation, the results show the value of having such a benchmarking tool by highlighting that price does not necessarily translate to performance (and its consistency) on the cloud, and that users do not necessarily benefit by procuring the most powerful server instances. Smart CloudBench is helpful for the users to make high level comparison of cloud offerings, but it does not provide a systematic way to compare any cloud specific attribute.

How can applications be deployed on the cloud to achieve maximum performance? The research [74] addressed the above question by proposing a benchmarking methodology in which a user provides a set of weights that indicate how important memory, local communication, computation and storage related operations are to an application. The user can either provide a set of four abstract weights or eight fine grain weights based on the knowledge of the application. The weights, along with benchmarking data collected from the cloud, are used to generate a set of two rankings - one based only on the performance of the VMs and the other takes both performance and costs into account. The authors hypothesized that by taking into account the requirements of an application, along with benchmarking data collected from the cloud, VMs can be ranked in order of performance and cost effectiveness so that a user can deploy an application on a cloud VM, which will maximize performance. In their work, the focus is on scientific High-Performance Computing (HPC) applications and maximum performance is defined as the minimum execution time of an application. The value of each weight ranges from 0 to 5, where 0 signifies that the memory and process, local communication, computation, or storage groups represented by the weight has no relevance to the application, and 5 indicates that the group is important to the application for achieving maximum performance. Overall benchmarking process comprised of six steps: (1) capture attributes of cloud VMs, (2) group attributes of cloud VMs, (3) benchmark cloud VMs, (4) normalize attribute groups, (5) provide weights to groups, and (6) rank cloud VMs. Two sets of ranks are generated; the first ranking is solely based on the performance of the VMs and the second ranking considers both performance and cost. The rankings are validated on three case study applications using two validation techniques. The case studies on a set of experimental VMs highlight that maximum performance can be achieved by the three top ranked VMs and maximum performance in a cost-effective manner is achieved by at least one of the top three ranked VMs produced by the methodology. It is deduced that high correlation between the ranks, which is an indication that the benchmarking methodology with fine-grain weights, can produce results close to reality as verified through the case studies.

While it is now widely established that running an HPC workload on top of IaaS resources induces a non-negligible performance overhead due to the hypervisor at the heart of every Cloud middleware, many people assume that this performance impact is counter-balanced by the massive cost savings brought by the Cloud approach. A TCO analysis of an in-house HPC facility was performed in [75]. This TCO model was then used to compare with the induced cost that would have been required to run the same platform with the same workload over a competitive Cloud IaaS offer. The approach to address this price comparison is three-fold. First a theoretical price - performance model based on the study of the actual cloud instances proposed by one of the major cloud IaaS actors Amazon Elastic Compute Cloud (EC2) is proposed. Then, based on the HPC facility TCO analysis, an hourly price comparison is made between the in-house cluster and the equivalent EC2 instances. Finally, based on the experimental benchmarking on the local cluster and on the Cloud instances, the model is updated to reflect the real system performance. Thus for each instance type, a pricing linear model based on linear regression analysis is computed. The model parameters are selected based on the two steps. First automated stepwise selection is performed and then from the meaningful parameters detected, the ones that are the most representative in the model via. R2 shrinkage are manually assessed. Among many parameters evaluated, it was established that the significant ones are: Processor speed (GFLOPS), Memory size (GB), Disk Size (GB) and number of GPU cores. This means that even though not all the cluster nodes have a perfect cloud instance match, it was still possible to determine what would be its equivalent price on EC2 if that matching instance was available. This information is later used to assess the interest of operating a given node class regarding renting an on demand instance with the same performance on the cloud. In both experiment sets, it was observed that for the same number of cores, the performance reached is generally the same but with a theoretical performance more than doubled for EC2 regarding gaia. This largely impacts the reached vs. theoretical ratio that is quite low for EC2. The results of the updated cost model showed that operating a local HPC platform is more cost effective on the hourly rate.

In contrast to traditional cluster computing systems, the workload in a cloud platform is much more heterogeneous, complex and dynamic and may pose special challenges for benchmarking efforts. These challenges arise from the characteristics of cloud file systems, including (1) system complexity, which makes it difficult to develop request processing models; (2) workload heterogeneity and dynamicity, which hamper efforts to identify representative workload behavior; (3) high data volume and large cluster scale, which make it challenging to replay the workload and reproduce system behavior; and (4) rapid system evolution, which requires benchmarks to accommodate changes in the underlying systems. Therefore, it is still challenging to generate realistic I/O workloads. System developers often use traditional file system benchmarks and make inaccurate assumptions on workload generation, yielding to misleading results. To address this problem, two-week I/O workload trace in a production cloud infrastructure at Alibaba Cloud Computing, which is one of the biggest cloud providers in Asia, was investigated [76]. One of key observations in earlier studies was that the request arrivals do not follow a Poisson process. Another was that the request arrival process presents multiple periodicities. The authors proposed a flexible framework iGen to mimic I/O request arrivals. One of the salient features of the iGen is that the request arrival process is modeled by three statistics properties, request arrival rate, inter-arrival time distribution, and request periodicity. According to these properties, the iGen can determine the sequence of requests and the inter-arrival time between two subsequent requests. The iGen model was then used to emulate a real workload that collected from Alibaba cloud platform. Experimentation results were synthesized and validated through a case study and it was employed that the request arrivals observed in the Pangu, showed high accuracy of the iGen.

This authors in [77] reported an early-stage performance evaluation that followed the Cloud Evaluation Experiment Methodology (CEEM) to benchmark GCE and also compared it with Amazon EC2. By employing four popular benchmarks, the study exhibited the fundamental performance of four GCE types, and also compared them with nine Amazon EC2 types to help understand the elementary capability of GCE for dealing with scientific problems. Based on the experimental results and analyses, the potential advantages of, and possible threats to applying GCE to scientific computing were analyzed. For example, GCE would be particularly suitable for applications that require frequent disk operations, while it may not well support single VM-based parallel computing. Following the same evaluation methodology, even different evaluators would be able to replicate and/or supplement this fundamental evaluation of GCE. Moreover, according to the outcome of this study, researchers and engineers can establish suitable GCE environments to carry out sophisticated and scientific case studies.

The private cloud is one typical cloud deployment model which is operated solely for a single organization, whether managed internally or by a third party. It usually hosts a number of corporate workloads, such as financial management, human resource management, internal mail, project management, development and test and so on, shared and accessible by different organization departments. It may also host a business critical customer-facing main website for a business. Workload isolation becomes very important when it mixes the internal utilities and externally visible applications. When coming to the "cloud" world, traditional benchmarks cannot serve the needs. A cloud benchmark needs to reflect not only the runtime performance as a traditional benchmark does, but also one or more cloud specific attributes, such as elasticity, deployment, resiliency and recovery. A cloud benchmark generally consists of a group of heterogeneous applications or traditional benchmarks (also called workloads) running in parallel, and representing typical customer usage of a cloud environment within a particular use case. The drivers simulating client loading of the workloads must include a variety of load behaviors, such as fixed load, periodically changing load level, spiky load changes and so forth, and even more complicated changes of transaction type mixtures. Additionally, the cloud benchmarking typically requires a number of simulated activities, including periodical deployment and removal of workloads or virtual machines, planned virtual machine or storage migrations, undesirable malicious attacks, unexpected VM failures or middleware failures and so forth. One way of defining benchmark metrics is based on observations of the internal mechanisms in a cloud. For example, an elasticity evaluation may be based on measuring a resource provisioning interval in the cloud. However, a more meaningful evaluation should be based on user-centric metrics. A user centric metric is a component metric that can be directly measured, calculated and compared by the cloud users, including workload consumers or the users who deploy and manage the workload life cycles (workload deployers, managers, maintainers, and so on). Therefore, user centric metrics are not applicable to the simulated activities that are not visible to the cloud user, for example infrastructure-level operations and tasks. Three different kinds of workload are introduced in [78]. Each workload represents current customer applications and system usage within a particular domain. The workloads appropriately exercise all aspects of the system hardware (CPU, memory, network and disk) and software (hypervisor, operating system, middleware). The workloads integrated into the benchmark are Tradelite (a variation of the DayTrader benchmark), the web service facet of SOABench, and TPC-E using an internal IBM DB2 kit. Each workload has different performance characteristics and architectural characteristics. Instead of monitoring the cloud's internal events, they can be directly measured and calculated by the cloud users. They are the key to ensure that the cloud service provider delivers the agreed terms of services to the cloud consumer.

The main costs associated with the hypervisor layer are its enter-and-exit operations, which can entail significant overhead, particularly for I/O processing (including disk and network I/Os). Different hypervisors implement I/O virtualization in different ways. Xen moves all the device drivers to Dom0, an initial domain started by the Xen hypervisor on boot and performs I/O processing on behalf of all guest VMs, while VMware installs device drivers in the hypervisor layer. Either approach results in some performance overhead. Another cost source is resource contention among co-located VMs. Virtualization promises some isolation among co-located VMs, so that the execution of one VM doesn't affect another VM's performance. The isolation mechanism works well for CPU and memory sharing, but it's less effective for shared resources such as processor caches (some VMs might use a lot of cache space, which can eventually affect collocated VMs' performance).The third cost source is complex virtualization operations such as live migration and virtual clusters, which incur performance overheads for the related VMs. Pre-copy migration, a technique that implements live migration of VMs and is widely used in both VMware and Xen hypervisor, creates several overheads. The popular benchmarks for virtualization environments such as SPECvirtsc2010, VMmark, and vConsolidate have two limitations: they measure only the server consolidation scenario's performance and leave other important scenarios (such as live migration and virtual clusters) untouched; and they have fixed, inflexible benchmark workloads that can't be configured on demand to satisfy different user requirements. To address the insufficiencies of existing benchmarking methods, a three-layer strategy fully is proposed in [79] to evaluate VM system performance. It includes a benchmark suite to measure various virtualization scenarios and an automated performance testing toolkit. It can collect performance data from the hardware, hypervisor, and VM layers. A new benchmark suite, Virt-B, is introduced that helps users understand the virtualization overhead of typical scenarios, including single machine virtualization, server consolidation, VM mapping, live migration, and virtual clusters. Finally, the toolkit implementation for automating the performance testing process is presented through several case studies. It is concluded that the proposed scenario-based three-layer benchmark method is an effective and efficient solution to comprehensively quantify the performance overheads and detect potential performance bottlenecks of VM systems.

An analysis of the performance of an I/O-intensive real scientific workflow on cloud environments using makespan (the turnaround time for a workflow to complete its execution) as the key performance metric is investigated in [80]. In particular, the impact of varying the storage configurations on workflow performance is assessed when executing on Google Cloud and Amazon Web Services. The study was aimed to understand the performance bottlenecks of the popular cloud based execution environments. Experimental results showed significant differences in application performance for different configurations. They also revealed that Amazon Web Services outperforms Google Cloud with equivalent application and system configurations. The Montage workflow, a well-known astronomy application, was used as a benchmark to quantify application performance. The Montage workflow is composed of thousands of computing jobs and manages over 20,000 data transfers. The instances of Montage on different storage deployment configurations were executed in both cloud systems.

Performance metrics such as makespan were then collected to compare the efficiency of these systems. This comparison unveils significant performance differences among configurations revealing the impact of the bottlenecks in the storage configuration. A remarkable difference between the application's performance on both cloud systems was also noticed despite the similarity of the execution environments (in terms of VM types and software) and configurations used. The study results revealed that the standard cloud environments may present performance issues for running I/O-intensive workflows. In particular, for workflows that operate over a large number of small files, the performance may be poor, as noticed for the Montage workflow. It was identified that overhead incurred on individual file transfer tends to be the culprit as well as the performance measures obtained from workflow runs on Amazon are up to 44% faster than runs conducted on Google.

Table 2.2 provides a comparison of above cited literature based on some study variables.

Table 2.2: Comparison of Different Cloud Benchmarking Studies

Study	**Model**	**Cloud Providers**	**Method**	**Tools**	**Metrics**	**Comparative Validation**
CloudBench [73]	Automated Performance Benchmarking	Amazon EC2, GoGrid, Rackspace	Experiment	TPC-W	General Performance evaluation	None
Benchmarking for Maximising Performance [74]	Weight Based	Amazon	Experiment	bonnie++, lmbench, sysbench	memory and process, local communication, computation and storage	Case study
EC2 vs. in-House HPC [75]	Amazon EC2 vs local cluster performance comparison	Amazon	Experiment	IOZone	Computation and storage evaluation	None
iGen [76]	Customized workload generator	Alibaba Cloud Computing	Experiment	iGen	Cloud file system evaluation	K-S test

Performance of Google Compute [77]	Google Compute Engine vs Amazon EC2 Performance Evaluation	Google Compute, Amazon EC2	Experiment	Iperf, Ping, STREAM, Bonnie++, NPB-MPI	Communication, Memory, Storage and Computation performance evaluation	None
Benchmarking Private Cloud [78]	Automated Performance evaluation based on user centric metrics	Not mentioned	Experiment	Workload generation tools for DayTrader, SOABench, TPC-E benchmarks	Baseline, elasticity, and deployment performance evaluation	None
Virt-B [79]	Automatic benchmark suite for single and virtual cluster	Local Datacenter	Experiment	SPECjbb, IOzone, Sysbench, Webbench	Performance evaluation of single machine virtualization, server consolidation, VM mapping, live migration, and virtual clusters	None
I/O-Intensive Workflow on Google and Amazon [80]	I/O-Intensive Workflow benchmarking between Amazon and Google Clouds	Amazon, Google	Experiment	Montage, Pegasus workflow management system	Performance evaluation of disk I/O and network benchmarking	None

2.3.1 Benchmarking Frameworks

Different benchmarking techniques and methodologies have been proposed in the literature. In this section, the most relevant studies are reviewed and discussed:

2.3.1.1 CloudCmp [32]

Migration of legacy systems to current state of the art is a challenging issue for industry practitioners and enterprise developers. CloudCmp is a framework aimed to estimate the cost and performance of legacy systems while porting and deploying it over the cloud. Three phases were involved in this process: In the service benchmarking phases, six cloud vendors (including Google AppEngine , Amazon AWS, Microsoft Azure , GoGrid, and Rackspace) were selected considering the web application deployment features necessary for cloud computing services. Four types of service benchmarking were applied including elastic compute cluster, persistent storage, intra-cloud networking, and wide-area delivery networking. Every cloud provider was assessed for performance and associated cost by running a collection of benchmarking test application. Since portability was a key concern in this environment, cloud services were tested using SPECjvm2008 Java benchmarking toolkit. The performance of individual services was tested by measuring starting and finishing time while cost effectiveness was measured by the cost per task. Server response time was measured by the time a VM was requested and when the VM was provisioned. However, this metric is limited considering the fact that not all services allow scaling request in this way. To measure the performance of storage system, total time to insert random records from data table was measured. This test showed the correlation between table size and disk operation which has a significant effect on the performance. Two types of network services were tested in this framework: Inter-cloud network was tested using iperf tool to establish a paired connection between two ends, measured available bandwidth and response time while wide area network was tested by sending ICMP ping requests to calculate network latency and packet loss. The second phase of application workload collection was aimed to represent workload of a user's legacy application. It was concluded that this can be achieved by collecting the application's request traces and driving an execution path for each request. In the final phase of performance prediction, each cloud provider profile and representation of workload was used as a measure for legacy applications to predict total running time and total cost of running time.

2.3.1.2 CloudStone [81]

CloudStone is a University of California, Berkeley open source project aimed at characterizing workload of social network websites using web 2.0 applications. The goal is to provide a benchmark for investigating fair performance assessment of implementation decisions under different deployment environments. The framework offers various AMI's (virtual machine image files compatible with Amazon's Elastic Compute Cloud) to facilitate the workflow. Cloudstone is based on three components: Olio, a work load generator and social event calendar that may support thousands of online users. A collection of open source tools for database and metric formulation for experimenting Olio on Amazon EC2 and a set of parameters for calculating the cost per month of applications under deployment.

The cloud benchmarking approach reflects the overall transaction performance of one specific application running over the cloud. The benchmarking results are intended to show the maximum number of users that can be served with one setup consisting of a virtual machine type and software configuration [29]. Furthermore, Cloudstone does not define a procedure to evaluate the cost of virtual machine migration [82].

2.3.1.3 HiBench [83]

HiBench, originally proposed by Intel Company, is a Hadoop benchmarking suite involving BigData storage and processing. It is aimed to characterize and evaluate Hadoop's parallel computing component (MapReduce) and database component (HDFS). A total of 11 workloads tasks were chosen for benchmarking including microbenchmarks, web search tasks, machine learning tasks, and HDFS benchmark. Different workload schemes are presented including WordCount, TeraSort , Bayesian Classification, K-means Clustering as an input to evaluate Hadoop performance. Although HiBench benchmark suite includes a wider variety of jobs, yet it fails to capture the different job mixes and job arrival rates that one would expect in production MapReduce clusters [84].

2.3.1.4 Yahoo Cloud Serving Benchmark (YCSB) [85]

Yahoo Cloud Serving Benchmark is a framework for benchmarking different cloud database storage and distributed data serving systems such as Cassandra, HBase, and PNUTS. YCSB emulates a synthetic workload generator that can be parameterized to vary the read/write ratio, access distributions. It was originally designed to evaluate Key-Value stores and hence primarily designed for single key operations or scans [86]. It can be configured on scalable serving system for database operations (read, insert and delete). Based on these operations, it measures database performance for throughput and response time. The framework consists of two layers: First layer (performance) is for general performance evaluation and here, latency is calculated as throughput is increased. The Second layer- (scaling) analyses system performance through scale-up and elastic speedup by adding more machines to the system. Five different types of random workloads were used including Workload A (50 percent reads and 50 percent updates), Workload B (read heavy workload), workload C (read only), Workload D (read latest) and workload E (short ranges). The YCSB package includes a standard workload executor for the core package to test the measures of performance, scalability and elasticity of serving servers.

2.3.1.5 CloudSuite [87]

CloudSuite is a benchmarking suite for scale-out datacenter services involving massive amounts of human-generated data. Scale-out workloads have many inherent characteristics that place them in the distinct class of desktop, parallel and traditional server workloads. These workloads operate across a large number of machines to process large datasets that usually do not share any state. In the research study, performance counters were analyzed for micro-architecture behavior for a wide range of scale-out workloads. Key sources of inefficiencies in frontend core and L2 cache, data-access and bandwidth were analyzed and identified. The results show significant over-provisioning of core micro-architectural resources as well as out-of-order execution elides stalls due to memory accesses. The results also suggest that the memory system behavior is closely related to traditional online transaction processing work-loads such as TPC-C, TPC-E and web backend. However, the workloads differ considerably from online transaction processing such as TPC-C. Further research in this area will result in widening the mismatch between the scale-out workloads and future server processors.

Table 2.3 provides a comparison of above cited literature based on some study variables. As shown in the table, CloudCmp supports both IaaS and PaaS platforms while all other frameworks are aimed at benchmarking a single cloud deployment model. Elasticity and Scalability is mostly achieved using load balancers or by using speed-up/ scale up of the scaling tier of the framework. Task, Throughput, instruction cache and read/ modify latencies are the core metrics, being evaluated by these frameworks.

Table 2.3: Comparison of Different Benchmarking Techniques

Variable	**CloudCmp [32]**	**CloudStone [81]**	**HiBench [83]**	**YCSB [85]**	**CloudSuite [87]**
Objective	Execution of applications on cloud	Model web 2.0 behavior in cloud computing	Hadoop (Map-reduce) applications	Benchmarking cloud systems	Workload management
Elasticity/ Scaling	Response time for provisioning new instances	Load Balancer	------	Speed up/ Scale up	--------
Storage	Disk I/O for inserting / fetching random entries	Database	HDFS	Different workload distribution	iSCSI storage array
Metrics	Task latency through SPECjvm 2008	Response Time	Throughput, resource utilization and job execution speed	Read/ Modify latency	Bandwidth, memory utilization, instruction cache, execution time
Experimentation	Variable Instances	Amazon EC2	Hadoop Cluster	Operational servers	Operational servers
Platform	IaaS/ PaaS	IaaS	PaaS	PaaS	IaaS

2.3.2 Benchmark Applications

Different Industry standard tools are available to benchmark computing systems. In this section, the most widely used tools are discussed.

2.3.2.1 SPECjvm [88]

This tool is developed for testing and benchmarking software systems' performance especially web services under platform independent Java Runtime Environment (JRE). It contains a set of test suits, based on file I/O, network I/O and other application computations to measure CPU, operating system and memory sub-system performance.

The tool is focused on performance evaluation of the hardware processor and memory subsystem of single application under Java Runtime Environment execution. It has low dependency on file I/O while network bandwidth related operations are not part of the benchmark. Application workload for the benchmarks mimics multiple general purpose application computations which are applicable to measure Java performance on a variety of client server based application environment.

2.3.2.2 Open-Source Benchmarking [89]

The Phoronix Test Suite is one of the comprehensive benchmarking toolkits for measurement of qualitative and quantitative test cases. With over 450 test profiles and 100 test suites, the tool supports stress testing of system, processor, graphics, memory and network related performance assessment. If a particular test is missing in the repository, it is automatically downloaded and installed.

A variety of test suites are available to benchmark target system performance that includes but are not limited to monitoring CPU, graphics, memory subsystem, disk I/O, ray-tracing application benchmarks and battery power consumption. The test profiles also support cascading test profiles whereby an existing profile can be extended.

2.3.2.3 The DaCapo Benchmarks [90]

This toolkit is designed to assess CPU and memory related performance metrics. Developed under Java platform, it consists of a set of open source testing applications with different workload for performance evaluation. It introduces time series and statistical metrics for measurement of static and dynamic properties such as code complexity, code size, heap composition, and pointer mutations.

The framework is aimed at a client-side applications environment that can be widely used to provide a compelling focus with minimal dependences for the community's innovation and optimizations as compared to other synthetic benchmarks. Experimental results have demonstrated that DaCapo benchmarks show much richer lifetime behaviors than SPEC [91].

2.3.2.4 IOzone Filesystem Benchmark [92]

IOzone is aimed to measure and generate a variety of file I/O operations for file system analysis of different operating systems. It supports I/O operations for read, write, re-read, re-write, read backwards, read strided, fread, fwrite, random read, pread ,mmap, aio_read and aio_write. By using the broad performance coverage for multiple operating systems including Linux, Solaris, MAC OS X and Windows, users can decide a more balanced platform that suits their application requirements.

2.3.2.5 Bonnie++ [93]

Bonie++ is a testing tool for experimenting a number of storage and file system operations. It simulates the usage of programs like Squid, INN, or Maildir format email by creating, deleting and reading a set of files with storage in GBs to measure file system performance. For RAID arrays storage devices, Bonnie++ supports multiple types of IO operations at the same time. It has also the facility to test more than 2G of storage on a 32bit or 64 bit machines.

2.3.2.6 Cachebench [94]

Cachebench is a benchmarking suite for measuring memory sub-system performance. The novel features of tool are building different test cases, execution and result generation using GNUPlot. Cachebench supports 8 different read, write and read/modify/write tests with features like automated build test, execution and result processing, safe from aggressive optimizing compilers. It mimics the performance of multiple levels of system cache present to establish peak computation rate given optimal cache reuse and to verify the effectiveness of high levels of compiler optimization.

2.4. Taxonomy of Large Scale Resource Management Frameworks

The modern day advancement is increasingly digitizing our lives which has led to a rapid growth of data. Such multi-dimensional datasets are precious due to the potential of unearthing new knowledge and developing decision making insights from them. Analyzing this huge amount of data from multiple sources can help organizations to plan for future and anticipate changing market trends and customer requirements. While the Hadoop framework is a popular platform for processing larger datasets, there are a number of other compute infrastructures, available to use in various applications domains.

Big Data applications might be viewed as the advancement of parallel computing, but with the important exception of the scale. The scale is the necessity arising from the nature of the target issues: data dimensions largely exceed conventional storage units, the level of parallelism needed to perform computation within a strict deadline is high and obtaining final results require the aggregation of large numbers of partial results. The scale factor, in this case, does not only have the same effect that it has in classical parallel computing, but it surges towards a dimension in which an automated resource management and their exploitation is of significant value [95].

An important consideration of modern applications is the massive amount of data that needs to be processed. Such data usually originate from different sets of devices (e.g., public web, business applications, satellites or sensors) and procedures (e.g., case studies, observational studies or simulations). Therefore, it is imperative to develop computational architectures with even better performance to support current and future application needs. Historically, this need for computational resources was provided by high-performance computing (HPC) environments such as computer clusters, supercomputers, and grids. In traditional owner-centric HPC environments, internal resources are handled by a single administrative domain [96] . Although organizations usually prefer to store their most sensitive data internally (on-premises), huge volumes of big data (owned by the enterprises or generated by third-parties) may be stored externally, some of it may already be on a cloud. Retaining all data sources behind the firewall may result in a significant waste of resources. Analyzing the data where it resides either internally or in a public cloud data center makes more sense [97], [98].

The primary object of the section is how to classify different big data resource management systems. We use various evaluation metrics for popular big data frameworks from different aspects. We also identify some key features which characterize big data frameworks as well as their associated challenges and issues and hence the need for a more generic framework that supports massively large data handling considering the scalability and elasticity specific requirements of application domain.

While the Hadoop framework is a popular platform for processing huge datasets in parallel batch mode using commodity computational resources, there are a number of other compute infrastructures that can be used in various applications domains. The primary focus of this section is to investigate popular big data resource management frameworks which are commonly used in cloud computing environment. Most of the popular big data tools available for cloud computing platform, including the Hadoop ecosystem, are available under open source licenses.

2.4.1 Hadoop

Hadoop [35] is a distributed programming and storage infrastructure based on the open-source implementation of the MapReduce model [99]. MapReduce is the first and current de-facto programming environment for developing data-centric parallel applications for parsing and processing large datasets. The MapReduce is inspired by Map and Reduce primitives used in functional programming. In MapReduce programming, users only have to write the logic of Mapper and Reducer while the process of shuffling, partitioning and sorting is automatically handled by the execution engine [99], [100]. The data can either be saved in the Hadoop file-system as unstructured data or in a database as structured data [101]. Hadoop Distributed File System (HDFS) is responsible to break large data files into smaller pieces known as blocks. The blocks are placed on different data nodes, and it is the job of the NameNode to notice what blocks on which data nodes make up the complete file. The NameNode also works as a traffic cop, handling all access to the files, including reads, writes, creates, deletes, and replication of data blocks on the data nodes. A pipeline is a link between multiple data nodes that exists to handle the transfer of data across the servers. A user application pushes a block to the first data node in the pipeline. The data node takes over and forwards the block to the next node in the pipeline; this continues until all the data, and all the data replicas, are saved to disk. Afterwards, the client repeats the process by writing the next block in the file [102].

YARN is the core Hadoop service to provide two major functionalities: Global resource management (ResourceManager) and Per-application management (ApplicationMaster). The ResourceManager is a master service which controls NodeManager in each of the nodes of a Hadoop cluster. It includes a scheduler, whose main task is to allocate system resources to specific running applications. All the required system information is tracked by a Resource Container which monitors CPU, storage, network and other important resource attributes necessary for executing applications in the cluster. The RessourceManager has a slave NodeManager service to monitor application usage statistics. Each deployed application is handled by a corresponding ApplicationMaster service. If more resources are required to support the running application, the ApplicationMaster requests the NodeManager and the NodeManager negotiates with the ResourceManager (scheduler) for the additional capacity on behalf of the application [35].

2.4.2 Spark

Apache Spark [32], originally developed as Berkeley Spark, was proposed as an alternative to Hadoop. It can perform faster parallel computing operations by using in-memory primitives.
A job can load data in either local memory or a cluster-wide shared memory and query it iteratively with much a great speed as compared to disk-based systems such as Hadoop MapReduce [99]. Spark has been developed for two applications where keeping data in memory may significantly improve performance: Iterative machine learning algorithms and interactive data mining. Spark is also intended to unify the current processing stack, where batch processing is performed using MapReduce, interactive queries using HBase and the processing of streams for real-time analytics using other frameworks such Twitter’s Storm. Spark offers programmers a functional programming paradigm with data-centric programming interfaces built on top of a new data-model called Resilient Distributed Dataset (RDD) which is a collection of objects spread across a cluster stored in memory or disk [100]. Applications in Spark can load these RDDs into the memory of a cluster of nodes and let the Spark engine automatically manage the partitioning of the data and its locality during runtime. This versatile iterative model makes it

possible to control the persistence and manage the partitioning of data. A stream of incoming data can be partitioned into a series of batches and is processed as a sequence of small-batch jobs. The Spark framework allows this seamless combination of streaming and batch processing in a unified system. To provide rapid application development, Spark provides clean, concise APIs in Scala, Java, and Python. Spark can be used interactively from the Scala and Python shells to rapidly query big datasets.

2.4.3 Flink

Apache Flink is an emerging competitor of Spark which offers functional programming interfaces, much similar to Spark. It shares many programming primitives and transformations in the same way as what Spark does for iterative development, predictive analysis and graph stream processing. Flink is developed to fill the gap left by Spark, which uses a mini-batch streaming processing instead of a pure streaming approach. Flink ensures high processing performance when dealing with complex big data structures such as graphs. Flink programs are regular applications which are written with a rich set of transformation operations (such as mapping, filtering, grouping, aggregating and joining) to the input datasets. The Flink dataset uses a table-based model, therefore application developers can use index numbers to specify a particular field of a dataset [99], [100].

Flink is capable to achieve high throughput and a low latency, thereby processing a bundle of data very quickly. It is designed to run on large-scale clusters with many thousands of nodes, and in addition to a standalone cluster mode, Flink provides support for YARN. For distributed environment, Flink chains operator subtasks together into tasks. Each task is executed by one thread [33]. Flink runtime consists of two types of processes: There is at least one JobManager (also called masters) which coordinate the distributed execution. It schedules tasks, coordinate checkpoints and coordinate recovery on failures. A high-availability setup may involve multiple JobManagers, one of which one is always the leader, and the others are standby. The TaskManagers (also called workers) execute the tasks (or more specifically, the subtasks) of a dataflow/ buffer and exchange the data streams. There must always be at least one TaskManager. The JobManagers and TaskManagers can be started in various ways: directly on the machines as a standalone cluster, in containers, or managed by resource frameworks like YARN or Mesos.

TaskManagers connect to JobManagers, announcing themselves as available, and are assigned work.

2.4.4 Storm

Storm [34] is a free open source distributed stream processing computation framework. It takes several characteristics from the popular actor-model and can be used with practically any kind of programming language for developing applications such as real-time streaming analytics, critical work flow systems and data delivery services. The engine may process billions of tuples each day in a fault-tolerant way. It can be integrated with popular resource management frameworks such as YARN, Mesos, Docker and many others. Apache Storm cluster is made up of two types of processing actors: Spouts and Bolts.

• Spout is connected to the external data source of a stream and is continuously emitting or collecting new data for further processing.

• Bolt is a processing logic unit within a streaming processing topology, each bolt is responsible for a certain processing task such as transformation, filtering, aggregating and partitioning.

Storm defines workflow as directed acyclic graphs (DAGs), called topologies with connected Spouts and Bolts as vertices. Edges in the graph define the link between the bolts and the data stream. Unlike batch jobs being only executed once, Storm jobs run forever until they are killed. There are two types of nodes in a Storm cluster: Nimbus (master node) and Supervisor (worker node). Nimbus, similar to Hadoop JobTracker, is the core component of Apache Storm and is responsible to distribute load across the cluster, queue and assign tasks to different processing units and monitor execution status. Each worker node executes a process known as the supervisor which may have one or more worker processes. Supervisor delegates the tasks to worker processes. Worker process then creates a subset of topology to run the task. Apache Storm does rely on an internal distributed messaging system, called Netty, for the communication between nimbus and supervisors. Zookeeper manages the communication between real-time job trackers (nimbus) and supervisors (storm workers).

2.4.5 Comparative Analysis of Large Scale Resource Management Frameworks

Big data in cloud computing, a popular research trend, is posing significant influence on current enterprises, IT industries and research communities. There are a number of disruptive and transformative big data technologies and solutions that are rapidly emanating and evolving in order to provide data-driven insight and innovation. Furthermore, modern cloud computing services are offering all kinds of big data analytics tools, technologies and compute infrastructure to speed up the data analysis process at an affordable cost. Although many distributed resource management frameworks are available nowadays, the main issue is how to select a suitable big data framework. The selection of one big data platform over the others will come down to the specific application requirements and constraints that may involve several tradeoffs and application usage scenarios. A brief comparison of big data frameworks is presented in table 2.4.

Table 2.4: Comparison of Big Data Frameworks

Framework / **Attribute**	**Hadoop**	**Spark**	**Storm**	**Flink**
Current Stable Version	2.8.1	2.2.0	1.1.1	1.3.2
Batch Processing	Yes	Yes	Yes	Yes
Computational Model	Map-reduce	Streaming(micro -batches)	Streaming(micro -batches)	Supports continuous flow streaming, micro-batch, and batch
Data Flow	chain of stages	Directed acyclic graph	Directed acyclic graphs (DAGs) with Spouts and Bolts	controlled cyclic dependency graph through machine learning
Resource Management	YARN	YARN/ Mesos	HDFS (YARN)/ Mesos	Zookeeper/YARN/ Mesos
Language Support	All major	Java, Scala,	Any	Java, Scala, Python

	languages	Python and R	programming language	and R
Job Management/ Optimization	MapReduce approach	Catalyst extension	Storm-YARN/ 3rd party tools like Ganglia	Internal optimizer
Interactive Mode	None (3rd party tools like Impala can be integrated)	Interactive Shell	None	*Scala Shell*
Machine Learning Libraries	Apache Mahout/ H2O	Spark ML and MLlib	Trident-ML/ Apache SAMOA	FlinkML
Maximum Reported Nodes (Scalability)	Yahoo Hadoop Cluster with 42,000 nodes	8000	300	Alibaba customized Flink Cluster with 1000s of nodes

Hadoop MapReduce has a clear edge on large-scale deployment and larger dataset processing. Hadoop is highly compatible and interoperable with other frameworks. It also offers a reliable fault tolerance mechanism to provide a failure-free mechanism for over a long period of time. Hadoop can operate on a low-cost configuration. However, Hadoop is not suitable for real-time applications. It has a significant disadvantage when latency, throughput and iterative job support for machine learning are the key considerations of application requirements.

Apache Spark is designed to be a replacement for batch-oriented Hadoop eco-system to run-over static and real-time datasets. It is highly suitable for high throughput streaming applications where latency is not a major issue. Spark is memory intensive and all operations take place in memory. As a result, it may crash if enough memory is not available for further operations (before the release of Spark version 1.5, it was not capable of handling datasets larger than the size of RAM and the problem of handling larger dataset still persists in the newer releases with different performance overheads). Few research efforts, such as Project Tungsten, are aimed to address the efficiency of memory and CPU for Spark applications. Spark also lacks its own storage system so its integration with HDFS through YARN or Cassandra using Mesos is an extra overhead for cluster configuration.

Apache Flink is a true streaming engine. Flink supports both batch and real-time operations over a common run-time to fulfill the requirements of Lambda architecture. However, it may also work in batch mode by stopping the streaming source. Like Spark, Flink performs all operations in memory, but in case of memory hog, it may also use disk storage to avoid application failure. Flink has some major advantages over Hadoop and Spark by providing better support for iterative processing with high throughput at the cost of low latency.

Apache Storm was designed to provide a scalable, fault tolerance, real-time streaming engine for data analysis what Hadoop did for batch processing. However, the empirical evidence suggests that Apache Storm proved to be inefficient to meet the scale up/ scale down requirements for real-time big data applications. Furthermore, since it uses micro-bath stream processing, it is neither very efficient where continuous stream process is a major concern nor it provides a mechanism for simple batch processing. For fault-tolerance, Storm uses Zookeeper to store the state of the processes which may involve some extra-overhead and may also result in message loss. On the other hand, Storm is an ideal solution for near real-time application processing where workload could be processed with a minimal delay with strict latency requirements.

Guenter et al. [103] conducted a conceptual survey on stream processing systems. However, their discussion was focused on some basic differences related to real-time data processing engines. Dilpreet et al. [104] provided a thorough analysis of big data analytics platforms that included Peer-to-peer networks, Field programmable gate arrays (FPGA), Apache Hadoop ecosystem, High-performance computing (HPC) clusters, Multi-core CPU and Graphics processing unit (GPU). Sara et al. [105] focused on machine learning libraries and their evaluation based on ease of use, scalability and extensibility. C.L. Philip et al. [106] discussed big data problems, challenges and associated techniques and technologies to address these issues. Several potential techniques including cloud computing, quantum computing, granular computing and biological computing were investigated and the possible opportunities to explore these domains were demonstrated. However, the performance evaluation was discussed only on theoretical grounds. A taxonomy and detailed analysis of the state-of-the-art in big data 2.0

processing systems was presented in [107]. The focus of the study was to identify current research challenges and highlight opportunities for new innovations and optimization for future research and development. Marcos et al. [108] reviewed multiple generations of data stream processing frameworks that provide mechanisms for resource elasticity to match the demands of stream processing services. The study examined the challenges associated with efficient resource management decisions and suggested solutions derived from the existing research studies. However, the study metrics are restricted to the elasticity/ scalability aspect of big data streaming frameworks.

As shown in table 2.5, performance evaluation of these resource management frameworks is a key research issue in on-going research in big data domain and it needs further investigation to cover application specific data requirements and challenges.

Table 2.5: Comparison and Application Areas of Related Research Studies

Study Reference	**Data Model**	**Resource Frameworks**	**Study Features**	**Evaluation/ Ranking Methodology**
[103]	Data Stream Processing Systems	Storm, Flink, Spark, Samza	A brief comparison of resource frameworks	✘
[104]	Batch and Stream Processing Systems	Horizontal scaling systems such as Peer-to-Peer, MapRedce/MPI, Spark and Vertical Scaling Systems such as: CUDA and HDL	Comparison of horizontal and vertical scaling systems	Theoretical comparison of resource frameworks
[105]	Batch and Stream processing Engines	MapReduce, Spark, Flink, Storm as well as machine learning	Machine learning libraries and their evaluation mechanism	Performance comparison with respect to machine

		libraries		learning toolkits
[106]	Batch and Stream processing frameworks	Hadoop, Storm and other big data frameworks	In-depth analysis of big data opportunities and challenges	✗
[107]	Batch and Stream processing frameworks	Hadoop, Spark, Storm, Flink, Tez as well as SQL, Graph, Bulk Synchronous Parallel Model	Analysis of current open research challenges in the field of big data and the promising directions for future research	✗
[108]	Stream processing engines	Apache Storm, S4, Flink, Samza, Spark Streaming and Twitter Heron	Classification of elasticity metrics for resource allocation strategies that meet the demands of stream processing services	Evaluation of elasticity/ scaling metrics for stream processing systems

2.5. Scope and Positioning of the Research Study

In cloud, provisioning of computing resources is offered in the form of virtual machines (VMs), being deployed on physical computing nodes on pay-per use pricing policy. Cloud providers specify their offerings to the clients on hourly, monthly, semiannual and annual basis with different performance indicators measured by themselves. Mostly these indicators don't provide comprehensive information about overall performance of virtual machines. For instance, a 4 GB standard instance from Rackspace is offered with 2 vCPU (weighted based on the size of the server), 4 GB RAM, 400 Mb/s network, 160 GB system disk with good disk I/O. For quantitative analysis of cloud performance and associated monetary cost, a comparison is generally required between cloud providers with similar VM specification. A much similar instance of Amazon t2.medium is offered with 2 vCPU, 4 GB memory with low to moderate network performance. For choosing a best-fit cloud provider, the user needs a good understanding of mapping from low, moderate and good in terms of disk I/O and network performance. However, because of the over-provisioning of cloud resources and underlying hardware, these are not necessarily the decisive indicators [29], [109]. The characterization and evaluation of cloud providers is the first step to understand what instances are more appropriate for a particular set of applications. Providing a benchmark and ranking methodology for investigating fair performance assessment of implementation decisions under different deployment environments is necessary to compare current variety of cloud providers [109].

The aim of the research study is to improve the decision accuracy while choosing a cloud provider for a given set of user preferences. The evaluation process helps to measure instance configuration in an objective way to find the most suitable ones based on benchmarking as well as real world experimentation to filter, select and monitor the VMs across multi-cloud environment based on QoS requirements of user jobs. Since cost and deadline are two key considerations in such an environment [110], [111], the analysis of price vs. performance of different cloud providers is proposed through the broker based resource provisioning framework to allow scientific community to find the most cost effective virtual machines that fit their application needs as well as reduce overall cost. From the earlier literature, it is evident that broker enabled QoS ranking, negotiation and monitoring framework based on user level QoS requirements that determine users' needs and utility is still a missing area of research which is addressed in this research study.

Chapter 3: Broker based Resource Provisioning Framework

In this chapter, a basic mechanism for representing and measuring QoS of IaaS computing resources across multiple clouds is described. In this proposed approach, the broker acts on behalf of the consumer to process all the SLA management tasks. Considering the issues involved while selecting a best-fit IaaS candidate among a set of cloud providers, we propose a mathematical model for cloud ranking as well as a framework based on job management, SLA negotiation, service provisioning and resource management is discussed in this chapter.

3.1 Background

Normally, user requirements can be categorized in two types: obligatory/critical of highest priority and trivial requirements with comparatively low priority [40]. The designed framework allows service filtering based on QoS requirements and then calculating providers' ranking based on performance of cloud infrastructure along with previous job experience to select the 'right cloud provider' for a particular set of quality attributes required to satisfy incoming job requirements. In this model, a higher ranked cloud provider for a particular set of quality attributes has more probability to be selected as compared to low ranked cloud providers. However, if the required minimum level (or in some cases the maximum level such as server response time) of critical QoS requirements does not meet the job quality constraints, the provider is removed from the list without taking into account the overall rank value of the cloud provider.

In virtual cloud, cloud provider, underlining public cloud infrastructure, and cloud service provider realizing cloud resources/ services may be different vendors. Resources of other cloud providers are normally borrowed to meet end user requirements. Cloud service provider does not itself own networking or data center resources. A cloud consumer can construct virtual cloud by leasing virtual machines from the cloud providers. A central entity (also known as broker or mediator, global cloud agent/coordinator) performs or facilitates multiple clouds to share resources. Cloud broker acts as an intermediary between service consumers and producers. Cloud consumers can find best provider and service through the matchmaking process of cloud broker.

The primary design goal of the proposed system is to facilitate user job execution by automating the entire process on hand and achieve economic efficiency on the other hand. The core components of the system are cloud customizer broker at user end and server side cloud provider component.

A cloud broker may also provide customers with some additional services, encryption and transfer of consumer data to the cloud and monitoring data life cycle management. Such broker is known as cloud enabler or cloud aggregator. Sometimes a broker integrates cloud services on behalf of customers to work together and sells the services under their own brand; such broker is known as cloud customizer or white label cloud service [112].

In this study, the cloud customizer involves following phases:

- Job Admission: Jobs are submitted by the users along with necessary information including task(s) to be executed, budget, deadline, QoS parameters etc.
- Runtime Estimation phase: When a job is submitted, the broker estimates the job characteristics and schedule amount of processing nodes considering the workload requirements.
- Discovery phase: Cloud customizer queries a list of resource/ service providers that satisfies the requirements.
- Resource Selection: Checks individual resource / service provider to confirm the service requirements. The cost of executing a task is obtained by querying cloud provider. A final priority order list is generated after confirming each individual provider.
- Scheduling Module: This upper layer scheduling module is responsible for creation of the virtual machine pool according to actual state information of the user job. Based on the priority order, the module aims to complete job within budget and deadline constraints.
- Resource monitoring: This module of broker continuously monitors secured resources against service abruption, violation of SLA, QoS and so on.
- Resource switchover: In case of service abruption or low QoS, provisioning of resources is re-evaluated to meet QoS and deadline constraints
- Release of Resources: Unused resources are released after successful execution of job

After provisioning of required resources, broker continuously monitors foreign cloud provider for server availability and other quality attributes. In case of successful job execution, broker receives feedback from the foreign cloud provider or a risk alert is generated in case of any violation of SLA agreement. If QoS requirements are not satisfying SLA agreement, foreign cloud provider is penalized for an agreed penalty which is returned to home cloud provider. In this model, broker also takes into account migration strategies when a foreign CP is not responding to avoid SLA violations. Figure 3.1 illustrates the process of resource provisioning:

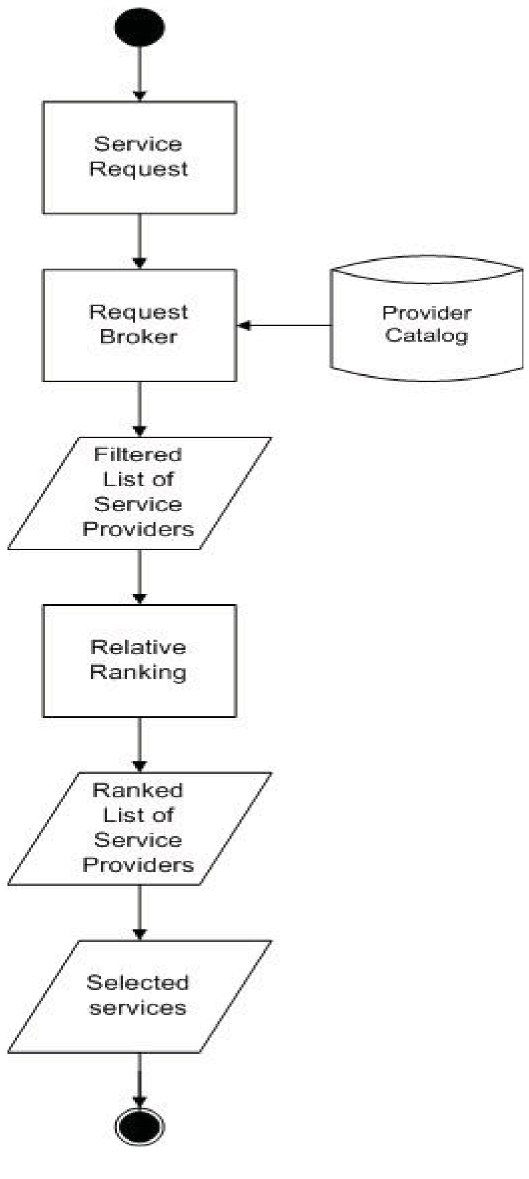


Fig. 3.1: General Process of Resource Provisioning, adapted from [2]

3.2 Negotiation Mechanism

A system model is considered that consists of a set of cloud providers { S_1, S_2, …, S_n }mediated through a broker. Each cloud provider offers *m* type of VM instances { VM_1, VM_2, …, VM_m }, each of which offers different CPU cores, disk storage and memory size. The incurred cost of a cloud provider S_i by c_{ij} for provisioning each VM of type VM_j and resource offered price by p_{ij}.

In case of resource requirement, a home CP submits VM provisioning request, denoted by { r_1, r_2, …, r_m}, where r_j represents resource request for VM_j. At the time of job completion, CP has to pay $\sum_{j=1}^{m} r_j \cdot p_j$.

3.2.1 Assumptions

The required assumptions in the selection process of foreign CP are as follows:

1. Every CP in the member clouds has the capacity to offer spare resources.
2. Only a pre-defined set of VM instances are offered.
3. Both providers and consumers are interested to maximize their objective of resource utilization without direct cooperation with each other (not peer-to-peer federation model).

Several negotiation strategies have been proposed in the literature [41], [113]. The basic steps of proposed negotiation process are as under:

Step 1: CPs in the member clouds provide the prices and SLA terms of their resources which is stored in the broker repository.

Step 2: Home CP submits a job (VM request) to the broker with all QoS parameters q_j.

Step 3: Broker filters the job requirements and provides home CP a list of best-fit cloud providers to accept one of the candidate foreign cloud provider.

Step 4: Upon acceptance, an agreement is established and resource is provisioned.

Step 5: SLA monitoring engine periodically examines any violation of SLA.

Step 6: If any serious violation of SLA is found, an alarm is generated by the SLA event generator of the SLA enforcement engine and it is notified to the enforcement engine of SLA; broker updates the foreign CP reputation ranking with a negative rating and starts from step 2 for migration strategy.

Step 7: Broker evaluates job performance with required QoS constraints.

Step 8: A notification is sent to home CP about job completion and agreement is terminated.

To reduce cost and avoid over-provisioning of resources, exact matching mechanism is used wherever possible in step 3. However, if this is not possible, an alternative exists:

Step 3*: If an exact match is not available, the broker must select a VM instance among a large number of dominating VM instances that may be available on several member clouds (stochastic least differential capacity).

Each CP is assigned an initial reputation rating which is stored in reputation repository. Based on the job experience and performance, this rating is dynamically updated for a provider involved as a foreign CP. In case of SLA violation, an additional operation is performed as under:

Step 7:* If actual CP performance violates SLA constraint, broker updates reputation repository with a negative rating for the CP otherwise it updates with positive rating.

The migration strategy, specified in step 7*, is the process of replacement of a CP with another one when a risk alarm is generated or when a foreign CP is not responding.

The primary objective of global federation marketplace is to provide a resource sharing platform for profit maximization of individual clouds by maximum utilization of resources within SLA constraints as well as perform well in social welfare [114], [115]. The maximum profit obtained when clouds operate in federation, known as maximum utility, can be defined as:

$$\boldsymbol{U\,(i,j)} = \mathbf{Max}\left(\sum_{i=1}^{n}\sum_{j=1}^{m} \boldsymbol{x_{ij}}\,\left(\boldsymbol{p_{ij}} - \boldsymbol{c_{ij}}\right)\right) \tag{1}$$

where x_{ij} represents the VM instances of type VM_j from provider S_i.

Once the agreement is established, a resource provisioning request, along with associated SLA, is submitted to SLA manager by the consumer. SLA manager dispatches this request to deployment manager to create and start requested resources. Deployment manager is responsible for interaction with other cloud providers for on-demand resource provisioning. Deployed VMs can be stopped or resumed at any time upon consumer request (within time frame of SLA). Once an agreement is terminated, all the created resources are released by the Cloud platform. Figure 3.2 depicts the architecture of VM placement across multi-cloud:

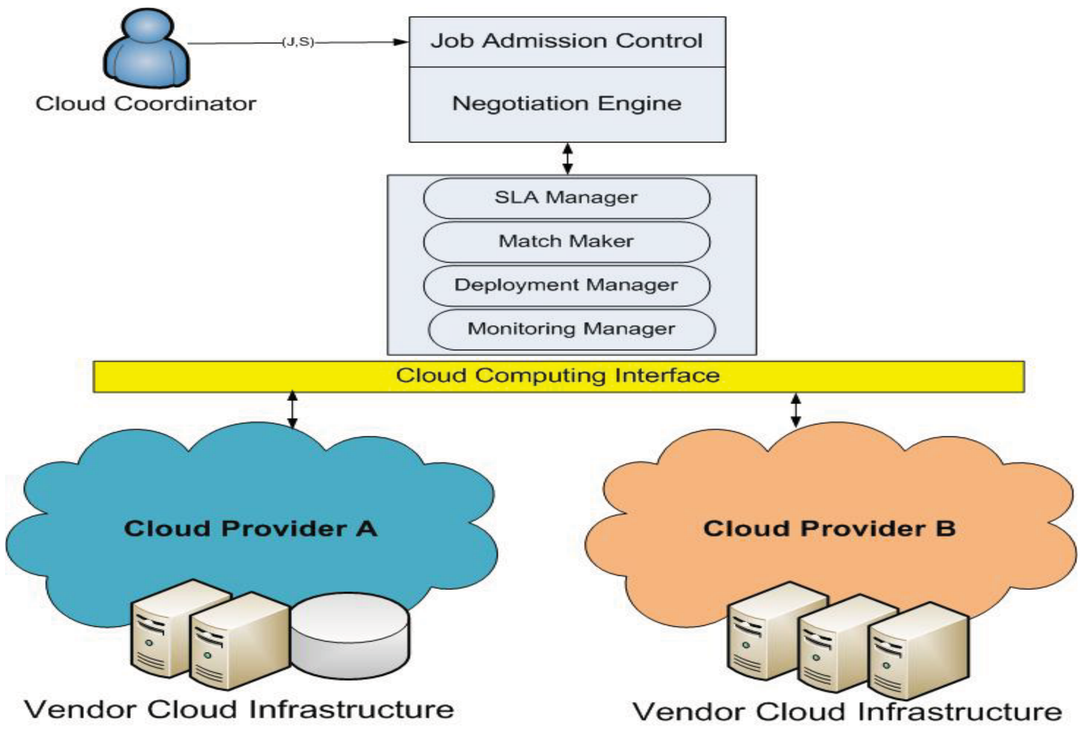


Fig. 3.2: Architecture of VM placement across Multi-cloud, adapted from [40]

3.3 IaaS Metrics for CP Ranking

IaaS metrics can be classified in four categories: Compute metrics, Memory Hierarchy, Network metrics and Storage metrics. In this study, few more metrics, apart from the metrics being proposed by [32] , are added for performance evaluation of the proposed broker based resource provisioning framework :

3.3.1 Compute

CPU Processing is a common performance benchmark for applications involving intensive workload. Processing performance is much dependant on clock speed, number of CPU cores and type of hardware.

Metrics Speed of VM, vCPU, outage length, availability, instance reboot time, execution time, server latency/ response time, Operating System, scaling latency.

3.3.2 Memory Hierarchy

Memory is the core element that determines the processing speed of an application. Many scientific and business applications are extremely dependant on memory system performance.

Under heavy memory contention, the memory latency may increase two or three times. Thus, the increasing performance gap between processors and memory systems imposes a memory bottleneck for such applications [116].

Metrics: Memory read/modify/write access time.

3.3.3 Network

Network performance determines how fast a VM can communicate with other clients over a network. Unlike most cloud storage and database applications, which are comparatively more tolerant to delay and jitter, media applications are far more demanding. The network delay with large jitter degrades user received media quality of latency sensitive multimedia applications [117], [118]. Hence, cloud providers with better network performance in terms of throughput vs. latency play a key role in the success of such business applications.

Metrics: availability, network latency, available bandwidth, response time, throughput.

3.3.4 Storage

Considering the nature of cloud computing environment where scientific applications and business workflows continuously involve storage I/O operations, it is important to determine how quickly storage devices allow applications to interact with data or files.

Metrics: input/output operations per second (IOPS), max restore time (backup), average access time, throughput.

Most of the cloud vendors guarantee performance of compute metrics at hypervisor level and these metrics are not reflected in service level agreements. On the other hand, network metrics are mostly reflected in SLA at data center level which are not specific to any particular user or consumer [119].

3.4 Metrics Evaluation

3.4.1 Server Latency/ Response Time (Compute)

Whenever a new VM request is made, a new instance of VM is provisioned, initialized and new IP is assigned to this instance. The time taken by this process, between a resource request and response, is referred as server latency/ response time which is formulated as:

$$\boldsymbol{t_i = t_{req} - t_{response}} \qquad (2)$$

so, average response time of *n* measurements is:

$$\mathbf{1 - \prod_{i=1}^{n}(t_i)\,/t_{max}} \qquad (3)$$

Where t_i is the time for IaaS instance to get available, t_{max} is the maximum acceptable time to complete this request and *n* is the total number of requests.

3.4.2 Outage Length (Compute)

Server Outage length (downtime), represented as a sum of the downtimes during a calendar period of normally one month is given as [120]:

$$\mathbf{\sum_{i=1}^{n} D_i} \qquad (4)$$

3.4.3 Memory Bandwidth (Compute)

A measure of peak memory bandwidth performance to identify how quickly operating system can get data into and out-of memory for processing.

3.4.4 Speed of VM (Compute)

Effective VM speed in MHz, can be calculated using SPECint, SPECfp benchmarks.

3.4.5 Scaling Latency (Compute)

Scalability refers to the ability to accommodate large loads or change in workload by provisioning of resources at runtime. This can be further categorized as scale-up (by making hardware stronger) or scale-down (by adding additional nodes). Throughput capacity gain by scalability, represented as α, can be measured as [121]:

$$\frac{\boldsymbol{T(\alpha s)}}{\boldsymbol{T(s)}} \qquad (5)$$

Where *s* is the number of VM instances and T(s) is the throughput function.

3.4.6 Availability (Compute)

Availability, the percentage of time a server is up, is given by [121]:

$$\frac{(\textit{\textbf{total up time}}) - (\textit{\textbf{total down time}})}{\textit{\textbf{total up time}}} \qquad (6)$$

3.4.7 Throughput (Network)

A throughput measurement (m_i=OPS) is the number of network I/O operations (file download / upload) over a specific time t_i. Based on multiple throughput measurement, throughput metric can be formulated for read or write requests. Average throughput per second can be measured as [121]:

$$T = \bar{m} = \frac{\sum_i^n m_i}{\sum_i^n t_i} \qquad (7)$$

3.4.8 Network Latency

Network latency is mostly measured by average ICMP ping time.

3.4.9 Network Availability

Network availability can be measured as [122]:

$$\frac{\textit{Total minutes of uptime of a Service network}}{\textit{total minutes in the Service month}} \qquad (8)$$

3.4.10 Input/ Output Operations per Second (Storage)

IOPS (input/output operations per second) is a common performance measurement for the maximum number of reads and writes operations and the mix of sequential and random access patterns to non-contiguous storage locations. Calculation of IOPS is based on the three factors: average latency, average seek time and disk rotational speed. For a single storage disk, IOPS can be calculated as [123]:

$$\frac{1}{\textbf{average seek time} + \textbf{average latency}} \qquad (9)$$

However, mostly multi-disk arrays are used in cloud environment so the same is formulated as:

$$(\textbf{Total IOPS} \times \%\,\textbf{READ}) + ((\textbf{Total IOPS} \times \%\,\textbf{WRITE}) \times \textbf{RAID IO Penalty}) \qquad (10)$$

3.5 QoS based Ranking Algorithm

In this study, past heuristics are used to predict the future quality conformance of a CP. For a particular CP S_i offering a set of quality attributes q_j, the feedbacks from the past performance using benchmarking and job execution experience over a time period T are collected. The proposed algorithm is extended from the one provided in [124]. Following two definitions are the fundamental concepts, used in this study.

3.5.1 Definition 1

The quality indicator value $\boldsymbol{F_{ij}^t}$ of a cloud provider S_i at time t in satisfying a quality requirement q_j is given as [124]:

$$\boldsymbol{F_{ij}^t} = \quad (\boldsymbol{\lambda_{ij}^t} - \boldsymbol{\tau_{ij}^t})/\,\boldsymbol{\tau_{ij}^t} \;\boldsymbol{\longrightarrow}\; [0, 1] \tag{11}$$

where λ_{ij}^t is the normalized value that of q_{ij} that was delivered by S_i in the past and τ_{ij}^t is what S_i promised to deliver at time t.

Cloud indicator value can be:

$$\begin{cases} > 0, & \textit{if performance is higher than promised value} \\ < 0, & \textit{if performance is lower than the promised value} \\ \boldsymbol{0} & \boldsymbol{\textit{same}} \end{cases} \tag{12}$$

3.5.2 Definition 2

A quality feedback report by broker monitoring component is a vector { S_i , t , R } where R is a pair-wise vector of conformance of { q_{ij} , F_{ij}^t }.

In case, in-house resources of home cloud provider are not enough to satisfy incoming job requests, a job request is submitted to broker along with a vector of QoS requirement { q_j , u_j, l_j } where the attribute q_j is the QoS requirement, u_j represents the priority of QoS attribute and l_j is the minimum (or in some cases maximum) QoS requirement needed by home CP. Different complex ranking techniques are proposed in the literature; in this study additive weighting scheme is used which results very close to other complex ranking techniques [124], [125].

The QoS rank of a CP, S_i , in fulfilling all quality criteria is defined by the following weighted sum [124]:

$$\frac{\left(\sum_{q_j \in Q} \boldsymbol{u_j} . \boldsymbol{\omega_{ij}} . \widehat{\boldsymbol{n.\lambda_{ij}^t}}\right)}{\sum_{q_j \in Q} \boldsymbol{u_j}} \tag{13}$$

Where

The normalized value of $\widehat{n\,\lambda_{ij}^t}$ represents S_i capacity for providing quality requirement q_j. It is calculated as the difference between evaluated QoS delivery of $\widehat{\lambda_{ij}^t}$ and required minimum (or maximum) quality criteria l_i as required by the home cloud provider:

$$\widehat{\boldsymbol{n\,\lambda_{ij}^t}} = (\,\widehat{\boldsymbol{\lambda_{ij}^t}} - l_i)/\,l_i \tag{14}$$

ω_{ij} represents the degree of match and is the similarity proportional weight between q_{ij} provided by S_i and the required quality concept q_j. It can be calculated as:

$$\boldsymbol{\omega_{ij}} = \begin{cases} \mathbf{1.0} & \boldsymbol{q_{ij} \equiv q_j} \\ \mathbf{0.5} & \boldsymbol{if\ q_{ij}\ is\ more\ general\ than\ q_j} \\ \mathbf{0} & \boldsymbol{otherwise} \end{cases} \tag{15}$$

Although cloud providers with higher quality of services should given more priority but this is not a case here. To reduce cost and get accurate QoS promised claims, cloud servers who provide exact match to the required incoming jobs are ranked relatively higher.

For each incoming job request, all cloud providers are parsed for the evaluated QoS and a QoS matching table, consisting of a V_{qos_j} vector of $\{S_{ij}\ ,\ \omega_{ij}\ ,\ \hat{\lambda}_{ij}\ \}$, is constructed as list of CPs meeting a particular quality requirement according to their matching weights. Resource selection and ranking algorithm for a list of ‘L’ cloud providers is presented as under:

Algorithm: QoSbasedRanking (CPList CP, QualityRequirements Q)

```
► CPList is the list of cloud providers, Q is the list of requirements that needs to be satisfied in order to successfully execute a particular job
1:  enqueue the list of QoS required values from Q: $CP_q=\{ [q_1 , u_1, l_1],\ldots, [q_s , u_s, l_s] \}$
    ► Initially the quality score for all cloud providers will be zero
2:  Score [$S_{ij}$] = 0.0
    ► Now parse each cloud provider $S_i$ for the required quality attribute $q_j$
3:  for each quality requirement $q_j$ in $CP_q$ do
4:    for each Cloud provider $S_{ij}$ in CP do
    ► First confirm whether there is any available data for the required quality attribute $q_j$ which is provided by cloud provider $S_i$
5:      Parse the list $V_{qos_j}$ of $q_j$ for $S_{ij}$
6:      if $S_{ij}$ is matched then
    ► Get the relative quality score of cloud provider $S_i$ for meeting the quality attribute $q_j$
7:        $w_{ij}$ =weight ($q_j$ , $q_{ij}$)
    ► Now calculate the evaluated capacity for QoS delivery of quality attribute $q_j$ as calculated in eq. 14
8:        $QoSScore_{Partial}= w_{ij}. (\hat{\lambda}_{ij} - l_{jj})/ l_j$
    ► Aggregate the quality of service score for the current requirement as well as the previously calculated requirement from the list Q
9:        Score [$S_{ij}$] = Score [$S_{ij}$] + $(u_j / \sum u_j)$. $QoSScore_{Partial}$
10:     else
    ► If a particular cloud provider does not satisfy the required quality criteria, remove it from the list
11:       Delete $S_{ij}$ from CP
12:     end if
13:   end for
14: end for
    ► Finally, return the prioritized list of providers
15: Return the sorted list of Score [$S_{ij}$]
```

3.5.3 Algorithm Complexity Analysis

The above algorithm is composed of two parts. For the worst case scenario of selection process (lines 3-14) with n quality concepts, the algorithm will have to process a list of l total providers for p possible providers with quality concepts meeting user requirements. This results in worst-case complexity of $O(n.l.p)$. Considering that provider list is constant, the final worst-case complexity is $O(n)$. In second part of algorithm (line 15), merge sort is used for prioritizing the list of providers which has a complexity of $O(n\,log\,n)$, so the total worst case complexity of proposed algorithm by using the additive property of big O is:

$$\mathbf{O}(\boldsymbol{n}) + \mathbf{O}(\mathbf{n\,log\,n}) = \mathbf{O}(\mathbf{n\,log\,n}) \tag{16}$$

3.6 Proposed Framework

The proposed cross-layer design consists of three modules; each can directly communicate with other, whenever required. Job Management Module is responsible for job admission, negotiation and provisioning of resources based on the feedback from ranking module. When the negotiation is successfully completed, Service Level Agreement is established between consumer and producer which specifies the Service Level Objectives based on QoS. SLA management module is responsible for monitoring the violations of SLA. In case of violation, alarm notifier will trigger an alarm and the event will be stored in violation data repository. Based on this violation data repository and CP catalog, which stores the promised QoS by each provider, QoS rank of each CP is calculated by the ranking module. The proposed framework is given in figure 3.3.

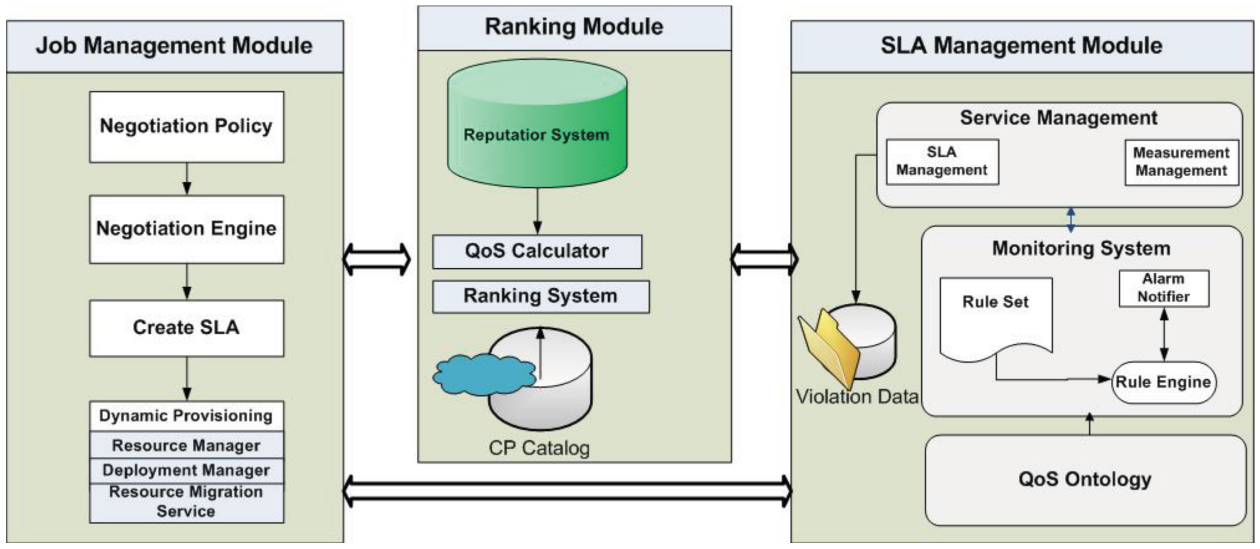


Fig. 3.3: Proposed Framework for QoS based Ranking in federated Cloud

3.6.1 Job Management Module

This module is responsible for managing the requests from the consumer of cloud federation along with their SLAs and negotiation contracts to interact with the match maker module in order to invoke resource provisioning mechanism between the consumer and provider which suits best to the requirements specified in form of SLA.

Four further subcomponents in this module are:

3.6.1.1 Negotiation Engine

This is an important component of the Job management module. With the negotiation terms, the strategies that need to be adopted in form of penalties in case of SLA violations from the cloud provider are mostly referred [41]. Moreover, this component is solely negotiating the QoS parameters such as availability, bandwidth, storage and so on, that are to be required by the consumer from the provider after the one time negotiation settlement with the provider along with their priorities. This also covers the cost factors for the resources that will be utilized once the settlements have been made. The contents of negotiation terms are exchanged in XML schema of SLA.

3.6.1.2 Resource Manager

This module ensures the provisioning of resources by the provider to its consumer once the request is initiated by the job management module taking into account the negotiation terms and settlements. The degree of provisioning SLA assurance however is evaluated by the SLA manager.

3.6.1.3 Deployment Manager

It is responsible for creation of the virtual machine pool according to actual state information of the user job. Based on the QoS requirements, the module aims to complete job within budget and deadline constraints.

3.6.1.4 Resource Migration Service

In case of early termination or service interruption due to low QoS, provisioning of resources is re-evaluated to meet QoS and deadline constraints. Necessary measures would be taken to migrate the VM resources to the subsequent best matched cloud provider.

3.6.2 Ranking Module

Since in-house cloud platform was not available for this research study, the proposed framework and policies were evaluated on real test-bed experiments. The experiments were conducted on different VM instances leased from the three cloud providers. Multiple data-centers of the three cloud providers were used to calculate average performance estimation. Our price-performance analysis model differs from the one, proposed in [126], as instead of calculating a simple product of average scores, we take into considerations several other factors, associated with their relative weights. For QoS calculation, the following model is proposed:

Consider a system model which consists of *n* cloud providers $\{S_1, S_2, \ldots, S_n\}$. Each cloud provider offers *m* type of VM instances $\{VM_1, VM_2, \ldots, VM_m\}$, each of which offers different CPU cores, disk storage and memory size. To compare performance of VMs, it is needed to assign weights to each top level evaluation metric (W_j) and associated benchmarking tools (w_j) taking into account their relative importance of each benchmarking experiment *j*. Each VM instance is compared term by term correspondingly for *k* set of benchmarking tools for evaluation of benchmarking metrics. The performance score is computed for each benchmarking experiment j. However, since a benchmarking tool may consist of l number of benchmarking test cases, the performances score ($\mathrm{P_j}$) for each of the experiment is calculated as under:

$$\mathbf{P_j} = \sum_{b=1}^{l} \boldsymbol{GeometericMean}(\boldsymbol{test\_case\_score}) \tag{17}$$

The price factor should also be incorporated in eq. 17 as a VM with high utilization cost will usually provider better performance results so performance comparison should be made on equal ground. Hence, ρ, the price-performance ratio is introduced in this study, so that VMs with different configuration and pricing schemes may be accurately compared:

$$\boldsymbol{\rho}_{\mathbf{j}} = {}^{\mathbf{P_j}}/_{\mathbf{C}} \tag{18}$$

where C represents the VM cost. Finally, overall score for a particular benchmarking experiment is computed as under:

$$(\boldsymbol{\alpha_j} \times \boldsymbol{\rho}_{\mathbf{j}}) + (\mathbf{100} - \boldsymbol{\alpha_j}) \times \mathbf{P_j} \tag{19}$$

where α_j modifier is the sensitive factor of price-performance analysis and may vary for different experiments. For each benchmark, the utility values of VM instance v_{ij} are computed that depicts the performance of a particular VM with respect to a particular benchmarking result. The utility value is normalized so that $\sum_{j=1}^{l} v_{ij}=1$ and $v_{ij} \in [0,1]$. The overall aggregation of utility values V_{ij} is then calculated as a sum of all v_{ij} for every evaluation metric. Finally, the utility function $f(V_{ij})$ associated with VM instance i is constructed as follows:

$$\boldsymbol{f(V_{ij}) = \sum_{j=1}^{l} v_{ij} * W_j} \tag{20}$$

VM instances with higher value of utility functions are ranked higher as compared to instances with lower score.

3.6.3 SLA Management Module

Service level agreement is defined as: *"a contract between a cloud provider and a cloud consumer, with details related to the provisioned resources along with the price charged, and the associated quality attributes while ensuring the use of provisioned resources provided by the CP"* [50]. SLAs are composed of service level objectives (SLOs) that define quality attributes such as availability as well as associated penalty in case of SLA violations. SLAs are used as a driving force for negotiation between different cloud providers. In this framework, SLAs of different cloud providers are created by benchmarking their QoS aspects and based on the user requirements, the candidate best fit cloud provider is selected. SLAng [127], a XML based language for specification of QoS aspects of service level agreements, is extended to provide a negotiation mechanism between different cloud providers.

3.6.3.1 SLA Service Management

This module checks individual resource providers to confirm the service requirements based on the schema in figure 3.4. The proposed schema is a group of SLOs that is derived for network, storage, memory and computing quality attributes. The first set of service level objectives defined in schema is related to the computing capacity of the resources followed by storage, network and memory. Attributes that are associated with the computing requirements are CPU cores, VM speed, availability, OS type, cost and boot time of VM instances. Quality metrics like Input/ Output operations per second and latency are also taken in account while representing the storage set of terms. Upload/ download speed and ping are other important attributes while representing the network set of requirements in SLA. Provider's location attribute is given in the schema in order to represent the geographical boundary of a particular datacenter so that resource outsourcing may be achieved with minimal latency using the most nearest data centers.

Figure 3.4 outlines the proposed schema to represent different parameter settings for QoS metrics. For instance, the xs *element namespace identifier* for QoS category $CPU_{Performance}$ is used to add a vocabulary for the $\mathrm{CPU}_{\mathrm{Performance}}$ related six QoS metrics which are then later on declared using float, byte and string data types. Each category of metrics can be either must required once in a new job submission ($\max_{\mathrm{Occurs}}$ = 1) or may not be part of a job requirement ($\min_{\mathrm{Occurs}}$ = 0). For instance, a network bound steaming job may only require network related metrics and any other QoS metrics may be assumed with default settings.

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="QoSModel">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Compute">
```

```
<xs:complexType>
  <xs:sequence>
    <xs:element name="CPUPerformance" minOccurs="0" maxOccurs="1">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:string">
            <xs:attribute type="xs:float" name="Availability"/>
            <xs:attribute type="xs:byte" name="CPUCores"/>
            <xs:attribute type="xs:byte" name="VMSpeed"/>
            <xs:attribute type="xs:float" name="Cost"/>
            <xs:attribute type="xs:string" name="OS"/>
            <xs:attribute type="xs:byte" name="BootTime"/>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
    <xs:element name="Storage" minOccurs="0" maxOccurs="1">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:string">
            <xs:attribute type="xs:short" name="DiskSpace"/>
            <xs:attribute type="xs:byte" name="IOPS"/>
            <xs:attribute type="xs:byte" name="Latency"/>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
    <xs:element name="Network" minOccurs="0" maxOccurs="1">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:string">
            <xs:attribute type="xs:byte" name="Upload"/>
            <xs:attribute type="xs:byte" name="Download"/>
            <xs:attribute type="xs:byte" name="Ping"/>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
    <xs:element name="Memory" minOccurs="0" maxOccurs="1">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:string">
            <xs:attribute type="xs:byte" name="RWM"/>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
```

```
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Mutual">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Violation_Clauses">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute type="xs:string" name="Compensation"/>
                      <xs:attribute type="xs:string" name="Exclusion_Clauses"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Provider">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Id">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute type="xs:string" name="Slo_ID"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
              <xs:element type="xs:string" name="Name"/>
              <xs:element type="xs:string" name="DataCenter"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Fig. 3.4: SLA Schema for QoS Metrics

The flow of events for SLA management process, based on the negotiation process as presented in section 3.2, is shown in figure 3.5. A list of SLAs for each cloud provider is parsed against any incoming job request and SLAs with the best matched configuration are returned to cloud consumer. Upon acceptance by consumer, an agreement is established and resource is provisioned.

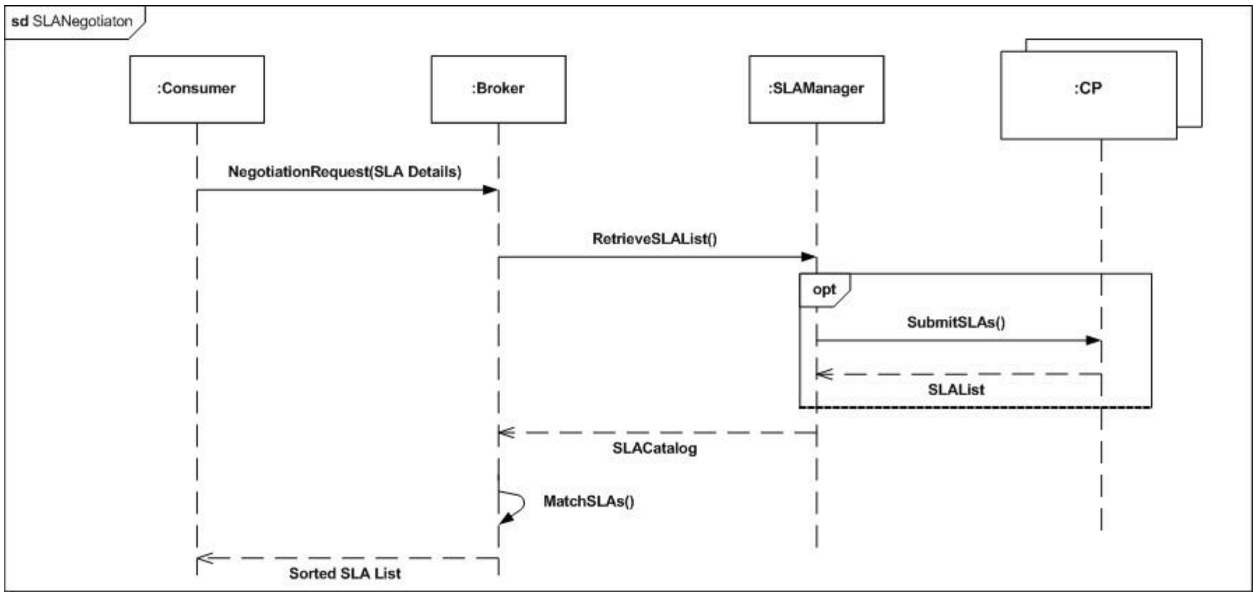


Fig. 3.5: SLA Negotiation Process for Job Submission

3.6.3.2 Resource Monitoring Module

This module continuously monitors secured resources against service abruption and violation of SLA. Taking into consideration the existing studies on SLA violation [128], [129], the flow of events is as under:

4 The module is invoked when the agreement is signed between home and foreign cloud providers on agreed upon QoS terms.
5 A small monitoring component is installed at the foreign cloud provider end to collect necessary statistical usage and evaluation data.
6 Monitoring component periodically updates the broker about usage data.
7 If case of any SLA violation, an alarm is triggered which is handled by alarm manager. The details of violation is recorded and compared against allowed threshold given in rule set. In

case of severe violation, resource switchover takes place and the VM is migrated to the candidate cloud provider.

Figure 3.6 presents the job termination process where a provisioned resource is requested to be terminated by cloud consumer. Although cloud consumers have the privilege to terminate any VM at any instance of time, however, the process is necessary to collect any violation of negotiation terms and provides a mechanism for developing a repository of violation data and thus a better ranking mechanism, required for future job satisfaction. Upon the consumer request, broker requests the monitoring component to exchange all execution and violation data. If foreign CP performance violates SLA constraints, broker updates reputation repository with a negative rating for the CP otherwise it updates with positive rating. Broker also notifies the foreign cloud provider to acknowledge the job termination which is then passed on to the consumer.

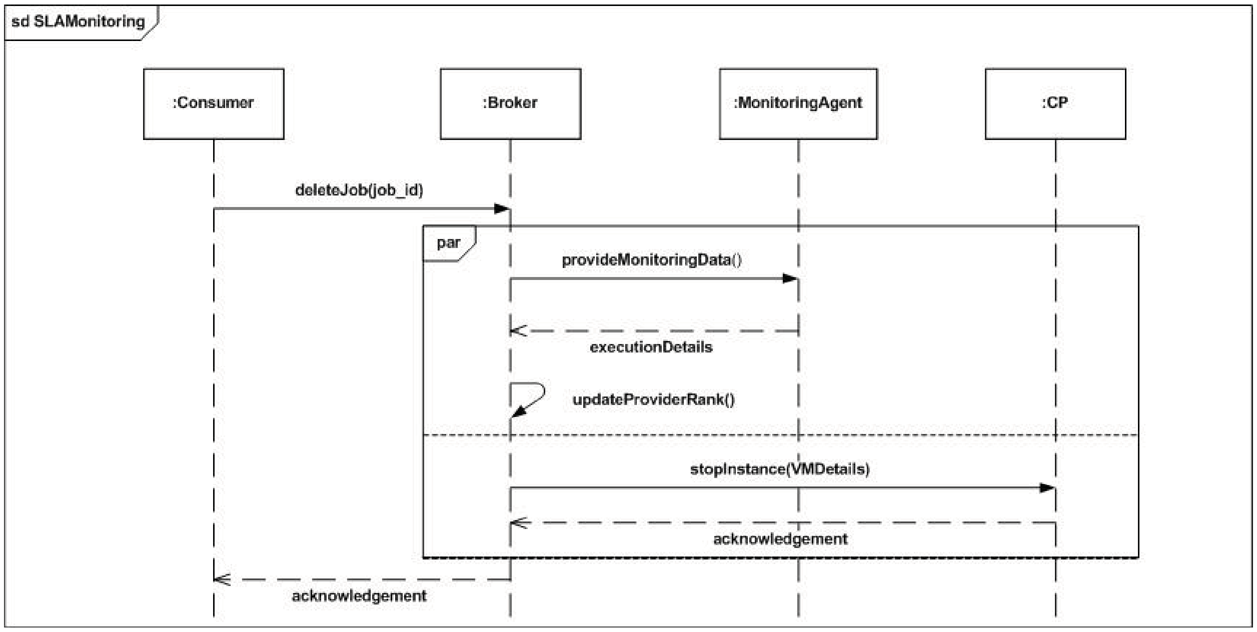


Fig. 3.6: SLA Monitoring and Termination Process

The ruleset is a repository of SLA rules along with allowed threshold. A part of rule-set is given in figure 3.7.

```
<?xml version="1.0"?>
<ruleset version="1.0+">
```

```
<rule ref="Compute.Availability">
<properties>
        <property name="threshold" value="-10%" />
        <property name="action" value="preempt" />
        <property name="message" value="Serious violation of availability SLO. Resource will
be preempted." />
</properties>
</rule>
<rule ref="Network.Download">
<properties>
        <property name="threshold" value="-15%" />
        <property name="action" value="warning" />
        <property name="message" value="Warning. Violation of network bandwidth SLO" />
</properties>
</rule>
</ruleset>
```

Fig. 3.7: SLA Violation Rule-set

Some rules are critical and violation of such rules may be resulted in resource preemption. However, some rules are non-critical in nature and foreign cloud provider may be warned for a pre-defined number of times before any further action is taken place.

Chapter 4: Implementation

In this chapter, the implementation detail of the designed framework is discussed. In the subsequent sections, the benchmarking methodology along with necessary tools is discussed, followed by experimental design and the proposed evaluation strategies.

4.1 Benchmarking Methodology

Since simulation results and data extraction from the studies conducted earlier do not present the actual figures to benchmark a particular cloud vendor, so different cloud vendors are evaluated on the real testbed experiments. The experiments were conducted on different 64-bit Linux VM instances, given in table 4.1, leased from the three cloud providers. Necessary statistical data for benchmarking was collected after a series of experiments conducted in a time span of two month. Multiple datacenters of the three cloud providers were used to calculate average performance estimation.

Table 4.1: List of VM instances used for Experimentation

Ref. ID	**Cloud Provider**	**VM instance**	**vCPU**	**Memory (GB)**	**Cost (US$)**	**Storage (GB)**
A1GB	Amazon	t2.micro	1	1	$0.013	EBS
A4GB	Amazon	t2.medium	2	4	0.052	EBS
A15GB	Amazon	m3.xlarge	4	15	0.266	2 x 40 SSD
RS1GB	Rackspace	1 GB Standard	1	1	0.06	40 RAID
RS 4GB	Rackspace	4GB Standard	2	4	$0.24	160 RAID
R7.5GB	Rackspace	7.5GB Compute v1	4	7.5	$0.23	50 ESB
G1GB	Google	g1-small	1	1.7	0.01	10240 SSD
G3.8GB	Google	n1-standard-1	1	3.75	$0.050	10240 SSD
G13GB	Google	n1-highmem-2	2	13	$0.126	10240 SSD

Tables 4.2-4.5 show the list of benchmarking tools used in the experiments:

Table 4.2: Network Benchmarking Tools

Tool	**Supported Metrics**
iperf [130]	Network bandwidth, throughput
ping	Network latency
speedtest-cli [131]	Network bandwidth

Table 4.3: CPU Benchmarking Tools

Tool	**Supported Metrics**
Dacapo [90]	CPU execution time
SPECjvm2008 [88]	CPU execution time
Phoronix Test Suites [89]	CPU execution time

Table 4.4: Storage Benchmarking Tools

Tool	**Supported Metrics**
iozone [92]	Throughput, Disk I/O
bonnie++ [93]	Storage latency, Disk I/O

Table 4.5: Memory Benchmarking Tools

Tool	**Supported Metrics**
Cachebench [94]	Memory read/modify/write access time
Ubench [132]	Memory throughput

4.2 Implementation Technologies

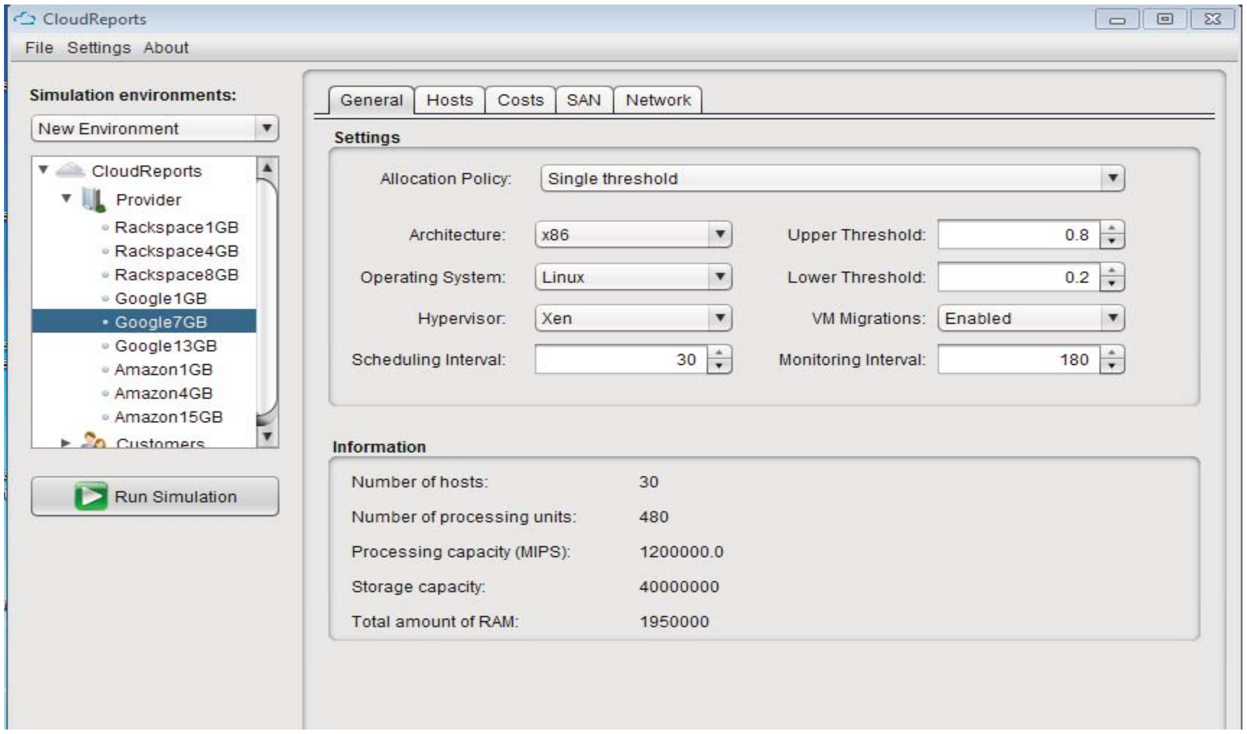
A two-fold cross validation experimentation strategy is used to evaluate the proposed framework. Real test-bed experiments are expensive, both terms of time and cost and hence first simulation environment was used to ensure the effectiveness of designed broker based resource provision framework. Based on simulation results, the evaluation policies were reassessed and then a series of experimentations, as presented in sections 5.4.1 and 5.4.1, were performed on real cloud providers. Table 4.1 presents the list of tools used during the experimentation process.

Table 4.6: Tools used in Simulation of the Proposed Framework

Tools	**Usage**
CloudSim (version 3.0)	CloudSim is a Java based Infrastructure as a Service toolkit to simulate data centers, virtual machines, hosts and job management in cloud computing environment. It can be used to simulate resource scheduling, cloud pricing and policy management within cloud or inter-cloud environment [133]. To simulate cloud computing environment, virtual machine provisioning mechanism as well as brokering policies were extended in order to evaluate the designed framework.
CloudReports (version 1.0)	CloudReports is a GUI wrapper built on the top of CloudSim to provide better graphical and visual representation of simulation environment. The tool provides additional analysis and reporting charts and metrics to synthesize simulation results [134]. Some components of CloudReports were extended to match the scenario of inter-cloud environment.
Hiberante ORM	To ensure better reliability of simulation results, it is necessary to repeat simulation experiments more than once. It, therefore, becomes necessary to save and retrieve the state of simulation objects (virtual machines, hosts, job preferences and policy mechanism) when a series of experiments take place. Hibernate ORM was used to map object states in database to ensure data persistence and reliability of experiments [135] .

CloudSim simulation is based on the configuration of datacenters, hosts and virtual machines. Datacenters are responsible for creating and managing hosts while VMs are created within hosts.

Consumer issues requests for number of required VMs and associated tasks in form of Cloudlets. Figure 4.1 depicts the multi-cloud simulation environment developed in CloudSim toolkit.



Settings

Allocation Policy: Single threshold

Architecture: x86

Operating System: Linux

Hypervisor: Xen

Scheduling Interval: 30

Upper Threshold: 0.8

Lower Threshold: 0.2

VM Migrations: Enabled

Monitoring Interval: 180

Information

```
Number of hosts:                 30
Number of processing units:      480
Processing capacity (MIPS):      1200000.0
Storage capacity:                40000000
Total amount of RAM:             1950000
```

Fig. 4.1: Cloud Simulation Environment

The default policy for DataCenterBroker is based on RoundRobin scheduling where resources are randomly allocated and workload is divided in different datacenters. To implement the designed framework, the DataCenterBroker class was extended to manage resource provisioning and scheduling policies. A broker extension was developed to divide workload according to user QoS requirements for optimal resource allocation. The cost model of CloudSim was also extended to calculate VM utilization cost per hour which is not implemented in CloudSim simulator. VM allocation policy was also modified to provide an inter-cloud outsourcing and migration mechanism for different datacenters. Workload management as well as resource leasing and migration across multiple datacenters provide an effective and reliable resource management strategy where every stakeholder is at win-win situation. The architecture of extended CloudSim toolkit is presented in figure 4.2.

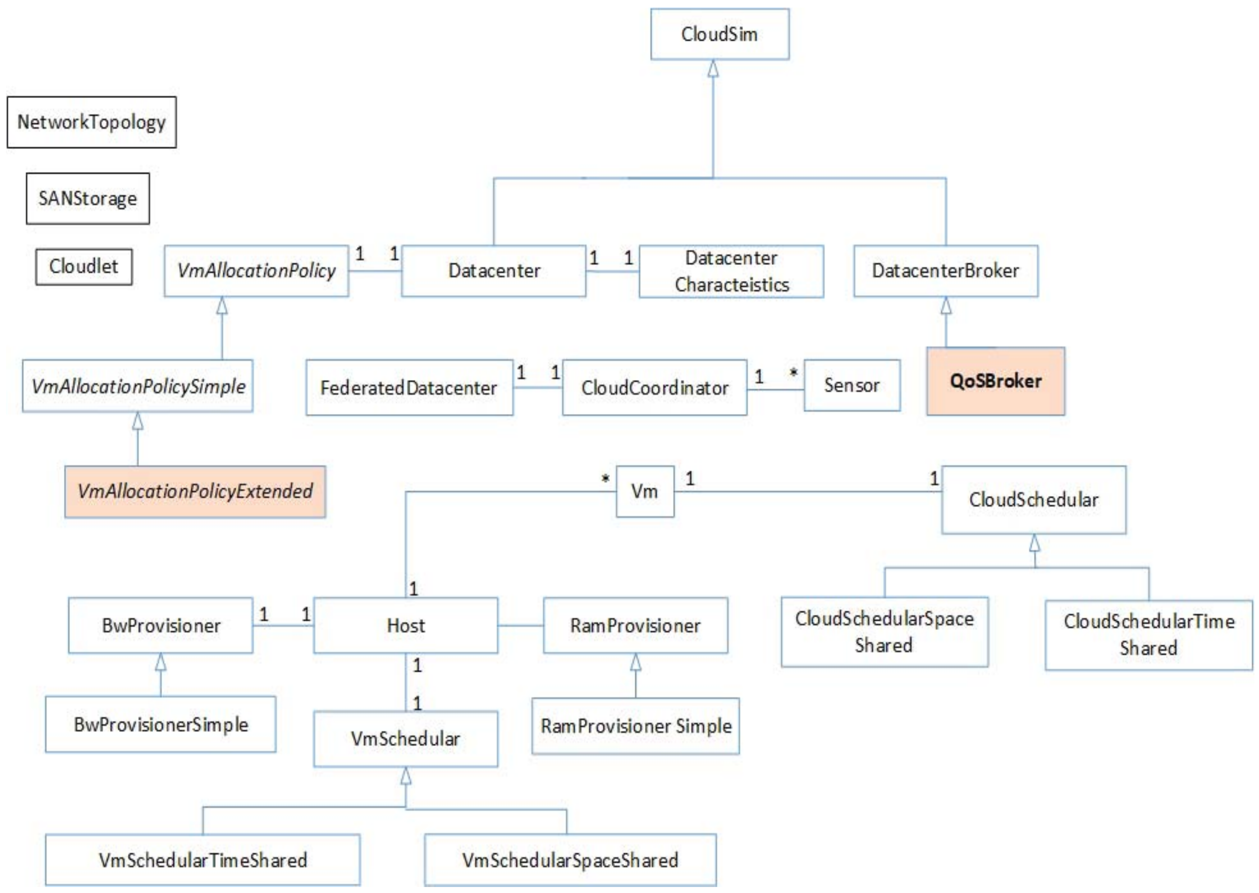
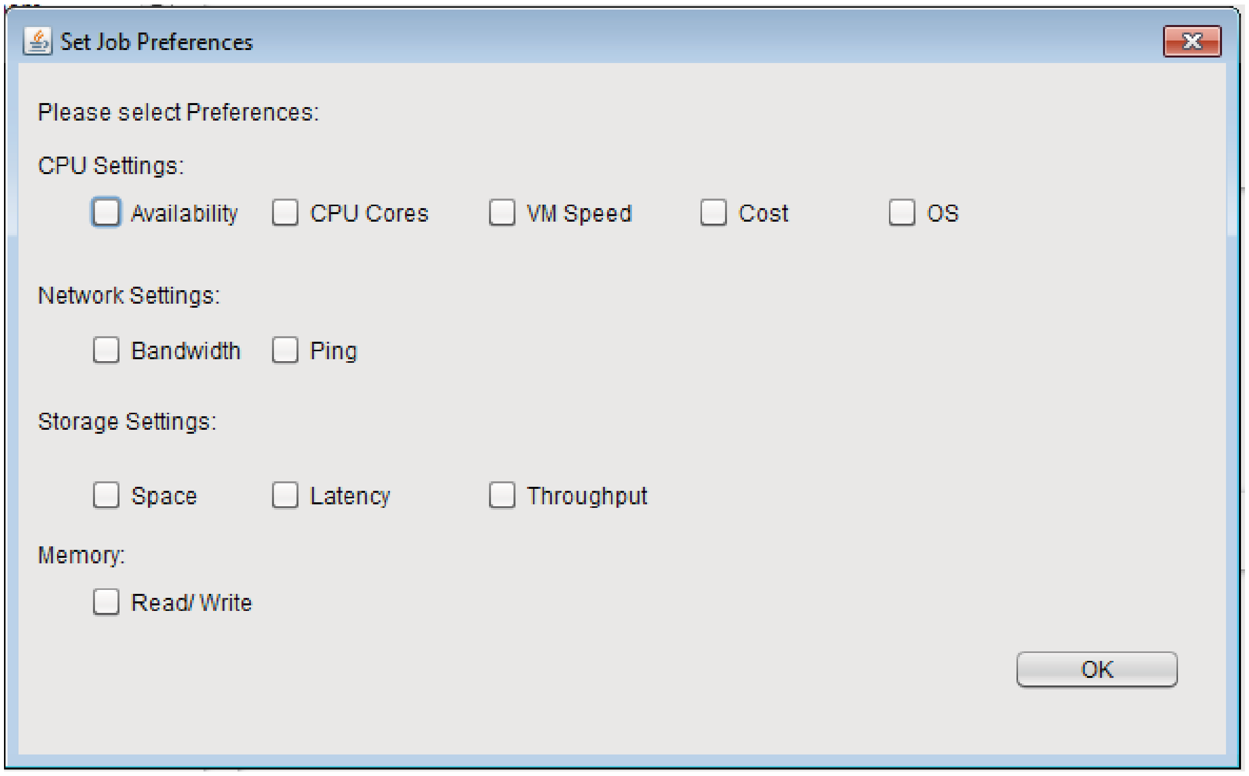


Fig. 4.2: Architecture of Extended CloudSim

The proposed policies were simulated by executing simulation several times using CloudReports. A set of experiments were performed and following variables were measured for synthesizing and calculating experimental results:

1. Cost
2. Processing load
3. Time

Upon a job submission, user is requested to provide job QoS preferences to classify incoming tasks as CPU/ network/ storage/ memory intensive as shown in figure 4.3. If local cloud resources are insufficient to meet incoming job demands, resources from other cloud providers are leased based on user preferences. The negotiation and brokering strategies are discussed in chapter 3.



Set Job Preferences

Please select Preferences:

CPU Settings:

☐ Availability ☐ CPU Cores ☐ VM Speed ☐ Cost ☐ OS

Network Settings:

☐ Bandwidth ☐ Ping

Storage Settings:

☐ Space ☐ Latency ☐ Throughput

Memory:

☐ Read/ Write

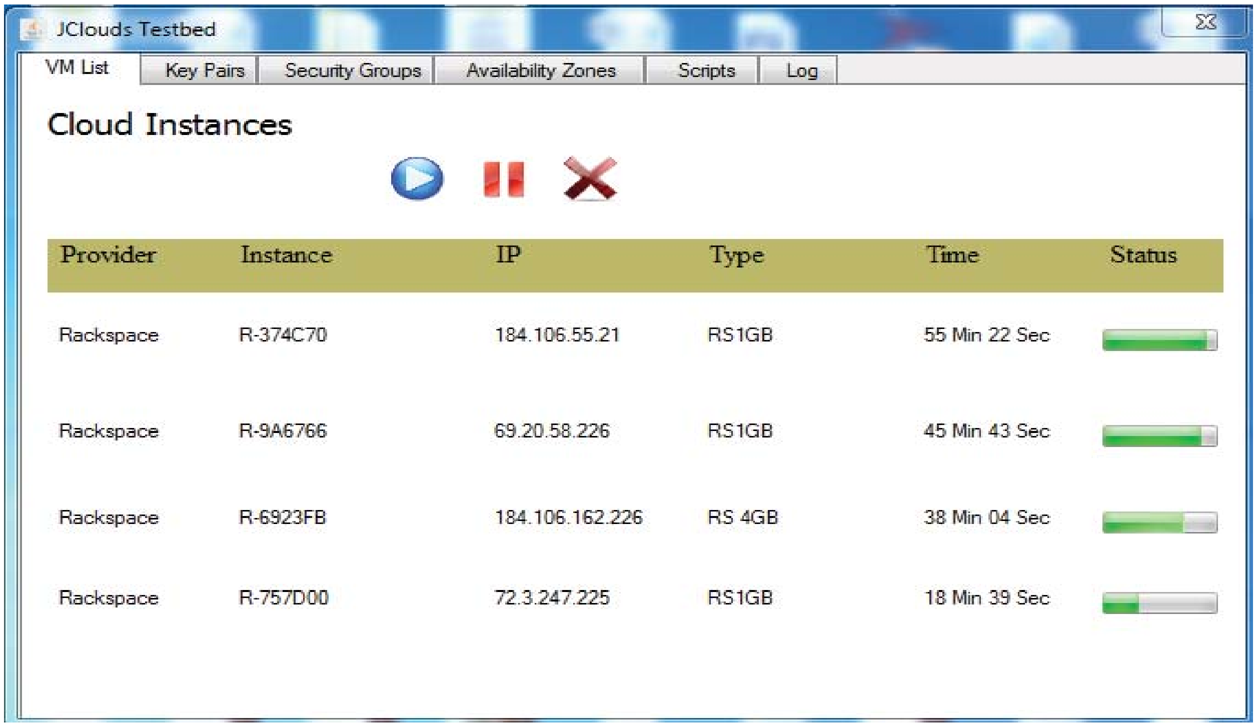
Fig. 4.3: Job Preferences for SLA Management

Real testbed experiments were performed using Apache jclouds while Cloudharmony API was invoked to collect necessary usage and performance statistics of different cloud vendors. Table 4.2 presents the list of tools, used during the testbed experiments while the broker based testbed prototype is shown in figure 4.4.

Table 4.7: Tools used in the Experimentation of the Designed Framework

Tools	**Usage**
Apache jclouds (version 1.8)	Apache jclouds is a multi-cloud toolkit, developed to provide a common interface for cloud specific features such as managing VM instances, load balancing and data storage across multi-clouds [136]. A GUI based interface was developed to perform necessary experiments in a user friendly way.

Cloudharmony API	Cloudharmony is aimed at providing reliable and updated performance analysis about availability, network outages, and performance analysis for various cloud providers [137].
BLAST+	BLAST+ (Basic Local Alignment Search Tool) is a command line utility for comparing biological sequence information such as the different proteins or the nucleotides of DNA sequences. The tool enables users to compare a search sequence query against a database of sequences to identify sequence that resemble with source query sequences under a specific threshold [138].
GNU Parallel	GNU parallel is a command line utility for Unix like operating systems. It allows users to execute shell scripts in parallel [139].



Provider	Instance	IP	Type	Time	Status
Rackspace	R-374C70	184.106.55.21	RS1GB	55 Min 22 Sec	
Rackspace	R-9A6766	69.20.58.226	RS1GB	45 Min 43 Sec	
Rackspace	R-6923FB	184.106.162.226	RS 4GB	38 Min 04 Sec	
Rackspace	R-757D00	72.3.247.225	RS1GB	18 Min 39 Sec	

Fig. 4.4: JClouds Experimentation Setup

4.3 QoS Evaluation Policies

To evaluate the designed framework, following policies were derived in order to evaluate the designed model:

4.3.1 Local Resource Allocation

As the name indicates, local resource allocation policy is used by autonomous cloud providers for placement of VM instances from the job pool. This policy is quite simple to use and no federation is required in this case. It ensures the notion of unlimited VM instances availability as long as VM pool is not fully occupied. However, further VM requests crossing the availability of the resources in the job pool are not entertained under the local resource allocation. This may result in less profit and higher denial of service for new incoming jobs.

The providers' profit acquired at a certain period of time Δt, using these policies is determined as:

$$\boldsymbol{Profit_p(\Delta t) = Revenue_p(\Delta t) - Cost\_of\_Operating\_VMs_p(\Delta t)} \qquad (21)$$

and $$\boldsymbol{Revenue_p(\Delta t) = VM_p(\Delta t).VM\_price\_per\_hour_p.\Delta t} \qquad (22)$$

where,

Revenue$_{p,}$ is obtained by multiplying the number of VMs allocated to tasks requested by the client and VM price per hour for a certain period of times. Cost_of_Operating_VM is the cost required to keep the nodes up in the datacenter during a certain period of time. The provider gains profit only if Revenue (Δt) $>$ Cost_of_Operating_VMs(Δt) in datacenter.

4.3.2 Broker based Outsource oriented Resource Allocation

In order to maintain a better reputation and to entertain excessive on-demand requests from the clients, this policy can be used by the cloud service providers. This policy accommodates the broker based resource provisioning model as proposed in the framework for cloud federation. Hence, this policy solves the issue of denial of service in earlier policies. If in-house resources of home cloud provider are not enough to satisfy incoming job, resources from foreign cloud providers are leased based on the QoS requirements of home cloud provider. The policy also ensures that the cost charged by the provider of the spare resource should be less than the one charged by the provider to its client for revenue perspective. Also the quality metrics required by the cloud provider must be satisfied accordingly for resources being leased. The revenue in this case, can be calculated from the number of outsourced VM by the provider as Revenue_o. Thus, the profit will be computed as:

$$\mathbf{Profit_o(\Delta t) = Revenue_o(\Delta t) - Cost\_of\_Outsourced\_VMs_o(\Delta t)} \qquad (23)$$

The provider will be in benefit if the outsourced VM instances would be less in price than the one offered by the provider in his own datacenter. Although an exception would be the case when provider may sometimes face a loss in revenue just to maintain the reputation of the services provided to the clients, as the price of outsourced VM may vary at times or a provider is accommodating the purchase from the expensive cloud provider in federation due to lack of spare resources from the other cloud providers to avoid any SLA violation for regular users.

4.4 Workload Generation

Cloud computing environment is inherently different from cluster and grid computing, being operated by R& D institutes and government agencies and workload traces are made available to researchers and general audience. Clouds are mostly operated by private venders and no public workload dataset is available for IaaS cloud computing platform [140]. For a generic workload model, the following three workload patters are mostly used by researchers [141]:

1. Uniform: Under this workload pattern, incoming job arrivals are mostly constant and a steady pattern is observed
2. Discrete/ Continues Distribution: In this scenario, job request rate is either increased or decreased at a gradual pace and discrete/ continuous deviation pattern may be observed
3. Workload Spikes: Such a pattern is also known as flash crowd or resource burst. In such an interval, large spikes of resource demands is anticipated that may affect overall provisioning of resource provisioning policies and VM placement mechanism. This may be a result of workload variation during day vs. nights, weekdays vs. weekends or any special occasions.

Poison distribution was used for discrete/ continuous workload patterns. However, workload pattern follow a log-normal like distribution instead of the poison process when request arrivals are not independent [76]. Motivated by this, we adapted the workload model proposed in [8] and [142] to generate realistic load streams during spikes. This model considers four time periods: t_0, t_1, t_2 and t_3. Incoming job requests arrive at a uniform rate until a resource burst is experienced at t_0. During the first phase, as shown in figure 4.5, a resource spike is observed and resource demand is anticipated as more and more resource requests arrive during a relatively shorter period of time, till it reaches to a maximum level at t_1. The phase, known as Ramp-up, can be presented as:

$$\text{Ramp-up} = t_1 - t_0 \tag{24}$$

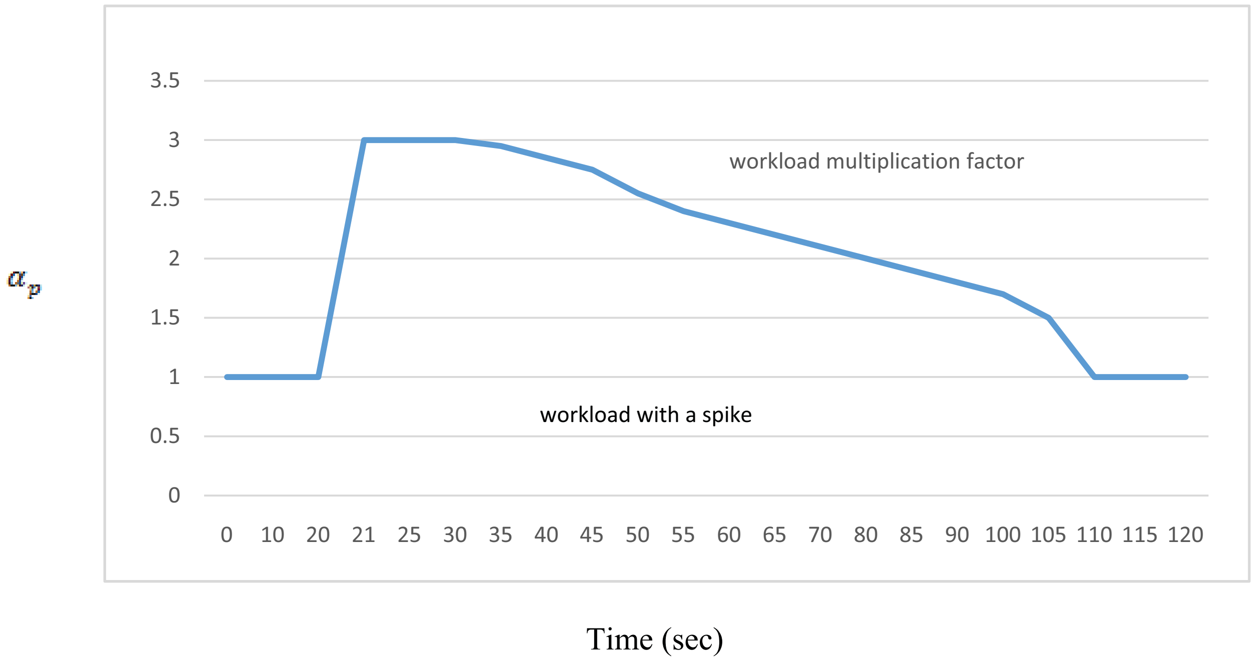


Fig. 4.5: Large Spike of Resource Demand

After Ramp-up, resource demand remains steady for a period till t_2. This phase is known as steady phase:

$$\text{Steady-phase} = t_2 - t_1 \tag{25}$$

In steady phase, job arrival rate is proportionally equal to job completion rate. Finally, the demand for resources gradually decreases and a normal workload pattern is again observed at t_3. The phase is known as Ramp-down:

$$\text{Ramp-down} = t_3 - t_2 \tag{26}$$

As shown in figure 4.6, time values and a scale-up factor U generate overall resource spike. The normal workload pattern is multiplied by a shape parameterization factorα_p is set to 1.0 during normal workload before t_0. During spikes (between t_0 and t_1), α_p gradually increase to U and remains unchanged until t_2. During ramp-down phase, $\alpha_p$gradually decreases until it reaches 1.0 at t_3.

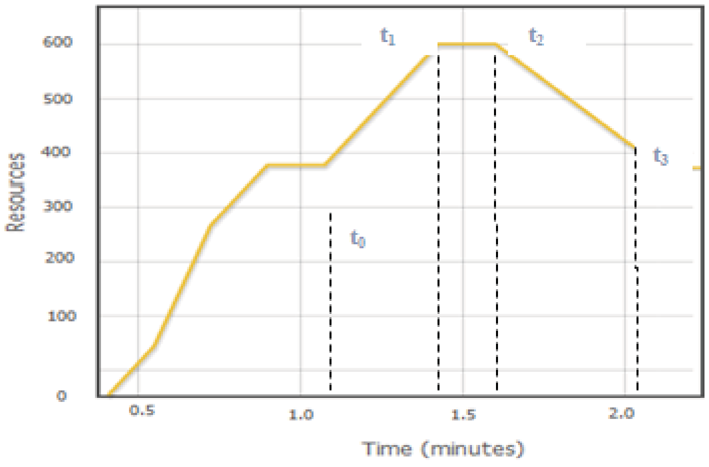


Fig. 4.6: Time Interval Distribution of Resource Spike

The final workload model is presented in the figure 4.7. Based on the workload pattern studies for cloud computing [140], [143], higher workload pattern was generated on weekdays as compared to lower number of requests during weekends to increase or decrease the data center load. Since we intend to study the behavior of the cloud resource provisioning policies in different situations of the load, the incoming job request load may exceed over 100% to reflect excess resource demands that may not be accommodated by using datacenter capacity. Lease or not to lease will be a key consideration for assessing the impact of cost, trust and goodwill. For Poisson distribution, Lamda (λ) was set to 10 for most of the times and some short random spikes were added to test the workload distribution of the system. However, number of incoming job requests during weekends were almost half of the normal load during the weekdays. It is worth mentioning that unlike real cloud environments, where resource demands can be obtained by analyzing usage history data (and the resultant value will be a specific threshold), cloud resource demand simulation initially starts up with zero workload and gradually warms up. Since the initial values are of little significance, these values, although considered in results, are not displayed in the workload model. The code for Poisson distribution is presented in Appendix-I.

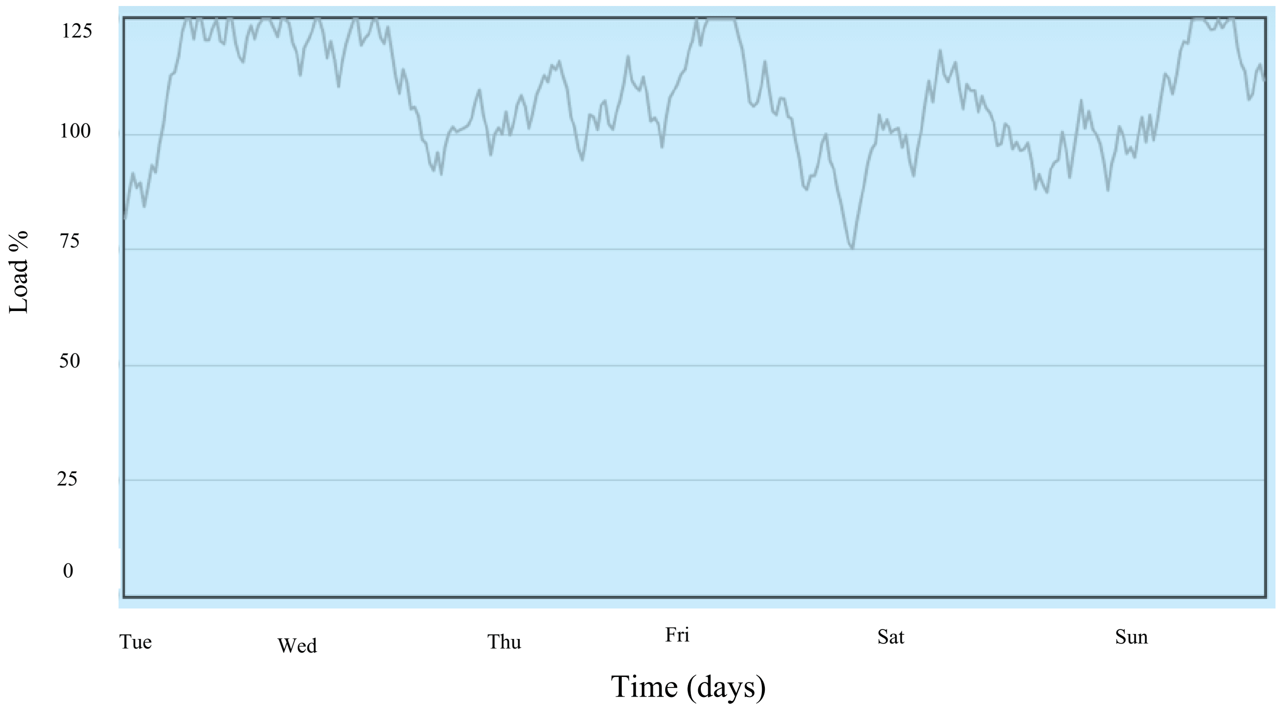


Fig. 4.7: Proposed Workload Model for Simulation

Chapter 5: Results & Discussion

In this chapter, first benchmarking results based on the four categories of metrics, discussed in earlier sections, are presented. In the second section, simulation results are presented while finally online experimentation results are presented in the last section of the chapter.

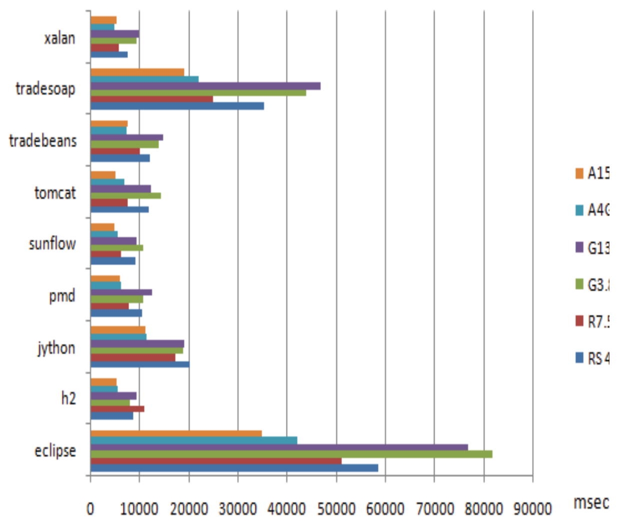
5.1 Benchmarking Results

Based on the literature, the three IaaS cloud providers were evaluated using the following categories of metrics: Compute, Memory Hierarchy, Network and Storage. Necessary statistical data for benchmarking was collected after a series of experiments conducted in a time span of one month. Multiple datacenters of the three cloud providers were used to calculate average performance estimation.

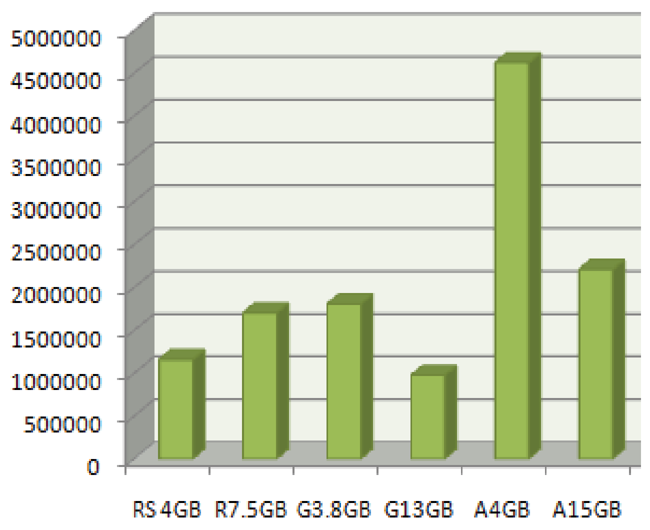
5.1.1 CPU Benchmarking

CPU Processing is a common performance benchmark for applications involving intensive workload. Processing performance is much dependant on clock speed, number of CPU cores and type of hardware. It is generally believed that VM instances with large number of CPU cores provide much better performance in terms of server execution and response time. However, experimentation results somehow depict that no significant performance gain was observed between VM instances with 2 vCPU and 4 vCPU compute instances. A maximum of 10-15% of performance variation was found between switching small instances to comparatively better server grade hardware. For instance, Decapo CPU benchmarking resulted in only 8% of performance gain while switching from Amazon t2.medium to Amazon m3.xlarge. Figures 5.1-5.3 present benchmarking results of Decapo, SPECJVM and Phoenix test suites respectively. Experimentation results demonstrate that Amazon m3.xlarge outperforms all other instances in terms of CPU performance.

A15GB outperformed other VM instances in terms of performance (as shown in figure 5.1a) but with relatively higher cost as compared to A4GB. However, the performance analysis, as presented in section 3.6.2, shows that A4GB instances are much more promising than other VM instances when the price versus performance score is taken into consideration (shown in figure 5.1b). Scientific applications, due to their inherently multi-threaded distributed nature, can significantly reduce computational cost by leasing such inexpensive VM instances instead of investing in relatively expensive VM instances with partial gain in overall performance.



(a) Benchmarking Result



(b) Price-performance score, $\alpha = 5$

Fig. 5.1: Benchmarking Result of Decapo Test suite

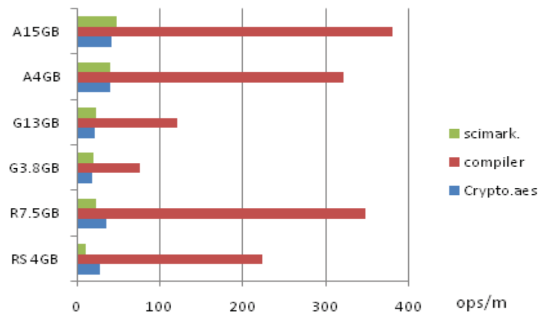
The price-performance comparison of all VMs, based on ranking procedure as outlined in section 3.6.2 and derived from figure 5.1 (b), is given in table 5.1.

Table 5.1: Decapo price-performance Analysis ($\alpha = 5$)

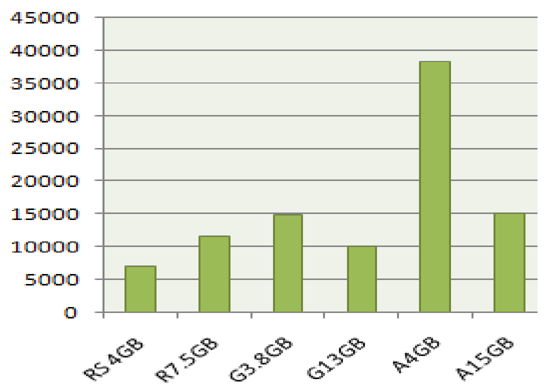
VM	RS 4GB	R7.5GB	G3.8GB	G13GB	A4GB	A15GB
RS 4GB	1:1	1.5:1	1.6:1	1: 1.2	4:1	1.9:1
R7.5GB	1.5:1	1:1	1.1:1	1:1.8	2.7:1	1.3:1
G3.8GB	1:1.6	1:1.1	1:1	1:1.9	2.6:1	1.2:1
G13GB	1.2:1	1.8:1	1.9:1	1:1	4.8:1	2.3:1
A4GB	1:4	1:2.7	1:2.6	1:4.8	1:1	1:2.1
A15GB	1:1.9	1:1.3	1:1.2	1:2.3	2.1:1	1:1

In table 5.1, each score is a ratio between two cloud providers. For instance, the score 1.5:1 of second column in first row shows that RS 4GB performance result is 1.5 times lower than R7.5GB. The most extreme ratio was between A4GB and G13GB with 1:4.8.

Scientific applications are inherently CPU and memory intensive and they often push computing resources to its full potential [144]. The sub-benchmarks of SPECjvm2008 are components of real world applications intended to test overall performance of modern CPU architecture and memory sub-system. Figure 5.2 depicts benchmarking results in ops/min (operations per minute) of three different test cases: Compiler for throughput measurement to reflect overall performance of the system, SciMark workload to test JVM code optimization targeted at L2 cache and memory subsystem and Crypto sub-benchmark with AES workload for performance testing of encryption and decryption protocols.



(a) Benchmarking Result



(b) Price-performance score, $\alpha = 10$

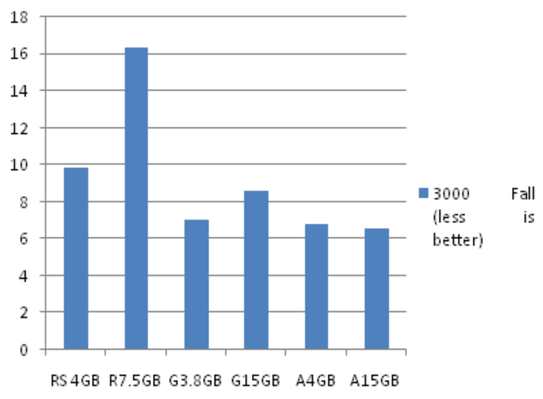
Fig. 5.2: Performance Classifications of three SPECJVM Test cases

As shown in Table 5.2, the most extreme ratio for SPECJVM benchmark was 5:1 among RS 4GB and A4B with 5:1.

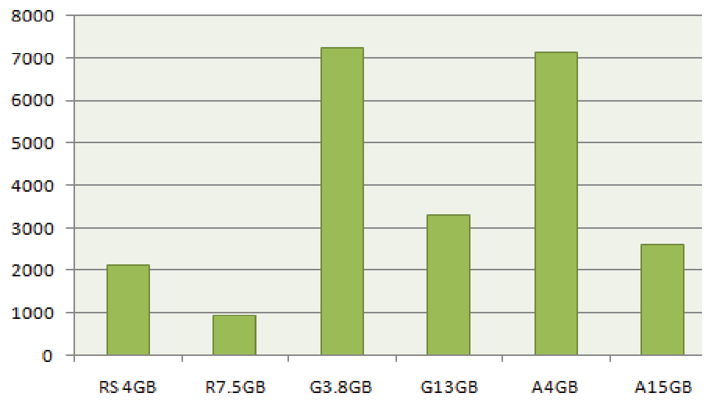
Table 5.2: SPECJVM price-performance Analysis ($\alpha = 10$)

VM	RS 4GB	R7.5GB	G3.8GB	G13GB	A4GB	A15GB
RS 4GB	1:1	1.6:1	2.1:1	1.4:1	5:1	2.1:1
R7.5GB	1:1.6	1:1	1.3:1	1:1.2	3.2:1	1.3:1
G3.8GB	1:2.1	1:1.3	1:1	1:1.5	2.6:1	1:1
G13GB	1:1.4	1.2:1	1.5:1	1:1	3.8:1	1.5:1
A4GB	1:5.3	1:3.2	1:2.6	1:3.8	1:1	1:2.5
A15GB	1:2.1	1:1.3	1:1	1:1.5	2.5:1	1:1

The Phoronix Test Suite is built for benchmarking real world applications and hardware performance comparisons including processor, system, memory and graphic performance. Bullet is a physics engine to simulate virtual environment that incorporates laws from the physical world. Figure 5.3 presents the experimentation results of 3000 fall bullet physics engine. The score of such experiments are inverted as VMs with better hardware configuration complete the simulation in relatively lesser time. The results are interesting and somehow differ from the earlier two CPU experiments as G3.8GB performed slightly better than A4GB in price-performance analysis.



(a) Benchmarking Result of 3000 fall bullet Phoronix test suite



(b) Price-performance score, $\alpha = 2$

Fig. 5.3: Performance Classifications of three Phoronix Test cases

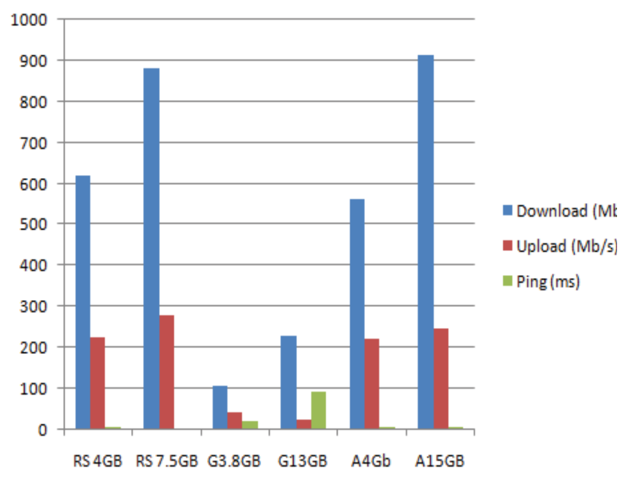
As given in table 5.3, the worst ratio was G3.8GB vs. R7.5GB with a performance variation of 1:7.7.

Table 5.3: Phoronix price-performance Analysis ($\alpha = 2$)

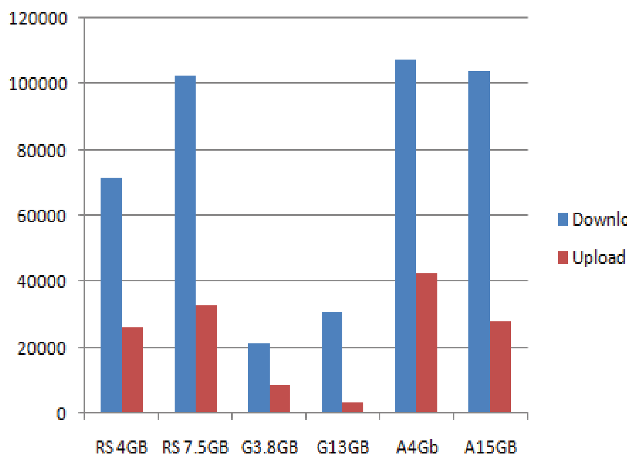
VM	RS 4GB	R7.5GB	G3.8GB	G13GB	A4GB	A15GB
RS 4GB	1:1	1:2.2	3.4:1	1.6:1	3.4:1	1.2:1
R7.5GB	2.2:1	1:1	7.7:1	3.5:1	7.6:1	2.8:1
G3.8GB	1:3.4	1:7.7	1:1	1:2.2	1:1	1:2.8
G13GB	1:1.6	1:3.5	2.2:1	1:1	2.2:1	1:1.3
A4GB	1:3.4	1:7.6	1:1	1:2.2	1:1	1:2.8
A15GB	1:1.2	1:2.8	2.8:1	1.3:1	2.8:1	1:1

5.1.2 Network Benchmarking

Network performance determines how fast a VM can communicate with other clients over a network. Unlike most cloud storage and database applications, which are comparatively more tolerant to delay and jitter, media applications are far more demanding. The network delay with large jitter degrades user received media quality of latency sensitive multimedia applications [117],[118]. Hence, cloud providers with better network performance in terms of throughput vs. latency play a key role in the success of such business applications. The network performance of the three cloud providers was evaluated using iperf, speedtest-cli and ping. The results are depicted in figure 5.4.



(a) Benchmarking Result



(b) Price-performance score, $\alpha = 5$

Fig. 5.4: Network performance of VM Instances

Table 5.4: Network price-performance Analysis ($\alpha = 5$)

(a) Download analysis

VM	RS 4GB	R7.5GB	G3.8GB	G13GB	A4GB	A15GB
RS 4GB	1:1	1.4:1	1:3.4	1:2.3	1.5:1	1.4:1
R7.5GB	1:1.4	1:1	1:4.8	1:3.3	1:1	1:1
G3.8GB	3.4:1	4.8:1	1:1	1.5:1	5:1	5:1
G13GB	2.3:1	3.3:1	1:1.5	1:1	3.4:1	3.3:1
A4GB	1:1.5	1:1	1:5	1:3.4	1:1	1:1
A15GB	1:1.4	1:1	1:4.9	1:3.3	1:1	1:1

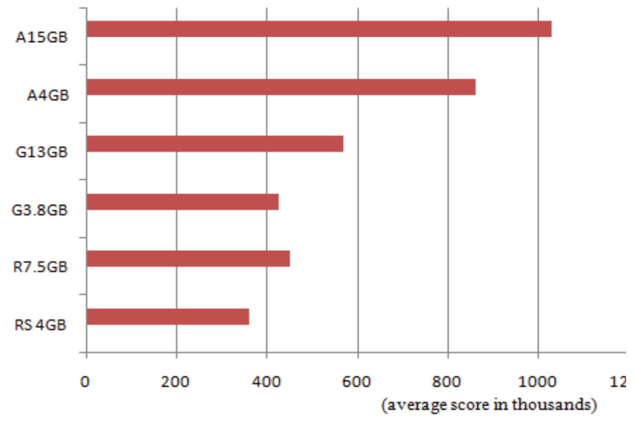
(b) Upload analysis

VM	RS 4GB	R7.5GB	G3.8GB	G13GB	A4GB	A15GB
RS 4GB	1:1	1.2:1	1:3.1	1:6.1	1.6:1	1.1:1
R7.5GB	1:1.2	1:1	1:3.8	1:7.6	1.3:1	1:1.2
G3.8GB	3.1:1	3.8:1	1:1	1:2	5:1	3.3:1
G13GB	6.1:1	7.6:1	2:1	1:1	10:1	6.6:1
A4GB	1:1.6	1:1.3	1:5	1:10	1:1	1:1.5
A15GB	1:1.1	1.2:1	1:3.3	1:6.6	1.5:1	1:1

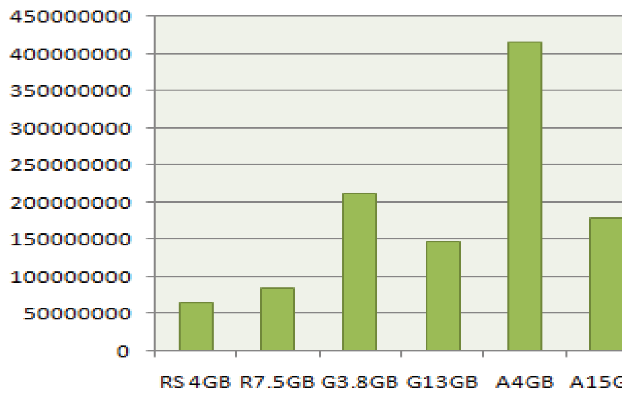
As shown in table 5.4, both Amazon and Rackspace provide much better network performance with notably higher download/ upload speed. Google VM instances are relatively poor in terms of network performance and latency. The most extreme ratios 1:5, between G3.8GB and A4GB in case of download and 1:10, between G13GB and A4Gb in case of upload were obtained in calculation results.

5.1.3 Memory Benchmarking

Memory is the core element that determines the processing speed of an application. Many scientific and business applications are extremely dependant on memory system performance. Under heavy memory contention, the memory latency may increase two or three times. Thus, the increasing performance gap between processors and memory systems imposes a memory bottleneck for such applications [116]. The three cloud providers were evaluated using Ubench and Cachebench benchmarking tools. Experimentation results are presented in figures 5.5 and 5.6. The results suggest that Google and Amazon VM instances are well optimized for memory related operations.



(a) Benchmarking Result



(b) Price-performance score, $\alpha = 2$

Fig. 5.5: Ubench Memory Benchmarking Score

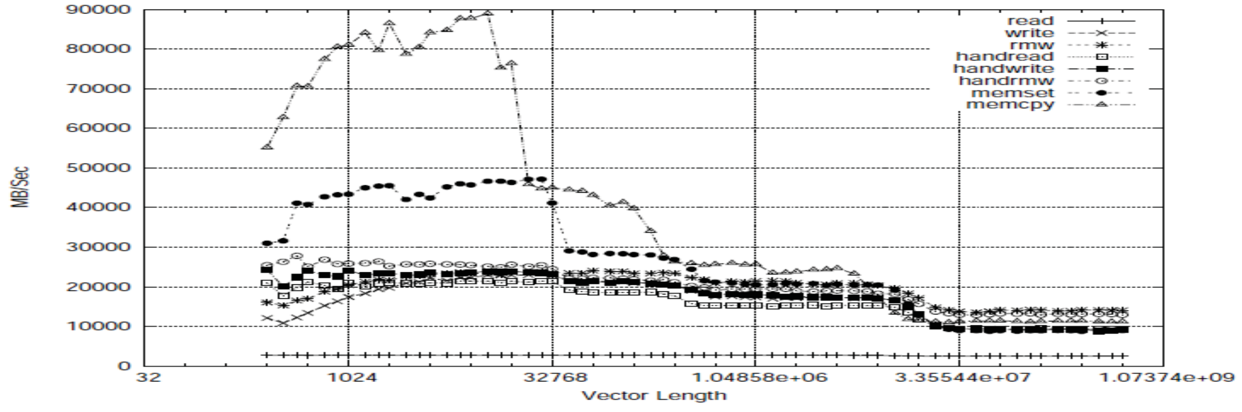
As summarized in table 5.5, Rackspace instances are far below in the memory related functions and operations. Likewise CPU benchmarking, no major performance degradation was observed between VM instances with less number of cores and memory. Comparative benchmarking results among different cloud providers show that the most extreme ratio was between RS 4 GB and A 4GB with 6.3:1.

Table 5.5: Ubench price-performance Analysis ($\alpha = 2$)

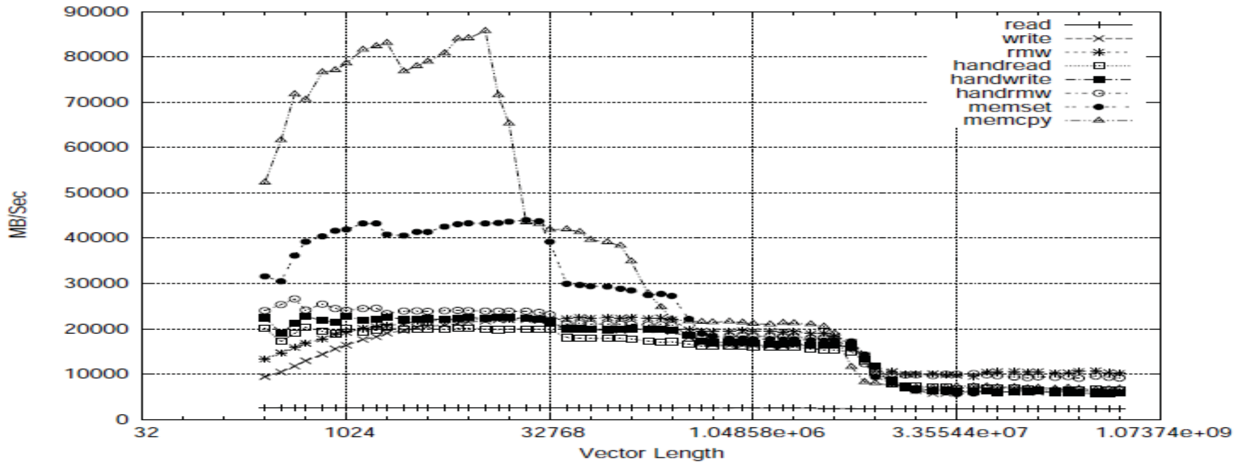
VM	RS 4GB	R7.5GB	G3.8GB	G13GB	A4GB	A15GB
RS 4GB	1:1	1.3:1	3.2:1	2.2:1	6.3:1	2.7:1
R7.5GB	1:1.25	1:1	2.6:1	1.8:1	5:1	2.2:1
G3.8GB	1:3.2	1:2.6	1:1	1:1.6	2:1	1:1.2

G13GB	1:2.2	1:1.8	1.5:1	1:1	2.9:1	1.2:1
A4GB	1:6.3	1:5	1:2	1:2.9	1:1	1:2.3
A15GB	1:2.7	1:2.2	1.2:1	1:1.2	2.3:1	1:1

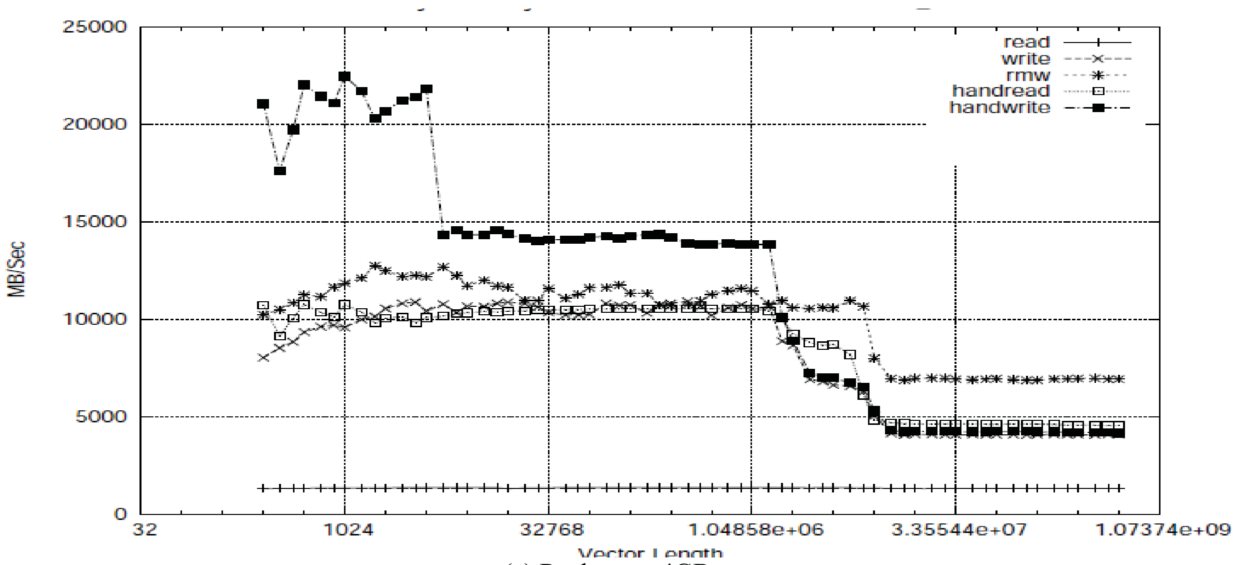
Cachebench experiments were performed for benchmarking memory system performance using read, write, rmw, handread, handwrite, handrmw, memset,memcpy operations. From figure 5.6, it is apparent that Amazon and Google VM instances have very identical memory performance while the performance of Rackspace VM instances was degraded during the memory related operations.



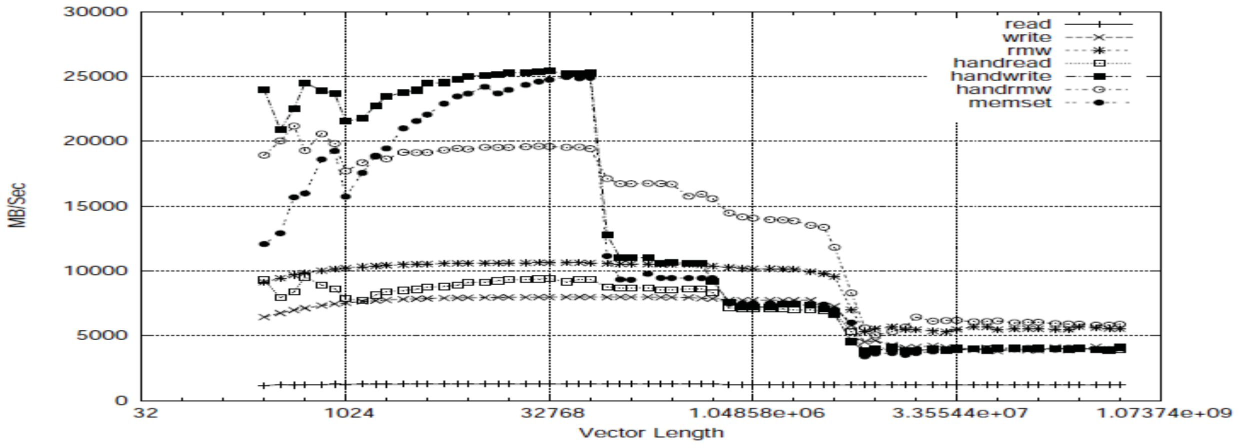
(a) Google 4GB



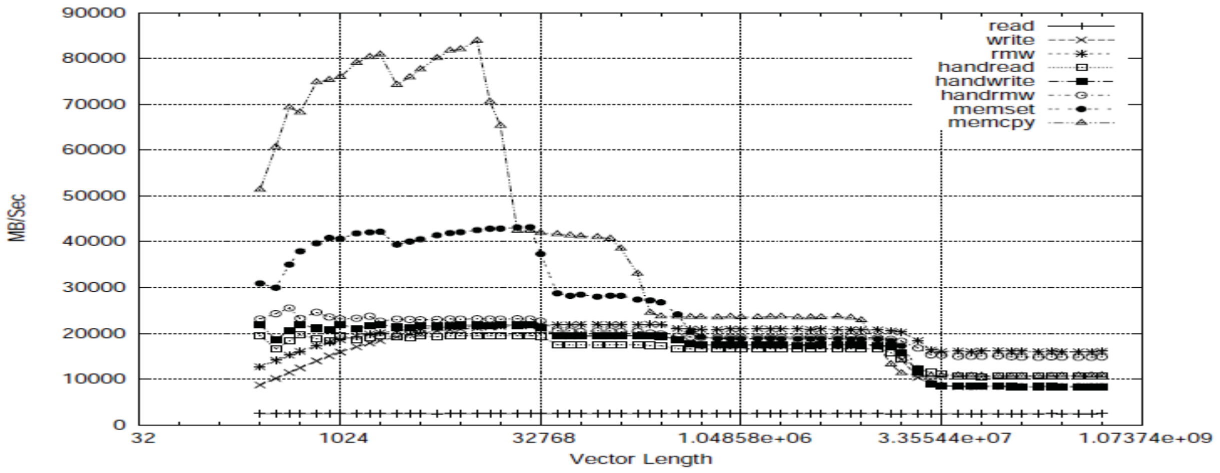
(b) Google 13GB



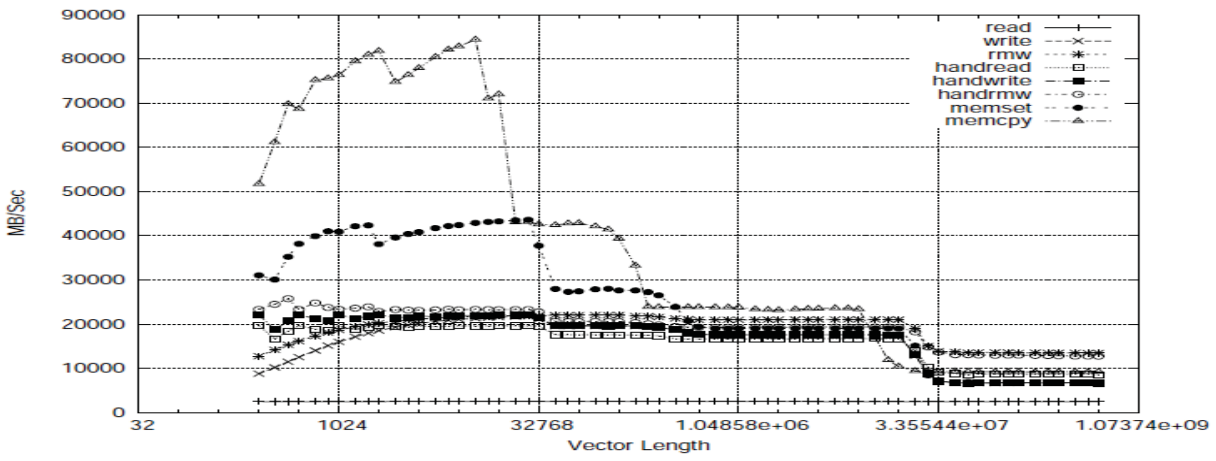
(c) Rackspace 4GB



(d) Rackspace 7.5GB



(e) Amazon 4GB



(f) Amazon 15GB

Fig. 5.6: Cachebench Results of three Cloud Providers

5.1.4 Storage Benchmarking

Considering the nature of cloud computing environment where scientific applications and business workflows continuously involve storage I/O operations, it is important to determine how quickly storage devices allow applications to interact with data or files. Experimentations results, as shown in figure 5.7, reveal a steady IO performance for both Rackspace and Amazon, however, performance variation of Google is a matter of discussion for future studies. The second sets of experiments with Bonnie suggest that Rackspace VM instances outperformed Amazon and Google. Furthermore, the two Google VM instances were failed to complete IO operations for file size greater than 4 GB. It clearly shows that such instances are not built for data intensive cloud applications.

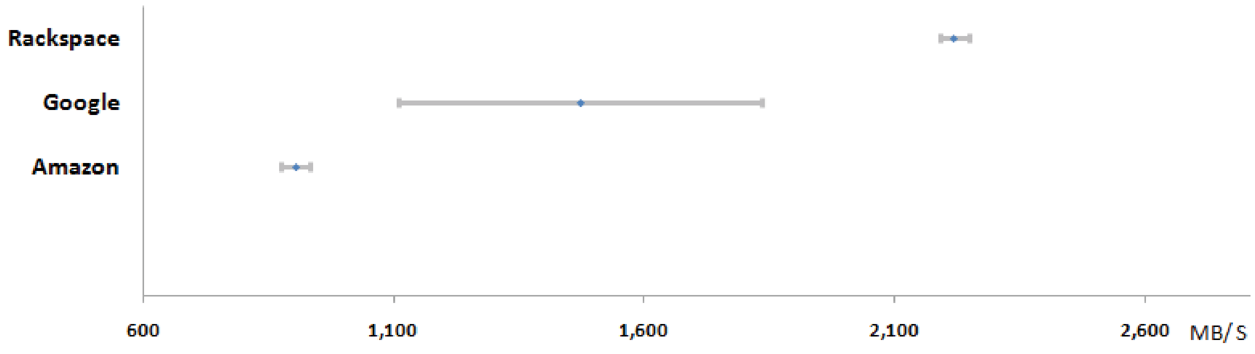


Fig. 5.7: IOZone Benchmarking Results of 4GB VM instances

Tables 5.5- 5.8 clearly present that Rackspace instances performed much better for all file operations ranging up to 4 GB file size performance test. Amazon instances completed most of the operations but got stuck at 4GB file operations for a block size larger than 1024. Google instances produced poor I/O performance results and could not even completed some of the 2GB file operations.

For I/O benchmarking through bonnie++, data size double of the size of RAM was provided. However, an exception was Google13GB instance which was failed on such data size and experiments had to be restricted at a maximum of 8 GB data size. As presented in figure 5.8, Rackspace VM instances provided better performance results while Amazon instances usage for CPU was lesser during the same operations. Google VM instances were ranked lower in all cases.

Table 5.6: Rackspace IOZone Results

		read	**reread**	**write**	**rewrite**	**fread**	**randread**
1048576	64	3382634	3342491	414765	1622381	3342491	3200420
	128	3474879	3542198	428442	1650396	3542198	3437660
	256	3474923	3613884	425441	1668508	3613884	3370747
	512	3568708	3655486	422748	1674525	3655486	3609581
	1024	3220115	3263166	419646	1606513	3263166	3236275
	2048	3166342	3194931	421690	1587983	3194931	3201494
	4096	2585597	2654133	412644	1463031	2654133	2649403
	8192	2548682	2609046	412064	1461604	2609046	2611398
	16384	2509041	2598688	403356	1398791	2598688	2563868
2097152	64	3278152	3345919	359845	1574608	3345919	3119587
	128	3383889	3458900	350208	1618113	3458900	3374822
	256	3424228	3526853	362374	1636934	3526853	3421101
	512	3492896	3562217	347045	1688271	3562217	3626314
	1024	3445013	3471726	362576	1635600	3471726	3455333
	2048	3087488	3149921	346430	1536836	3149921	3103852
	4096	2685739	2824306	334507	1435786	2824306	2770320
	8192	2444680	2504014	339638	1415473	2504014	2472057
	16384	2453515	2621600	327991	1439147	2621600	2619419

4194304	64	3275259	3352209	307455	560398	3352209	3140798
	128	3416583	3534116	306743	634439	3534116	3497915
	256	3616094	3701592	304603	644555	3701592	3589240
	512	3633373	3709537	325815	665320	3709537	3629200
	1024	3448707	3490488	310532	639957	3490488	3505940
	2048	3169432	3227281	301190	642106	3227281	3304097
	4096	2670099	2650404	310398	630980	2650404	2732625
	8192	2510975	2621199	295546	646799	2621199	2574707
	16384	2444184	2581151	298815	734278	2581151	2546598

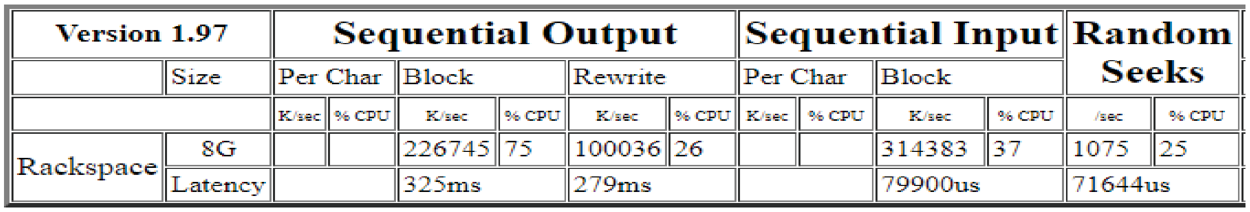
Table 5.7: Amazon IOZone Results

		read	reread	write	Rewrite	fread	randread
1048576	64	7683280	7790139	1561024	2843327	7733715	7425597
	128	7625398	7718243	1562168	2954019	7674230	7413469
	256	6886885	7297944	1558606	2903058	7216184	7113474
	512	7010885	7451880	1572599	2905390	7392919	7379609
	1024	7041112	7537912	1580947	2922542	7524873	7471464
	2048	7058934	7527630	1585666	2937079	7470246	7529782
	4096	7039591	7603396	1579509	2932694	7410221	7574391
	8192	6990282	7417194	1564600	2868599	7375513	7391938
	16384	4477227	4519216	1511803	2653602	4772285	4403688
2097152	64	7624571	7737939	1564781	2861291	7636254	7342401
	128	7619136	7679006	1567393	2945218	7619994	7359048
	256	6955842	7229094	1570127	2867821	7272889	7040683
	512	7005569	7400208	1566001	2923157	7380184	7362892
	1024	7025325	7472238	1581715	2893412	7426468	7446901
	2048	6992303	7568922	1587800	2938170	7534121	7533173
	4096	6976387	7528137	1585369	2929502	7576987	7565921
	8192	6998061	7500835	1572799	2920470	7429121	7515543
	16384	4717534	4688879	1498704	2691751	4495077	4771458
4194304	64	7685057	7702841	320908	237367	7747029	7275890
	128	7605748	7654761	322318	310238	7673047	7294465
	256	7002515	7220764	322788	310109	7283219	7147925
	512	7009886	7333265	9316	10106	7369795	7289750

	1024	-------	-------	-------	-------	-------	-------
	2048	-------	-------	-------	-------	-------	-------
	4096	-------	-------	-------	-------	-------	-------
	8192	-------	-------	-------	-------	-------	-------
	16384	-------	-------	-------	-------	-------	-------

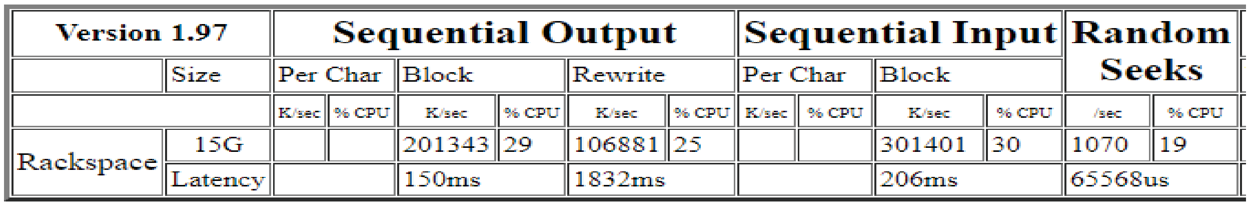
Table 5.8: Google IOZone Results

		read	reread	write	rewrite	fread	Randread
1048576	64	6697510	6837484	1554915	3214291	6136114	6422298
	128	6102229	6565583	1603815	3169300	6118461	6016621
	256	5682019	5842756	1628130	3172743	5712410	5658494
	512	5996694	6077152	1648113	3291345	5922849	6001915
	1024	6165706	6116282	1660558	3375207	5905742	5529965
	2048	5689811	6133957	1649127	3341095	5906940	6164617
	4096	6050499	6167798	1661041	3179284	5978093	6189830
	8192	6093470	6202173	1617139	3323221	5863727	6197811
	16384	5349355	5456357	1516517	2894779	5150632	5420453
2097152	64	7080690	7262363	1310443	2427523	6187930	6595291
	128	6229510	6245482	1354827	2447068	5876038	5874750
	256	5791190	5863847	1307913	2398689	5512901	5666815
	512	6019172	6052822	1339378	2471109	5832160	5920393
	1024	5873699	6033391	1346250	2444392	5924995	6051023
	2048	-------	-------	-------	-------	-------	-------
	4096	-------	-------	-------	-------	-------	-------
	8192	-------	-------	-------	-------	-------	-------
	16384	-------	-------	-------	-------	-------	-------
4194304	64	-------	-------	-------	-------	-------	-------
	128	-------	-------	-------	-------	-------	-------
	256	-------	-------	-------	-------	-------	-------
	512	-------	-------	-------	-------	-------	-------
	1024	-------	-------	-------	-------	-------	-------
	2048	-------	-------	-------	-------	-------	-------
	4096	-------	-------	-------	-------	-------	-------
	8192	-------	-------	-------	-------	-------	-------
	16384	-------	-------	-------	-------	-------	-------



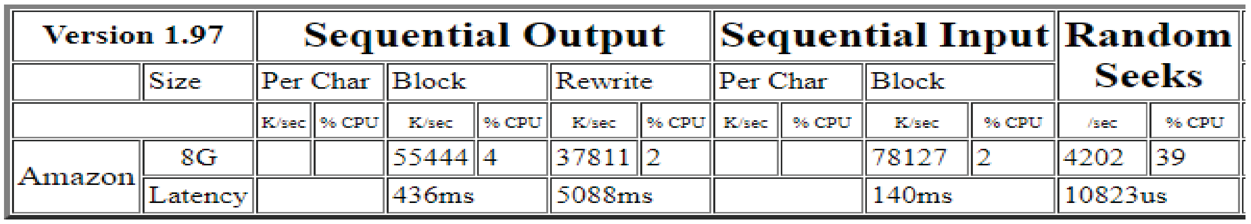
Version 1.97		Sequential Output						Sequential Input				Random Seeks	
	Size	Per Char		Block		Rewrite		Per Char		Block			
		K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	/sec	% CPU
Rackspace	8G			226745	75	100036	26			314383	37	1075	25
	Latency			325ms		279ms				79900us		71644us	

(a) Rackspace 4GB



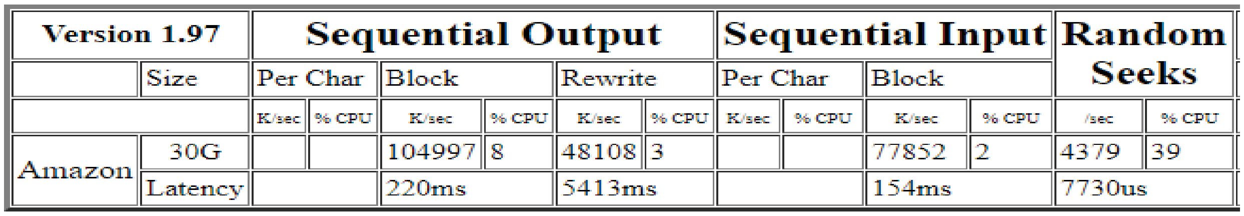
Version 1.97		Sequential Output						Sequential Input				Random Seeks	
	Size	Per Char		Block		Rewrite		Per Char		Block			
		K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	/sec	% CPU
Rackspace	15G			201343	29	106881	25			301401	30	1070	19
	Latency			150ms		1832ms				206ms		65568us	

(b) Rackspace 7.5GB



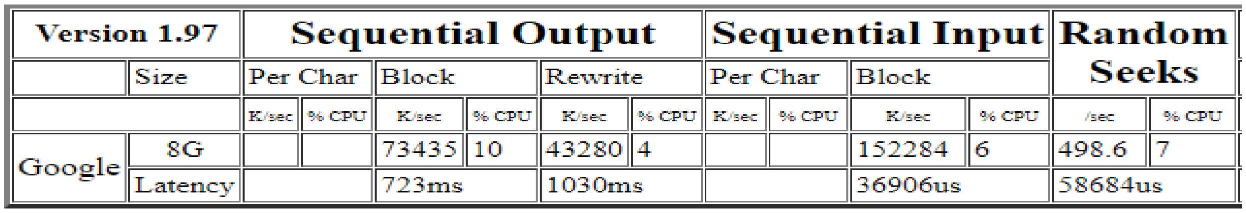
Version 1.97		Sequential Output						Sequential Input				Random Seeks	
	Size	Per Char		Block		Rewrite		Per Char		Block			
		K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	/sec	% CPU
Amazon	8G			55444	4	37811	2			78127	2	4202	39
	Latency			436ms		5088ms				140ms		10823us	

(c) Amazon 4GB



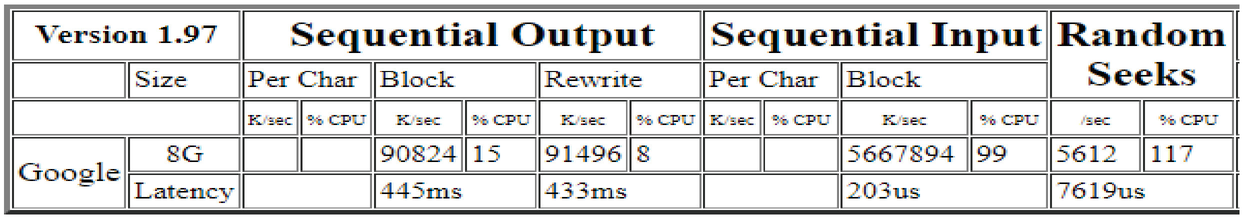
Version 1.97		Sequential Output						Sequential Input				Random Seeks	
	Size	Per Char		Block		Rewrite		Per Char		Block			
		K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	/sec	% CPU
Amazon	30G			104997	8	48108	3			77852	2	4379	39
	Latency			220ms		5413ms				154ms		7730us	

(d) Amazon 15GB



Version 1.97		Sequential Output						Sequential Input				Random Seeks	
	Size	Per Char		Block		Rewrite		Per Char		Block			
		K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	/sec	% CPU
Google	8G			73435	10	43280	4			152284	6	498.6	7
	Latency			723ms		1030ms				36906us		58684us	

(e) Google 4GB



Version 1.97		Sequential Output						Sequential Input				Random Seeks	
	Size	Per Char		Block		Rewrite		Per Char		Block			
		K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	/sec	% CPU
Google	8G			90824	15	91496	8			5667894	99	5612	117
	Latency			445ms		433ms				203us		7619us	

(f) Google 13GB (Tested on 8GB File Size)

Fig. 5.8: Bonnie File System Benchmarking Results

5.2 Ranking Cloud Providers for Scientific Computing

For overall price-performance ranking of the three cloud providers, additive weighting scheme was used as discussed in section 3.6.2. Scientific applications are usually hardware dependant and may be compute and I/O intensive [145]- [146], compute and bandwidth intensive [147]-[148] or compute and memory intensive [149], [150], [151]. Based on the research studies and necessary ranking procedure as outlined in section 3.6.2, weights were assigned to each evaluation metric as given in table 5.9.

Table 5.9 : List of Metrics and associated weights

Metric	**Top level weight(W_j)**	**Benchmark (j)**	**Second level (w_j)**
CPU	0.4	Phoronix	0.4
		SPECjvm	0.4
		Dacapo	0.2
Memory	0.3	Cachebench	0.6
		Ubench	0.4
Network	0.15	Upload	0.4
		Download	0.4
		Ping	0.2
Storage	0.15	Bonnie	0.6
		IOZone	0.4

For all VMs, the utility value against each benchmark is obtained by multiplying the relative normalized score with the relative second level weight of the benchmark w_j. For every CPU metric, the score is aggregated, resulting in cumulative utility value (Vj). The utility function is then calculated by multiplying top level weight (Wj) by cumulative utility value. Finally, total ranking score is calculated as a sum of utility functions for all evaluation metrics. The ranking scores for all VMs are given in table 5.10 where RS 4GB, R7.5GB, G3.8GB, G13GB, A4GB, A15GB are represented as VM1, VM2,…, VM6 respectively. The results clearly indicate that A4GB outperformed all other VMs in terms of price-performance evaluation metrics, followed by G3.8GB, A15GB, R7.5GB, 4GB, and G13GB. The final results are somehow different from the benchmarking studies conducted earlier [109], [152] as the computational cost, being the driving factor of cloud computing model, was not considered in these studies and hence the performance evaluation results may not be on equal ground.

Table 5.10: Aggregate Utility Ranking based on Performance Metrics

Metric	Benchmark	VM_1	v_{1j}	V_{1j}	$f(V_{1j})$	VM_2	v_{2j}	V_{2j}	$f(V_{2j})$	VM_3	v_{3j}	V_{3j}	$f(V_{3j})$
CPU	Phoronix	0.09	0.04	0.09	0.036	0.04	0.02	0.1	0.04	0.31	0.12	0.21	0.08
	SPECjvm	0.07	0.03			0.12	0.05			0.15	0.06		
	Dacapo	0.09	0.02			0.14	0.03			0.14	0.03		
Network	Download	0.16	0.06	0.18	0.027	0.23	0.09	0.23	0.035	0.05	0.02	0.06	0.009
	Upload	0.18	0.07			0.23	0.09			0.06	0.02		
	Ping	0.23	0.05			0.25	0.05			0.08	0.02		
Memory	Cachebench	0.09	0.05	0.07	0.021	0.07	0.04	0.07	0.021	0.28	0.17	0.24	0.072
	UBench	0.06	0.02			0.08	0.03			0.19	0.07		
Storage	Bonnie	0.23	0.14	0.22	0.033	0.19	0.11	0.18	0.027	0.08	0.05	0.12	0.018
	Iozone	0.21	0.08			0.18	0.07			0.18	0.07		
Total					**0.117**				**0.123**				**0.179**

Metric	Benchmark	VM_4	v_{4j}	V_{4j}	$f(V_{4j})$	VM_5	v_{5j}	V_{5j}	$f(V_{5j})$	VM_6	v_{6j}	V_{6j}	$f(V_{6j})$
CPU	Phoronix	0.14	0.06	0.12	0.05	0.31	0.12	0.35	0.14	0.11	0.04	0.14	0.06
	SPECjvm	0.10	0.04			0.39	0.16			0.16	0.06		
	Dacapo	0.08	0.02			0.37	0.07			0.18	0.04		
Network	Download	0.07	0.03	0.05	0.008	0.24	0.10	0.26	0.039	0.24	0.10	0.22	0.033
	Upload	0.03	0.01			0.30	0.12			0.20	0.08		
	Ping	0.03	0.01			0.23	0.04			0.18	0.04		
Memory	Cachebench	0.16	0.10	0.15	0.045	0.26	0.16	0.31	0.093	0.15	0.09	0.15	0.045
	UBench	0.13	0.05			0.38	0.15			0.16	0.06		
Storage	Bonnie	0.06	0.04	0.06	0.009	0.25	0.15	0.24	0.036	0.18	0.11	0.18	0.027
	Iozone	0.04	0.02			0.22	0.09			0.17	0.07		
Total					**0.112**				**0.308**				**0.165**

5.3 Simulation Results

For simulation setup, the workload pattern as presented in figure 4.7 was used as a reference model. Rackspace 7.5GB VM instances was used as a reference, charged at US $0.23 per hour with an additional cost of US$ 0.12 per GB outgoing bandwidth usage. During simulation, each data center supported contained 15 physical servers, each capable to host a maximum of 8 VM instances. Simulation data for a period of one week was collected. The request arrival rate varied over time; during weekdays the average workload was set around 1575 VM requests while the same was reduced to 975 for weekends. Simulation was performed for both local resource allocation (LRA) and broker based outsource oriented resource allocation policies (BOR). A marginally lower random probability was also assigned to job failure during the course of execution.

Each individual simulation run was performed for one hour. In this section, simulation result for a workload pattern with varying load is presented. Resource demand was at low during the start and gradually increased with the passage of time. Resource surge was reached at 100% during the mid of simulation and further job requests was denied by LRA while resource outsourcing was started for BOR. The load pattern is given in figure 5.9.

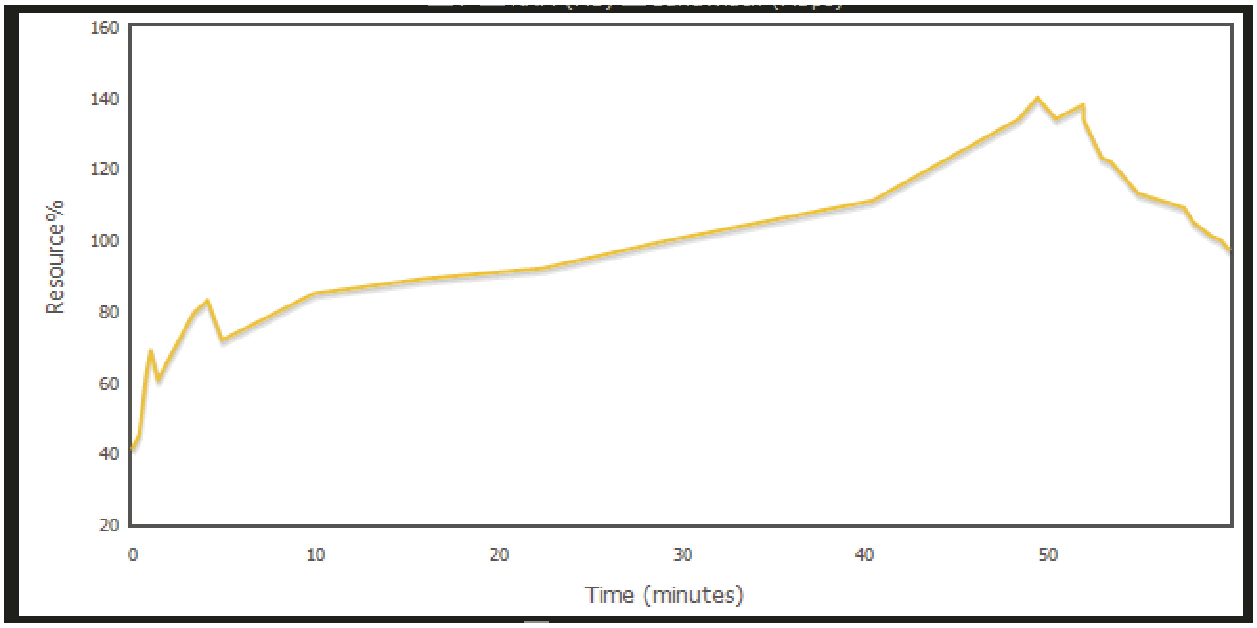


Fig. 5.9: Resource Demand on Varying Loads

For LRA, the policy was set to utilize maximum capacity of resource pool. However, upon the full utilization of the VM pool, further incoming requests were rejected until some VMs were de-allocated after job completion. The resource utilization pattern is presented in the following figure:

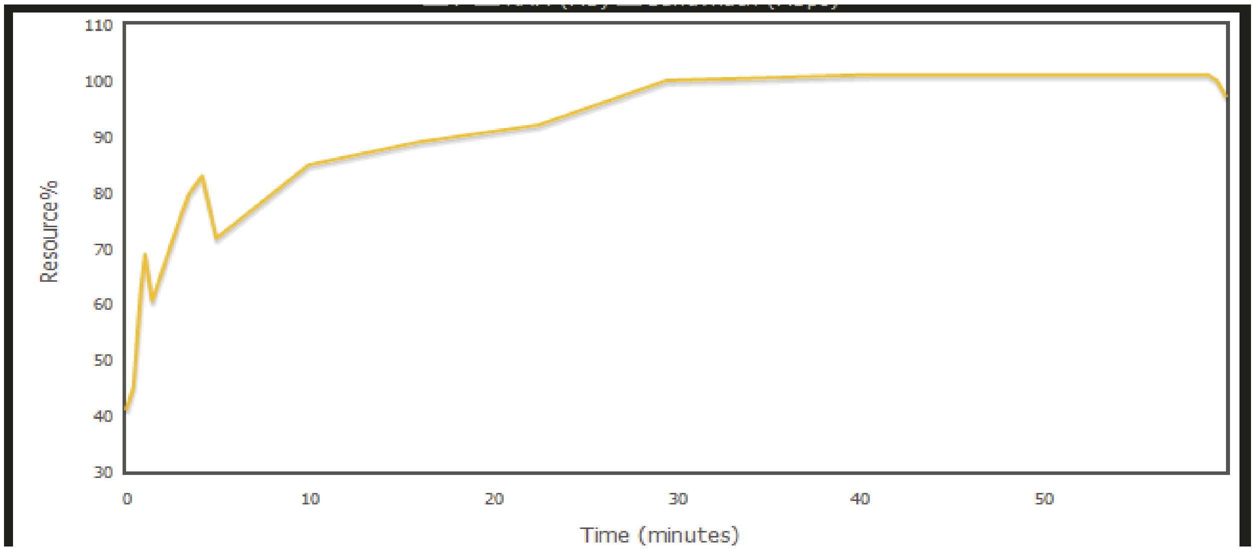
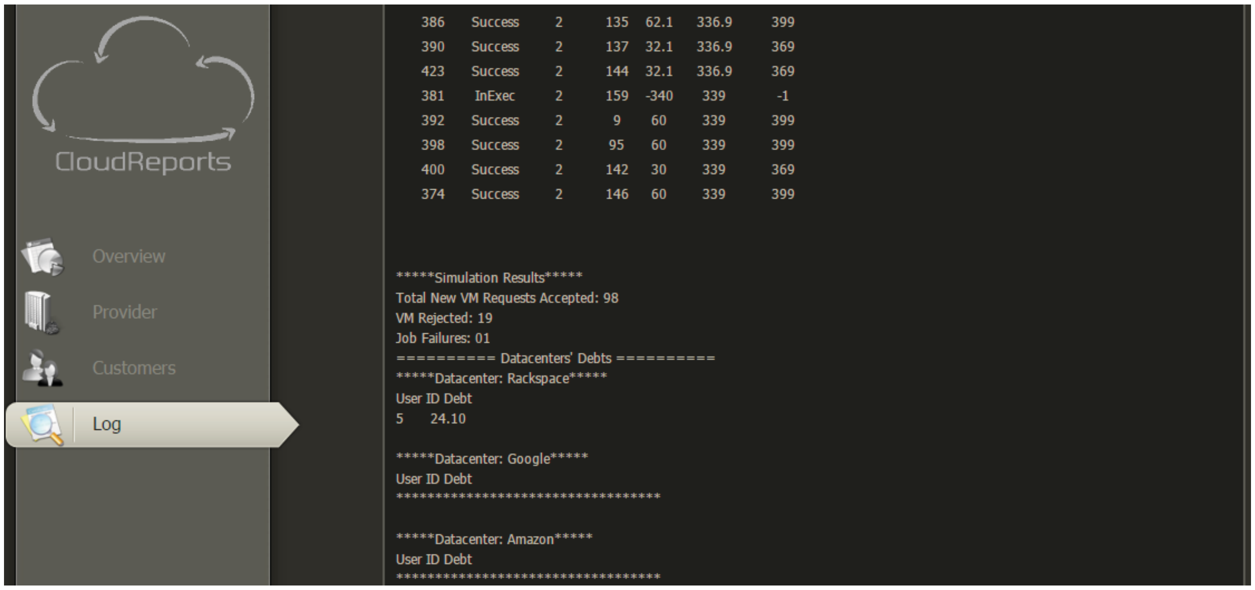


Fig. 5.10: Overall Resource Utilization of LRAI Policy

As shown in figure 5.10, the resource surge reached at maximum during the mid of simulation, and further job requests were not entertained and hence a steady utilization graph at 100% is achieved for a period time. However, in the end of simulation, after completion of the job cycle for few VMs, the job pool was once again available for future job requests. Simulation results for this one hour of experiment are given in figure 5.11.



```
    386   Success   2     135  62.1   336.9    399
    390   Success   2     137  32.1   336.9    369
    423   Success   2     144  32.1   336.9    369
    381    InExec   2     159  -340    339      -1
    392   Success   2       9    60    339     399
    398   Success   2      95    60    339     399
    400   Success   2     142    30    339     369
    374   Success   2     146    60    339     399



*****Simulation Results*****
Total New VM Requests Accepted: 98
VM Rejected: 19
Job Failures: 01
========== Datacenters' Debts ==========
*****Datacenter: Rackspace*****
User ID Debt
5    24.10

*****Datacenter: Google*****
User ID Debt
**********************************

*****Datacenter: Amazon*****
User ID Debt
**********************************
```

Fig. 5.11: Results of LRAI Policy

As depicted in the figure 5.11, a total of 98 new VM requests were accepted while 19 VMs were rejected during the peak workload. Total user debt was US$ 24.10 for one hour of simulation. Execution of one job was failed but since the offered capacity did not allow migration to other data centers, this was resulted in SLA violation and hence compensation penalty had to be paid for such occurrences.

The BOR policy was aimed to fully utilize local resource pool as well as lease available capacity from other cloud providers based on particular QoS requirements, whenever required. This policy results in better economic efficiency and economics of scale. As a result, user jobs being rejected by LRA can now be outsourced which increases overall profit of cloud providers. During simulation, local resource pool was used till the mid of simulation where resource demand was within the capacity of local datacenter. However, the broker component was requested to outsource further incoming requests based QoS requirements. Before the start of simulation, cloud user was requested to provide necessary QoS requirements which are critical to complete job execution. The feedback was then matched with SLA of foreign cloud providers with the reference cloud provider, Rackspace in this case. During the outsourcing phase, only those VM instances from foreign cloud providers were selected who match with user SLA requirements. Since foreign cloud providers cannot satisfy all such requests, few jobs were rejected during simulations. The resource utilization pattern is presented in figure 5.12.

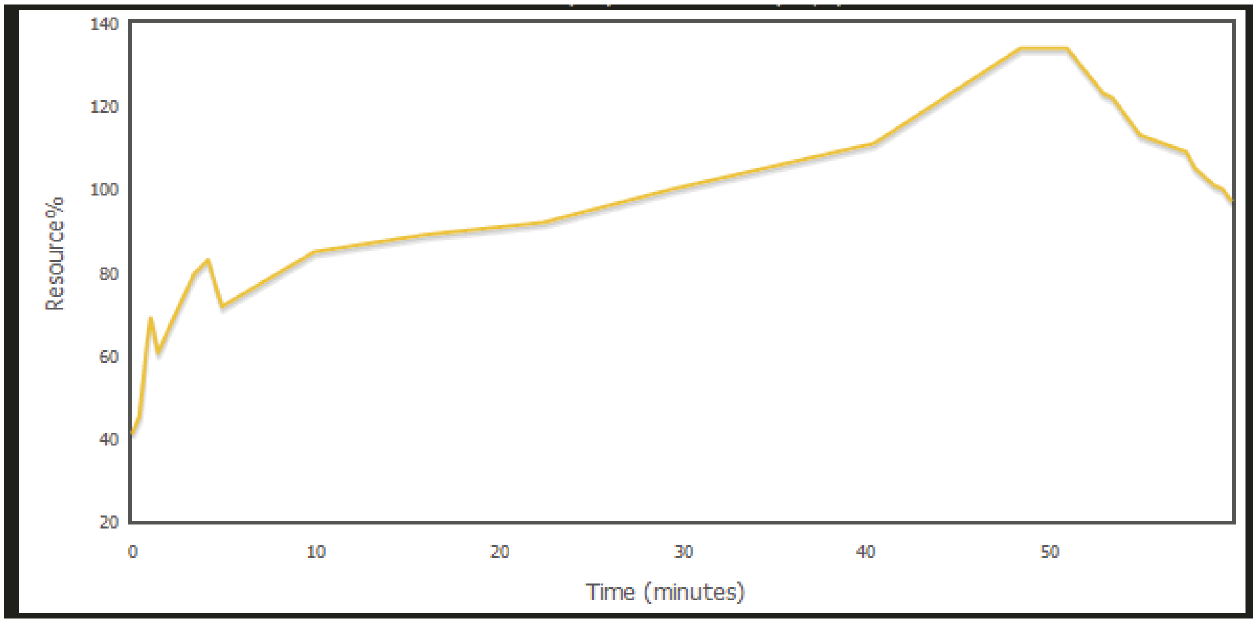
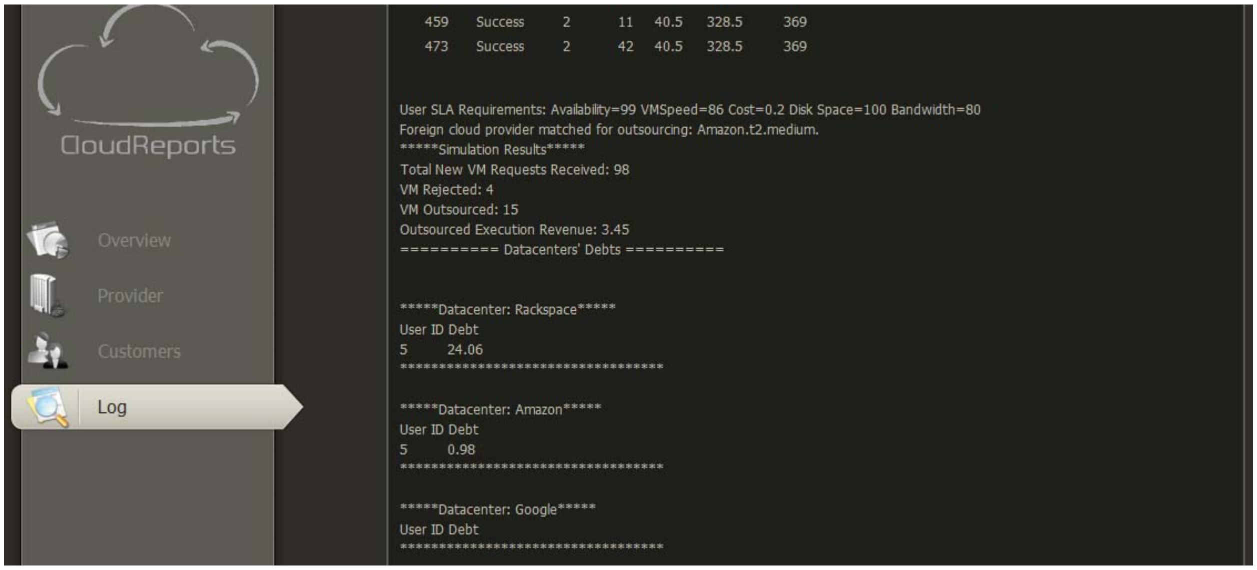


Fig. 5.12: Resource Utilization of BOR Policy

BOR policy resulted in managing resource demand above 100% and incoming job requests were outsourced to foreign cloud providers. For the SLA requirements of this particular simulation experiment, Amazon t2.medium was selected as a candidate cloud provider for resource outsourcing. After the mid of simulation, where the local resources were almost fully utilized, outsourced VM for the candidate cloud provider were leased for incoming job requests. However, since the foreign cloud provider was also managing its own workload, some VM requests were rejected due to relatively high resource demand within the datacenter of foreign cloud provider. However, overall job satisfaction and resource utilization was much higher for this particular kind of experiment. Much similar with LRA, one job was failed on the local servers; however this job was outsourced to foreign cloud provider to avoid any SLA violation.

The results of simulation are depicted in figure 5.13.



```
    459    Success    2       11   40.5   328.5     369
    473    Success    2       42   40.5   328.5     369


User SLA Requirements: Availability=99 VMSpeed=86 Cost=0.2 Disk Space=100 Bandwidth=80
Foreign cloud provider matched for outsourcing: Amazon.t2.medium.
*****Simulation Results*****
Total New VM Requests Received: 98
VM Rejected: 4
VM Outsourced: 15
Outsourced Execution Revenue: 3.45
========== Datacenters' Debts ==========


*****Datacenter: Rackspace*****
User ID Debt
5      24.06
**********************************

*****Datacenter: Amazon*****
User ID Debt
5      0.98
**********************************

*****Datacenter: Google*****
User ID Debt
**********************************
```

Fig. 5.13: Simulation Results of BOR Policy

As presented in figure 5.13, a total of 112 VM requests were accommodated under this policy and only 4 job requests were declined. Only one job failure did occur during the execution, but this particular VM was migrated to the foreign cloud provider to avoid any SLA violation. A total of 17 VM requests were outsourced to foreign cloud provider. Total revenue of this simulation was US$ 26.53 while the total cost of outsourced VMs was US$ 0.98.

Overall revenue was calculated using billing of Rackspace, Amazon and Google. Experimentation results were calculated through a series of iterations and normalized profit values were calculated. Figure 5.14 presents profit for two polices for 8 hours of simulation. The results clearly depict that since outsourcing policy results in more resource utilization, overall revenue gain is increased as compared to local resource utilization policy.

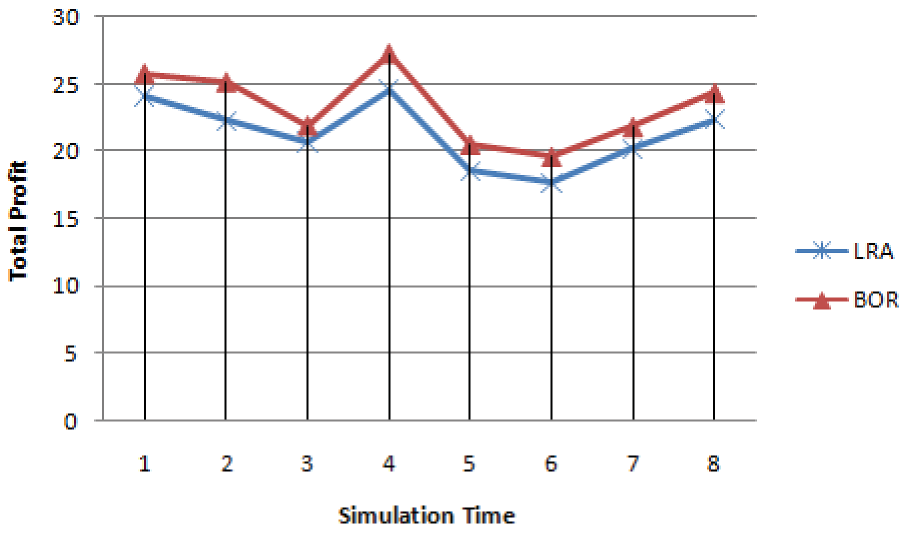


Fig. 5.14: Policy Comparison for 8 hour Simulation

Simulation results suggest that the policy results in 10-15% profit gain in comparison to local resource utilization. Figure 5.15 presents total number of rejected job requests during 8 hours of simulation. As the number of jobs being rejected increases for LRA, the chance increases to outsource such jobs for BOR, if resources can leased from foreign cloud providers based on the particular QoS requirements.

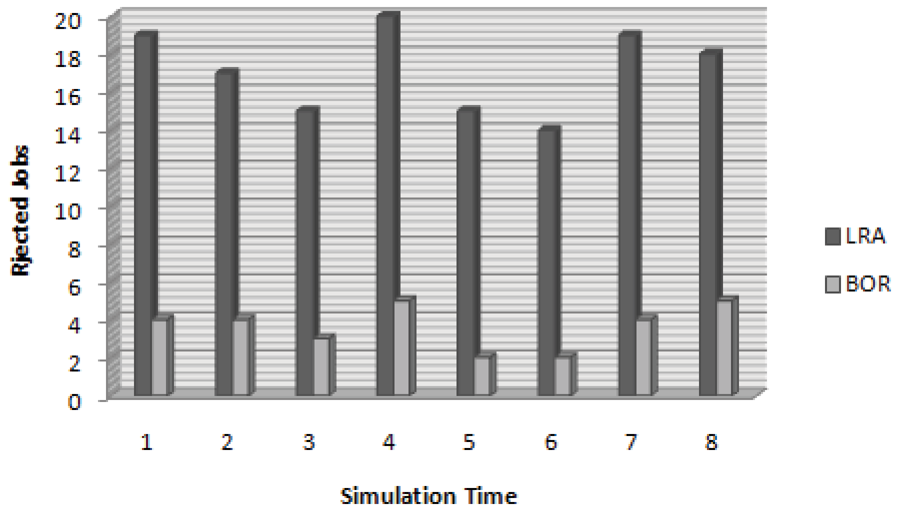


Fig. 5.15: Rejected Job Comparison of Two Polices

The workload pattern, given in figure 4.7, was used as a baseline model for simulation experiment of one week. The workload was sliced in 24 parts for each day to simulate and log experimentation data for synthesizing overall data. At the end, normalized statistics for each day was recorded for both policies which are given in figure 5.16.

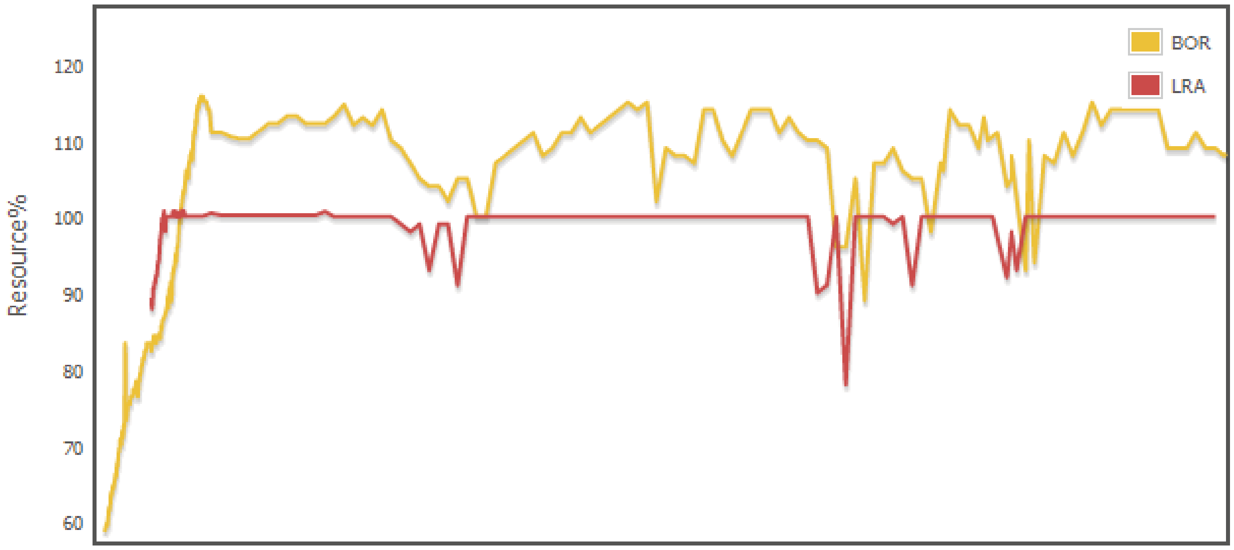


Fig. 5.16: Overall Resource Utilization of LRA and BOR Policies

Since load resource demand was anticipated at the start, both policies showed the same trend during simulation. However, BOR managed better resource demands on one hand and resulted in gaining users' trust by avoiding SLA violation on the other hand.

Initially the load patterns produced same results the two policies. However, OORA showed tendencies to accept more workload during the peaks and gain better economic efficiency. Furthermore, it was also able to gain users' trust and avoid SLA violations. Table 5.11 synthesizes simulation results for one week of simulation.

Table 5.11: Simulation Results of 7 days Experiments

Total Workload for 7 days: 9825 VM Requests

Policy	Requests Completed	Requests Rejected	Requests Outsourced	Job Failures	SLA Penalty ($)	Total Profit ($)
BOR	9424	401	1354	15	3.5	2156.31
LRA	7973	1746	----	106	24.38	1944.62

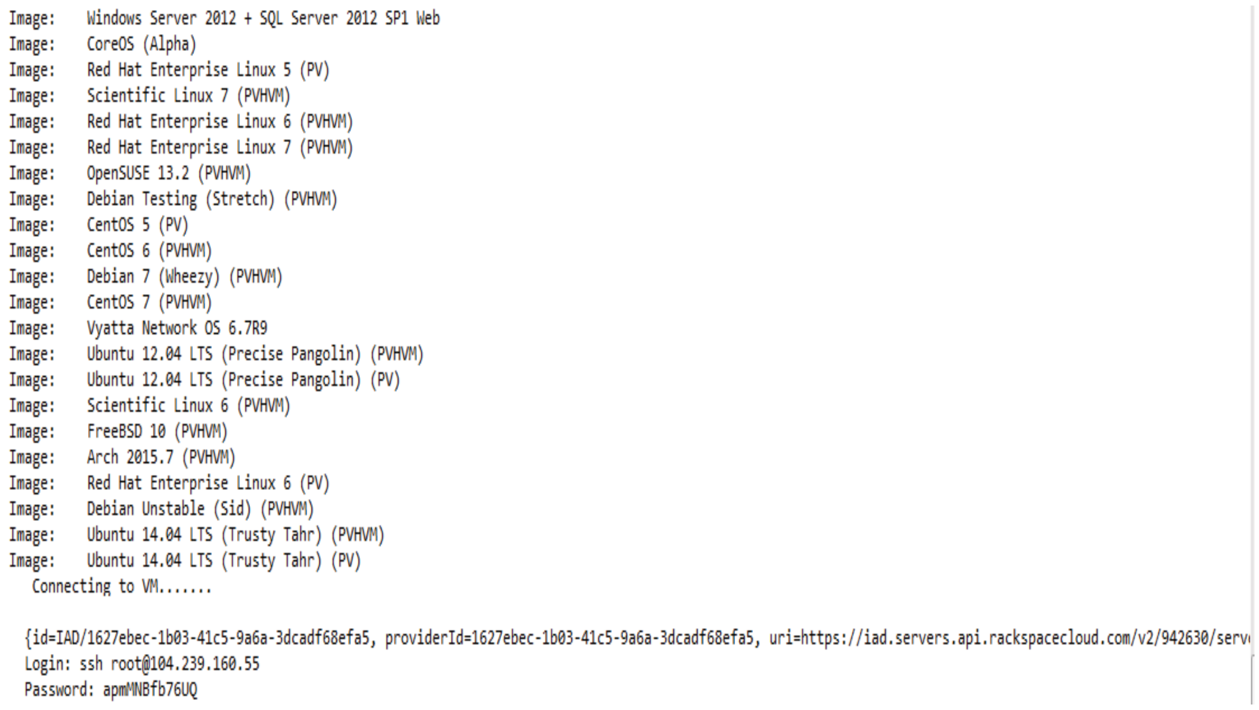
As the result suggest, BOR not only gained higher profit margin but also attained more user satisfaction in terms of lower job rejection and failure rate. On the other hand, LRA policy may discourage cloud users because of higher rejection and SLA violations due to job failures and hence the reputation of such cloud provider would be affected for its regular consumers.

5.4 JClouds Testbed Results

Deployment of virtual machines across multi-cloud is a challenging issue for cloud developers as every cloud vendor offers its own set of libraries, APIs and SDKs. One method to address this issue is to use a portable library such as JClouds which currently supports uniform access to 30 IaaS cloud providers by encapsulating functionality in a level of common abstraction. All the experiments were conducted on three cloud providers- Rackspace, Google Compute and Amazon Elastic Compute Cloud. For most of these experiments, United States based cloud data centers were utilized. For the two case studies, Amazon A15 and R7.5GB standard instance were selected respectively for experimentation purposes while resource outsourcing was performed whenever required.

Although JClouds provides a common API for all cloud providers, but the connectivity configuration differs in case of cloud providers. Rackspace instance creation is much simpler than the other two cloud providers as necessary user credentials are the only requirements for VM provisioning and deployment. Figure 5.17 presents JClouds log of a Rackspace VM instance, being created in *rackspace-cloudservers-us*. Upon the successful completion of the process, linux root login details and password were returned. The 64 bit Ubuntu 14.04 LTS (Trusty Tahr) VM machine was created using the following hardware profile:

compute1-8, name=7.5 GB Compute v1, location={scope=ZONE, id=IAD, description=IAD, parent=rackspace-cloudservers-us, iso3166Codes=[US-VA]}, processors=[{cores=4.0, speed=1.0}], ram=7680, volumes=[{type=LOCAL, size=0.0, bootDevice=true, durable=true}], supportsImage=Predicates.alwaysTrue()}



```
Image:    Windows Server 2012 + SQL Server 2012 SP1 Web
Image:    CoreOS (Alpha)
Image:    Red Hat Enterprise Linux 5 (PV)
Image:    Scientific Linux 7 (PVHVM)
Image:    Red Hat Enterprise Linux 6 (PVHVM)
Image:    Red Hat Enterprise Linux 7 (PVHVM)
Image:    OpenSUSE 13.2 (PVHVM)
Image:    Debian Testing (Stretch) (PVHVM)
Image:    CentOS 5 (PV)
Image:    CentOS 6 (PVHVM)
Image:    Debian 7 (Wheezy) (PVHVM)
Image:    CentOS 7 (PVHVM)
Image:    Vyatta Network OS 6.7R9
Image:    Ubuntu 12.04 LTS (Precise Pangolin) (PVHVM)
Image:    Ubuntu 12.04 LTS (Precise Pangolin) (PV)
Image:    Scientific Linux 6 (PVHVM)
Image:    FreeBSD 10 (PVHVM)
Image:    Arch 2015.7 (PVHVM)
Image:    Red Hat Enterprise Linux 6 (PV)
Image:    Debian Unstable (Sid) (PVHVM)
Image:    Ubuntu 14.04 LTS (Trusty Tahr) (PVHVM)
Image:    Ubuntu 14.04 LTS (Trusty Tahr) (PV)
   Connecting to VM.......

  {id=IAD/1627ebec-1b03-41c5-9a6a-3dcadf68efa5, providerId=1627ebec-1b03-41c5-9a6a-3dcadf68efa5, uri=https://iad.servers.api.rackspacecloud.com/v2/942630/serv
  Login: ssh root@104.239.160.55
  Password: apmMNBfb76UQ
```

Fig. 5.17: JClouds log for Rackspace VM Connection

However, VM deployment for Amazon and Google Compute involve few other security mechanisms. For instance, a RSA private key *pem* file is required to Amazon VM instance deployment. Although this configuration is a well suited for a centralized VM placement scenario but it may not be applicable in distributed broker based resource provisioning framework. To address this issue, a small shell script (Appendix A) was executed to override the default security mechanism by authenticating user based on security credentials instead of private key certificate. The security credentials were then provided to the coordinator of home cloud provider for resource outsoaring. Furthermore, since each cloud provider impose necessary restrictions on it users, for instance deployment of only 15 instances of standard 1GB VMs are possible on a Rackspace account, multiple accounts were used to validate the designed experimentation model.

Although monitoring tools are nowadays provided with cloud instances, however, considering the unique and particular requirements of the monitoring system where each and every violation had to be recorded and matched against SLOs, a small script was developed to monitor VM performance during experimentation. The script was comprised of Linux utilities such as VmStat, Iostat, iftop, Collectl and so on to collect necessary performance and usage statistics. The script was prely installed on outsourced VMs, before provisioning the VMs to home cloud provider.

5.4.1 Case Study I: Cross-Species Sequence Comparisons

Bioinformatics is an emerging interdisciplinary field of scientific research where the statistical, mathematical, engineering and computer science models are used to analyze biological data. Genetic similarities and differences can be compared using different bioinformatics methods. One of the common use of bioinformatics techniques is to identify candidate genes and nucleotides to better understand the genetic basis of disease, desirable properties and nucleic acid and protein sequences [153]. DNA sequencing is made with aim of determining the precise order of nucleotides within a DNA molecule. Each DNA strand is composed of four bases: adenine, guanine, cytosine, and thymine. To determine the precise order of four bases with DNA, different methods and tools can be used to identify precise location and sequences of genomes of numerous types and species of life. DNA sequencing may be used to determine the sequence of individual genes, larger genetic regions such as clusters of genes or operons, full chromosomes or entire genomes. This information is useful to various fields of biology and other sciences, medicine, forensics, and other areas of study. DNA sequencing techniques have been applied in numerous fields including molecular biology, evolutionary biology, metagenomics, medicine and forensics [154].

Knowledge about the sequence of genes in different species at varying evolutionary distances helps to identify coding and functional non-coding sequences among organisms as well as sequences that are unique for a given organism. For instance, many of the homo sapiens (Human) genes are identical or similar to those found in other species. The comparison process requires at least two DNA sequences, related by convergent evolution or divergent evolution from a common ancestor to determine the level of similarity of two sequences. Hence, cross-species sequence comparisons is a useful technique to understand genome, both in similarity and differences, to study changes in human genomes in different diseases [155], [156]. The flow of events for cross-species sequence analysis is presented in figure 5.18.

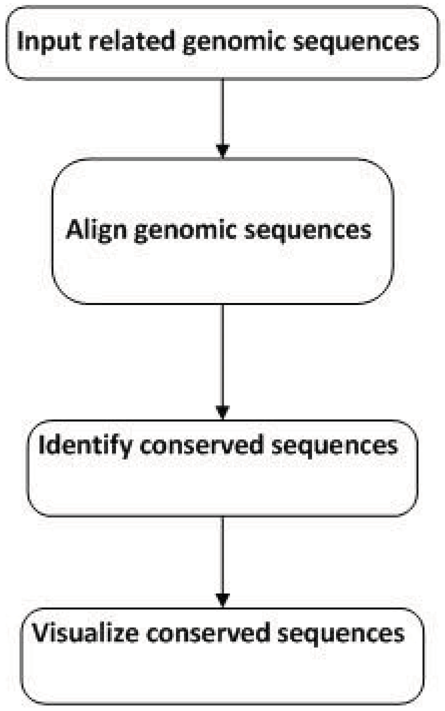

Input related genomic sequences
Align genomic sequences
Identify conserved sequences
Visualize conserved sequences

Fig. 5.18: Cross-species Sequence Analysis Process, adapted from [155], [156]

DNA sequencing is an essential resource to study health and genetic diseases in human being. Two large genomes that have been sequenced with the human reference genome are zebrafish and mouse genome. Zebrafish are biologically much similar to human genes which make them a reference study model to understand genes role in different human disease. According to a scientific study, automatic and manual annotation for more than 26,000 protein-coding genes showed 70% of identical similarity with human protein genes to zebrafish protein genes while 84% genes associated with human diseases have a counterpart in zebrafish [157].

Different studies have revealed that zebrafish may spontaneously develop any type of tumor known from human with similar morphology [158]. The research in genetic similarity is advancing the understanding of muscle and organ development. Zebrafish genes have been used to assert the evolution and formation of causal gene in muscular dystrophy disorders. These are also helpful in for modelling paediatric diseases in 80% of cases by altering the activity of genes in zebrafish embryos [159]. However, the process will take several years to read through the entire human genome to locate the sequence of genes in other species which is impractical. Hence, sophisticated tools and technological methods are required to carry out this process.

A useful bioinformatics tool in this area of research is BLAST+ (Basic Local Alignment Search Tool) [138], to search entire genomic libraries for identical or similar sequences. The software can be used as either a command line standalone version for local alignment search or by using the online version of the tool. Considering the nature of workload for two genomes with a large sequences, the search process may be much computationally expensive. For instance, On a Core i-5 processor with 4 GB RAM, the similarity search for two small sized genomes, zebrafish.1.protein.faa and mouse.1.protein.faa, may take computational time up to 15 hours. This computational delay is not affordable in bioinformatics and genetic engineering applications where organism's genomes are continuously manipulated. With the introduction of cloud computing environment in scientific applications, single instance based BLAST cloud AMI has been introduced by three cloud providers: Amazon Web Services (AWS), Google Compute Engine (GCE), and Microsoft Azure. The purpose of this case study was to distribute sequence searches across multiple instances using the designed framework, which is not currently supported in the cloud environment.

5.4.1.1 Experimentation Procedure

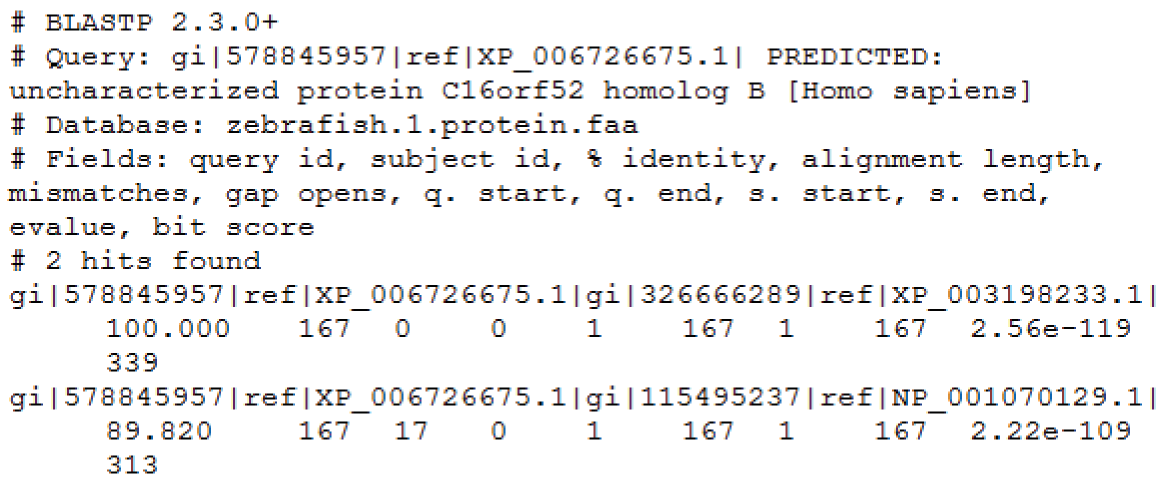
For the experimentation purposes, VM instances of 64-bit Amazon m3.xlarge (Amazon A15) BLAST AMI hosted on Ubuntu 12.04 were used as a reference cloud provider VMs. BLAST+ server images for Google Compute Engine were leased, when required. Since dedicated instances for BLAST+ package are not installed on Rackspace servers, these instances were pre-installed and configured with necessary BLAST+ packages and tools.

Each search query consisted of hundreds of FASTA sequences of zebrafish and mouse proteins that were compared with human protein structure. These query files were hosted on a FTP server and were parsed by GNU parallel [139] to create a parallel pipeline based on the number of available virtual machines. A simple FASTA file with two sequences is presented in the figure 5.19.

```
>gi|51467976|ref|NP_001003855.1| alpha-(1,6)-fucosyltransferase [Danio rerio]

MRPWTGSWRWIALVLLAWGTLLFYIGGHLVKDSEHAPRSSRELAKILTKLERLKQQNEDLRRMAQSLRIPE
GQSDGPISS

GRLRSLEEQLSRAKQKIQSFQRLSGEGPGKDHEELRRKVENGVRELWYFVRSEVKKLPLMETGAMHKHVD
TLMQDLGHQQ

RSVMTDLYHLSQADGAGDWREKEANELSDLVQNRIMYLQNPQDCSKARKLVCNINKGCGYGCQLHHVV
YCFMIAYGTQRT

LILESQNWRYATGGWETVFKPVSDTCTDRTGASTGHWSGEAHDRDVQVVELPIVDSLHPRPPYLPLAVPE
DLAPRLQRLH

GDPSVWWVSQFVKFLVRPQAWLEKEIQETCLKLGFKHPIIGVHVRRTDKVGTEAAFHPIEEYMVHVEDHY
QSLAQRMHVD

KKRVYLATDDPSLLQEAKTKYPDYEFISDNSISWSAGLHNRYTENSLRGVILDIHFLSRTNYLVCTFSSQVC
RVAYEIMQ

TLHPDASSYFYSLDDIYYFGGQNAHNQIAIYPHQPRNSDDIPLEPGDVIGVAGNHWDGYSKGINRKTGRTG
LYPSYKVKE

KIETVKYPTYPEADKLLKKP

>gi|130487273|ref|NP_001076269.1| leucine-rich repeat-containing protein 30 [Danio rerio]

MCSKLEVLSLANNHLTGLPASLSALVGLKKLNLSHNNITHIPGCVYTMRNLVFLQLACNNLENIADQIQAL
TDLKILIVE

GNCIHSLPKMLCCLTKLELLNVDFNDIQNVPAEMHKLKRLEKLACHPLDKGLHIMHNPLLKPIKEVLDGGL
QALYCYLKAT
```

Fig. 5.19: FASTA File with Two Sequences

In figure 5.19, first FASTA query of 580 characters represents amino acid molecule type of alpha-(1,6)-fucosyltransferase. The best identical match of this sequence with other species is Cyprinus carpio fish with 100% similarity value. However, the FASTA sequence of zebrafish.1.protein with human.1.protein.faa query returned very interesting pattern:



```
# BLASTP 2.3.0+
# Query: gi|578845957|ref|XP_006726675.1| PREDICTED:
uncharacterized protein C16orf52 homolog B [Homo sapiens]
# Database: zebrafish.1.protein.faa
# Fields: query id, subject id, % identity, alignment length,
mismatches, gap opens, q. start, q. end, s. start, s. end,
evalue, bit score
# 2 hits found
gi|578845957|ref|XP_006726675.1|gi|326666289|ref|XP_003198233.1|
     100.000    167   0     0     1     167   1     167   2.56e-119
     339
gi|578845957|ref|XP_006726675.1|gi|115495237|ref|NP_001070129.1|
     89.820     167   17    0     1     167   1     167   2.22e-109
     313
```

Fig. 5.20: FASTA Sequence Comparison of zebrafish.1.protein with human.1.protein.faa

Homologs of the C16orf52 gene show that this gene is conserved in chimpanzee, Rhesus monkey, dog, cow, mouse, rat, chicken, zebrafish, fruit fly, mosquito, and frog as well as human. Table 5.12 shows total number of sequences used during the experimentation. It is worth mentioning that a subset of protein databases was used as complete database search was not feasible at this stage.

Table 5.12: Total Number of Sequences

Species	Number of Sequences
Human	24, 534
Zebrafish	18,314
Mouse	22, 802

Each experiment consisted of FASTA sequence comparison of the zebrafish and mouse genes against human genes to construct region of similarity in cross species. The experimentation was aimed at reducing overall searching time for query sequences so cost and other factors were ignored. Since FASTA sequences are stored in a text format, the cloud coordinator of reference cloud provider (Amazon in this case) parallelized the process by splitting the search query into a chunk of 200K FASTA file to distribute the workload across multiple VM instances. These query files were then pipelined into parallel to speed up the execution process. Each input job file was assigned to a single core of worker virtual machines. Overall flow of events is presented in figure 5.21.

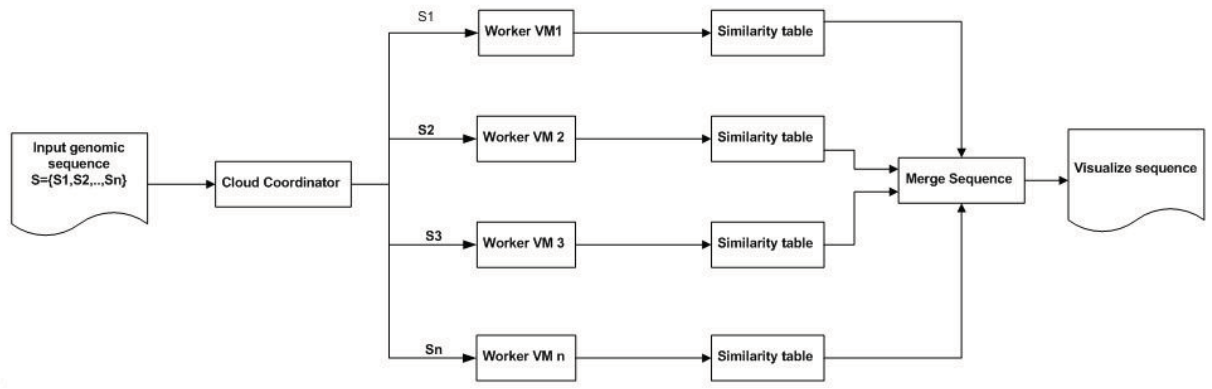


Fig. 5.21: Bio-sequence Analysis Parallelization Pipeline for BLAST+

The SLA, as shown in table 5.13, was used to enforce service level objectives (SLOs) including availability, incoming bandwidth, outgoing bandwidth, CPU performance and storage.

Table 5.13: SLA Monitor settings for BLAST+

SLA Objective	**Value**	**Threat Threshold**
Availability	≥99.9%	99.9%
Memory	≥4GB	----
CPU Power	≥92 % (broker unit)	90%
Storage	≥10GB	
Incoming Bandwidth	≥400 Mbit/s	300 Mbit/s
Outgoing Bandwidth	≥100 Mbit/s	70 Mbit/s

SLA settings suggest that a single failure of virtual machine may result in resource switchover as real time applications may not afford resource unavailability during their execution. For the set of experiments, ten local Amazon instances were provided while five instances from other two cloud providers could be leased based upon the workload and job completion deadline. Table

5.14 shows the experimentation results of a single experiment which clearly depicts that total execution time was reduced when more virtual machines were utilized. With one VM instance of Amazon xlarge, the experimentation process was completed in around 8 hours while with 18 virtual machines, the experimentation time was reduced to around 44 minutes. For the comparison purposes, two types of experiments were performed. In the first set of experiments, referred as local cloud completion, local in-house resources were utilized and the experiments were restricted to total number of available BLAST+ instances, which in this case, were a maximum of ten instances. On the contrary, multi-cloud completion time scheme used cloud outsourcing model based on the SLA requirements of the experiments. VM resources, matching SLA requirements, were leased from other two cloud providers and the sequence comparison queries were executed using both in-house and outsourced VMs. Experimentation was repeated for a total of 3 times to calculate average values.

Experimentation results, as shown in table 5.14, show that the designed broker based multi-cloud resource management framework achieved significant performance improvement by speeding up overall job completion time. Only one instance failure on G13GB was recorded for violation of network latency during experimentation and the VM was migrated to available Rackspace instance for switchover. In contrast to other models, such as CloudBLAST [160] or CloudBurst [161], which are Hadoop MapReduce based alignment models and are technology dependant, the designed model achieved much better results by using the designed multi-cloud resource management model.

Table 5.14: Experimentation Results of BLAST+ for two Policies

No. of virtual machines			**BOR Completion Time HH:MM:SS**	**LRA Completion Time HH:MM:SS**
Amazon	**Google**	**Rackspace**		
1	0	0	08:14:22	---------same as BOR----
4	0	0	02:57:38	---------same as BOR----
8	0	0	01:31:18	---------same as BOR----
10	1	1	01:06:36	01:18:51
10	4	4	00:44:15	---------same as above----

The results in table 5.14 also present an interesting picture of parallelism that can be achieved during such experimentations. The resultant job completion curve tend to decrease at a sharp rate when few virtual machines were used but with the higher increment of virtual machine instances, very small increase was observed in parallelism. This was due to the network overhead, involved in the worker virtual machines and the cloud coordinator within inter-cloud and intra-cloud for transfer of query and result files. The execution curve is presented in the figure 5.22.

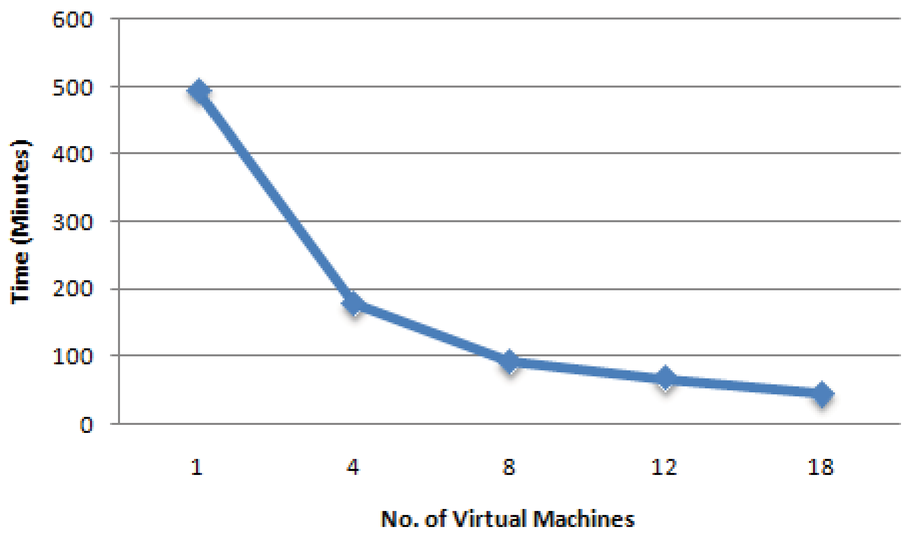


Fig. 5.22: Level of parallelism for different virtual machines

The BLAST+ output files were parsed by individual works and final results were stored and synthesized at cloud coordinator end. The more number of hits shown in the results, the more data sequences are aligned. The e-value confidence was set to 1e-50 to get the high quality similarly aligned search sequences. Finally BLAST+ package tools were used to visualize the similarity results which are shown in figure 5.23. The comparative protein structure distribution which represents total number of hits, as shown in figure, clearly depicts that the three species human, zebrafish and mouse are very much related to each other in protein structure and hence further investigation of similarity can deepen the understanding of muscle and organ development in human being.

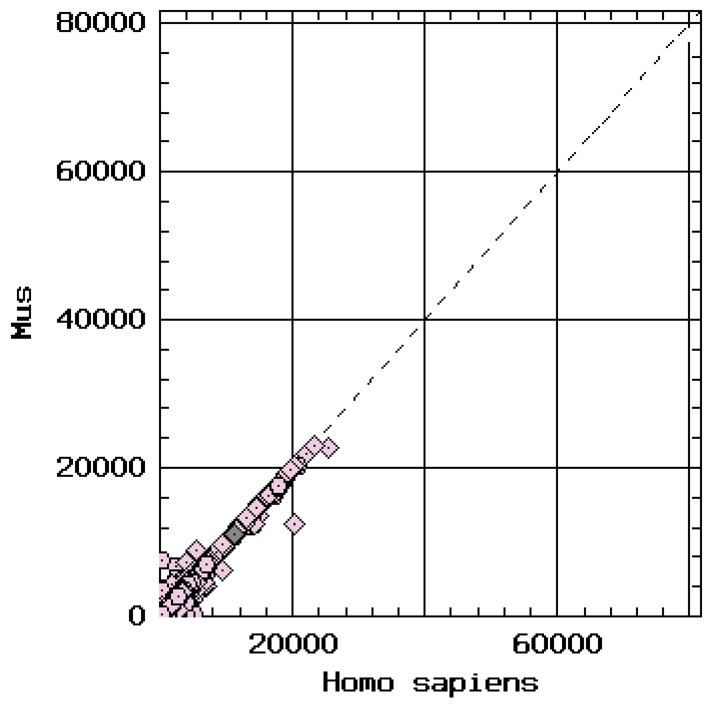


Fig. 5.23: Similarity Distribution of Zebrafish with Human and Mouse

5.4.2 Case Study II: POV-RAY and Wikipedia Dumps

Two datasets for the real test-bed experimentation were used: POV-Ray ray tracing program [162] for CPU benchmarking and Wikipedia dumps [163] for testing network and I/O performance. Several iterations of standard benchmark.pov scene rendering were performed to calculate overall CPU performance. A maximum of ten R7.5GB VM instances were used during the course of experimentation while resources were leased from other cloud providers, if required. Unlike the first case study, this case study was used a stress testing model where virtual machines' performance was expected at their peak and workload demands remained at peaks as given in figure 4.7. Failing to meet the required execution deadline was considered as a job failure and necessary measures were taken to resume the process on other available resources. Experimentation was performed over a span of 4 months starting from April 2015. The output of the standard benchmark is given in figure 5.24.

Fig. 5.24: Render output of benchmark.pov

The input of scene data is the definition of 3D objects, materials, lights and a camera object. One of the major problems in ray-tracing is the large amount of computational time required for the elaboration of the image [164]. To address this issue, parallelization pipeline technique was used by partitioning the scene data into a sub-set of partitions. The coordinator of the home cloud submitted initial scene tasks to every available worker VMs. Worker VMs produced the corresponding scene data and returned the scene data to cloud coordinator which was stored and the worker VM was assigned a new task. At peaks, when the incoming job demand was high, resources were leased from other cloud providers. Based on the SLA of reference cloud provider, best-matched VM instances were selected by the broker for resource outsourcing. In order to evaluate the brokering strategy, service level objectives (SLOs) were used.

As a second experiment, a Java based module was developed to retrieve and parse HTML and XML files from Wikipedia dump. The tool was primarily aimed to test network and I/O performance.

The SLAs for both of the experiments are given in Tables 5.15 and 5.16 including availability, incoming bandwidth, outgoing bandwidth, CPU performance and storage.

Table 5.15: SLA Monitor settings for POV-Ray

SLA Objective	**Value**	**Threat Threshold**
Availability	≥99.5%	99%
Memory	≥4GB	----
CPU Power	≥85 % (broker unit)	82%
Storage	≥10GB	

Table 5.16: SLA Monitor settings for Wikipedia Dump

SLA Objective	**Value**	**Threat Threshold**
Availability	≥99 %	98%
Incoming Bandwidth	≥400 Mbit/s	300 Mbit/s
Outgoing Bandwidth	≥100 Mbit/s	70 Mbit/s
Storage	≥10GB	

The monitoring component of the broker agent was responsible to monitor SLOs. Failing to complete these SLOs resulted in SLA violations and an alarm was triggered by the monitoring component which could, at worse, resulted in resource switchover and the executing tasks were migrated to the candidate cloud instances.

5.4.2.1 Experimentation Results

The primary goal of the experiments was to characterize the relative performance of the proposed broker based QoS enabled resource management framework. This was done by experimenting the jclouds testbed on many different ensembles and comparing the scores computed using the performance metric for execution, SLA violations, job rejection and overall utility in terms of cost and time. Several iterations of the POV-Ray were performed on a series of VM instances and the average execution time was recorded. Figure 5.25 depicts average execution time for 1-6 VM instances which clearly shows a gradual decrease in average VM execution time.

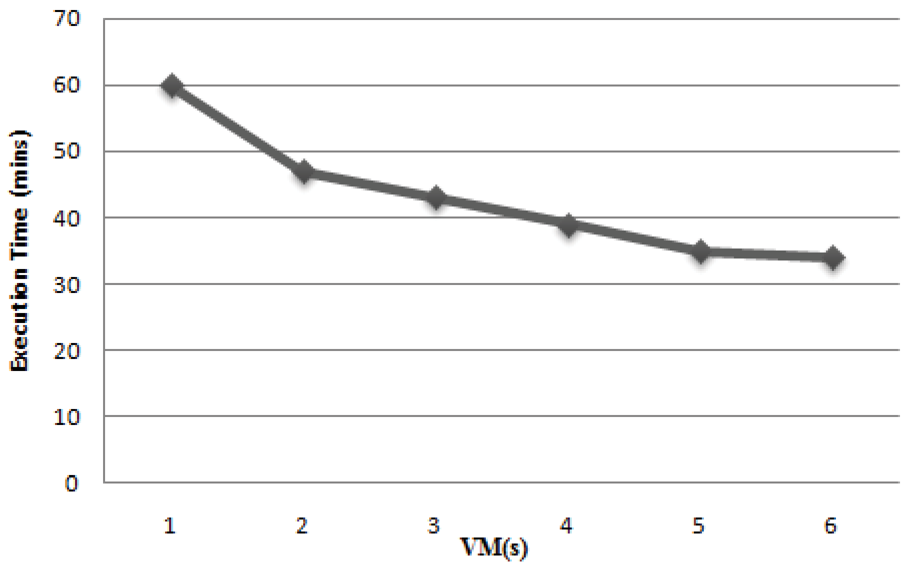


Fig. 5.25: Average Execution Time

Results for execution time also show that reduction in the average time taken for POV-RAY rendering scenario involving more than ten virtual machines. After this available amount of resources, the extra resources enable more requests to be served, however they are requested when average execution time is high. The higher parallelism level is achieved with the extra resources leased from other cloud providers in peak times.

Figure 5.26 shows a histogram of execution time for 140 VM instances where 17 VM instances were leased from Amazon and only 4 VM instances were requested from Google. The frequency distribution clearly depicts that Amazon VM instances outperformed other instances based on execution time and hence were mostly utilized, whenever required.

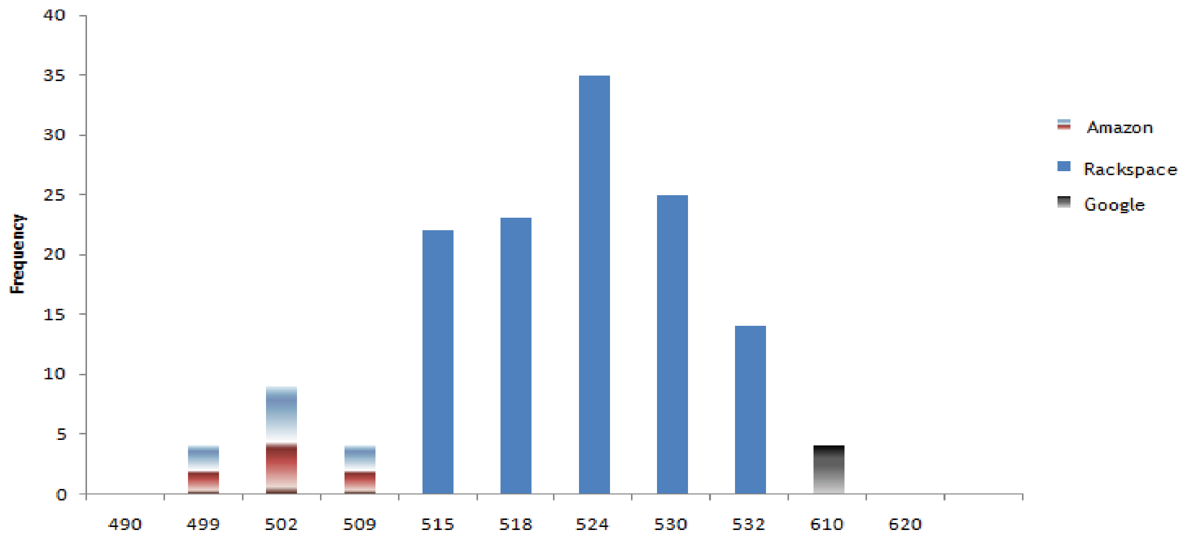


Fig. 5.26: Frequency of POV-RAY job completion for different cloud providers

Table 5.17 summarizes overall experimentation results for two datasets. Very few SLA violations were recorded when resources were outsourced to Amazon VM instances. However, Google SLA violations were relatively higher, majorly resulted due to availability and deadline issues.

Table 5.17: Total SLA Violations

	Iterations **No. of Experiments**	**24** **60**	**36** **50**	**48** **40**	**60** **30**	**72** **20**
POV-RAY No. of SLA Violations						
Amazon		2 (5%)	1 (4%)	3 (7%)	3 (5%)	4 (8%)
Google		1 (25%)	2 (15%)	2 (15%)	3 (10%)	5 (12%)
Wikipedia Dump No. of SLA Violations						
	Iterations **No. of Experiments**	**24** **20**	**36** **18**	**48** **16**	**60** **14**	**72** **12**
Amazon		0	0	0	0	0
Google		0	0	0	2 (7%)	2 (7%)

Table 5.18 presents experimentation results for four month setup. Very few job requests were turned down by the designed broker enabled QoS based resource provisioning policy and hence the policy resulted better, both in terms of SLA violations and economic gains, with no compromise on performance. However, for local isolated resource provisioning scheme, job rejection rate was much higher which resulted in relatively much less revenue.

Table 5.18: Comparison of Proposed Broker vs. Local Resource Utilization Policies

Scenario	Policy	Total Iterations	Failed	Revenue ($)
POV-RAY	**BOR**	8400	52	1814
	LRA		1826	1512
Wiki Dump	**BOR**	3600	23	774
	LRA		612	687

The performance comparison results of the two policies are presented in table 5.19.

Table 5.19: POV-RAY Performance Results for Two Policies

Metrics	BOR	LRA	Improvement
Execution Time (mins)	358	610	42%
Execution & Scheduling Time (mins)	424	812	47%

Based on the results from tables 5.18 and 5.19, it can be safely concluded that the designed BOR framework not resulted in higher gain in profit but also achieved faster execution time as compared to the base line isolated LRA policy. The testbed results compliment the simulation results which clearly indicate that by using the designed model, cloud providers in a federation can utilize the resources at maximum without risking their reputation for the varying nature of demands from their clients. Economic efficiency, which is the basic building block of cloud computing model, is also achieved by providing a resource sharing platform based on particular QoS requirements for better resource utilization.

5.5 Comparison with State of the Art Implementations

In this section, we provide a brief state-of-the-art about the proposed generic broker based QoS ranking and resource selection framework. Over the last few years, many researches have been conducted in the field of VM placement across multiple clouds for efficient resource management. However, the majority of them deal with these aspects separately such as the issues like negotiation [24], [25], provisioning of resources [26], [27] and resource monitoring [28] are addressed in isolation and a comprehensive broker based framework for job placement seems to be an important area of ongoing research.

A generic architecture for a Cloud service broker operating in an Inter-cloud environment is proposed in [40]. The primary goal of the broker is to select the most suitable cloud provider that satisfies the QoS aspects of user jobs and provide a uniform interface to manage and monitor deployed services. There key points that make this work different from our solution are as follows: Most of the cloud vendors guarantee performance of compute metrics at hypervisor level and these metrics are not reflected in service level agreements. Instead of providing a theoretical model for SLA matching such as proposed in [40], we provide a comprehensive match making process by applying necessary benchmarking procedure to reflect near to optimal QoS aspects for cloud ranking. Secondly, instead of just simulating the evaluation results on a cloud simulator, we evaluated the cloud performance results on three real cloud providers which present real world evaluation results. Thirdly, SLA schema for QoS metrics, price-performance analysis and monitoring SLA violations are few other exceptions that are not addressed in [40].

A prototype implementation of a cloud coordinator which envisions a marketplace that enables brokers and providers to improve performance, reliability, and scalability of elastic applications by leveraging resources from multiple Clouds in order to seamlessly meet applications' SLAs by scaling them across various data centers is proposed in [165]. The architecture and design of the coordinator is primarily focused on speeding up the application execution time and enhancing the cloud capacity to meet shorter deadlines with the use of inter-cloud resources. However, their work is focused on deployment of virtual machines and the strategies for job management, monitoring and performance evaluation are not part of the cloud coordinator.

Chapter 6: Conclusion and Future Work

6.1 Conclusion

Quality of Service is a broad topic in the field of computer networks and distributed systems. Although various QoS standards are available for other fields of computer networks, much research is still needed to define QoS standards and metrics in the field of cloud computing. Most of the contemporary researches deal with cloud providers operating in isolation which may introduce numerous challenges. The issue of VM placement across multiple clouds was addressed in this research to provide a resource sharing platform where the objectives of minimizing resource rejection and efficient service level agreement management were incorporated in the designed framework. The aim of this research study was to improve the decision accuracy, while choosing a cloud provider for a given set of user QoS preferences. The main objective of this research work was the development and evaluation of a QoS based ranking framework for IaaS computing resources across multiple clouds for resource negotiation, provisioning of physical resources, monitoring and ranking, based on job execution experience.

Cloud providers were assessed based on the performance evaluation and best-fit selection was made to meet service level objectives. Results proved that broker based cloud federation framework provided a win-win scenario for both cloud providers and consumers by avoiding job denial and achieving better SLA management and resource utilization. The designed framework also provided efficient job management among clients so the rejection of the incoming request from the clients and SLA violations was minimal.

6.2 Contributions

The contributions made in this thesis primarily include the introduction of a new framework for provisioning of optimal resources across multiple clouds. This was achieved through a novel broker based QoS ranking and resource selection framework to address the issues of negotiation, provisioning, monitoring and ranking of physical resources based on job execution experience. To the best of our knowledge, this is the first effort in this domain that addresses these issues by proposing a framework for QoS based ranking in federated cloud, introduces a new schema for defining SLOs, investigates cloud price/ performance analysis and outlines the strategies for resource monitoring and switch-over to meet SLA constraints. This dissertation has contributed to the theory and practice of cloud computing by proposing the following:

1. The definition of the SLA schema and SLA service management model to represent QoS aspects of service level objectives.

2. The development of SLA resource management module that also supports monitoring and resource migration capabilities.

3. A price/ performance oriented QoS based ranking model to select the right cloud provider that may meets user requirements based on service level objectives.

4. Investigation of cloud performance metrics by thoroughly benchmarking three different cloud providers to evaluate QoS based performance patterns of different cloud vendors.

5. Better resource selection and allocation strategies to meet user requirements and job deadlines while optimizing time, cost and resource selection constraints.

The benefits resulting from the developed federated broker include:

- Provisioning of optimal software and hardware resources to achieve the best performance at lower cost
- Efficient and scalable so that multiple cloud datacenters be utilized to provide uninterrupted services of cloud platforms
- Flexibility in assigning QoS metrics to user jobs to better adjust cost, price and quality tradeoffs

The proposed framework was validated by implementing a software prototype. The references implementation was tested using two real world case studies involving computationally intensive processing for performance analysis of the proposed framework. The outcomes of the study reveal that the job execution and management can significantly be improved while preserving the QoS at an optimal level.

6.3 Limitations

The thesis has presented the suggestions on provisioning of optimal software and hardware cloud computing resources to ensure a better QoS, whereby a framework for ranking cloud providers and monitoring SLA violations was developed. The proposed research model is validated by using two general case studies. Since these two case studies cannot fully cover every aspect of scientific and business applications, we cannot make any safe assumptions on the validity for other types of user applications.

The research work presented in this dissertation is also subject to certain limitations and assumptions. Since time was a major constraint, limiting some planned research activities which are outlined in the future work section.

While the dynamic nature of cloud computing comes with great benefits, it may also pose certain challenges for cloud resource broker to ensure availability and performance of deployed user services. It will be valuable to extend the proposed framework by introducing adaption rules for price changes and performance degradation at runtime with or without user involvement for enforcing the SLA. Likewise, cloud reputation ratings could be further examined to analyze their effect on different aspects of the system. Additionally, further work could be carried out to evaluate the overheads surrounding other popular IaaS solutions such as 1 &1 Cloud, CenturyLink and Microsoft Azure.

In addition to the QoS and pricing considerations, the match-making process could be further enhanced to incorporate further details about network topology and memory effects in SMP environments (effects of shared cache and memory buses) as these properties heavily influence the data transfer performance. Besides that, strategies to predict application needs in terms of computing resources can be investigated to meet a predefined deadline.

Different benchmarking tools were used to cover QoS aspects of multiple cloud providers. These tools may not be sufficient to cover all possible patterns of overhead involved in virtualization platform. Further research is required to study micro-benchmarks that are particularly aimed at evaluating cloud performance metrics. The experiments presented in this study only cover a subset of performance related virtualization properties. Other performance-relevant properties such as fine-grained resource allocation using caps, shares or priorities might be an interesting area of research in this particular domain. Finally, an in-depth survey of the root causes of many of the performance related issues discovered in the study results could ascertain specific improvements, needed to be made by cloud providers to enhance QoS specific aspects.

As a final assumption, we do not argue that our model and framework for resource management are complete. We are however confident that in future work we can evaluate the usability of our proposed broker based framework in different scientific applications.

6.4 Future Work

Although the research study aids IaaS resource providers and consumers by maximizing profit and trust as well as provisioning of best-matched resources, there are still further areas to explore in future:

- In the designed model, resource switchover takes place and the VM is migrated to the candidate cloud provider whenever any serious violation of SLA is occurred. However, current state of executed tasks are not saved and hence execution is retrieved from the beginning rather than restarted from the last stable checkpoint. Fault tolerance mechanism can be incorporated in the proposed model for better workload and deadline management.

- Strategies for future workload prediction could be integrated in the designed model to predict the future workload requirements of the cloud users. The two evaluation strategies are calculating profit based on incoming job requests and these are unable to predict future job requirements. An adaptive workload prediction model [166], integrated with the designed framework can offer better resource provisioning and management mechanism.

- The primary objective of global cloud federation marketplace is to provide a resource sharing platform for profit maximization of individual clouds by maximum utilization of resources. However, the decision about how and when a cloud provider should trade resources with other cloud providers to maximize net profit in long run is dynamic optimization problem and needs further research.

Appendix-A

Algorithm Poisson_Distribution (Knuth)

```
init:
        L ← $e^{-\lambda}$
        k ← 0
        p ← 1.

do:
        k ← k + 1.
        Generate uniform random number u in [0,1]
        p ← p × u.
while p > L.
return k − 1.
```

Script to authenticate user based on security credentials

```
#!/bin/bash

Output='/etc/ssh/sshd_config'
Expression=('s/^#PermitRootLogin yes/PermitRootLogin no/' 's/PermitRootLogin yes/PermitRootLogin no/' 's/^#PermitEmptyPasswords yes/PermitEmptyPasswords no/' 's/PermitEmptyPasswords yes/PermitEmptyPasswords no/' 's/^#PasswordAuthentication yes/PasswordAuthentication no/' 's/PasswordAuthentication yes/PasswordAuthentication no/' 's/^#X11Forwarding yes/X11Forwarding no/' 's/X11Forwarding yes/X11Forwarding no/')


for i in "${ Expression [@]}"
{
   sudo sed -i "$i" $ Output
}
```

References

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

[2] L. Wang, G.V. Laszewski, M. Kunze, J. Tao. *Cloud computing: A perspective study*. in *Grid Computing Environments (GCE) workshop*.2008.

[3] *The NIST definition of cloud computing, recommendations of the National Institute of Standards and Technology*. [cited July 05, 2014], Available from: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[4] R. Buyya, C.S. Yeo, S. Venugopal. *Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities*. in *HPCC. 10th IEEE International Conference on High Performance Computing and Communications* pp. 5-13,2008.

[5] *Cloud Computing*. Accessed: Nov 20, 2014, Available from: http://en.wikipedia.org/wiki/Cloud_computing.

[6] N. Turner. *Cloud computing: a brief summary. Lucid Communications Limited*. Available from: http://www.lucidcommunications.co.uk/Content/whitePapers/CloudComputing.pdf.

[7] *Cloud computing*. [cited August 20, 2014], Available from: http://www.explainthatstuff.com/cloud-computing-introduction.html.

[8] D.J. Armstrong, *Enhancing quality of service in cloud computing through novel resource management*. 2012, University of Leeds).

[9] *Amazon Web Services*. Accessed: Jule 20, 2014, Available from: http://aws.amazon.com.

[10] *Enomalism Virtualized Management Dashboard* Available from: http://www.virtustream.com.

[11] *Eucalyptus - Elastic Utility Computing Architecture*. Accessed: November 12, 2014, Available from: http://www.eucalyptus.com.

[12] *OpenNebula - The Engine for Data Center Virtualization and Cloud Solutions*. Accessed: August 06, 2014, Available from: http://www.opennebula.org.

[13] *Google Cloud Platform*. Accessed: August 04, 2014, Available from: https://cloud.google.com.

[14] *Rackspace Cloud.* Accessed: August 02, 2014, Available from: http://www.rackspace.com/cloud.

[15] *Cloud Harmony Reports.* Accessed: July 22, 2014, Available from: https://cloudharmony.com/reports/editions/state-of-the-cloud-compute/basic/state-of-the-cloud-compute-0714.pdf.

[16] *OpenStack Cloud Computing Platform.* Accessed: July 24, 2014, Available from: http://www.openstack.org.

[17] T. Vondra , J. Sedivy, *Maximizing utilization in private IaaS clouds with heterogenous load through time series forecasting.* International Journal on Advances in Systems and Measurements. **6**(1), pp. 149-165, 2013.

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

[19] *Amazon Cloud Service Goes Down...* Accessed: May 19, 2014, Available from: http://bits.blogs.nytimes.com/2012/10/22/amazon-cloud-service-goes-down-and-takes-some-popular-web-sites-with-it.

[20] *Case study: Amazon outages shake faith in the cloud.* Accessed: June 09, 2014, Available from: http://www.ft.com/cms/s/0/a5cba44c-a264-11e0-9760-00144feabdc0.html.

[21] *Techniques for implementing information services with tentant specific service level agreements US 20140101299 A1.* Accessed: June 13, 2014, Available from: http://www.google.com/patents/US20140101299.

[22] R. Buyya, R. Ranjan, R.N. Calheiros. *Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services*. in *Proceedings of the 10th international conference on Algorithms and Architectures for Parallel Processing*. pp. 13-31,2010.

[23] *Use Cases and Functional Requirements for Inter-Cloud Computing. Technical Report, Global Inter-Cloud Technology Forum, 2010.* Accessed: July 08, 2014, Available from: http://www.ttc.or.jp/files/8614/1214/5480/GICTF_Whitepaper_20100809.pdf.

[24] S. Hudert, H. Ludwig, G. Wirtz, *Negotiating SLAs- An pproach for a generic negotiation framework for WSAgreement.* Journal of Grid Computing. **7**(2), pp. 225-246, 2009.

[25] M. Chhetri, J. Lin, S. Goh, J. Yan, J.Y. Zhang, R. Kowalczyk. *A coordinated architecture for the agent-based service level agreement negotiation of Web service composition*. in *Software Engineering Conference*. pp. 90-99,2006.

[26] A. McCloskey, B. Simmons, H. Lutyya. *Policy-based dynamic provisioning in data centers based on SLAs, business rules and business objectives*. in *Network Operations and Management Symposium*. IEEE. pp. 903-906,2008.

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

[28] O. Rana, M.Warnier, T.B. Quillinan, F. Brazier. *Monitoring and reputation mechanisms for service level agreements*. in *5th international workshop on Grid Economics and Business Models, GECON '08*. Springer-Verlag. pp. 125-139,2008.

[29] A. Lenk, M. Menzel, J. Lipsky, S. Tai, P. Offermann. *What are you paying for? performance benchmarking for infrastructure-as-a-service offerings*. in *International Conference on Cloud Computing*. IEEE. pp. 484-491,2011.

[30] K.R. Jackson. *Performance of HPC applications on the Amazon web services cloud*. in *2nd international conference on cloud computing*. IEEE. pp. 159 - 168,2010.

[31] J. Scheuner, P. Leitner, J. Cito, H. Gall. *Cloud WorkBench- Infrastructure-as-Code based cloud benchmarking*. in *6th IEEE International Conference on Cloud Computing Technology and Science (CloudCom'14)*.2014.

[32] A. Li, X. Yang, S. Kandula, M. Zhang. *CloudCmp: comparing public cloud providers*. in *10th ACM SIGCOMM conference on Internet measurement*. ACM,2010.

[33] H.-K. Kwon , K.-K. Seo, *A Fuzzy AHP based Multi-criteria Decision-making Model to Select a Cloud Service.* International Journal of Smart Home. **8**(3), pp. 175-180, 2014.

[34] A. Rezgui , S. Rezgui. *A stochastic approach for virtual machine placement in volunteer cloud federations*. in *International Conference on Cloud Engineering (IC2E), Boston, USA*. IEEE. pp. 277 - 282,2014.

[35] S.K. Garg, S. Versteeg, R. Buyya, *A framework for ranking of cloud computing services.* Future Generation Computer Systems. **29**(4), pp. 1012-1023, 2013.

[36] M. Salama, A. Zeid, A. Shawish, X. Jiang, *A Novel QoS based framework for cloud computing service provider selection.* International Journal of Cloud Applications and Computing. **4**(2), pp. 48-72, 2014.

[37] V.C. Emeakaroha, M.A. Netto, R.N. Calheiros, I. Brandic, R. Buyya, C.A.D. Rose, *Towards autonomic detection of SLA violations in cloud infrastructures.* Future Generation Computer Systems. **28**(7), pp. 1017-1029, 2012.

[38] J. Tordsson, R.S. Montero, R. Moreno-Vozmediano, I.M. Llorente, *Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers.* Future Generation Computer Systems. **28**(2), pp. 358-367, 2012.

[39] S. Chaisiri, B.S. Lee, D. Niyato. *Optimal virtual machine placement across multiple cloud providers*. in *Services Computing Conference, APSCC*. IEEE. pp. 103-110,2009.

[40] F. Jrad, J. Tao, A. Streit, *SLA based service brokering in Intercloud environments.* CLOSER, pp. 76-81, 2012.

[41] F. Faniyi, R. Bahsoon, G. Theodoropoulos. *A dynamic data-driven simulation approach for preventing service level agreement violations in cloud federation*. in *International Conference on Computational Science, ICCS 2012*. pp. 1167-1176,2012.

[42] S. Bardhan , D. Milojicic. *A mechanism to measure quality-of-service in a federated cloud environment*. in *2012 workshop on Cloud services, federation, and the 8th open cirrus summit*. ACM. pp. 19-24,2012.

[43] S.K. Garg, S. Versteeg, R. Buyya. *SMICloud: A framework for comparing and ranking cloud services*. in *Fourth IEEE International Conference on Utility and Cloud Computing (UCC)*. pp. 210-218,2011.

[44] D. Bao, Z. Xiao, Y. Sun, J. Zhao. *A method and framework for quality of cloud services measurement*. in *3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*. IEEE. pp. 358-362

[45] Armstrong, Django, K. Djemame. *Towards quality of service in the cloud.* in *the 25th UK Performance Engineering Workshop*.2009.

[46] *ISO/IEC NP 19086-2 Metrics*. Accessed: July 11, 2014, Available from: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=67546.

[47] T. Guha, *Investigation of service selection algorithms for grid services*. 2009, University of Saskatchewan.

[48] Y. Liu. *QoS Computation and Policing in Dynamic Web Service Selection*. in *WWW2004, New York. IEEE computer Society*. pp. 66–73,2004.

[49] S. Ran, *A model for web services discovery with QoS.* ACM SIGecom Exchanges. **4**(1), pp. 1-10, 2003.

[50] Al-Ghuwairi, Abdel-Rahman, J. Cook, *Modeling and Enforcement of Cloud Computing Service Level Agreements. Technical Report, 2012.*

[51] D. Tsesmetzis, I. Roussaki, E. Sykas, *Modeling and simulation of QoS aware web service selection for provider profit maximization.* Simulation. **83**(1), pp. 93–106, 2007.

[52] L. Zhao, Y. Ren, M. Li, K. Sakurai, *Flexible service selection with user-specific QoS support in service-oriented architecture.* Journal of Network and Computer Applications. **35**(3), pp. 962-973, 2012.

[53] L. Qi, W. Dou, X. Zhang, J. Chen, *A QOS-aware composition method supporting cross-platform service invocation in cloud environment.* Journal of Computer and System Sciences. **78**(5), pp. 1316 -1329, 2012.

[54] L.Wu, S.K. Garg, R. Buyya, *SLA-based admission control for a software-as-a-service provider in cloud computing environments.* Journal of Computer and System Sciences. **78**(5), pp. 1280- 1299, 2012.

[55] S. Gogouvitis, K. Konstanteli, S. Waldschmidt, G. Kousiouris, G. Katsaros, A. Menychtas, D. Kyriazis, T. Varvarigou, *Workflow management for soft real-time interactive applications in virtualized environments.* Future Generation Computer Systems. **28**(1), pp. 193-209, 2012.

[56] J. García, D. Ruiz, A. Ruiz-Cortés, J. Parejo. *QoS aware semantic service selection: An optimization problem*. in *IEEE Congress on Services-Part I*. pp. 384–388,2008.

[57] J. Garcıa, I. Toma, D. Ruiz, A. Ruiz-Cortés. *A service ranker based on logic rules evaluation and constraint programming*. in *2nd ECOWS non-functional properties and service level agreements in service oriented computing workshop*.2008.

[58] J. García, D. Ruiz, A. Ruiz-Cortés. *On user preferences and utility functions in selection: A semantic approach*. in *Service-Oriented Computing - ICSOC 2007 Workshops, ser. Lecture Notes in Computer Science*. Springer Berlin Heidelberg. pp. 105–114,2007.

[59] A. Ruiz-Cortés, O. Martín-Díaz, A. Duran, M. Toro, *Improving the automatic procurement of web services using constraint programming.* International Journal of Cooperative Information Systems. **14**(4), pp. 439–467, 2005.

[60] Z.U. Rehman, F.K. Hussain, O.K. Hussain. *Towards multi-criteria cloud service selection*. in *Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. IEEE. pp. 44-48,2011.

[61] W. Zeng, Y. Zhao, J. Zeng. *Cloud service and service selection algorithm research*. in *GEC '09 Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*. pp. 1045-1048,2009.

[62] S. Sundareswaran, A. Squicciarini, D. Lin. *A brokerage-based approach for cloud service selection*. in *IEEE 5th International Conference on Cloud Computing*. pp. 558-565,2012.

[63] *Microsoft, IBM Surge Ahead of Amazon In Cloud Revenue Growth*. Accessed: August 24, 2014, Available from: http://www.forbes.com/sites/joemckendrick/2014/07/28/ibm-microsoft-surge-ahead-of-amazon-in-cloud-revenues-analysts-estimate/.

[64] *CSMIC, 2011. Service Measurement Index (Version 1.0). Carnegie Mellon University Silicon Valley, Moffett Field, CA, USA*. [cited July 06, 2014], Available from: http://csmic.org/wp-content/uploads/2011/09/SMI-Overview-110913_v1F1.pdf.

[65] Z. Yu , L. Zhang, *QoS-aware SaaS Services Selection with Interval Numbers for Group User.* Journal of Software. **9**(3), pp. 553-559, 2014.

[66] P. Choudhury, M. Sharma, K. Vikas, T. Pranshu, V. Satyanarayana, *Service Ranking Systems for Cloud Vendors.* Advanced Materials Research. **433-440**, pp. 3949-3953, 2012.

[67] Y. Zhang, H. Liu, B. Deng, F. Peng. *A Reliable QoE-aware Framework for Cloud Service Monitoring and Ranking*. in *2013 International Conference on Electrical and Information Technologies for Rail Transportation (EITRT2013)*. Springer Berlin Heidelberg. pp. 401-409,2014.

[68] Z. Zheng, X. Wu, Y. Zhang, M. Lyu, J. Wang, *QoS Ranking Prediction for Cloud Services.* Journal of IEEE Transactions on Parallel and Distributed Systems. **24**(6), pp. 1213-1222, 2012.

[69] J.C. Luo, L. Zhen, G. Lu, L. Zhang, C.-Z. Xu, N. Sun, *CloudRankD: Benchmarking and Ranking Cloud Computing Systems for Data Processing Applications.* Frontiers of Computer Science. **6**(4), pp. 347-362, 2012.

[70] T. Chauhan, S. Chaudhary, V. Kumar, M. Bhise. *Service Level Agreement Parameter Matching in Cloud Computing*. in *2011 World Congress on Information and Communication Technologies (WICT)*. IEEE. pp. 564 - 570,2011.

[71] C. Vazquez, R. Krishnan, E. John. *Cloud Computing Benchmarking: A Survey*. in *International Conference on Grid & Cloud Computing and Applications (GCA 2014)*.2014.

[72] *Benchmarking*. Accessed: May 07, 2014, Available from: https://en.wikipedia.org/wiki/Benchmarking.

[73] M.B. Chhetri, S. Chichin, B.Q. Vo, R. Kowalczyk. *Smart CloudBench-Automated Performance Benchmarking of the Cloud*. in *Sixth International Conference on Cloud Computing*. IEEE. pp. 414-421,2013.

[74] B. Varghese, O. Akgun, I. Miguel, L. Thai, A. Barker, *Cloud Benchmarking For Maximising Performance of Scientific Applications.* IEEE Transactions on Cloud Computing, 2016.

[75] J. J. Emeras, S. Varrette, V. Plugaru, P. Bouvry, *Amazon Elastic Compute Cloud (EC2) vs. in-House HPC Platform: A Cost Analysis.* IEEE Transactions on Cloud Computing, 2016.

[76] Z. Ren, B. Xu, W. Shi, Y. Ren, F. Cao, J. Lin, Z. Ye. *iGen: A Realistic Request Generator for Cloud File Systems Benchmarking*. in *9th International Conference on Cloud Computing*. IEEE. pp. 343 - 350,2016.

[77] Z. Li, L. OBrien, R. Ranjan, M. Zhang. *Early Observations on Performance of Google Compute Engine for Scientific Computing*. in *5th International Conference on Cloud Computing Technology and Science*. IEEE. pp. 1-8,2015.

[78] B. Sun, B. Hall, H. Wang, D.W. Zhang, K. Ding. *Benchmarking Private Cloud Performance with User-Centric Metrics*. in *International Conference on Cloud Engineering*. IEEE. pp. 311 - 318,2014.

[79] K. Ye, Z. Wu, B.B. Zhou, X. Jiang, C. Wang, A.Y. Zomaya, *Virt-B: Towards Performance Benchmarking of Virtual Machine Systems.* IEEE Internet Computing. **18**(3), pp. 64 - 72, 2014.

[80] H. Nawaz, G. Juve, R.F.D. Silva, E. Deelman. *Performance Analysis of an I/O-Intensive Workflow Executing on Google Cloud and Amazon Web Services*. in *International Parallel and Distributed Processing Symposium Workshops*. pp. 535 - 544,2016.

[81] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, A. Klepchukov, S. Patil, A. Fox, D. Patterson. *Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0*. in *Cloud Computing and Its Applications*.2008.

[82] W. Voorsluys, J. Broberg, S. Venugopal, R. Buyya. *Cost of virtual machine live migration in clouds: A performance evaluation*. in *Proceedings of the 1st International Conference on Cloud Computing*. pp. 254-265,2009.

[83] S. Huang, J. Huang, J. Dai, T. Xie, B. Huang. *The HiBench benchmark suite: Characterization of the MapReduce-based data analysis*. in *26th International Conference on Data Engineering Workshops (ICDEW)*. IEEE. pp. 41-51,2010.

[84] Y. Chen, *Workload-Driven Design and Evaluation of Large-Scale Data-Centric Systems*. 2012.

[85] B.F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, R. Sears. *Benchmarking cloud serving systems with YCSB*. in *1st ACM symposium on Cloud computing*. ACM. pp. 143-154,2010.

[86] A.J. Elmore, S.Das, D.Agrawal, A.E. Abbadi. *Zephyr: live migration in shared nothing databases for elastic cloud platforms*. in *proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM. pp. 301-312,2011.

[87] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A.D. Popescu, A. Ailamaki, B. Falsaf, *Clearing the clouds: a study of emerging scale-out workloads on modern hardware.* ACM SIGPLAN Notices. **47**(4), pp. 37-48, 2012.

[88] *SPECjvm*. Available from: www.spec.org/jvm2008.

[89] *Phoronix Test Suite Suites*. Available from: http://openbenchmarking.org.

[90] *dacapo*. Available from: www.dacapobench.org.

[91] S.M. Blackburn, R. Garner, C. Hoffmann, A.M. Khan, K.S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S.Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J.E.B. Moss, A. Phansalkar, D. Stefanovi, T. VanDrunen, D.v. Dincklage, B. Wiedermann, *The DaCapo benchmarks: Java benchmarking development and analysis.* ACM Sigplan Notices. **41**(10), pp. 169-190, 2006.

[92] *IOZone*. Available from: www.iozone.org.

[93] *bonnie*. Available from: www.coker.com.au/bonnie++.

[94] *CacheBench*. Available from: www.icl.cs.utk.edu/projects/llcbench/cachebench.html.

[95] M. Gribaudo, M. Iacono, F. Palmieri, *Performance Modeling of Big Data-Oriented Architectures*. Computer Communications and Networks. 2016: Springer.

[96] G. Mateescu, W. Gentzsch, C. Ribbens, *Hybrid computing – where HPC meets grid and cloud computing.* Future Generation Computer Systems. **27**(5), pp. 440–453, 2011.

[97] *Big Data in the Cloud: Converging Technologies*. [cited 2017], Accessed: August 11, Available from: https://www.intel.com/content/dam/www/public/emea/de/de/documents/product-briefs/big-data-cloud-technologies-brief.pdf.

[98] E.C. Inacio , M.A. Dantas, *A survey into performance and energy efficiency in HPC, cloud and big data environments.* International Journal of Networking and Virtual Organisations. **14**(4), pp. 299-318, 2014.

[99] D. Wu, S. Sakr, L. Zhu, *Big Data Programming Models. In Handbook of Big Data Technologies.* Handbook of Big Data Technologies pp. 31-63, 2017.

[100] S. García, S. Ramírez-Gallego, J. Luengo, J.M. Benítez, F. Herrera, *Big data preprocessing: methods and prospects.* Big Data Analytics. **1**(1), 2016.

[101] *BIG DATA WORKING GROUP Big Data Taxonomy*. 2014.

[102] J. Hurwitz, A. Nugent, F. Halper, M. Kaufman, *Big data for dummies*. 2013: John Wiley & Sons.

[103] G. Hesse , M. Lorenz. *Conceptual Survey on Data Stream Processing Systems*. in *International Conference on Parallel and Distributed Systems*. IEEE. pp. 797-803,2015.

[104] D. Singh , C.K. Reddy, *A survey on platforms for big data analytics.* Journal of Big Data 2. **2**(1), 2015.

[105] S. Landset, T.M. Khoshgoftaar, A.N. Richter, T. Hasanin, *A survey of open source tools for machine learning with big data in the Hadoop ecosystem.* Journal of Big Data. **2**(1), 2015.

[106] C.L.P. Chen , C.-Y. Zhang, *Data-intensive applications, challenges, techniques and technologies: A survey on Big Data.* Information Sciences. **275**, pp. 314–347, 2014.

[107] F. Bajaber, R. Elshawi, O. Batarfi, A. Altalhi, A. Barnawi, S. Sakr, *Big Data 2.0 Processing Systems: Taxonomy and Open Challenges.* Journal of Grid Computing. **14**(3), pp. 379–405, 2016.

[108] M.D.d. Assuncao, A.d.S. Veith, R. Buyya, *Distributed Data Stream Processing and Edge Computing: A Survey on Resource Elasticity and Future Directions*. 2017.

[109] L. Gillam, B. Li, J. O'Loughlin, A.P.S. Tomar, *Fair benchmarking for cloud computing systems.* Journal of Cloud Computing: Advances, Systems and Applications. **2**(1), 2013.

[110] M. Malawski, G. Juve, E. Deelman, J. Nabrzyski. *Cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds*. in *International Conference on High Performance Computing, Networking, Storage and Analysis*. pp. 1–11,2012.

[111] R.V.d. Bossche, K. Vanmechelen, J. Broeckhove. *Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads*. in *IEEE 3rd International Conference on Cloud Computing (CLOUD)*. pp. 228-235,2010.

[112] *Cloud broker*. Accessed: July 11, 2014, Available from: http://searchcloudprovider.techtarget.com/definition/cloud-broker.

[113] S.Selvarani , G.S. Sadhasivam, *A Novel SLA based Task Scheduling in Grid Environment.* International Journal of Applied Information Systems. **1**(1), pp. 14-19, 2012.

[114] H. Li, C. Wu, Z. Li, F. Lau, *Virtual Machine Trading in a Federation of Clouds: Individual Profit and Social Welfare Maximization.* IEEE/ACM Transactions on Networking,

[115] L. Mashayekhy , D. Grosu. *A coalitional game-based mechanism for forming cloud federations*. in *IEEE/ACM Fifth International Conference on Utility and Cloud Computing*. pp. 223-227,2012.

[116] C. Hristea, D. Lenoski, J. Keen. *Measuring memory hierarchy performance of cache-coherent multiprocessors using micro benchmarks*. in *ACM/IEEE 1997 Conference on Supercomputing*. pp. 45-45,1997.

[117] L. Cheng , C.L. Wang, *QoS Scheduling for Latency-sensitive Cloud Applications*.

[118] S.K. Barker , P. Shenoy. *Empirical evaluation of latency-sensitive application performance in the cloud*. in *first annual ACM SIGMM conference on Multimedia systems*. pp. 35-46,2010.

[119] *Practical Guide to Cloud Service Level Agreements Version 1.0*. Available from: http://www.cloudstandardscustomercouncil.org/2012_Practical_Guide_to_Cloud_SLAs.pdf.

[120] M. Kerai. *SLA Metrics, Measurement and Manipulation - Identifying flawed SLA metrics in IT outsourcing contracts*. Accessed: July 11, 2015, Available from:

http://www.xceedgroup.com/_literature_116197/SLA_Metrics,_Measurement_and_Manipulation_-_XIPWP04.

[121] M. Klems, D. Bermbach, R. Weinert. *A runtime quality measurement framework for cloud database service systems*. in *Eighth International Conference on Quality of Information and Communications Technology (QUATIC)*. IEEE. pp. 38-46,2012.

[122] *Service Level Agreement (SLA) Internet Dedicated Services*. Available from: http://www.verizonenterprise.com/resources/terms/sla-ip-vpn-dedicated-services_en_xg.pdf.

[123] W.N. Spencer, *Advances in disk technology: performance issues.* Computer. **31**(5), pp. 75 - 81 1998.

[124] L.H. Vu, M. Hauswirth, K. Aberer. *QoS-based service selection and ranking with trust and reputation management*. in *International conference on the Move to Meaningful Internet Systems*. Springer Berlin Heidelberg. pp. 466-483,2005.

[125] M. Ouzzani , A. Bouguettaya, *Efficient Access to Web Services.* IEEE Internet Computing, pp. 34-44, 2004.

[126] *Price-Performance Analysis of Top 10 Cloud IaaS Providers, Top 10 Cloud IaaS Providers Comparison, North American IaaS Providers Benchmark Report*. 2017 Accessed: February 18, 2017, Available from: http://connect.cloudspectator.com/2017-cloud-iaas-providers-comparison.

[127] L.D. Davide, J. Skene, W. Emmerich. *SLAng: a language for service level agreements*. in *The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems*. pp. 100-106,2003.

[128] A. Sahai, V. Machiraju, M. Sayal, L.J. Jin, F. Casati. *Automated SLA Monitoring for Web Services*. in *13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management: Management Technologies for E-Commerce and E-Business Applications* pp. 28-41,2002.

[129] V.C. Emeakaroha, M.A.S. Netto, R.N. Calheiros, I. Brandic, R. Buyya, C.A.F.D. Rose, *Towards autonomic detection of SLA violations in cloud infrastructures.* Future Generation Computer Systems. **28**(7), pp. 1017–1029, 2012.

[130] *iperf*. Available from: https://iperf.fr.

[131] *speedtest*. Available from: www.pypi.python.org/pypi/speedtest-cli.

[132] *ubench*. Available from: https://archive.org/download/tucows_69604_Ubench/ubench-0.32.tar.gz.

[133] *Cloudsim*. Available from: http://www.cloudbus.org/cloudsim/.

[134] *Cloudreports*. Available from: https://github.com/thiagotts/CloudReports/.

[135] *Hibernate ORM*. Available from: http://hibernate.org/orm/.

[136] *Apache jclouds*. Available from: https://jclouds.apache.org/.

[137] *Cloudharmony API*. Available from: https://cloudharmony.com/docs/api.

[138] *BLAST+*. Accessed: August 04, 2015, Available from: https://blast.ncbi.nlm.nih.gov/Blast.cgi.

[139] *GNU Parallel*. Accessed: August 08, 2015, Available from: http://www.gnu.org/software/parallel/.

[140] A.N. Toosi, *On the Economics of Infrastructure as a Service Cloud Providers*. 2014.

[141] A. Antoniou, *Performance Evaluation of Cloud Infrastructure using Complex Workloads*. 2011.

[142] P. Bodik, A. Fox, M.J. Franklin, M.I. Jordan, D.A. Patterson. *Characterizing, modeling, and generating workload spikes for stateful services*. in *Proceedings of the 1st ACM symposium on Cloud computing*. ACM. pp. 241-252,2010.

[143] A.N. Toosi, R.K. Thulasiram, R. Buyya. *Financial option market model for federated cloud environments*. in *IEEE/ACM Fifth International Conference on Utility and Cloud Computing*.2012.

[144] A. Fries, J.P.i.d. Mora, R. Sirvent, *Java-based communication in a High Performance Computing environment.* EAS Publications Series. **45**(1), pp. 103-106, 2010.

[145] D. Zhao, Z. Zhang, X. Zhou, K.W. T. Li, D. Kimpe, I. Raicu, F.S. Fusion. *Toward supporting data-intensive scientific applications on extreme-scale high-performance computing systems*. in *IEEE International Conference on Big Data*.2014.

[146] D.A. Reed, *Scalable Input/ Output: Achieving System Balance.* 2003: The MIT Press.

[147] *CIO WebBusiness*.

[148] S. Sindhu , S.A. Mukherjee. *A Dynamic List Scheduling Algorithm for Scheduling HPC Application in a Cloud Environment*. Available from: http://searchdl.org/public/book_series/AETS/7/178.pdf.

[149] Z. Zhong, V. Rychkov, A. Lastovetsky. *Data partitioning on heterogeneous multicore platforms*. in *IEEE International Conference on Cluster Computing (CLUSTER)*. pp. 580-584,2011.

[150] N.K. Govindaraju, S. Larsen, J. Gray, D. Manocha. *A memory model for scientific algorithms on graphics processors*. in *ACM/IEEE SC Conference*.2006.

[151] R. Bader, M. Brehm, R. Ebner, H. Heller, L. Palm, F. Wagner. *TeraFlops Computing with the Hitachi SR8000-F1: From Vision to Reality*. in *High Performance Computing in Science and Engineering, Munich*. pp. 3-8,2002.

[152] C.B. Rodamilans, A. Baruchi, E.T. Midorikawa. *Experiences Applying Performance Evaluation to Select a Cloud Provider*. in *Recent Advances in Computer Engineering, Communications and Information Technology*.2014.

[153] *Bioinformatics*. Accessed: July 14, 2015, Available from: https://en.wikipedia.org/wiki/Bioinformatics.

[154] *DNA sequencing*. Accessed: July 18, 2015, Available from: https://en.wikipedia.org/wiki/DNA_sequencing.

[155] K.A. Frazer, L.Elnitski, D.M. Church, I. Dubchak, R.C. Hardison, *Cross-species sequence comparisons: a review of methods and available resources.* Genome Research. **13**(1), pp. 1-12, 2003.

[156] G.S. Gerhard, *Comparative aspects of zebrafish (Danio rerio) as a model for aging research.* Experimental gerontology. **38**(11), pp. 1333-1341, 2003.

[157] *Zebrafish Genome Found Strikingly Similar to Humans*. Accessed: July 20, 2015, Available from: http://www.sci-news.com/genetics/article01036.html.

[158] H. Feitsma , C. Edwin, *Zebrafish as a cancer model.* Molecular Cancer Research. **6**(5), pp. 685-694, 2008.

[159] K. Howe, M.D. Clark, C.F. Torroja, J. Torrance, C. Berthelot, M. Muffato, S. McLaren, *The zebrafish reference genome sequence and its relationship to the human genome.* Nature. **496**(7446), pp. 498-503, 2013.

[160] A. Matsunaga, M. Tsugawa, J. Fortes. *Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications*. in *IEEE International Conference on eScience*.2008.

[161] M.C. Schatz, *Cloudburst: highly sensitive read mapping with mapreduce.* Bioinformatics. **25**(11), 2009.

[162] *Persistence of Vision Raytracer*. Available from: http://www.povray.org.

[163] *Wikimedia Downloads*. Available from: https://dumps.wikimedia.org.

[164] A. Fava, E. Fava, M. Bertozzi, *MPIPOV: a parallel implementation of POV-Ray based on MPI.* Recent Advances in Parallel Virtual Machine and Message Passing Interface. **1697**, pp. 426-433, 2002.

[165] R.N. Calheiros, A.N. Toosi, C. Vecchiola, R. Buyya, *A coordinator for scaling elastic applications across multiple clouds.* Future Generation Computer Systems. **28**(8), pp. 1350-1362, 2012.

[166] S. Islam, J. Keung, K. Lee, A. Liu, *Empirical prediction models for adaptive resource provisioning in the cloud.* Future Generation Computer Systems. **28**(1), pp. 155-162, 2012.

List of Publications

Journal Publications

Serial#	Paper Title	Journal	Indexing/ Abstracting
1	A price-performance analysis of EC2, Google Compute and Rackspace cloud providers for Scientific Computing By Saeed Ullah, M. Daud Awan, M. Sikandar Hayat Khiyal	J. Math. Computer Sci. 16 (2016)	ISI SCI
2	Cloud Computing Framework for Resource Management System: A Case Study of Cross-Species Sequence Comparisons By Saeed Ullah , M. Daud Awan, M. Sikandar Hayat Khiyal	Journal of Agricultural and Biological Sciences Vol. 11, No. 7, July 2016	ISI ZooRec
3	Big Data in Cloud Computing: A Resource Management Perspective By Saeed Ullah , M. Daud Awan, M. Sikandar Hayat Khiyal, IF 1.344	Scientific Programming (18)	ISI Web of Science
4	Broker based QoS Centric Resource Provisioning Framework with Financial Options By Sahar Arshad, Saeed Ullah, Shoab Ahmed Khan, M. Daud Awan and M. Sikandar Hayat Khiyal	Journal of Engineering and Applied Sciences (Vol. 10, No. 8, May 2015)	Elsevier Scopus

Conference Publications

Serial#	Paper Title	Conference Name
1	Service Provisioning of Spot Virtual Machines based on Optimal bidding in Cloud Computing By Saman Safdar , Saeed Ullah , Zakia Jalil, M. Daud Awan, M. Sikandar Hayat Khiyal	MDSRC - 2015 Proceedings, 14-15 November, 2015 Wah/Pakistan
2	A Survey of Cloud Computing Variable Pricing Models By Sahar Arshad, Saeed Ullah, Shoab Ahmed Khan, M. Daud Awan and M. Sikandar Hayat Khiyal	Proceedings of the 10th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE-2015)