ADDIS ABABA UNIVERSITY
COLLEGE OF NATURAL SCIENCES

Distributed Score Based Job Scheduling Algorithm for Cloud Computing Environment

Natnael Argaw

A Thesis Submitted to the Department of Computer Science in Partial Fulfillment for the Degree of Master of Science in Computer Science

Addis Ababa, Ethiopia
June 2016

Addis Ababa University

College of Natural Sciences

Natnael Argaw

Advisor: Mulugeta Libsie (PhD)

This is to certify that the thesis prepared by *Natnael Argaw*, titled: *Distributed Score Based Job Scheduling Algorithm for Cloud Computing Environment* and submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the Examining Committee:

Name Signature

Date

Advisor: Dr. Mulugeta Libsie _____

Examiner: Dr. Dida Midekso _____

Examiner: Dr. Yaregal Assabie _____

Abstract

Cloud computing, the long-held dream of computing as a utility, is transforming a large part of the IT industry, making software even more attractive as a service and shaping the way computer hardware is designed and purchased. It is observed that the capabilities of cloud computing environment have not been used well since their core sub system, job scheduler, lacks various salient features. Consequently, this research work identified resource utilization, response time and load balancing as the major problems of job schedulers in the cloud. Thereby, we formulated a new scheduling algorithm called distributed score based job scheduling algorithm for cloud computing environment.

This algorithm mainly uses the score of resources. Processor speed, processing elements/cores, size of primary memory, size of secondary memory, and network bandwidth are parameters considered in this component. Furthermore, a component called group state manager is used to categorize resources with the intention of improving the degree of resource utilization and load balancing among resources in the cloud. In addition, it is responsible to group jobs based on their resource demand. Consolidation of jobs per resource as well as per host is also performed by a component called job consolidator. The output of this components is fed to another component called group level adaptive job scheduler. The purpose of incorporating group level adaptive job scheduler in the main architecture is to enable contextual modification of the proposed scheduling algorithm on each group. Furthermore, components dedicated for interrupt threshold calculation and job prioritization are incorporated to solve issues related to starvation.

Moreover, taking various scenarios which depict the minimum and maximum capacity of the computing environment as well as different requirements of incoming jobs we evaluated our scheduling algorithm with respect to two standard scheduling algorithms, namely First Come First Served and Round Robin job scheduling algorithms. It is observed that the proposed scheduling algorithm outperforms the counterparts from the perspective of response time, resource utilization, and load balancing.

Keywords: Cloud Computing, Job Scheduling, Server Score, Adaptive Job Scheduling, IaaS, Group Based Scheduling

Acknowledgments

Though only my name appears on the cover of this thesis, many great people have contributed to its success. I owe my gratitude to all those people who made this thesis possible and because of whom my graduate experience has been one that I will value forever.

My deepest gratitude is to my advisor, Dr. Mulugeta Libsie. I am deeply grateful to him for helping me sorting out the technical details of the research work, giving insightful comments and constructive criticisms, encouraging the use of correct grammar and consistent notation in my writings and for carefully reading and commenting countless revisions of this document. His patience and support helped me overcome many difficult situations and finish this thesis work. I hope one day I would become as good advisor and teacher to my students as he has been to me.

None one of this would have been possible without the love and patience of my family and friends. Their support and care helped to overcome setbacks and stay focused on my graduate study.

Finally, I appreciate the financial support from Addis Ababa University that funded parts of the research discussed in this manuscript.

Table of Contents

List of Figures

List of Algorithms

List of Tables

Acronyms and Abbreviations

ABC	Activity Based Costing
ACO	Ant Colony Optimization
BA	Bees Inspired Algorithm
BCO	Bee Colony Optimization
BYOD	Bring Your Own Device Policy
DSBJSA	Distributed Score Based Job Scheduling Algorithm
EA	Evolutionary Algorithm
FAN	File Area Network
FCS	Finite Capacity Scheduling
GA	Genetic Algorithm
HRMS	Heterogeneous Resource Management Systems
IaaS	Infrastructure as a Service
IBS	Incentive Based Scheduling
MBO	Marriage In Bees Optimization
NAS	Network Attached Storage
NFM	Network File Management
NISA	Nature Independent Scheduling Algorithm
OSS	Operation Support System
PaaS	Platform as a Service
PSO	Particle Swarm Optimization
ROI	Return On Investment
SaaS	Software as a Service
SCP	Supply Chain Planning
SI	Swarm Intelligence
SLA	Service Level Agreement
SOA	Swarm Optimization Algorithm
VDI	Virtual Desktop Infrastructure
VM	Virtual Machine
VMM	Virtual Machine Manager

1. Introduction

1.1 Background

Computing as a utility is a dream that dates from the beginning of the computing industry. A set of new technologies that have come along that, along with the need for more efficient and affordable computing, has enabled on-demand systems like the Cloud. Cloud computing refers to applications and services that run on a distributed network using virtualized resources and accessed by common Internet protocols and networking standards [1]. It is distinguished by the notion that resources are virtual and limitless and that details of the physical systems on which software runs are abstracted from the user. Cloud computing takes the technology, services, and applications that are similar to those on the Internet and turns them into a self-service utility. It is growing fast because of its attractive features such as on demand services, easy to use, cheap and pay as use scheme, etc. [2, 3, 4].

Most scholars separate cloud computing into two distinct sets of models: deployment model and service model. Deployment model defines the purpose of the cloud and the nature of how the cloud is located. Cloud computing deployment models are classified into 4 namely: public cloud, private cloud, hybrid cloud and community cloud [16]. The public cloud offers applications, storage and other services to the general public by a service provider. This is based on "pay -as-you-go" model. A public cloud is constructed with a view to offer unlimited storage space and increased bandwidth via Internet to all businesses. Private cloud is a cloud infrastructure build exclusively for a single organization, deployed within certain boundaries like firewall settings whether managed internally or by a third party and hosted internally or externally. Users are charged on the basis of per Gigabyte usage along with bandwidth transfer fees. Data stored in the private cloud can only be shared amongst users of an organization. Hybrid clouds combine the advantages of private and public clouds, offers flexibility, control and security of multiple deployment models. In community cloud, the infrastructure is shared between the organizations with similar interests and requirements whether managed internally or by a third party and hosted internally or externally.

On the other hand, as cloud computing has developed, different vendors offer clouds that have different services associated with them. The portfolio of services offered adds another set of definitions called the service model. Service models elaborate the particular types of services that

one can access on a cloud computing platform. There are three universally accepted cloud services: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS).

Cloud computing is capable to provide massive computing or storage resources without the need to invest money or face the trouble to build or maintain such huge resources. Consumers only need to pay for using the services just like they do in case of other day to day utility services such as water, gas, electricity, etc. Cloud computing is now being used in many applications that are beyond distribution and sharing of resources. The distributed resources are useful only if the cloud resources are scheduled. Using optimal scheduler results in high performance cloud computing whereas poor schedulers produce substandard results [5]. Scheduling algorithms are used for dispatching user tasks or jobs to a particular resource or data [6]. Scheduling is a challenging job in the cloud because the capability and availability of resources vary dynamically. The goal of job scheduling is to properly dispatch parallel jobs to slave node machines according to scheduling policy meeting certain performance indexes and priority constraints to shorten total execution time and lower computing cost and improve system efficiency.

There are a number of models designed to improve the performance of cloud computing. However, each and every proposal is aimed to overcome a part of known scheduling problems. The tradeoffs between scheduling problems could not be resolved completely. Currently, employed job scheduling algorithms encounter critical problems. Resource utilization, load balancing, reconfiguration cost, communication latency, execution time, virtual machine migration and starvation are among the most known setbacks of scheduling algorithms.

The aim of this thesis is to develop a distributed score based job scheduling model for cloud computing environment. The model mainly uses the score of distributed resource centers to determine their category. Categories are defined to simplify the job-to-resource dispatching and increase resource utilization. Thresholds are also used to overcome the issue of overloaded servers. In addition, regardless of other scheduling algorithms which use per job server selection, the proposed algorithm introduced a mechanism to perform per job-queue reconfiguration of servers. Similarly, regardless of other scheduling algorithms which allocate each job sequentially and independently, the proposed algorithm allocates jobs in proportion based manner. It is

believed that this model showed the efficient way to overcome the aforementioned problems of currently deployed job scheduling models.

1.2 Motivation

Cloud computing has become the major destiny for high scale computation jobs and even for any ordinary computing requirements. Currently, the technology strives to satisfy large computational requirements through well-equipped and distributed computing systems rather than using standalone high capacity servers. Moreover, cloud provides resources as a service on demand base. This makes it handy for the customers and showed a radical diversion on the trend of the computing industry.

However, the full capability of such computing systems does not exist as it is perceived by the technology conceivers. Although, cloud is equipped with high performance resources, it is observed that lack of resource provisioning as well as efficient job scheduling restrained it from working in its full capability. For one who observed the aforementioned issues in the cloud and wants to solve it from its root must work on the major cause, scheduling problem.

Hence, in this research we have given a great emphasis to the scheduling and resource provisioning domains in the cloud right after the identification of high level problems. Obviously, striving to solve the scheduling problem in the cloud and delivering an improved scheduling algorithm as a result does not only boost the performance of the cloud but also the satisfaction of customers who are the major entities in the cloud. This is because, improving the scheduling performance improves the overall cloud performance and the better cloud performance the higher service level agreement between the customer and cloud vendor.

On the other hand, the findings of this research work have a great impact for the upcoming efforts to solve the scheduling problem. Components conceived and implemented in this research work could be adopted to overcome gaps observed in other domains which have scheduling problem. Hence, we would say this is the secondary reason to conduct this research work.

Generally, the nontrivial impact of scheduling algorithm in the performance of the cloud as well as the indicators that show how small the cloud is being used in comparison with to what extent it could have been utilized is the major reason to launch this research work.

1.3 Statement of the Problem

A cloud computing platform dynamically provisions, configures, reconfigures, and deprovisions servers as needed. Servers in the cloud can be physical machines or virtual machines. They provide the utility services based on the pay-as you go model. Users can host different kinds of applications on the cloud ranging from simple web applications to scientific workloads. These applications are delivered as services over the Internet. It is now being used in many applications that are beyond distribution and sharing of resources. Many leading IT vendors, such as Amazon, Google, Microsoft, IBM, HP, Cisco, etc. believe that cloud computing is the next logical step in controlling IT resources, as well as a primary means to lower total cost of ownership. Similarly, a need to develop an optimal model that can efficiently dispatch IT resources is their main agenda [9].

Various researches have been conducted to solve problems in job scheduling for cloud computing [8]. However, most of these studies concentrate on solving only one of the known problems of scheduling. The approach that most papers used ensue nonlinear dependency among performance indexes which we call them tradeoffs. For instance, while trying to solve resource utilization and load balancing problems most research works use an iterative modification of the whole resource centers, which in turn leads to high cost of reconfiguration and poor response time. In addition, some researchers stress on load balancing issues. However, the approach they used in balancing load causes a creation of many resource status checking interrupts, which can clearly degrade the performance of the scheduler.

In addition, most research works, [7-15] and [59-100], are intended to solve only one of the following problems such as resource utilization, load balancing, reconfiguration cost, communication latency, execution time, virtual machine migration, starvation, etc. In addition, researches could not come upon a greater degree of resource utilization, load balancing and reconfiguration cost.

This thesis is aimed to solve resource utilization, load balancing and reconfiguration cost problems by employing a new algorithm as well as by altering the preexisting components of job scheduling algorithms for the cloud. In addition, by considering the complexity of the public cloud, this thesis is designed and a working prototype is developed to overcome the scheduling problems that exist mainly in public cloud computing environment.

1.4 Objectives

General Objective

The general objective of this research is to design and develop a distributed score based job scheduling model for cloud computing environment.

Specific Objectives

- Reviewing related literature in the area of cloud computing and more specifically job scheduling for the cloud.
- Analyzing the components of the current job scheduling algorithms in line with the architecture of cloud computing.
- Identifying QoS constraints for Server Score Calculation.
- Deriving formulas to perform proportional and concurrent job allocation.
- Developing a prototype to show the performance of the distributed score based job scheduling model for cloud computing.
- Testing the Prototype.

1.5 Methods

The following methods will be applied in the course of the research.

- Literature review

  The research works that have been done with the intention of solving job scheduling in cloud computing environment will be evaluated. Moreover, elicitation as well as analyses of the gaps that are not covered by the related research works will be performed.

- Model Composition

  The gaps identified in various literatures and related works along with the proposed solution to solve these problems will be used to conceive and implement a new scheduling algorithm.

- Prototype Development

  Various document processing, data analysis and model designing tools will be used to compile the document. Moreover, Netbeans IDE with Java programming language will be used to develop the prototype. Specifically, we will use Cloudsim simulation toolkit to simulate the cloud and evaluate the performance of the proposed scheduler in various simulated models.

- Testing and Evaluation

  An intensive testing will be made on the proposed algorithm and its performance will be evaluated in terms of its goals and contributions. Specifically, the performance of the proposed scheduler in relation to other scheduling algorithms will be tested by running a concurrent and long term test.

1.6 Scope and Limitations

The main focus of this research is designing, modeling and prototype development of a distributed score based job scheduling model for cloud computing environment. However, the research work does not give much emphasis on creating virtual machine scheduling policy. It uses a time shared virtual machine scheduling mechanism. Besides that, this research work does not address issues related to securing cloudlets in the cloud.

1.7 Application of Results

The result of this research work could exhibit a radical improvement on performance and resource utilization in cloud computing. The approach can also be extended for other large scale multi user computing systems such as Grid Computing. In addition, researchers conducting researches on areas such as CPU scheduling, multi programmed computer systems, and large scale computing systems are among the immediate beneficiaries of the results of this research work.

1.8 Organization of the Rest of the Thesis

The rest of this thesis is organized into five major Chapters. This section gives an overview on the contents of each Chapter.

Chapter Two gives theoretical framework on cloud computing in general and job scheduling algorithms in particular. Furthermore, Cloud Computing and its characteristics, enabling technologies for the cloud are discussed briefly. Besides, service delivery models IaaS, PaaS, and SaaS are discussed concisely. Various features of job scheduling along with their taxonomy are also discussed in this Chapter.

In Chapter Three, major contributions on improving the scheduling algorithm for the cloud are described. Furthermore, various scheduling algorithms currently in use by various cloud vendors are briefly discussed in two major categories as swarm optimization algorithms and nature

independent scheduling algorithms. Swarm optimization algorithms sub section describes scheduling algorithms which are inspired by the natural problem solving behavior of social animals. On the other hand, the nature independent job scheduling algorithm describes some job scheduling algorithms derived by various mathematical problem solving techniques and/or models.

Chapter Four presents the proposed distributed score based job scheduling algorithm. In this Chapter the big picture of the proposed model as well as issues considered during its design are put concisely. Besides that, relationships between components and their high level description is given in the first part of this Chapter. Moreover, the major components used in the big picture are exploited and discussed independently and briefly.

Chapter Five presents topics that are concerned on describing the implementation details of the proposed model. Tools used to develop the prototype as well as their significance specific to this prototype development are described. Steps followed during prototype development are illustrated by coupling it with the pictorial illustration of outputs gained from each component and even methods inside it. Additionally, this Chapter presents results gained after comparing the proposed algorithm with other scheduling algorithms. Results are illustrated using tabular and graphical view.

The last section in this thesis, Chapter Six, summarizes the major concepts raised in this research work. In addition, contributions of the research and future works that could be done to improve the performance of the proposed model are briefly described.

2. Literature Review

This Chapter deals with the review of important concepts concerning the research work. It provides a detailed definition of cloud computing, different models, architectures, and services of the cloud. The technologies that drag cloud into existence and, enabling technologies of the cloud are also discussed briefly. Specifically, features of job scheduling along with its taxonomy are put succinctly in this Chapter.

2.1 Cloud Computing

Cloud computing, the long-held dream of computing as a utility, is transforming a large part of the IT industry, making software even more attractive as a service and shaping the way computer hardware is designed and purchased. It is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction [22]. Cloud is distinguished by the notion that resources are virtual and limitless and that details of the physical systems on which software runs are abstracted from the user [1].

Cloud computing has led technological giants to invest heavily in cloud infrastructures while the Software-as-a-Service, Platform-as-a-Service and Infrastructure-as-a-Service models are constantly on the rise. Organizations are now able to fulfill their infrastructure demands based on rising requirements while minimizing their IT expenditures. The pay for use model along with on-demand resource allocation and scalability allows even the smallest organizations to reach a wider audience without any financial or managerial constraints. Consequently, many organizations are migrating to cloud computing as it is reducing their capital investment and data center space needed for the construction of huge computing environment.

2.1.1 Characteristics of Cloud Computing

Cloud computing is an evolutionary change that is characterized by a revolutionary new approach to how computing services are produced and consumed. The following paragraphs depict the characteristics of the cloud; of which Broad network access, On-demand self-service, Resource pooling, Measured service, and Rapid elasticity are the salient ones [22].

a. Broad Network Access: In the earliest times, computing resources were scarce and costly. To conserve those resources, usage was limited based on priority and criticality of workloads.

Similarly, network resources were also scarce. IP-based networks were not in prevalent usage. Consequently, access to ubiquitous high-bandwidth, low-latency networks did not exist.

Over time, costs associated with the network have been diminished because of manufacturing scalability, commoditization of associated technologies, and competition in the marketplace [17]. As network bandwidth increased, network access and scalability which are the enablers for the cloud have also increased accordingly.

b. On-Demand Self-Service: is the primary characteristic of the cloud. In computing environment other than cloud, the ability of end users to self-provision resources fundamentally disrupts most of the legacy processes such as capacity planning, network management, and security. The other downstream of the aforementioned situation is extreme inefficiency and waste of resources as it allows unwanted resources to be presented in one end while the others are starving. Cloud-based architectures, however, are designed and built with self-provisioning in mind [17]. This premise implies the use of fairly sophisticated software frameworks and portals to manage provisioning and back-office functions. Self-service cloud offerings provide easy-to-use and intuitive user interfaces that equip users to productively manage the service delivery lifecycle [18]. The benefit of self service from the users' perspective is a level of empowerment and independence that yields significant business agility. Besides, one benefit often overlooked from the service provider's or IT team's perspective is that the more self-service that can be delegated to users, the less administrative involvement is necessary.

c. Resource Pooling: is a fundamental premise of scalability in the cloud. Without pooled computing, networks, and storage, a service provider must have provision across multiple silos[1]. Besides, the idea of multi-tenancy is fundamental to cloud computing. Service providers are able to build network infrastructures and data architectures that are computationally very efficient, highly scalable, and easily incremented to serve the many customers that share them. Multi-tenancy spans the layers at which services are provided [41]. Therefore, in the absence of resource pooling and multi-tenancy, the economics of cloud computing do not make financial sense [17].

d. Measured Service: implies that usage of pooled resources is monitored and reported to the consumer, providing visibility into rates of consumption and associated costs. Cloud computing

[1] Discrete, independent resources with few or no interconnections.

is usage driven in which consumers pay for only what resources they use and therefore are charged or billed on a consumption based model. Cloud computing platforms provide mechanisms to capture usage information that enables chargeback reporting and/or integration with billing systems [18]. The value here from a user's perspective is users are able to pay only for the resources they use, ultimately helping them keep their costs down. From a provider's perspective, it allows to track usage for charge back and billing purposes.

e. Rapid Elasticity: is the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible [42]. It is established on dynamic infrastructure and often the foundation for the dynamic infrastructure is a standardized, scalable, and secure physical infrastructure. Usually, there is high level of redundancy to ensure high level of availability, but mostly it is designed in a way that it is easy to extend as usage growth demands it, without requiring architecture rework.

A dynamic computing infrastructure is critical to effectively supporting the elastic nature of service provisioning and de-provisioning as requested by users while maintaining high levels of reliability and security [18]. The consolidation provided by virtualization, coupled with provisioning automation, creates a high level of utilization and reuse, ultimately yielding a very effective use of capital equipment.

2.1.2 Cloud Computing Architecture

Cloud computing architecture refers to the components and subcomponents required for cloud computing [43]. Many organizations and researchers have defined the architecture for cloud computing. As it is depicted in Figure 2-1 [35], the cloud architecture can be divided into two major parts, namely the core stack and the management.

The core stack subsumes Resource, Platform and Application layers.

a. The Resource Layer: is the infrastructure layer which is composed of physical and virtualized server, storage and networking resources. The server layer is composed of multiple sub layers. Each sub layer is composed of a set of physical servers. Physical servers provide computational resources to cloud users. Besides, each server hardware is managed by the hypervisor. In turn, the hypervisor runs the virtual machine manager (VMM). The VMM manages virtual machines running on the physical server [44]. The storage layer is composed of

sub layers, which consist of storage components. It stores cloud data and/or provides file system services. Storage could be either local storage or network storage. Local storage is connected directly to the server, while network storage servers are connected to the storage over public network. The Network layer enables the cloud components to communicate with each other. Communication can be horizontal within the same layer or vertical across multiple layers [45].

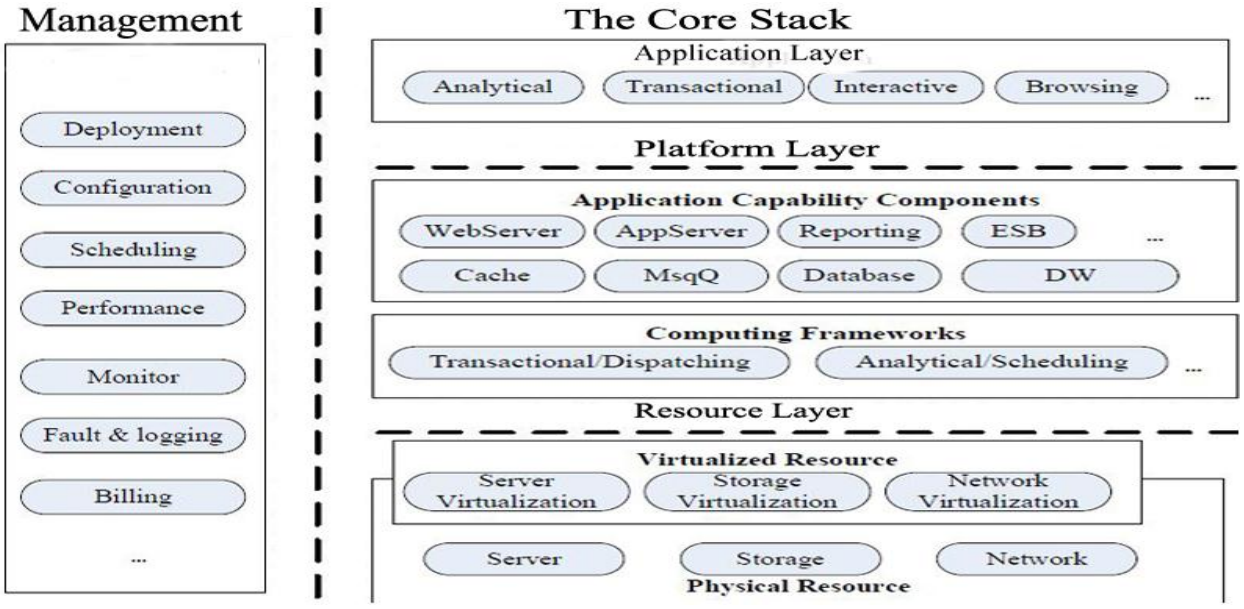

Figure 2-1: The Architecture of the Cloud

b. The Platform Layer: is the most complex part which could be divided into two major parts as computing frameworks and application capability components. It consists of operating systems and application frameworks. The purpose of the platform layer is to minimize the burden of deploying applications directly into VM containers [46]. For example, Google app engine operates at the platform layer to provide Application Program Interface (API) support for implementing storage, database and business logic of typical web applications.

The application server and other components support the same general application logic as before with either on-demand capability or flexible management, such that no components will be the bottleneck of the whole system.

c. The Application Layer: consists of the actual cloud applications. Different from traditional applications, cloud applications can leverage the automatic-scaling feature to achieve better performance, availability and lower operating cost. Each layer is loosely coupled with the layers above and below, allowing each layer to evolve separately. The architectural modularity allows

cloud computing to support a wide range of application requirements while reducing management and maintenance overhead [46].

On the other hand, the management part of the cloud reference architecture subsumes the software and technologies designed for operating and monitoring applications, data and services residing in the cloud. Cloud management tools help ensure cloud computing-based resources are working optimally and properly interacting with users and other services.

2.1.3 Models of Cloud Computing

The cloud computing model allows access to information and computer resources from anywhere that a network connection is available. There are diverse dimensions to classify cloud computing models, two commonly used categories used by many are: Service model and Deployment model.

2.1.3.1 Service Models

The cloud is often referred to as a stack, a pyramid-like structure with low-level resources (servers, networks, operating systems and so forth) at the bottom and end user software services at the top, with tools for developers and testers in between. These categories are not hard and fast, and in fact the lines between them have been blurring for some time. As it is shown in Figure 2-2 [36], there are three broad labels that are typically applied to the tiers of the cloud stack.

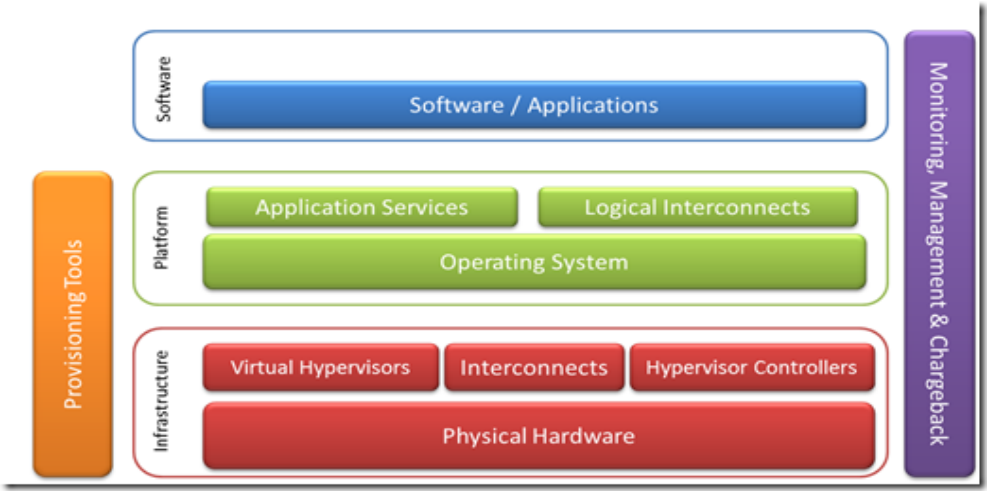

Figure 2-2: Cloud Service Model

a. Infrastructure as a Service (IaaS)

Infrastructure as a Service is the capability to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run software [19]. It helps companies to move their physical infrastructure to the cloud with a level of control similar to what they would have in a traditional on-premise data center. Infrastructure as a Service provides the closest resemblance to the in house data center compared to other services types.

The core infrastructure components are storage, servers, network, and management tools for infrastructure maintenance and monitoring. Each of these components has created a separate market niche. Infrastructure as a Service platforms offer highly scalable demand based resources that in turn make the service well-suited for workloads that are temporary and experimental.

b. Platform as a Service (PaaS)

Platform as a Service is a group of services that abstracts application infrastructure, operating system, middleware, and configuration details, and provides developer teams with the ability to provision, develop, build, test, and stage applications [20]. The aim of Platform as a Service is to enable rapid creation and release of applications without having to invest on the physical infrastructure and software tools that support them. It facilitates application deployment through self-service, on-demand tools, resources, automation, and a hosted platform runtime container. Besides, Platform as a Service provides access to scalable runtime and programming environments with embedded data structures. This in turn eliminates the need for an installation kit. As well, developers no longer have to configure and wait for physical servers or virtual machines or to copy files from one environment to another as they move through the application life cycle.

Platform as a Service provides several advantages for developers and lowers the barrier for creating cloud-aware applications. The rise of agile software development has made PaaS a popular mechanism for enhancing collaboration and accelerating application lifecycles [21].

c. Software as a Service (SaaS)

Software as a Service involves access to collections of software that are controlled by the provider and executed on that organization's infrastructure. It is basically the delivery of commercial software over the Internet. On-demand Software as a Service applications are based

on a recurring subscription fee and typically are a pay as you go model. The cost may increase as the usage of the application increases. In this model, costs are directly aligned with usage.

A typical Software as a Service deployment does not require any hardware and can run over the existing Internet access infrastructure [21]. Sometimes, changes to firewall rules and settings may be required to allow the Software as a Service application to run smoothly. A SaaS application can be configured using APIs but multi-tenant SaaS applications cannot be completely customized.

The SaaS vendor assumes all the support, training, infrastructure and security risks in exchange for the recurring subscription fees. The SaaS service model is designed to deliver business applications anywhere, anytime which in turn requires the SaaS vendor to employ dedicated support teams and staff that make themselves available to customers on short notice. Architecturally, the preferred SaaS model is multi-tenant [21].

As an import trend of cyberspace in the future, the aforementioned services of the cloud have attracted much attention from the IT industry. Many research institutions and companies have launched their own cloud platforms, which have virtual machine schedulers to manage the infrastructure resource pool. The virtual machine scheduling modules in these platforms are built in the platform and it is hard for developers to re-program. Since developers cannot design and implement special policies in the platform, the flexibility of the virtual machine scheduler is poor [47]. Furthermore, the unchangeable and firm interface of the infrastructure schedulers leads to poor portability and resource allocation. Accordingly, this research work considers the infrastructure as a service as its concern to formulate the best infrastructure scheduling algorithm.

2.1.3.2 Deployment Models

A deployment model defines the purpose of the cloud and the nature of how the cloud is located. There are four common deployment models for cloud services loosely determined by who has access to the cloud services and they are Public Cloud, Private Cloud, Community Cloud, and Hybrid Cloud [22].

a. Public Cloud

Public clouds, which are the major concerns of this research work are the latest evolution of computing, offering tremendous value to businesses in terms of better economics, agility, rapid

elasticity, etc. The public cloud infrastructure is operated by a cloud service provider and the services are offered over the Internet. This very nature of public clouds offers various advantages such as better Return on Investment (ROI) and faster time to market, while also raising concerns about lack of visibility, security, reliability, etc. Public clouds are well suited to meet the collaborative needs of today's global workforce distributed across different geographies and time zones.

Public clouds offer rapid elasticity and seemingly infinite scalability with ability to consume resources on a pay-per-use basis. Typically, public clouds are operated and managed at data centers belonging to service providers and shared by multiple customers (multi-tenancy). Such a shared model helps reduce vendor costs, which manifests itself in better cloud economics. The degree of visibility and control depends on the specific public cloud delivery model. For instance, there is less visibility and control in a public cloud than a private cloud because the underlying infrastructure is owned by the service provider [1].

b. Private Cloud

With a private cloud, users have immediate access to IT resources that are running on internal systems and/or in a data center. Computing, networking and storage can be provisioned from a Web interface, just like public cloud.

All services are deployed behind the company's firewall and are subject to existing mechanisms for physical, electronic and procedural security in the facility. Accordingly, private cloud offers a high degree of control and data security, making it a popular option for enterprises uncomfortable with storing information on someone else's infrastructure.

In addition, private cloud affords many options in hardware and is not predicated on multi-tenancy. This means that users benefit from dedicated performance. In some cases, organizations may also rely on third-party providers to administer entire servers that are reserved for their software. This arrangement is called a managed private cloud [48].

c. Community Cloud

A community cloud is one where the cloud has been organized to serve a common function or purpose. It may be for one organization or for several organizations, but they share common concerns such as their mission, policies, security, regulatory compliance needs, and so on. A community cloud may be managed by the constituent organization(s) or by a third party [1].

d. Hybrid Cloud

A hybrid cloud combines multiple clouds where those clouds retain their unique identities, but are bound together as a unit [48]. To enable such a setup, the user applies a tool such as Eucalyptus that allows for data and application migration between the multiple environments. Shared APIs are essential to proper hybrid cloud computing. A hybrid cloud may offer standardized or proprietary access to data and applications, as well as application portability.

2.1.4 Enabling Technologies of Cloud Computing

The evolution of cloud computing is made possible through technologies such as hardware advancement technologies, computing technologies, virtualization, distributed file system, and Web x.0 technologies [34]. These technologies are commonly termed as cloud-enabling technologies. Cloud computing provides an opportunity for enterprises to use these technologies by eliminating the need for deeper understanding and expertise in using them. Some enabling technologies of cloud are put succinctly in this topic.

a. Hardware Advancements

Single-core and multi-thread computing model is unable to meet the intensive computing demand of the cloud. On the other hand, multi-core CPU is characterized by low electricity consumption, efficient space utilization, and favorable performance which in turn helps cloud providers build energy-efficient and high performance data centers.

In addition, cloud computing provides services in a multi-tenant environment where network is serving as the "glue" function. Services in a cloud are delivered to cloud consumers through networks. Besides, the network advancement technologies highly influenced the taxonomy of the cloud, which is based on the networking relationship between cloud consumers and providers [34].

b. Computing Technologies

Distributed computing is the most obvious predecessor technology that enabled the inception of cloud computing. It is referred to as an infrastructure that delivers storage and compute resources from multiple distributed computers to address larger computational problems.

c. Virtualization

Over the past few years, researchers have been striving for a Utility Computing Model. Cloud computing service providers provide resources by means of virtualization. Applying these techniques, however, goes along with handling over ultimate control of data to a third party.

Virtualization is a framework or methodology of dividing the resources of a computer into multiple execution environments, by applying one or more concepts or technologies such as hardware and software partitioning, time-sharing, partial or complete machine simulation, emulation, quality of service, and many others [25]. The main aim of virtualization is to manage the workload of machines by transforming traditional computing into more scalable, efficient, and economical system.

Virtualization can trace its roots back to the mainframes of the 1960's and 1970's. Initially, the definition of virtualization was considered as a method of logically dividing mainframes to allow for multiple applications to run simultaneously [24]. In recent years, virtualization has gained popularity in many different areas such as server consolidation, information security and most importantly cloud computing.

Virtualization can be utilized in many different ways and can take many forms aside from just server virtualization. Generally, the main types of virtualization include application, desktop, user, storage and hardware virtualization [27].

i. Application Virtualization

Application Virtualization is defined as the ability to deploy software without modifying the host computer or making any changes to the local operating system, file system, or registry [28]. Using this virtualization technology, organizations can deploy custom and commercial software across the enterprise without installation conflicts, system changes, or any impact on stability or security. Consequently, it allows users to access the application, not from their workstation, but from a remotely located server.

ii. Desktop Virtualization

Desktop virtualization, often called client virtualization, is a virtualization technology used to separate a computer desktop environment from the physical computer [29]. Desktop virtualization is considered a type of client-server computing model because the "virtualized"

desktop is stored on a centralized, or remote, server and not the physical machine being virtualized.

Desktop virtualization "virtualizes desktop computers" and these virtual desktop environments are "served" to users on the network. This technology lets users to seamlessly interact with the virtual desktop. Another benefit of desktop virtualization is that it allows remote desktop access from any location.

Virtual Desktop Infrastructure (VDI) is a popular method of desktop virtualization. This type of desktop virtualization uses the server computing model, as the desktop virtualization in this scenario is enabled through hardware and software. VDI hosts the desktop environment in a virtual machine that runs on a centralized or remote server.

It also allows the user's operating system (OS) to be remotely stored on a server in the data center, allowing users to access their desktop virtually from any location.

iii. User Virtualization

User virtualization is similar to desktop virtualization, but provides users with the ability to maintain a fully personalized virtual desktop when not on the company network. This kind of virtualization decouples a user's profile, settings and data from the operating system and stores this information into a centralized data share either in the data center or the cloud. User virtualization solutions provide consistent and seamless working environments across a range of application delivery mechanisms. Users can basically log into their “desktop” from different types of devices like smart phones and tablets.

Although user virtualization is most closely associated with desktop virtualization, in fact, this technology can be used to manage user profiles on physical desktops as well. As the range of currently used operating systems expands, and the use of multiple devices by workers to perform their jobs escalates, user virtualization can support the creation of a "follow-me" identity that will allow access to a workspace without being tied into only a single device or a single location [23]. With more companies migrating to bring ones own device policy (BYOD)[2], desktop and user virtualization are becoming increasingly popular [30].

[2] Is a set of rules governing a corporate IT department’s level of support for employee-owned PCs, smart phones and tablets.

iv. Storage Virtualization

Storage virtualization is the act of abstracting, hiding, or isolating the internal functions of a storage system or service from applications, host computers, or general network resources, for the purpose of enabling application and network-independent management of storage or data. Storage virtualization has become the buzzword in the storage industry, especially with the increased acceptance of storage networks [31]. In addition, the scheme supports application and network independent management of storage.

As it is depicted in Figure 2-3 [32], the storage virtualization model can be divided into four main layers namely storage devices, block aggregation layer, file/record layer, and application layer.

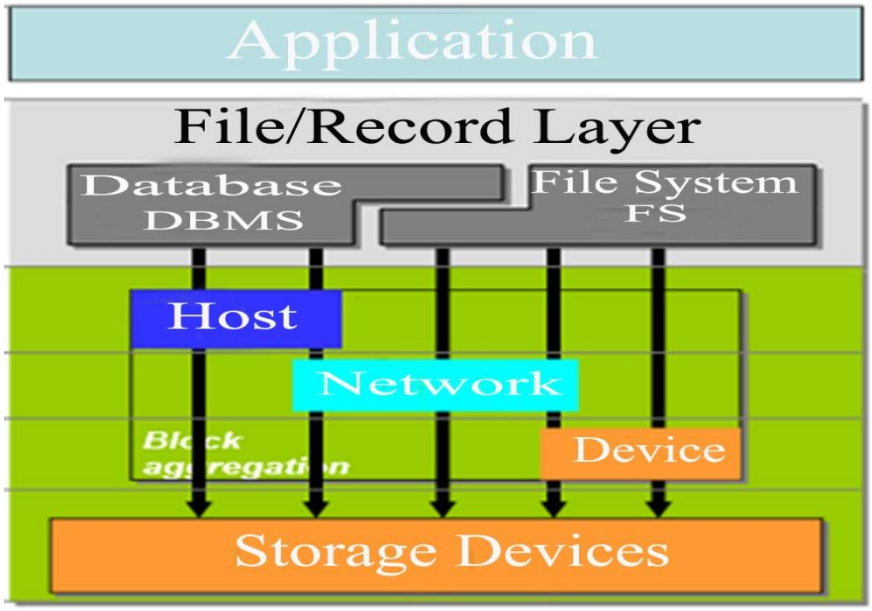

Figure 2-3: A Layered View of Storage Virtualization

- **Application Layer** makes a read or write request for the operating system.
- **File/ Record Layer** unites multiple storage devices into a single logical pool of files. It is a vital part of both file area network (FAN) and network file management (NFM) concepts. File virtualization, which is provided by this layer provides location transparency of files [31]. Users look to a single mount point that shows all their files. Data can be moved from a tier-one network attached storage (NAS) to a tier-two or tier-three NAS or network file server automatically.

- **Block Aggregation (Virtualization)** deals with the block level disk services, and it refers to a storage service that provides a flexible, logical arrangement of storage capacity to applications and users while abstracting its physical location. Furthermore, rather than disk virtualization, block virtualization virtualizes several physical disks to present a single logical device.

  Block virtualization usually accepts the logical addresses initiated by the application layer and converts to its physical address. This transformation can take place in the host, somewhere in the network or at the storage [31]. In this way virtualization enables administrators to provide the storage capacity when and where it is needed while isolating users from the potentially disruptive details of expansion, data protection and system maintenance.
- **Storage Devices layer** is the one which subsumes storage devices for which block aggregation layer resolves physical address. Furthermore, it is computing hardware that is used for storing, porting and extracting data files and objects.

v. Hardware Virtualization

Hardware virtualization is a form of virtualization that uses one processor to act as if it were several different processors. The user can then run different operating systems on the same hardware, or more than one user can use the processor at the same time. This type of virtualization requires a virtual machine manager called a hypervisor.

A hypervisor or virtual machine monitor is a piece of computer software, firmware or hardware that creates and runs virtual machines [40]. On a computer on which a hypervisor is running, one or more virtual machines are defined as host machines. Each virtual machine is called a guest machine. The hypervisor presents the guest operating systems with a virtual operating platform and manages the execution of the guest operating systems. Multiple instances of a variety of operating systems may share the virtualized hardware resources.

d. Distributed File System

Cloud computing utilizes the distributed data storage technology to store data. In this scheme, it is possible to access files from multiple hosts, via a computer network. It enables multiple machines to share files and storage resources.

e. Web x.0 Technologies

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network [34]. Cloud computing, to a large extent, leverages the concept of Service-Oriented Architecture (SOA)[3], especially in the SaaS and PaaS layers. Cloud computing is an infrastructure emphasizing on the solution of "how to deliver applications".

2.1.5 Architecture of Virtualized Cloud Technology

In cloud computing, memory is virtually allocated to the users in the servers which requires a platform on which a hypervisor runs. The virtualization model consists of cloud users, service models, virtualized models and its host software as well as their hardware. The basic architecture of virtualized cloud is depicted in Figure 2-4 [26].

A cloud is a pool of virtualized resources across the Internet that follows a pay-per-use model and can be dynamically reconfigured to satisfy user requests via on-the-fly provisioning/de provisioning of virtual machines [50]. Cloud computing already leverages virtualization for load balancing via dynamic provisioning and migration of virtual machines among physical nodes.

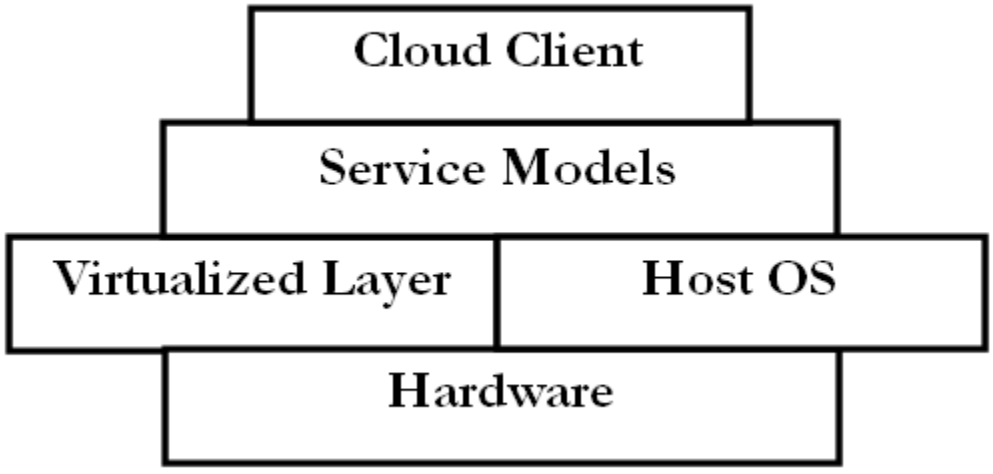

Figure 2-4: Basic Virtualized Cloud Technology Architecture

Subsequently, the high level description of each layer is given below.

[3] It is an architectural pattern in computer software design in which application components provide services to other components via a communications protocol, typically over a network independent of any vendor, product or technology.

a. Cloud Client: consists of computer hardware and/or computer software that relies on cloud computing for application delivery, or that is specifically designed for delivery of cloud services and that, in either case, is essentially useless without it [49].

b. Service Models: consists of the three service models namely SaaS (Software as a Service), PaaS (Platform as a Service), and IaaS (Infrastructure as a Service).

c. Virtualized Layer and Host OS: A host operating system is the primary operating system installed on a cloud server. On the other hand, the virtual layer helps to subsume virtual operating systems that may operate within the host OS. When virtual operating systems are required to deal with computer resources in different ways, one or more virtual operating systems are installed within the host operating system, allowing the operating systems to function simultaneously.

d. The Hardware Layer: consists of computer hardware and/or computer software products that are specifically designed for the delivery of cloud services, including multi-core processors, cloud-specific operating systems and combined offerings.

2.2 Job Scheduling in Cloud Computing Environment

The requirements for large resources and the presence of multiple replicas of these resources scattered at geographically-distributed locations makes job scheduling in the cloud different from the other kind of computing environments. When we come to the economic perspective of cloud computing, customers only use what they need, and only pay for what they actually use. Resources are available to be accessed from the cloud at any given time, and from any location via the Internet [34].

Cloud computing, a provider of dynamic services, uses very large scale and virtualized resources over the Internet. Resource is an entity consumed or borrowed to accomplish or support activities. Resources in the cloud are job associated. Some virtual resources that are being used by a job at a given moment may only stay in the computing track as far as the period of the job is not over. On the contrary, priorities might be used to assign resources for the seeking job. This kind of prioritization bases on either general rule or the characteristics of jobs.

Factoring more on cloud resources, a job can be identified as an activity that occurs over time. A job can have fixed or variable duration. Usually, it interacts with one or more resources and

consumes or replenishes resource capacity. As well, job reserves all or part of a resource temporarily. A job could interact with other jobs through shared resources and job constraints. Most jobs in the cloud may have temporal constraints based on events. Similar to resources, a given job can be prioritized over the others based on some criteria or constraints.

In multi programmed computing systems, like the cloud, there are a number of jobs competing for the same resource at the same time. The sub system that makes a choice for this kind of situations is called scheduler and the algorithm it uses is called scheduling algorithm [33].

2.2.1 Objective of Scheduling

Generally, schedulers generate the mapping of tasks to resources based on some particular objectives. Schedulers employ a function that takes into account the necessary objectives to optimize a specific outcome. The commonly used scheduling objectives in a cloud computing environment are related to the tasks completion time, load balancing, and resource utilization.

The scheduler uses a specific strategy for mapping the tasks to suitable cloud resources in order to satisfy user requirements. However, the majority of these scheduling strategies are static in nature. They produce a good schedule given the current state of cloud resources and do not take into account changes in resource availability. On the other hand, dynamic scheduling considers the current state of the system. It is adaptive in nature and able to generate efficient schedules, which eventually minimizes the completion time of tasks as well as improves the overall performance of the system.

2.2.2 Salient Scheduling Approaches

Scheduling applications can be classified based on various ways: the scheduling environment, the nature of a scheduling problem, and scheduling strategies.

a. Scheduling Environment

From an environment point of view, there are five types of job scheduling.

i. Job Shop Scheduling

Job shop scheduling is an optimization problem in which ideal jobs are assigned to resources at particular times. It is the problem of scheduling a set of I job types on J machines. Job type i consists of j_i stages, each of which must be completed on a particular machine. The pair (i,j) represents the j^{th} stage of the i^{th} job and has processing time $p_{i,j}$. The completion time of job

i is the completion time of the last stage j_i of job type i [37]. Assuming that we have n_i jobs of type i, the objective is to find a schedule that minimizes the maximum completion time, called the makespan, subject to the following restrictions:

1. The schedule must be non-preemptive. That is, once a machine begins processing a stage of a job, it must complete that stage before doing anything else.
2. Each machine may work on at most one task at any given time.
3. The stages of each job must be completed in order.

ii. Factory Scheduling

Factory scheduling, often termed finite scheduling or finite capacity scheduling (FCS), is a supply chain planning (SCP) technology designed to translate an operating plan into a granular set of daily manufacturing activities to fulfill planned orders. Supporting technology dissects the operating plan into specific work processes, optimizing the allocation of production resources against ongoing change and constraints to produce to the daily schedule [51].

iii. Flexible Manufacturing System Environment Scheduling

Flexible Manufacturing System Environment scheduling is an algorithm to manage highly dynamic systems with considerable product mix and manufacturing environment [52].

iv. Production Scheduling

Production scheduling, which is almost the same as factory scheduling, assigns a set of resources to a variety of production requirements in specified time frames, optimizing performance measure whilst satisfying a set of constraints [51].

V. Management Scheduling

Management scheduling is extended from the foregoing scheduling applications. It is a general management scheduling activity that can be either simple or complicated. Management scheduling can be used in any manufacturing environment or organization.

b. The Nature of a Scheduling Problem

The nature of the entities to be scheduled determines the scheduling approach to be applied. Hence, this Section compares and contrasts different scheduling approaches corresponding to the scheduling problem.

i. Static Scheduling vs. Dynamic Scheduling

Static scheduling means that scheduling is done in a pre-production period or that the scheduled environment is static. This kind of scheduling is done at compile time, in which the characteristics of a parallel program are known before the program execution [37].

Dynamic scheduling, on the other hand, means that scheduling is executed in a volatile environment, which necessitates frequent changes to requirements. The ability to efficiently migrate virtual machines between servers or data centers is crucial for the efficient and dynamic resource management [38].

ii. Online Scheduling vs. Offline Scheduling

Online scheduling implies that while production is in progress, scheduling is being done at the same time without influencing the production process. Offline scheduling, however, takes place when scheduling is completed in production down time [53].

iii. Real Time Scheduling vs. Dynamic Scheduling

Real time scheduling means that a scheduling activity is executed in a real time environment. All these activities can vary with changes in time or constraints. Dynamic scheduling, in its turn, means that the scheduling activity has to be revised, owing to the dynamic changes created within an environment.

c. Scheduling Strategy

This Section compares and contrasts various scheduling strategies.

i. Rescheduling vs. Scheduling

Scheduling means to perform a list of tasks on a given timetable. Rescheduling, on the other hand, means repeating previous scheduling activities under new scheduling strategies because of the unsatisfactory scheduling results obtained from the previous scheduling.

ii. Reactive Scheduling vs. Predictive Scheduling

Predicative scheduling constitutes the process by means of which times and machines are fixed for all operations of all orders before starting production. This method overcomes the dynamic workload fluctuations and resource conflict problem [39].

Reactive scheduling, in its turn, pertains to the process of revising an existing schedule that reacts to unexpected events. This process can also be viewed as an endeavor to reschedule a set

of jobs under modified constraints. It is an extension of predictive scheduling in a dynamic environment. Whenever reactive scheduling is warranted, immediate response on scheduling can be effected reactively. From the attitudes point of view, this is a positive on time scheduling activity.

iii. Interactive Scheduling vs. Active Scheduling

Interactive scheduling has internal knowledge and responsibility to cope with any changes introduced from outside, and to effect local scheduling without adversely affecting the entire schedule timetable.

Active scheduling is a continuous rescheduling process based on predictive scheduling. Rescheduling is a continuous process whereby the schedule can be improved and new events can be integrated as and when they occur. The rescheduling process takes care of such new data immediately [54].

IV. Intelligent Scheduling vs. Knowledge Based Scheduling

Intelligent scheduling is a wide ranging scheduling concept. The term "intelligent", however, could have various meanings, as a scheduling response could be either interactive or reactive, the problem or environment could be dynamic or real time, the method or solution could be knowledge based, and the scheduler could be intelligent based on the design of knowledge, and so forth. Intelligent scheduling can, ultimately, be defined as a comprehensive, high level scheduling process effected with the latest knowledge, which relates to a complex environment.

Knowledge based scheduling means that the scheduling solution is based on a knowledge based design, such as knowledge based analysis of problems, knowledge based methods, knowledge based strategies, knowledge based schedulers and knowledge based system design. For this reason, knowledge based scheduling is a more rational, intelligent and altogether better form of scheduling.

2.2.3 Service Scheduling in Cloud

As a prominent characteristic of resource management, service scheduling makes cloud computing different from other computing paradigms. Since cloud service is actually a virtual product on a supply chain, the scheduling can be classified into two basic categories: user-level and system-level [55]. The user-level scheduling deals with the problem raised by service provision between providers and customers. It mainly refers to economic concerns such as

equilibrium of supply and demand, competition among consumers and cost minimization under elastic consumer preference. On the other hand, the system-level scheduling handles resource management within a data center.

Data center consists of many physical machines which accept millions of tasks handed from the user. Besides, resource management has a significant impact on the performance of a data center. Other requirements such as Quality of Service (QoS), Service Level Agreement (SLA), resource sharing, fault tolerance, reliability, real time satisfaction, etc. are the major considerations during resource management [56].

Furthermore, service provisioning in cloud is based on SLA. It represents a contract signed between the customer and the service provider stating the terms of the agreement including non-functional requirements of the service specified as QoS, obligations, and penalties in case of agreement violations. Thus, there is a need of scheduling strategies considering multiple SLA parameters and efficient allocation of resources.

2.2.4 Scheduling Problems in Cloud and Schematic Methods

Scheduling problem is the problem of matching elements from different sets, which is formally expressed as a triple (E, S, O), where E is the set of examples, each of which is an instance of a problem, S is the set of feasible solutions for E, and O is the object of the problem. As it is shown in Figure 2-5 [55], we can classify scheduling problems into two namely: optimization problem or NP-hard problem and decision problem or easy problem.

An optimization problem requires finding the best solution among all the feasible solutions in set S. On the other hand, decision problem requires a specified feasible solution $s \in S$, problem needs a positive or negative answer to whether the object O is achieved. Obviously, optimization problem is harder than decision problem, because the specified solution only compares with one threshold solution in decision problem, instead of all feasible solutions in optimization problem.

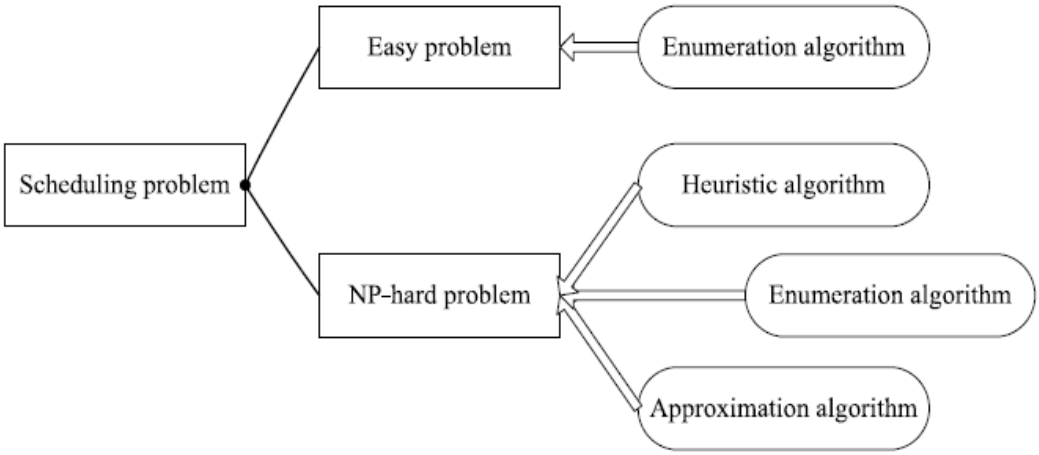

Figure 2-5: Schematic Methods

Optimization problems can be solved by enumeration method, heuristic method or approximation method [56]. In enumeration method, an optimal solution can be selected if all the possible solutions are enumerated and compared one by one. Exact enumerative algorithms have exponential time complexity in the worst case. However, for some Non-deterministic Polynomial-time hard (NP-hard)[4] problems in weak sense, when the number in one instance is relatively small, it can be solved by a pseudo-polynomial algorithm, the time complexity of which is bounded by a polynomial expression of the input size and the maximum number of the problem.

Moreover, there is another kind of enumeration, called implicit enumeration, which evaluates all possible solutions without explicitly listing all of them. Dynamic programming is a practicable implicit enumeration method to solve combinational optimization problems. Exhaustive enumeration is not feasible for scheduling problems, because only a few special cases of NP-hard problems have exactly-solvable algorithms in polynomial time [55].

In that case heuristic is a suboptimal algorithm to find reasonably good solutions which are reasonably fast. It iteratively improves a candidate solution with regard to a given measure of quality, but does not guarantee the best solution. To be more precise, approximation rate rH(e) is introduced to evaluate the accuracy of heuristic algorithms [55].

[4] It is a class of problems that are, informally, "at least as hard as the hardest problems in Non-deterministic Polynomial-time ".

$$r_H(e) = \frac{H(e)}{OPT(e)} \quad (1)$$

where H(e) is the value of the solution constructed by heuristic H for instance e, and OPT(e) is the value of the optimal solution for e.

If there is an integer K, all the instances satisfy OPT(e) ≥ K. This asymptotic ratio rH can be used to measure the quality of approximation algorithm. The closer rH approaches one, the better the performance is achieved by heuristics.

Approximation algorithms are used to find approximate solutions to optimized solution. These algorithms are used for problems when exact polynomial time algorithms are known. It uses relaxing some constraints imposed on the original problem so that the solution might be easy to obtain and have a good approximation to that in the original problem.

2.2.5 Non Functional Requirements of Cloud Services

Cloud computing represents a solution for applications with high scalability needs where usage patterns, and therefore resource requirements, may fluctuate based on external circumstances such as exposure or trending. However, in order to take advantage of the cloud's benefits, software engineers need to be able to express the application's needs in quantifiable terms. Additionally, cloud providers have to understand such requirements and offer methods to acquire the necessary infrastructure to fulfill the users' expectations.

This Section discusses the prominent non-functional requirements of cloud services stated as QoS.

a. Response Time

This requirement describes how much time it takes from the moment a user sends a request to the system until a complete response is provided [57]. In web applications, response time is affected by request transmission and processing, and response transmission. The factors that account for it are resource capabilities, processing power, memory, disk, network latency and the load produced by other processes running in the server or the number of concurrent requests.

For complex requests, this may also involve calls to external systems, or to other subsystems, in which case the host's internal network characteristics and other resources' load may be taken into account.

b. Availability

This is the total time the service is available. When considering this requirement, it is necessary to take into account the provider's own uptime. For example, if a provider has an uptime of 99.5%, it would be impossible to deploy an application with a higher uptime [58]. Recoverability is also another factor that accounts for uptime.

c. Requests Per Unit of Time

This requirement describes the number of requests the system can handle successfully per unit of time, and can also be referred to as the system's throughput. Resource allocation and usage has an impact in this parameter. Additionally, the number of requests can have an inverse relationship with the response time for the service [57].

d. Fault tolerance

It is the ability to operate under erroneous conditions [57]. In the case of the cloud, non-software errors can be generated either at the physical or the virtual machines hosting the service. While the first case is usually out of the developer's control, virtual machine faults can be handled by different means, for example, by spawning new instances, or having backup VMs to respond to failures.

2.3 Summary

In this Chapter we have reviewed literatures on cloud computing and supportive technologies. Moreover, the major characteristics and architecture of the cloud, enabling technologies, and various models of cloud computing environment are presented. The various types of scheduling algorithms along with their objective are also discussed.

Furthermore, Concepts which could facilitate the formulation of the solution domain are acquired. Literatures revised and as well summarized in the early sections of this research work happens to have a greater contribution for the effort to know cloud computing and its salient concepts. Similarly, literatures written about scheduling algorithms in general and scheduling in cloud computing environment improved our level of understanding on domain specific topics.

3. Related Work

This Chapter begins with an overview of scheduling algorithms, demands and illustration of the current scheduling problems. The second part briefly describes various contributions related to our current research work. Mainly, this part is divided into swarm optimization algorithms and nature independent scheduling algorithms. The first section describes algorithms which are inspired by the natural problem solving behavior of social animals. On the other hand, the later describes some job scheduling algorithms derived by various mathematical problem solving techniques and/or models.

Finally, the common gaps of the reviewed works as well as the way the newly proposed algorithm fills those gaps is described briefly.

3.1 Overview

Job scheduling is one of the core and challenging theoretical issues in a cloud computing environment [59]. In this environment, the scheduler needs to consider virtualized resources and users' required constraint to get better match between applications and resources. When users consider a variety of network resources related to quality of service, job scheduling becomes more complicated. Furthermore, millions of users share cloud resources by submitting their computing tasks to the cloud system. Consequently, scheduling these millions of tasks is a challenge to cloud computing environment.

The development of cloud computing requires technology about virtualization and resource allocation for which job scheduling plays a vital role. For cloud vendors to upraise, they should have a competent scheduling algorithm, an algorithm that achieves high system throughput, improves the load balance and minimizes the total job processing time while matching the job requirements with available virtualized resources. Besides that, efficient utilization of resources must be important and for that scheduling plays a vital role. Similarly, scheduling algorithms play a significant role to get maximum benefit from cloud resources. Consequently, developing efficient cloud computing resource dispatcher and increasing user satisfaction with the upraised scheduling technology is among the ultimate goals of cloud vendors.

Similarly, optimal resource allocation or task scheduling in the cloud should decide optimal number of systems required in the cloud so that the total cost is minimized and the Service Level

Agreement (SLA) is upheld. Cloud computing is highly dynamic, and hence, resource allocation problems have to be continuously addressed, as servers become available/non available while at the same time the customer demand fluctuates.

In recent years, scheduling research has had an increasing impact on practical problems, and a range of scheduling techniques have made their way into real world application development. With these mounting successes and advances, it might be tempting to conclude that the chief technical hurdle underlying the scheduling problem has been overcome. However, such a conclusion presumes a rather narrow and specialized interpretation of scheduling. It ignores much of the process and broader context of scheduling in most practical environments.

Consequently, job scheduling has always been a core research area and hence a plethora of papers related to this topic can be found in the past decades. However, the novelty of the cloud and the excess communication cost prevents well known task schedulers applicability. In addition, most job scheduling systems in cloud computing only considers how to increase job scheduling efficiency or how to meet the Quality of Service (QoS) requirements. They rarely give description which considers the importance of the aforementioned two aspects together. Besides that, most scheduling algorithms do not consider the network impact of cloud applications.

What follows are summaries of the major contributions of known job scheduling algorithms for cloud computing. For simplicity, this topic is organized into two major sub topics. The two categories are mainly differentiated based on their similarity with the collective problem solving skill of social animals. Those algorithms inspired by the natural problem solving skill of social animals is stated under the title Swarm Optimization Algorithms (SOA). On the other hand, algorithms that are derived by some mathematical models and are not dependent on the nature of social animals are described under Nature Independent Scheduling Algorithms (NISA).

3.2 Swarm Optimization Algorithms

Swarm intelligence (SI) is the collective behavior of decentralized, self-organized systems, natural or artificial [60]. Specifically, it is defined as the collective problem-solving capability of social animals. It is also the direct result of self-organization in which the interactions of lower-level components create a global-level dynamic structure that may be regarded as intelligence. These lower level interactions are guided by a simple set of rules that individuals of the colony

follow without any knowledge of its global effects. Various job scheduling algorithms inspired by swarm intelligence are described below.

3.2.1 Genetic Algorithms (GA)

Genetic Algorithms are a kind of Evolutionary Algorithms (EA). EA's main strategy is to find the optimal points by utilizing the stochastic search operators such as natural selection, mutation and recombination to the population. In addition, it exploits historical information to speculate on new search areas with improved performance [61]. Several forms of these algorithms have been introduced such as evolutionary strategy, evolutionary programming, genetic algorithm and so on.

Genetic algorithm is often used to solve job scheduling problems in the cloud. It is a search heuristic that mimics the process of natural evolution based on a population of candidate solutions. It is routinely used to generate useful optimized solutions. There are a number of papers inspired by genetic algorithms.

For instance, [62] used Improved Genetic Algorithm (IGA) for the automated scheduling policy. It employs the use of shortest genes and dividend policy in economics to select an optimal or suboptimal allocation for the VM requests. This optimized algorithm is used for the automated scheduling policy. The research mainly basis on the deep research on Infrastructure as a service cloud systems. Besides that, the authors implemented flexible Virtual Machine (VM) allocation technique assuming it can yield maximum usage of physical resources. Moreover, the simulation experiments indicate that the dynamic scheduling policy performs much better than that of the Eucalyptus, Open Nebula, Nimbus IaaS cloud, etc. The tests illustrate that the speed of the IGA is almost twice the traditional GA scheduling method in Grid environment and the utilization rate of resources always higher than the open-source IaaS cloud systems.

Similarly, [63] proposes an adaptive, multi-objective job shop scheduling algorithm to deal with internal and external disruptions faced in real life cloud environment. In addition, the authors developed an asexual reproduction genetic algorithm with multiple mutation strategies to solve the multi-objective optimization problem. Experimentation was conducted in two stages. In the first stage, single day experimentation is conducted. The purpose of this experiment is to validate the effectiveness of the proposed model in finding effective solutions to optimize the selected objectives. Population size, number of generations, mutation 1 rate, and mutation 2 rate are the

input parameters considered for this testing. Moreover, best among all, best among pair and a combination of s1 and s2 are used as the three major selection strategies in the research work. The second problem is a four job, three machine problem adapted from other related papers. The due date were formed based on the total processing time of the individual jobs. The proposed model were compared with other adaptive scheduling algorithms. The authors were able to acquire better optimized solution on the mean tardiness for jobs to effectively meet customer due date requirements and minimizes mean flow time to reduce the lead time jobs spend in the system. However, the algorithm is evaluated manually which obviously leads to incorrect deduction.

Most genetic algorithms have a local search problem. In [64], the local search ability of genetic algorithms is improved by implementing annealing after the selection, crossover and mutation. Besides, the paper mainly considers QoS requirements like completion time, bandwidth, cost, and reliability of different types of tasks. Here annealing is implemented after the selection, crossover and mutation, to improve local search ability of genetic algorithm.

Similarly, the work in [65] presented an optimized algorithm for task scheduling based on Activity Based Costing (ABC). The assumption in this paper is the development of a scheduling algorithm that optimizes all the performance parameters of the system and users improve performance of the system. The priority level for each task is assigned and uses cost drivers.

Furthermore, in cloud computing environment, there are intensive workflows constrained by cost. The authors in [66] proposed a solution for this kind of cloud problem. The assumption in this paper is once we know the execution cost in relation to execution time then optimizing will be a routine task. The algorithm is managed to minimize the cost under certain user designated constraints. In addition, computational capability of virtual machines is considered though not as expected.

Genetic algorithms impose a random initial population generation technique. Consequently, these populations are not so much fit. Furthermore, when these primary populations (schedules) are further mutated with each other, there are very much less chances that they will produce better child than themselves. Breakthrough to fill the aforementioned gap of GA is proposed in [67]. This algorithm uses both Min-Min and Max-Min techniques to generate the initial robust populations.

On the other hand, the work in [68] calls GA scheduling function in every cycle of scheduling. This function creates a set of task schedules and evaluates the quality of each task schedule with user satisfaction and virtual machine availability. The function iterates the genetic operation to make it optimal. Throughput, simulation time, average virtual machine utilization, average processing cost and number of tasks are all the parameters used in this paper. However, it is observable that the iterative call of GA function causes a tremendous degradation of scheduling performance.

Nowadays, finding a scheduling algorithm which is not constrained by tradeoffs between resource utilization and execution time is rare. However, some papers such as [69] designed a promising algorithm to perform the abovementioned problem. In this paper, the scheduler instantiates one task per virtual machine, assuring that at any given time, every host processing will execute one virtual machine instance. The nodes are given by a cluster. Each cluster has an independent master task scheduling node and the slave task assigned node which derives from a cluster under the control of the master node. The designation between master and slave tasks is differentiated by the type and demand of the incoming jobs.

Even though the nodes are from the same cloud they might have different hardware and scheduling is done without flaws. The master node is responsible for scheduling all the tasks, monitoring their implementation, re-run the task that has failed, or disposing of errors. The master node collects information about the nodes that are participating, their memory and processing capacity and transmission rate. The slave node is responsible for executing the task assigned by the master node. Upon receipt from the assignment of the master node, the slave node starts to find a suitable computing node. Data is distributed through these nodes. It also considers the work load on each node. Similarly, the algorithm balances the computation of nodes. However, the communication between cluster level task scheduler and the availability of only one master node might yield poor execution time and failure respectively.

Generally, genetic algorithms may reserve previous populations that may help in minimizing the number of executions that may take to reach to candidate solutions. However, this situation slows the converge since consecutive populations may die out, i.e., these kinds of algorithms are slower in finding optimal solutions due to the need of handling population variations.

Furthermore, since task assignment is an NP-complete problem, Genetic Algorithms are not the best.

3.2.2 Particle Swarm Optimization (PSO)

Particle swarm optimization is a heuristic global optimization method and also an optimization algorithm, which is based on swarm intelligence. It comes from the research on bird and fish flock movement behavior [71]. PSO consists of a population called swarm and each member of the swarm is called a particle [70]. The particles search the global optimum with a set of velocity. Since the particles modify and update the position with respect to itself and its neighborhood, it has the capability to do both local and global searches [72].

One among many researches inspired by particle swarm optimization is the work in [73]. This paper is adopted to solve task scheduling problem in grid environment. Each particle represents a possible solution and the position vector is transformed from the continuous variable to the discrete variable. This approach aims to generate an optimal schedule so as to get the minimum completion time while completing the tasks. The results of simulated experiments show that the particle swarm optimization algorithm is able to get a better schedule than genetic algorithms.

Similarly, the work in [74] presented a particle swarm optimization based heuristic to schedule applications to cloud resources that takes into account both computation cost and data transmission cost. It is used for workflow application by varying its computation and communication costs. The first step in this algorithm is calculating the average computation cost of all tasks on all the compute resources. This cost can be calculated for any application by executing each task of an application on a series of known resources. As the computation cost is inversely proportional to the computation time, the cost is higher for those resources that complete the task quicker. The average value of communication cost between resources per unit data is represented in a persistent model. Given the inverse relationship between communication and time the authors considered the size of input and output data of each task. In the first phase of allocation computation of the mapping of all tasks in the workflow, irrespective of their dependencies is done. This mapping optimizes the overall cost of computing the workflow application. To validate the dependencies between the tasks, the algorithm assigns the “ready” tasks to resources according to the mapping given by PSO. After dispatching the tasks to resources for execution, the scheduler waits for polling time. This time is for acquiring the status

of tasks, which is middleware dependent. Depending on the number of tasks completed, the ready list is updated, which will now contain the tasks whose parents have completed execution. The average values for communication between resources according to the current network load is updated successively. As the communication costs would have changed, the re computation of the PSO mappings is done. Experiment is conducted by feeding jobs of varying sizes. Also, the execution cost of each task varies among all the compute resources used. Moreover, the performance of the proposed heuristic is analyzed by varying each of these in turn. The average results of the proposed algorithm is used to present the results of the acquired from the experiment. The experimental results show that PSO can achieve cost savings and good distribution of workload onto resources.

In addition, the work in [75] presented a new task assignment algorithm that is based on the principles of particle swarm optimization (PSO). It follows a collaborative population-based search, which models over the social behavior of bird flocking and fish schooling. PSO system combines local search methods (through self-experience) with global search methods (through neighboring experience), attempting to balance exploration and exploitation. The paper discussed the adaptation and implementation of the PSO search strategy to the task assignment problem. The effectiveness of the proposed PSO-based algorithm is demonstrated by comparing it with the genetic algorithm, which is well-known population-based probabilistic heuristic, on randomly generated task interaction graphs. Simulation results indicate that PSO-based algorithm is a viable approach for the task assignment problem.

All through the above description of PSO based algorithms, it is obvious to notice its better computational speed over genetic algorithms. The main difference between the PSO approaches compared to GA is that PSO does not have genetic operators such as crossover and mutation. Particles update themselves with the internal velocity; they also have a memory important to the algorithm. Also, in PSO only the ‘best’ particle gives out the information to others. It is a one way information sharing mechanism, the evolution only looks for the best solution. Compared to GAs, the advantages of PSO are that PSO is easy to implement and there are few parameters to adjust.

However, PSO cannot work out the problems of scattering and optimization [77]. Besides, it is a continuous technique that is poorly suited to combinatorial problems. Lack of mathematically proved speed of the convergence is the other shortcoming of PSO [76].

3.2.3 Ant Colony Optimization (ACO)

Ant Colony Optimization (ACO) was inspired by the pheromone-based strategy of ants foraging in nature. The foraging behavior of ants is based on finding the shortest path between source and their nests. During the foraging process, ants leave their pheromone trails on the path when they return to their nest from the source, so the other members of the colony find the path by using the pheromone trails and pheromone level. If the selected path is the shortest path, then the pheromone level will be reinforced; otherwise it will evaporate as time passes.

The work in [78] is one of the ant colony inspired algorithms that monitors real time virtual machines on performance parameters and schedules fast resources. By real-time monitoring virtual machine of performance parameters, once judging overload, it schedules fast cloud resources using ant colony algorithm to bear some load on the load-free node. This algorithm is designed in a way it can fairly balance load among various resource centers. It finds the nearest idle node and allows it to bear some load meeting the performance and resource requirements of load thus achieving the goal of load balancing. The model is analyzed and meets the goals and requirements of self-adaptive cloud resources scheduling and improve the efficiency of the resource utilization.

In [79], a multi-objective ant colony algorithm for the virtual machine placement problem is proposed. The goal is to efficiently obtain a set of non-dominated solutions (the Pareto set) that simultaneously minimize total resource wastage and power consumption. A feasible and complete solution of the formulated multi-objective VM placement problem is considered as a permutation of VM assignment. The algorithm initially initializes the parameters and all the pheromone trails are set to $\tau 0$. In the iterative part each ant receives all VM requests, introduces a physical server and starts assigning VMs to hosts. This is achieved by the use of a pseudo-random-proportional rule, which describes the desirability for an ant to choose a particular VM as the next one to pack into its current host. This rule is based on the information about the current pheromone concentration on the movement and a heuristic which guides the ants towards choosing the most promising VMs. A local pheromone update is performed once an artificial ant

has built a movement. After all ants have constructed their solutions, a global update is performed with each solution of the current Pareto set. The performance of the proposed ant algorithm is compared to that of a multi-objective grouping genetic algorithm, a single-objective ACO (SACO) algorithm and a single-objective FFD algorithm. The programs for the proposed algorithm, MGGA algorithm and FFD heuristic were coded in the Java language and ran on an Intel Pentium® Dual-Core processor with 2.50 GHz CPU and 3 GB RAM. The settings for various parameters in VMPACS have a direct effect on the algorithm performance. Appropriate parameter values were determined on the basis of preliminary experiments. The final parameter settings were determined to be NA = 10, M = 100, $\alpha = 0.45$, $\rho l = \rho g = 0.35$, and $q0 = 0.8$. In the case of the MGGA algorithm the population size is 12. The initial population was generated randomly. The crossover rate is 0.7 and the mutation rate is 0.05. The maximum number of generations for each search process is 10.

With the above configuration, problem instances are randomly generated. The instances were a demand set of CPU and memory utilizations for 200 VMs. The number of servers was set to the number of VMs in order to support the worst VM placement scenario, in which only one VM is assigned per server. The simulation is done on the homogeneous server environments which could be mentioned as the major problem. Moreover, after the VM placement algorithm was finished, if there were several non-dominated solutions, a solution belonging to the set of non-dominated solutions was randomly chosen. Every test was repeated with 20 runs for each instance and the average results over 20 independent runs are reported. The paper also introduced the linear correlations of CPU and memory utilizations into the instances and used randomly generated sequences of CPU and memory utilizations in the experiments that had several correlations. The proposed algorithm outperforms multi group genetic algorithm from the perspective of power consumption and memory utilization.

Tawfeek *et al.* [80] proposed an algorithm which uses ant colony optimization in which random optimization search approach is used for allocating the incoming jobs to the virtual machines. This algorithm uses a positive feedback mechanism and imitates the behavior of real ant colonies in nature to search for food and to connect to each other by pheromone laid on paths traveled. A good task scheduler should adapt its scheduling strategy to the changing environment and the types of tasks. In this paper a cloud task scheduling policy based on ant colony optimization algorithm compared with different scheduling algorithms FCFS and Round Robin, has been

presented. Algorithms have been simulated using Cloudsim toolkit package. Experimental results showed that the ant colony optimization slightly outperformed FCFS and round-robin algorithms.

Another paper in [81] follows priority based dynamic resource allocation. The assumption here is considering multiple SLA parameters and resource allocations by preemption mechanism for high priority tasks, execution can improve resource utilization in the cloud. Generally, the main highlight of the paper is that it provides dynamic resource provisioning and attains multiple SLA objectives (such as memory, network bandwidth, and required central processor time) through priority based scheduling. This algorithm is recommended for situations where resource contention is brutal and it provides high degree of resource utilization.

Generally, ant colony optimization algorithm is a novel stochastic search technology, which simulates the social behavior of ant colony. It can be applied to complex combinatorial optimization problems and achieve good results. However, this algorithm has shortcomings such as slow evolution, falling in to local optimum, and so on.

3.2.4 The Bees-Inspired Algorithm (BA)

In nature, honey bees have several complicated behaviors such as mating, breeding and foraging. These behaviors have been mimicked for several honey bee based optimization algorithms. One of the famous mating and breeding behavior of honey bees inspired algorithm is Marriage in Honey Bees Optimization (MBO) [85]. The algorithm starts from a single queen without family and passes on to the development of a colony with family having one or more queens. The other type of bee-inspired algorithms mimic the foraging behavior of the honey bees. These algorithms use standard evolutionary or random explorative search to locate promising locations. Then the algorithms utilize the exploitative search on the most promising locations to find the global optimum. Generally, bees-inspired algorithm is an improved version of the Genetic Algorithm (GA). The main purpose of the algorithm is to improve local search while keeping the global search ability of GA.

Bee Colony Optimization (BCO) was proposed to solve combinatorial optimization problems in [83]. BCO has two phases called forward pass and backward pass. A partial solution is generated in the forward pass stage with individual exploration and collective experience, which will then be employed at the backward pass stage. In the backward pass stage the probability information

is utilized to make the decision whether to continue to explore the current solution in the next forward pass or to start the neighborhood of the new selected ones. The new one is determined using probabilistic techniques such as the roulette wheel selection.

In [84], a population-based meta heuristic inspired by the natural foraging behavior of honey bees is proposed. It starts with a number of scout bees being placed randomly in search space. The best sites visited in point of view fitness are chosen as selected bees which will be foraged their close sites to carry out a neighborhood search. Two standard functional optimization problems were used to test the Bees Algorithm and establish the correct values of its parameters and another eight for benchmarking the algorithm. As the Bees Algorithm searches for the maximum, functions to be minimized were inverted before the algorithm was applied. The following parameter values were set for this test: population n= 45, number of selected sites m=3, number of elite sites e=1, initial patch size ngh=3, number bees around elite points nep=7, number of bees around other selected points nsp=2.

On the other hand, the work in [85] presents a scheduling algorithm based on the reproduction behavior in bees colony. It is called marriage in honey bees optimization algorithm. The algorithm starts with three user–defined parameters; these are, the queen's spermatheca size representing the maximum number of matings in a single mating–flight, the number of broods that will be born by the queen, and the amount of time devoted to brood care signifying the depth of local search or number of attempts made to improve a solution. At the start of a run, the queen is initialized at random. The worker is then used to improve the queen's genotype. Afterwards, a number of mating–flights are carried out. In each mating–flight, the queen moves between the different states in the space based on her energy and speed, where both are generated at random before each mating–flight starts. At the start of a mating–flight, a drone is generated at random and the queen is positioned over that drone. The transition made by the queen in the space is based on her speed which represents the probability of flipping each bit in the drone's genome. Therefore, at the start of a mating–flight, the speed is usually high and the queen makes very large steps in the space. While the queen's energy decreases, the speed decreases. Subsequently, the neighborhood covered by the queen decreases. In each step made by the queen in the space, the queen is mated with the drone encountered at that step using the probabilistic rule If the mating is successful (ie. the drone passes the probabilistic decision rule), the drone's sperm is stored within the queen's spermatheca. Each time a drone is generated, half of his genes are

marked at random since each drone is haploid by definition. Therefore, the genes that will be transmitted to the broods are fixed for each drone. After the queen completes her mating–flight, she starts breeding. For a required number of broods, the queen is mated with a randomly selected sperm from her spermatheca. The worker is then used to improve the resultant brood. After all broods are being generated, they are sorted according to their fitness. The best brood replaces the queen if the brood is fitter than the queen. All broods are then killed and a new mating–flight is undertaken until all mating–flights are completed or an assignment that satisfies all clauses is encountered. The behavior of the proposed model is scrutinized and the results are compared against random walk scheduling algorithm. The annealing stage undertaken by the queen during her mating–flight in conjunction with the use of haploid–crossover and mutation, proved useful in the set of SAT problems solved here. In addition, a small to average number of broods produced better results.

In most cases, bees colony algorithm is best for optimization over all other swarm intelligence algorithms. However, these algorithms still are not better in selecting the nearest resources.

3.3 Nature Independent Scheduling Algorithms

In this part, algorithms which are not inspired by the behavior of social animals are described. Moreover, for the sake of better understandability we categorized them based on their major purposes namely; load balancing, execution time, resource utilization and availability & reliability.

3.3.1 Load Balancing

Cloud load balancing is a type of load balancing that is performed in cloud computing [86]. It is the process of distributing workloads across multiple computing resources. Cloud load balancing reduces costs associated with data management systems and maximizes availability of resources. What follows is the description of researches aiming at increasing the efficiency of load management in cloud computing systems.

Among many contributions, the work in [87] developed a load balancing framework for high-performance clustered storage systems. It provides a general method for reconfiguring a system facing dynamic workload changes. It simultaneously balances load and minimizes the cost of reconfiguration. The framework is evaluated in two different ways. First, using a description of a weeklong workload from an internally deployed system, the various implemented functions and optimization techniques are explored. Second, the model in a real-world scenario—a recent hardware upgrade and consolidation effort in one of our data centers are examined. Optimization methods for various imbalance and cost functions using a data center storage consolidation scenario is compared. In this scenario, the effects of taking an existing stand-alone system with e-mail workload from MS Exchange Servers and integrating it into the existing six-node system is analyzed. The first step involves rolling the existing hardware with its data into a cluster. The second step, and the one targeted by the framework, involves moving data between the nodes of the combined system to achieve a more balanced load. This framework happen to have practical contribution on using high-level workload descriptions from periodic collections of performance data, its applicability to real-world scenarios for consolidating data center storage, and the use of high-level empirical performance models. The effectiveness of this framework is demonstrated by balancing the workload on different cloud system simulators.

Another job scheduling algorithm that aims at improving the load management and resource utilization is presented in [88]. This paper presented the implementation of an efficient QoS based meta-scheduler and backfill strategy based light weight virtual machine scheduler for dispatching jobs. The user centric meta-scheduler deals with selection of proper resources to execute high level jobs. The system centric Virtual Machine (VM) scheduler optimally dispatches the jobs to processors for better resource utilization. The authors also present their thoughts on scheduling heuristics that can be incorporated at data center level for selecting ideal host for VM creation. The implementation can be further extended at the host level, using Inter VM scheduler for adaptive load balancing in cloud environment.

A similar scheduler presented in [12] considers the load of each resource center while dispatching jobs. The principle of this threshold based technique is to efficiently allocate the cloud computing resources where user's application changes its resource requirement dynamically with respect to time. Resource allocation policies are concerned with efficient utilization of resources for optimum use of cloud server capacity. A cloud system is a collection of data centers located in different parts of the world, in which each data center is a collection of host servers. During the allocation of resources to a cloud service or cloud application, scheduling algorithms should be applied on the both levels namely, data center level and host server level. At the data center level, appropriate data center is chosen where host server having enough resources to fulfill the dynamic requirement of application with respect to time exists. Whereas, at the host server level resource allocation to the application being scheduled, is considered. Two concepts of virtual machine scheduling algorithms are applied. First concept is related to selection of a data center that is having some of the host server with free resources to assign to the new cloud application or cloud service. Second concept is to apply the threshold based comparison of data center limit and host server limit. The simulation of this work is performed using CloudSim toolkit by developing a cloud environment where load balancing and server consolidation algorithms can be applied in various levels. In this work both have been applied on host server level in cloud environment. At host level, the host server provides its resources to virtual machine up to upper threshold as demand comes. In the CloudSim simulator, VmScheduler is used in virtual machine level which works on two policies namely TimeShared and SpaceShared. In the VM level, each virtual machine divides the resources among Cloudlets

running on it using cloudletScheduler which also uses two policies namly TimeShared and SpaceShared. CloudSim simulation tool of cloud computing is java based tool to create cloud environment with data center and host server. Entire functionality of CloudSim toolkit is implemented in java classes to perform cloud operation. Java classes of datacenter, datacenterBroker, datacenterCharacterstics, host, vm,vmAllocationPolicy, vmScheduler, cloudlet and cloudletScheduler are the main classes of the CloudSim simulator. In the first step of establishing cloud environment, datacenters are created using the host server having its characteristics and physical host server and its resources description. However, the authors implemented an iterative threshold checker in between two successive job allocations. This in turn causes a high reconfiguration cost. In addition, it affects the efficiency of the model.

A two level task scheduling mechanism based on load balancing is presented in [11]. The first level scheduling is from the users' applications to the virtual machine, and the second is from the virtual machine to host resources. The first level scheduler creates a VM description and the second level scheduler finds the appropriate host to the VM according to the task description. The problem with this approach is that they schedule the virtual machine to the host with lightest load each time. Besides, task response time and demand for resources are considered in the paper, i.e., machines with highest load might not participate in the scheduling process even though they are able to accept smaller Jobs. Hence, this model is prone to ill resource utilization.

A load balancing algorithm which considers different processing powers of VMs and assigns the tasks/requests to the most powerful VM and then to the lowest and so on is presented in [89]. Hence, the amount of hardware resources available to each VM is constrained by the total processing power available within the host. The choice of virtual machine is based on Central Processing Unit (CPU), memory, storage, bandwidth, etc. that is optimal for an application.

Verma *et al.* [90] investigated the design of a power-aware application placement controller (pMapper) to ensure a workload placement on heterogeneous virtualized server clusters. The assumption here is that the performance of the workloads can be characterized only by CPU utilization. It takes into account the cost-aware application placement problem as well as minimizing power subject to a fixed performance requirement. The authors claim the superiority of their proposal under most settings over other power unaware algorithms as well as power aware algorithms both theoretically and experimentally.

3.3.2 Minimum Computational Time

Topcuoglu *et al.* [91] presented two novel scheduling algorithms for a bounded number of heterogeneous processors with an objective to simultaneously meet high performance and fast scheduling time, which are called the Heterogeneous Earliest-Finish-Time (HEFT) algorithm and the Critical-Path-on-a-Processor (CPOP) algorithm. The HEFT algorithm selects the task with the highest upward rank value at each step and assigns the selected task to the processor, which minimizes its earliest finish time with an insertion-based approach. On the other hand, the CPOP algorithm uses the summation of upward and downward rank values for prioritizing tasks. Another difference is in the processor selection phase, which schedules the critical tasks onto the processor that minimizes the total execution time of the critical tasks. In order to provide a robust and unbiased comparison with the related work, a parametric graph generator was designed to generate weighted directed acyclic graphs with various characteristics. The comparison study, based on both randomly generated graphs and the graphs of some real applications, shows that the proposed scheduling algorithms slightly surpass previous approaches in terms of both quality and cost of schedules, which are mainly presented with schedule length ratio, speedup, frequency of best results, and average scheduling time metrics.

In addition, the purpose of the work in [92] is to propose an approach for the storage and analysis of very large images on the cloud. This service has been implemented using multiple distributed and collaborative agents. For this purpose, a region-oriented data structure is utilized, which allows storing and describing image regions as jobs to be carried out. The basic idea is to decrease processing time for parallelized tasks using an appropriate load balancer to distribute and assign tasks to agents with less workload.

Parsa and Entezari-Maleki in [93] proposed a new task scheduling algorithm named RASA. It is composed of two traditional scheduling algorithms; Max-Min and Min-Min. RASA uses the advantages of Max-Min and Min-Min algorithms. The experimental results show that RASA outperforms the existing scheduling algorithms in large scale distributed systems. It is used to reduce make span. However, the deadline of each task, arriving rate of the tasks, cost of the task execution on each of the resources, and cost of the communication are not considered.

The Min-Min algorithm in [14] starts with a set of all unmapped tasks. The machine that has the minimum completion time for all jobs is selected. Then the job with the overall minimum

completion time is selected and mapped to that resource. The ready time of the resource is updated. This process is repeated until all the unmapped tasks are assigned. Time complexity of the Min-Min algorithm when we have **R** resources and **T** tasks is $\mathbf{O(T^2R)}$. Hence, Min-Min has a time complexity problem. The scheduler repetitively checks the capacity of each server and incoming jobs which in turn badly affects the performance of scheduling. In addition, tasks estimated for maximum completion time might starve.

In the Max-Min Algorithm [15], the machine that has the minimum completion time for all jobs is selected. Then, the job with the overall maximum completion time is selected and mapped to that resource. The ready time of the resource is updated. This process is repeated until all the unmapped tasks are assigned. This algorithm takes $\mathbf{O(T^2R)}$ time, when we have **R** resources and **T** tasks. The per job level machine selection, which is the main characteristics of this algorithm, constrained the algorithm to have low performance.

The major objective of the scheduling model presented in [5] is to reduce the make span. There will be one parent queue which stores the jobs according to their arrival time, i.e. first in first out approach. Following that the availability of the requested resources will checked and sorting resources into two different queues, available and partially available will be undertaken. The available queue will be checked if the task is dependent or independent and according to that the jobs will be stored in their respective queues. The partially available queue have tasks which will need data resources from other data centers. Then they will be sorted in different queues named cat1, cat2… and so on. This will be done on the basis of resources they need. For example if task1 need resources from data center location1 and data center location3 and similar for task2 and task4. Then tasks 1, 2, 4 will be in one category say cat1. Similarly for other tasks we will make cat2, cat3 and so on till we finish the tasks in the partially available queue.

Once the major queues are constricted as, independent, dependent, and cat1, cat2…..catN three three different queues will further be constracted based on priority, High, Mid, Low. The priority is decided based on four major factors namely time, space, resources and profit.

While calculating priorities of the task, the newly calculated priority will be compared with the previous ones. Then according to that resorting occurs. Tasks from high priority to low priority will be used to perform resource to job mapping. However, the best server selection in this

algorithm takes a large amount of time. It uses maximum number of iterations to find a better server. Moreover, this model does not consider deadline and load balancing constraints.

Similarly, the work in [94] described a system which scales to a large number of storage nodes and objects and aims to minimize latency and bandwidth costs during system reconfiguration. It deals with an optimization problem that selects a subset of objects from hot-spot servers and performs topology-aware migration to minimize reconfiguration costs. The evaluation study achieved cost-effective load balancing and time-responsive in computing placement decisions.

The research in [10] proposed simple and straight forward Greedy algorithm for virtual machine scheduling in the cloud. In this algorithm, the first node with suitable resources identified is allocated the VM. This means that the greedy algorithm exhausts a node before it goes on to the next node. It is easy to implement and also the allocation of VMs does not require any complex processing. However, this technique encounters a major drawback on resource utilization.

In [8], two algorithms were proposed for scheduling tasks into the cloud computing. They consider the computational complexity and the computing capacity of the processing elements to schedule the task. This algorithm works in private cloud environment where the resources are limited. In addition, the load balancing and energy utilization techniques they have implemented have high cost of reconfiguration.

Jobs are randomly assigned to the available VM in [9]. Consequently, any process can be assigned to any VM. This method has no overhead and has very less complexity. The problem with this approach is that it does not consider the size of the process so sometimes it considers a machine which is overloaded.

Dakshayini and Guruprasad [95] proposed a new scheduling algorithm based on priority and admission control scheme. In this algorithm priority is assigned to each admitted queue. Admission of each queue is decided by calculating tolerable delay and service cost. As stated by the authors, this algorithm with the proposed cloud architecture has achieved very high service completion rate with guaranteed QoS. As this policy provides the highest precedence for highly paid user service-requests, overall servicing cost for the cloud also increases.

3.3.3 Degree of Reliability and Availability

Yin *et al.* [13] uses multiple real-time sources of resource availability data, from which it decides to acquire or release resources. This approach only considers how to assign jobs for a single server. However, the cloud is a combination of hundreds or thousands of resource centers.

The work in [96] investigated the effectiveness of rescheduling using cloud resources to increase the reliability of job completion. Specifically, schedules are initially generated using cloud resources. A job in their study refers to a bag-of-tasks application that consists of a large number of independent tasks; this job model is common in many science and engineering applications. They have devised a novel rescheduling technique, called rescheduling, using clouds for reliable completion and applied it to three well-known existing heuristics.

A new fault tolerant scheduling algorithm, MaxRe, is proposed in [97]. This algorithm incorporates the reliability analysis into the active replication schema and exploits a dynamic number of replicas for different tasks. It is stated that both the theoretical and experimental analyses prove that the *MaxRe* algorithm's schedule can certainly satisfy user's reliability requirements. However, the reliability analysis is performed more often than tolerable and the algorithm did not show a better execution time.

Delavar *et al.* [98] proposed a reliable scheduling algorithm in cloud computing environment. In this algorithm a job is divided into sub jobs. In order to balance the jobs the request and acknowledge time are calculated separately. The scheduling of each job is done by calculating the request and acknowledges time in the form of a shared job.

3.3.4 Maximum Resource Utilization

The mechanism to efficiently utilize resources in cloud computing and gain maximum profits is addressed in [7]. The algorithm is also called credit based scheduling algorithm. This algorithm evaluates the entire group of tasks in the task queue and finds the minimal completion time of all tasks. The problem with this model is that it only considers the probability of a resource to be free soon after executing a task so that it will be available for the next waiting, but processing time of a job is not considered.

Ambike *et al.* [99] proposed a differentiated scheduling algorithm with non-preemptive priority queuing model for activities performed by cloud user. In this approach one web application is

created to do some activity like one of the file uploading and downloading then there is the need of efficient job scheduling algorithm. The QoS requirements of the cloud computing user and the maximum profits of the cloud computing service provider are achieved with this algorithm.

The authors in [100] used statistical models to predict resource requirements for cloud computing applications. Such a prediction proposal can help system design to reach scalability, job scheduling, resource allocation, and workload management. Also, they proposed an initial design of a workload generator in order to evaluate alternative configurations without the overhead of reproducing a real workload. They argued that an effective prediction model is a prerequisite for a good workload generator.

3.4 Summary

This Chapter covered the review of cloud schedulers. The most significant contributions are addressed. Mainly, the review is conducted in two major categories depending on the algorithms' similarity to the concept of swarm intelligence. The first category discussed algorithms which are inspired by the natural problem solving skill of social animals. In this category, different forms of algorithms characterizing swarm intelligence namely; GA, PSO, ACO, and BA are described. On the other hand, algorithms which have no swarm intelligence are described in the second category. The review shows the effort of researchers to solve the unaddressed problems of scheduling in cloud and their shortcomings.

Throughout our review, we realized that SOA exhibits a better optimization solution than other types of schedulers. SOA's potential is indeed fast-growing and far-reaching. These algorithms offer an alternative, untraditional way of designing efficient schedulers that neither require centralized control nor extensive preprogramming. The review also showed some of their shortcomings.

For instance, the path ways to solutions in SOA are neither pre-defined nor pre-programmed. Most of the time, the path ways are emergent. Consequently, most SI algorithms are not suitable for time critical resource provisioning [82]. Moreover, lack of parameter tuning techniques is another major drawback of SI algorithms. In fact, since many parameters of SI systems are problem-dependent, they are often either empirically pre-selected according to the problem characteristics in a trial-and-error manner, or even better adaptively adjusted on run time as it is

done in [78] and the particle swarm optimization [74]. In addition, SI systems could suffer from a stagnation situation or a premature convergence to a local optimum.

On the other hand, the nature independent algorithms are described in the second category. The major problems observed in this kind of algorithms can be generalized into three. When we see the execution time of an algorithm, it is mainly affected by a frequent reconfiguration of servers. This kind of shortcoming is observed in [7, 8, 12, 15, 93, 97]. In addition, poor load management which minimizes their contribution on execution time and resource utilization is observed in [5, 9, 14]. Similar, but in the resource utilization aspect, some algorithms have a lower capacity of utilizing resources [9, 10, 11].

Technically speaking, most cloud scheduling algorithms are designed in a way they exhibit either better resource utilization and good load management or smaller execution time. However, to the best of our knowledge, there is no task scheduling and resource provisioning algorithm that can concurrently yield the best degree of resource utilization, load management and execution latency.

Therefore, this research proposes a scheduling algorithm that could not be affected by the existing tradeoff between resource utilization, execution time, and load management.

4. Distributed Score Based Job Scheduling Framework

This Chapter presents the Distributed Score Based Job Scheduling Framework for the cloud. The Chapter is mainly composed of two major parts namely: Design Considerations and System Architecture. The design considerations Section presents various determinants for high performance scheduling algorithm. In the system architecture Section, components of the proposed model along with detail description and complete algorithms are presented.

4.1 Design Considerations

The aim of this design is to propose a job scheduling algorithm that can yield an optimized resource allocation and job dispatching. The main factors considered during the design are summarized below.

a. Resource Utilization

The energy consumption of underutilized resources, particularly in the case of cloud computing environment, accounts for a substantial amount of the actual energy use. Furthermore, a resource allocation strategy that takes into account resource utilization leads to a better energy efficiency. This, in clouds, extends further with virtualization technologies in that tasks can be easily consolidated.

Task consolidation is an effective method to increase resource utilization. It also reduces energy consumption. Besides, resource categorization which has direct impact on the resource utilization is used.

b. Load Balancing

Design of scheduling algorithm for cloud should take the allocation and migration of reconfigurable virtual machines into consideration. Moreover, the scheduling architecture should enclose persistence layer and intermediary evaluation node to balance the load among various resource centers. Similarly, the architecture employs threshold based task allocation component which accounts directly for load balancing and resource categorization.

c. Computational Time

Cost and time factors are in the centre of resource provisioning on a “pay-as-you-go” basis. As a principle, the higher the cost paid by the user, the faster resources could be allocated and the smaller execution time a resource provider could ensure.

In the new scheduling model, two major techniques namely: task consolidation and resource categorization are designed in a way they yield a better computational time. The task consolidation component is employed based on the information generated from the threshold computing node. The threshold computing node generates the job consolidation rate based on the number and attributes of unprocessed jobs and the capacity of servers in each group. The assumption in the latter approach, resource categorization, is once resources are categorized based on their score then it will be easy to locate the appropriate group of servers for similar bunch of incoming jobs.

Besides that, resource reconfiguration rate, which is the major affecting factor of cloud performance is considered. Here, the new approach employed a threshold based reconfiguration interrupt generator to minimize unwanted reconfiguration cost.

d. Fault Tolerance

The architecture uses segment of virtual memory which has been assigned a direct byte-for-byte correlation with some portion of the resource. This memory mapped file will be used to capture the status of tasks to be processed, being processed, and accomplished before system failure. In addition, the persistence model is used to protect the scheduler against failure during resource dispatching. The state of the algorithm is periodically saved in the database. Thus, the scheduler can resume its work from where it was left.

e. Scalability

Most of the Distributed Score Based Job Scheduling Algorithm (DSBJSA) components are independently designed and managed to interact through persistence model. It is carefully designed in a way it can yield high cohesiveness among nodes in the component and low coupling among major components. This facilitates the scalable implementation of the architecture as each component can be distributed across different servers.

Since the architecture handles only the resource allocation and delegates the management of job submission and execution to the participating brokers and providers, thus most of the threads in DSBJSA are light weight and short lived.

4.2 System Architecture

Most scheduling algorithms formulated so far are developed from other preexisting approaches which are purposefully designed to solve provisioning problems of that time and scenario. For instance, scheduling algorithms we have been using for various computing environments such as in operating systems of Personal Computers (PCs) are shown to be very promising and they indeed yield optimized resource usage. Such schedulers could also be mentioned as the major contributors for the derivation of the current high performance scheduling techniques.

However, this does not necessarily mean that they are suitable algorithms for such relatively high scale performance computing systems like the cloud. Cloud in its nature, especially its resource provisioning demand, is different from other computing systems. Among the many factors, the frequent variation of users as well as the frequent variation of resource demand in cloud environment are the major ones. Besides that, the cloud resource provisioning model, "Pay-as-you-go", is also another factor that makes a lot of differences.

Consequently, cloud scheduling algorithms should utilize additional parameters and components to efficiently manage resources. To the best of our knowledge, currently employed scheduling algorithms, especially in the cloud computing architecture, are not primarily designed for cloud environment rather derived from other problem domains with a bulk of modification. This could also be mentioned as the major reason for the unpromising efficiency of schedulers in the cloud. Thus, the cloud environment needs a better scheduler which considers the unique behaviors of the cloud in general and its resources in particular.

Furthermore, intense reviews conducted on various cloud related literatures in general and resource provisioning models in particular revealed that the absence of some salient components inside cloud schedulers affect the efficiency of resource provisioning negatively.

As it is shown in Figure 4.1, the architecture is composed of six major components namely score calculator, group state manager, job consolidator, job prioritizer, group level adaptive resource dispatcher, and threshold calculator.

The demonstration for the aforementioned diagram could be described depending on the sequential flow of the architecture. The resource center shown at the first layer of the architecture imitates the overall characteristics of the cloud computing. This environment depicts the overall characteristics of the cloud in general and a particular service model, IaaS. Various nodes which

compose the resources of the cloud are simulated in this component. Resource centers have various hosts and each host has one or more virtual machines. Therefore, the aforementioned diagram of resource center encompasses Servers, Hosts, and Virtual Machines in each Host.

The new model is designed in a way each resource center has one additional embedded component called score calculator. The score calculation process takes various QoS constraints such as Millions of Instructions Per Second (MIPS), RAM, Storage, Processing Entity, and Bandwidth as its parameters.

The immediate accessing component of the output returned by the score calculator is the group state manager. Here, there are two major tasks to be done, namely group launcher and modifier. An invocation of group state manager given the initial state of the scheduler employs the group launcher. On the other hand, if the component is triggered by an interrupt generated by other components, then the group state modifier component will be used to alter the previous category of resource centers.

The purpose of group state manager is to define the belongingness of resources and jobs to a specific group. Here, the component primarily checks the group data holding persistent model. Then the attribute of resources indexed by their score and jobs will be altered by the name of the group of resources to which they belong. This is salient approach that minimizes the time needed to allocate jobs to the appropriate resources. The output of this component is a persistent model which stores a categorized list of resources and jobs at a given moment.

The group level job consolidation is followed by the specific consolidation phases, namely server level and host level job consolidation. In the server level consolidation, jobs in the same group are managed to be consolidated per data center. This mechanism bases on data center to job consolidation ratio. Furthermore, the host level job consolidator performs job to virtual machines tagging. This process is dedicated to consolidate the maximum number of jobs per host a fair manner. Considering the best case scenario, the flow of the scheduling architecture ends after the execution of group level adaptive job scheduler. In this phase, well tagged resources and consolidated jobs will be mapped to the corresponding virtual machine in a host.

The scheduling process is also tuned by the state of a component called threshold calculator. A threshold calculator is responsible to manipulate interrupts generated during resource to job assignment, accomplishment and other miscellaneous errors arousal. This component is

composed of three major components, namely interrupt handler, interrupt classifier and decision maker. The interrupt handler component accepts various types of interrupts from data centers and job allocators, and returns acquired values to the interrupt classifier. The interrupt classifier counts interrupts by their corresponding type and passes results to for the decision maker. Given a condition in which the amount of interrupts generated exceeds the specified threshold, the decision maker component generates cloud level reconfiguration interrupt to the group state manger and job enhancement request for the job priortizer.

Furthermore, there are some cases in which cloud level interrupt precedes the accomplishment of some jobs in the job queue. Here, the remaining jobs may starve beyond the tolerable range. Particularly, the absence of prioritizing rescheduled jobs over the upcoming jobs causes a great possibility of starvation. Consequently, a component namely job prioritizer is used to enahnce the score of rescheduled jobs. Every time a reconfiguration interrupt is generated the job prioritizer fetches the remaining jobs and enhances their resource requirement level for the phase of rescheduling. This technique helps to minimize the degree of starvation.

In the following Sections a detailed description of each component, flow of events and algorithms used are briefly discussed.

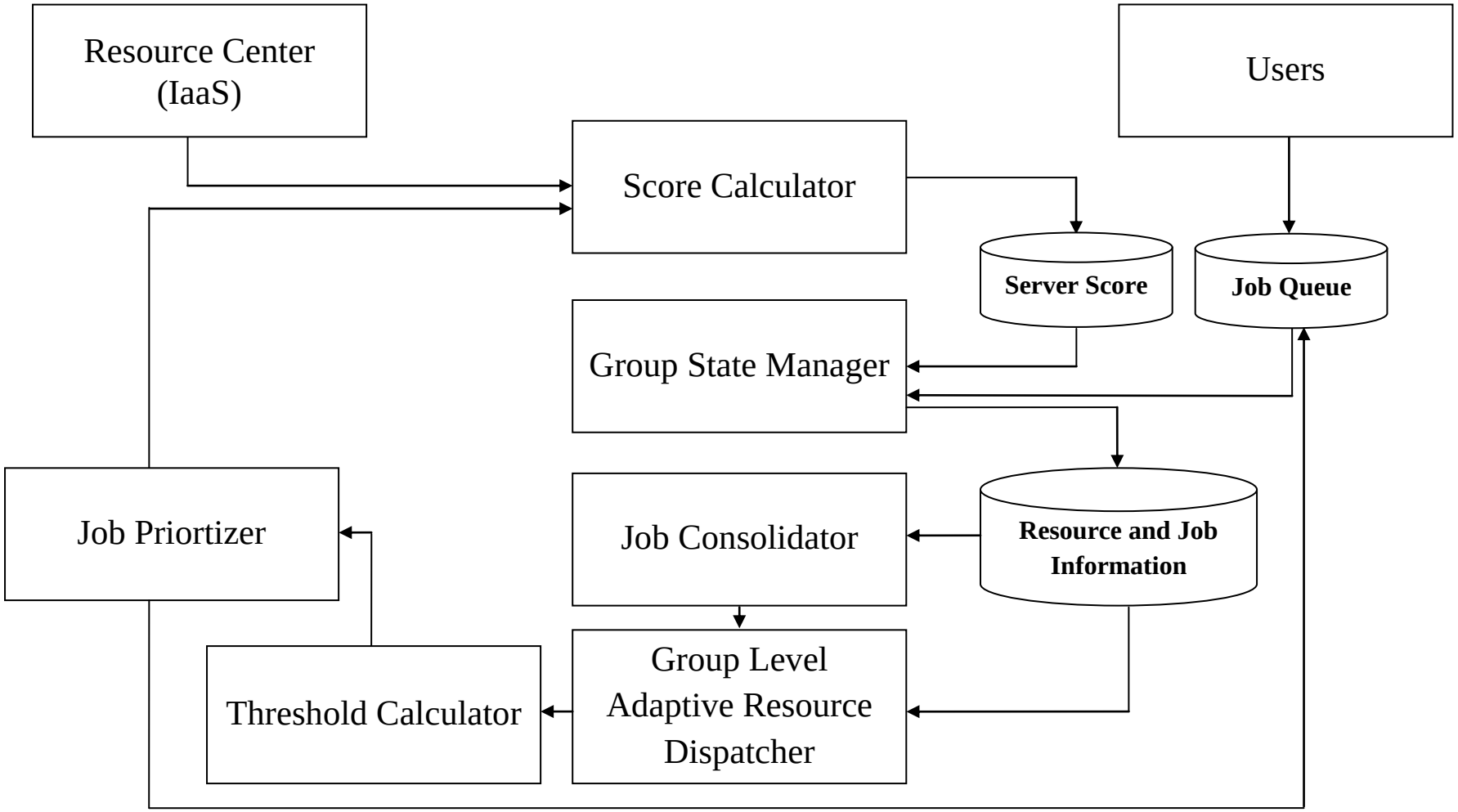

Figure 4-1: High Level Architecture of Distributed Score Based Job Scheduler

4.2.1 Score Calculator

Resources have various MIPS, RAM, Storage, Processing Entity, and Bandwidth according to their behavior. In order to rank servers capacity, a scoring system is necessary. Score based scheduling couples scheduling with a reputation system. As a result, this scheduling algorithm can choose high performance resources, so that it can improve reliability and overall cloud performance. This approach is also helpful to undergo Incentive Based Scheduling (IBS), which focuses on punishing (for example, exclusion) volatile, selfish, or malicious resources. Consequently, score based scheduling encourages volunteers to donate their resources eagerly and reliably.

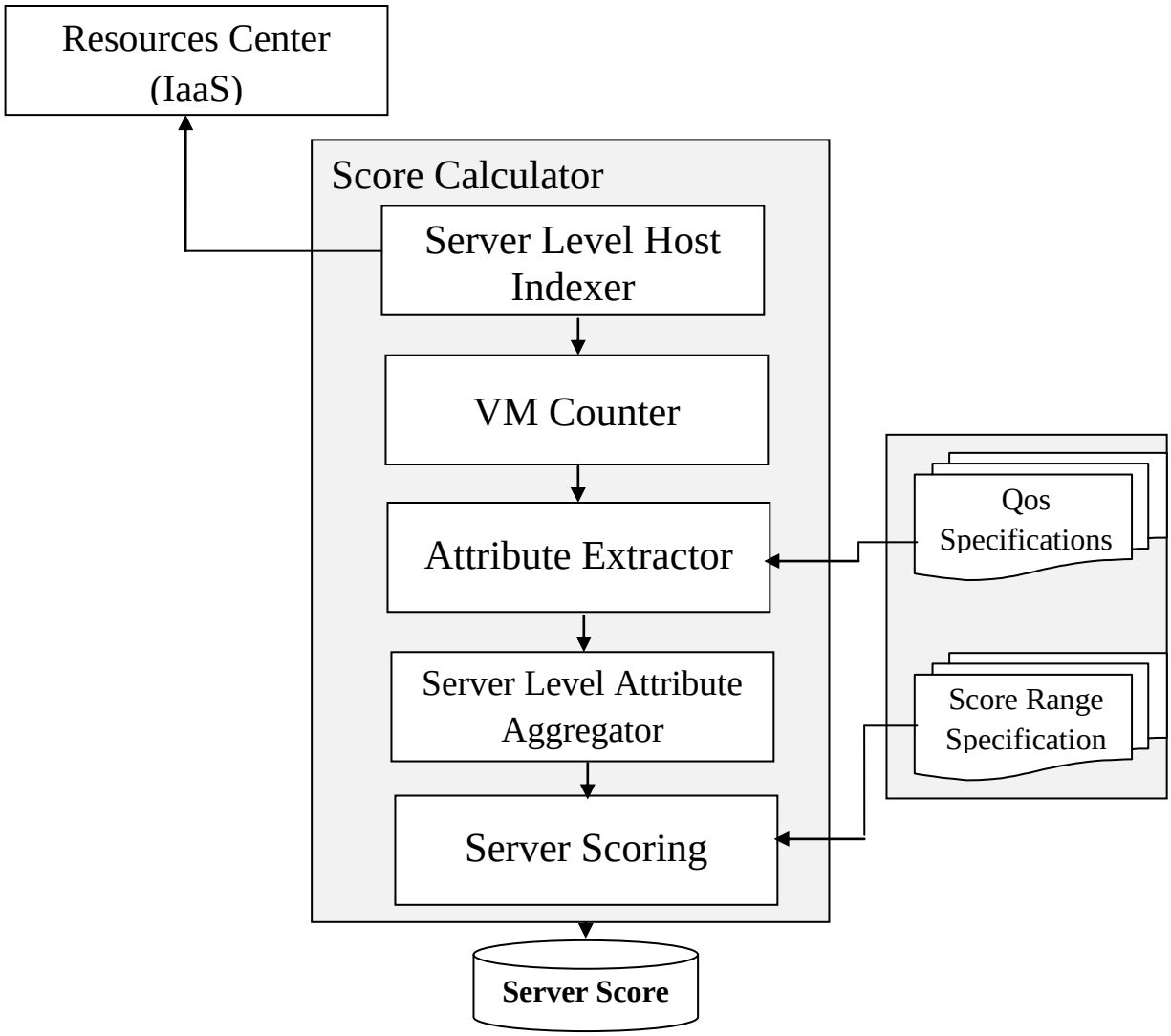

Figure 4-2: Architecture of Score Calculator

In addition, scoring is needed to perform efficient job consolidation in the specific server. The score for a given server is assigned between 1 and 10. The one with the greater score value will have a greater willingness to accept a larger number of consolidated jobs with higher demand of resources.

The score calculator is an iterative component that is responsible to measure the performance of the existing virtual machine. Various approaches could be used to calculate servers' performance though, DSBJSA's sub component shown in Figure 4-2 uses MIPS, RAM, Storage, Processing Entity, and Bandwidth as major parameters for score calculation.

As it is stated in Algorithms 4-1 and 4-2, the score calculation has two iterative and aggregative steps. Let us say we have a resource center R, hosts in each resource center **$H=\{h_1,h_2,.....,h_n\}$,** and virtual machines in each host referred to as **$VM=\{h_1v_1,....h_1v_n\},\{\ h_2v_1,....h_2v_n\ \},.....,\{\ h_nv_1,....h_nv_n\ \}$.** Here, the execution of score calculator starts with searching for available hosts using Server Level Host Indexer. The first object acquired during retrieval could be referred to as **h_1**. The second layer, VM Counter, checks for the validity of existing virtual machines on the upcoming scheduling cycle. Virtual machines that could not be handled by the existing host capability will be filtered using this component. Concurrent to the virtual machine affordability checkup, an iterative host attribute extraction takes place. This component, Attribute Extractor, launches an iterative search function that iteratively fetches host attributes based on the QoS parameters of each virtual machine in host **h_1**. Once the performance of each virtual machine is computed, the attributes of each host is returned to the component, Attribute Extractor.

The Server Level Attribute Aggregator is responsible for summing up host level scores with the purpose of acquiring the server state. Given the state of the server, Server Scoring component takes over. This component accepts the state of the server as an input and returns its group state. In some occasions, there might be a server whose score is less than tolerable. In this case, an isolator function, which is a part of score calculator and stated in Algorithm 4-3 discards them from the resources list ready for the upcoming scheduling. Finally, this component commits the overall acquired information of resource centers, hosts, and virtual machines into a persistent model.

Algorithm 4-1: Server Level Attribute Aggregation

```
Input: SERVER NAME, SERVER ID, MIPS, RAM, STORAGE, PROCESSING ENTITY,
and BANDWIDTH
Output: capacity: List  // List of the capacity of each data center
Begin
  For each data_center IN Cloud
     Counter=1;
     For each Host in data_center
        Capacity [ID] = Capacity [ID] + datacenter. Get(MIPS, RAM,
         STORAGE, CORE, BW)
     End For
  End For
  Return capacity
End
```

Algorithm 4-2: Server Score Calculator

```
Input: capacity: Collection
Output: Score: List  // List of scores per data center
Begin
  For i=0;i< capacity.size;i++
     Capacity.MIPS= (Capacity.MIPS/ Max_Mips)*100
     Capacity.BW= (Capacity.BW/ Max_BW)*100
     Capacity.PE= (Capacity.PE/ Max_PE)*100
     Capacity.STORAGE= (Capacity.STORAGE/ Max_Storage)*100
     Capacity.RAM= (Capacity.RAM/ Max_RAM)*100
     Score[i]=Average(Capacity)
  End For
  Return Score
End
```

Algorithm 4-3: Server Isolation

```
Input: Score: Collection
Output: Rejected List: List  // List of scores per data center
Begin
  For i=0;i< Score.size;i++
     If Score[i]< Min_Threshold Then
         Rejected_List.Add(Score[i])
         Score[i].Remove()
     End If
  End For
  Return Rejected_List
End
```

4.2.2 Group State Manager

In order to consider distinct features of cloud resources, a scheduler needs a resource grouping method, which ensures that resources with similar properties (such as capability, performance, availability, work-load, reputation/trust, volatility, etc.) are grouped together. The scheduler should also apply scheduling algorithms depending on the characteristics of each group.

Coupling a resource grouping method with scheduling helps to schedule and manage tasks efficiently. Moreover, there are benefits by coupling a resource grouping method with scheduling. For instance, resource grouping makes it possible to form homogeneous groups according to the score of resources which are in turn composed of Server Name, MIPS, RAM, Storage, Processing Entity, and Bandwidth of resource centers. A scheduler can apply various replication, result certification, and fault tolerant algorithms to each homogeneous group. In other words, the coupling method makes it possible that a scheduler selects and applies replication and fault tolerant algorithms suitable for the characteristics of each group. For example, a scheduler could take many replicas for volatile groups. A scheduler can frequently check result correctness for distrusted groups.

Thus, the resource grouping method directly affects reliability, completion time, and result correctness. It improves reliability of computation and performance. However, in the existing cloud systems, resource grouping is not tightly related with scheduling. As a result, overhead and performance degradation is the usual observation.

As it is shown in Figure 4-3**,** resource grouping technique employed in the new architecture is fully dependent on the score calculator. Assume we have a list of score of servers **S={s_1,s_2,s_3,s_5....,s_n}** acquired from the score calculator component. The group state manager is responsible for checking the score repository and generalize it to an appropriate group.

The assumption here is groups are labeled in range of integers 1,2, …,n where n is an integer that is directly proportional to the size of the cloud environment. Hence, the overall goal of group state manager shown in Figure 4-3 is to categorize servers to their respective groups based on their score. For that, this component mainly employs a simple mathematical technique called rounding to categorize equivalent scores to their respective group. The rounding technique usually rounds some fractional scores to their corresponding (nearest integer type score).

Let us perform scenario based clarification on Algorithm 4-2, which is the main node of the component shown in Figure 4-3. Let us say the score of **S_1=3.4, S_2=2.4, S_3= 2.9, and S_4=1.8**. Taking this scenario, there are two major steps to be performed to get the expected result out of this component. The first phase is to get the performance of the cloud in general and passing that value to group level job consolidator as it is shown in Algorithm 4-3.

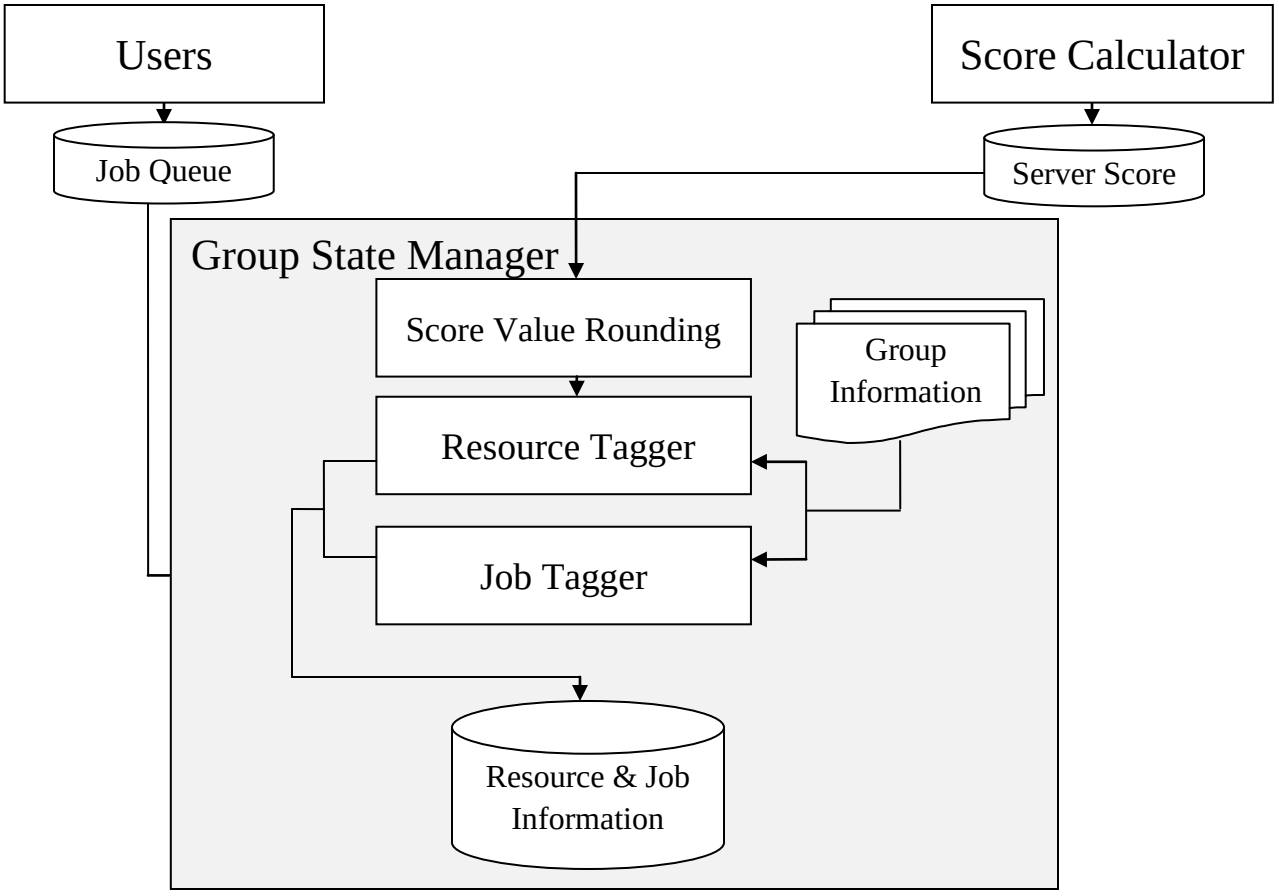

Figure 4-3: Architecture of Group State Manager

Following that, the group level consolidator, Algorithm 4-4 decides on how many servers should be categorized in each server. Hence, the first group to be established in the given case is the one with s_2 and s_4 as its members. This is because the group state manager primarily rounds **s_1** and **s_2** around their nearest non floating point number, in this case 2. What follows is, the state manager establishes groups that enclose servers with score of **s_1** and **s_3**. Generally, given the above list of scores, the group state manager will round off them into two major numbers, in our context groups which contains **s_2** and **s_4** on one side and **s_1** and **s_3** on another.

On the other hand, Algorithm 4-5 is a part of this component which tags jobs by their respective group of execution. This component is mainly dependent on the consolidation limiter which decides on how much cloudlet should be assigned to each group of resources. The number and length of cloudlets that belongs to each group is handled by a collection data structure.

Algorithm 4-4: Group State Manager

```
Input: Score: Collection
       Group_Criteria: List
Output:Group: List  // List of scores per data center
Begin
  For ID=0;ID< Score.size;i++
     Round(Score[ID], Half_Down)
     Switch(Score[ID])
         Case 1:
               Group[1].Add(ID) // Adding the id of datacenter
               Break
         Case N:
               Group[N].Add(ID) // Adding the id of datacenter
               Break
         Default: //
  End For
  Return Group
End
```

Algorithm 4-5: Job Tagger

```
Input: Cloudlet: List
       Group: List
Output:Group_Job_info: List // List of Cloudlet to group binding info
Begin
Consolidation_limit= ConsolidationLimiterDC(Cloudlet, Group)
  For i=0;i< Group.size;i++
    J=0
    While((Group_Job[i].length + Cloudlet[j])<=Consolidation_limit[i])
           Group_Job[i]=Binder(Cloudlet[j],Group[i])
           Cloudlet.Remove()// substitutes the 1st  index with 2nd
           Next
     End While
     If Cloudlet.Size >0 then
        Reconfigure_list.Add(Cloudlet)
     End If
  End For
  Return Group_Job_info //cloudlets-group information.
End
```

4.2.3 Job Consolidator

In the context of scheduling, the task consolidation problem is the process of assigning a set of jobs $j = \{j_1, j_2, j_3, \ldots, j_n\}$ to a set of virtual machines $VM = \{vm_1, vm_2, vm_3, \ldots, vm_m\}$ available in a given host with the purpose of maximizing resource utilization and minimizing energy consumption. Furthermore, task consolidation aims at yielding effective usage of cloud resources by consolidating a set of tasks into a small number of virtual machines. In the task consolidation problem, the resource allocated to a particular task must sufficiently provide the resource usage of the given task.

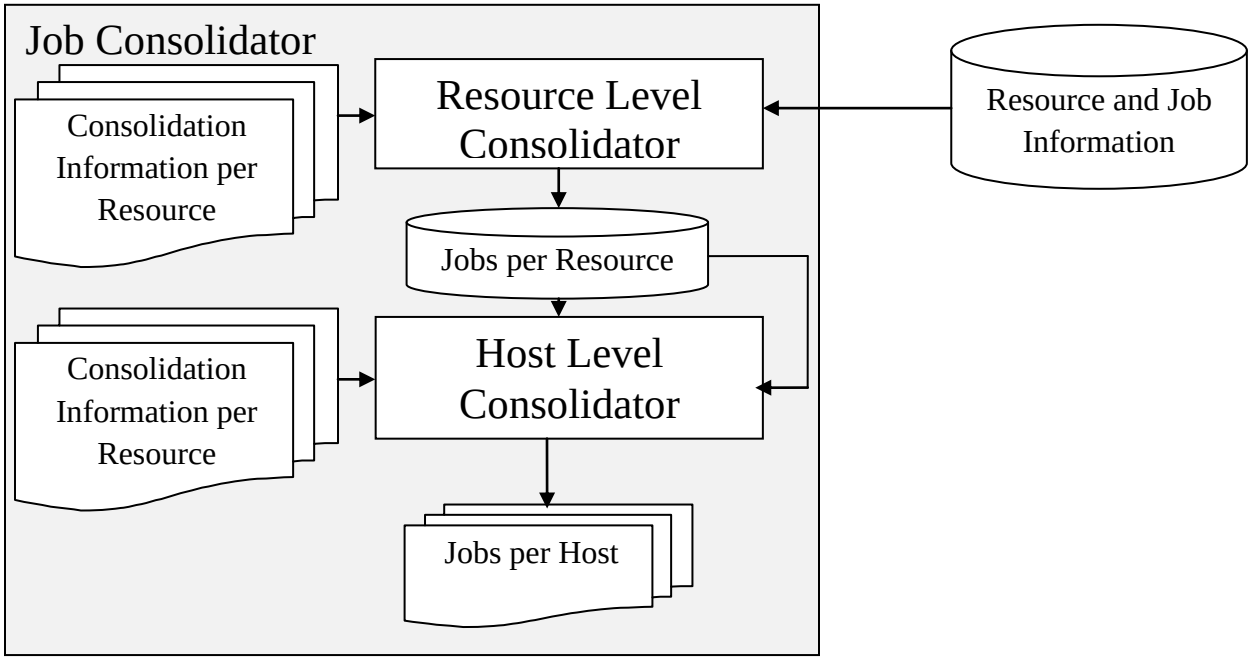

Figure 4-4 Architecture of Job Consolidation

Looking at Figure 4-4 one can realize that the flow of the job consolidation component is fully dependent on the number of virtual machines in the resource center. Moreover, Algorithm 4-6 shows the dependency of job consolidator on the maximum concatenation limit setter called consolidation limiter. The sub component of consolidation limiter depicted in Algorithm 4-6 is designed in a way it can balance the proportion of jobs and the available capacity of resources. In this context, there are two limits to be referred by the job consolidator namely: resource level consolidation limit and host level consolidation limit. In the resource level consolidation limiter shown in Algorithm 4-6 the maximum number of jobs that could be consolidated per a given

server is set. On the other hand, host level consolidation shown in Figure 4-4 is responsible to decide the maximum number of jobs to be assigned for each host in a given server.

The resource level consolidator is used to avoid excess load on a specific server. The maximum limit of consolidating jobs on a resource is decided by referring to the result of resource level consolidator. The latter consolidation limiter is used to limit the maximum consolidation rate in a specific host.

Algorithm 4-6: Resource Level Job Consolidator

```
Input: Group_Job_info[i]: List // this function is called at each GL
consolidator
Output: Resource_Job: ist of List
Begin
  Consolidation_limit= ConsolidationLimiter(Group_Job_info[i])
  J=0
  While((Resource_Job[i].length + Group_Job_info.Cloudlet[j])
  <=Consolidation_limit)
    Resource_Job[i]= Bind(Group_Job_info.Cloudlet[j], Resource_Job[i])
    Group_Job_info.Cloudlet[j].remove()
    Next
   End While
Return Resource_Job
End
```

Algorithm 4-7: Host Level Job Consolidation

```
Input: Resource_Job[i]: List // this function is called at each RL
consolidator
Output: Resource_Job: Collections
Begin
  Consolidation_limit= ConsolidationLimiter(Resource_Job[i])
  J=0
  While((Host_Job [i].length + Resource_Job.Cloudlet[j])
  <=Consolidation_limit)
    Host_Job [i]= Bind(Resource_Job.Cloudlet[j], Host_Job)
    Resource_Job.Cloudlet[j].remove()
    Next
   End While
Return Host_Job
End
```

4.2.4 Group Level Adaptive Resource Dispatcher

Although the coupling method has a lot of advantages, existing scheduling mechanisms, however, did not consider properties such as volatility, availability, and credibility that strongly affect QoS such as reliability, performance, and correctness. Moreover, they did not provide scheduling mechanism based on group characteristics. In other words, those scheduling algorithms do not apply different scheduling algorithms to each group according to their property. As a result, they happen to be a major reason for the degradation of reliability and performance of the cloud.

Consequently, this research work proposes a new score oriented group based scheduling model, which adapts to a dynamic cloud computing environment. As it is shown in Figure 4-5, components namely score calculator, group state manager and job prioritizer generate the major inputs for this component. Moreover, as it is presented in Algorithm 4-8 the results of score calculator as well as group state manager have determinant factor on this component. The modification phase mainly considers the volatility of hosts to determine issues related to cloudlet to virtual machine binding.

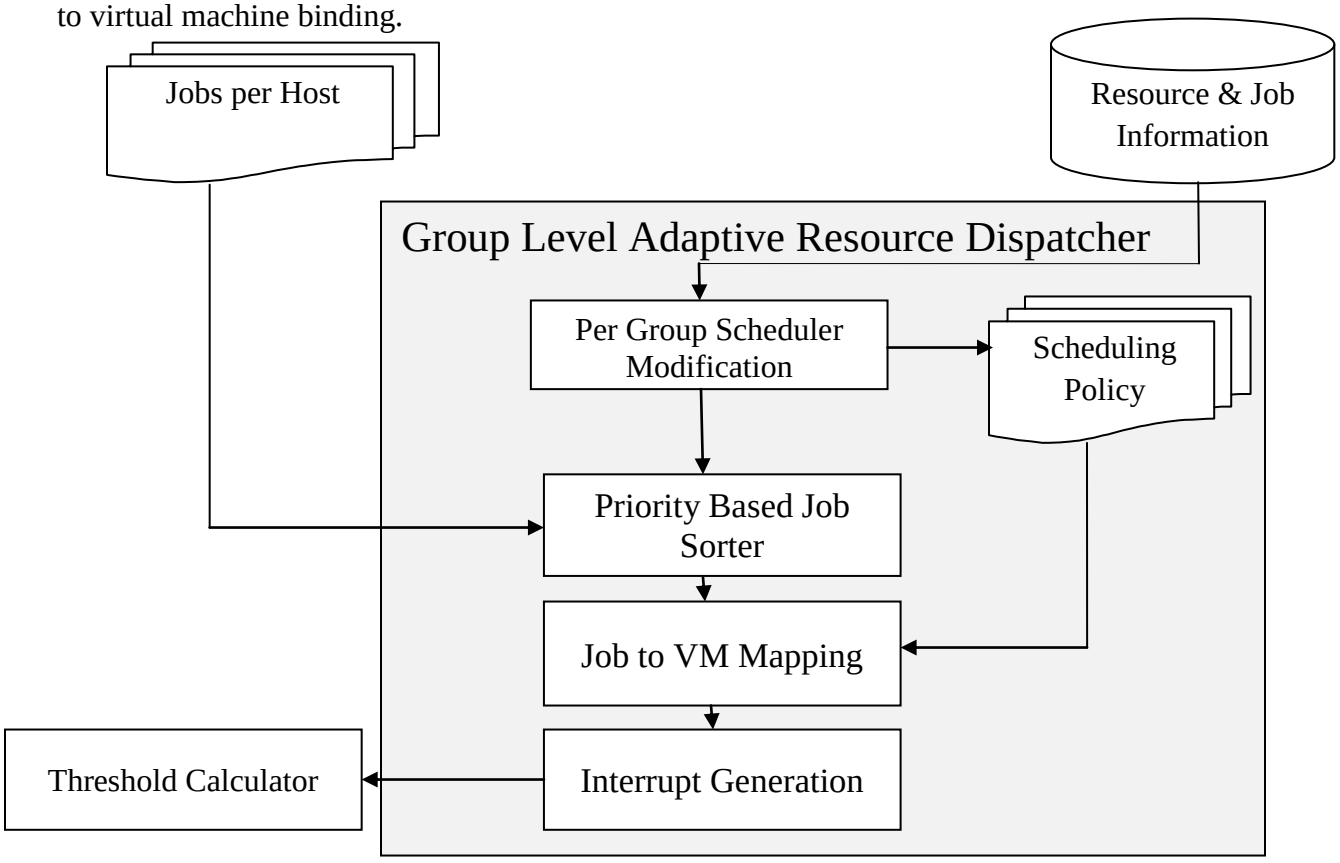

Figure 4-5: Architecture of Group Level Adaptive Resource Dispatcher

Besides that, this component determines in which way to use resource center from the perspectives such as dedication, volatility, availability, and credibility. Then it applies different scheduling, replication, result certification, and fault tolerance algorithms to each group. The replication and result certification process performs decision on the number of resource centers to accept the same job and the quality of the server respectively. Moreover, fault tolerance algorithm will be used to keep the scheduling process online. Accordingly, this scheduling component as well as its implementation improves reliability and performance.

Algorithm 4-8: Group Level Adaptive Resource Dispatcher

```
INPUT: Host_Job: Collection
OUTPUT:Void
BEGIN
   While (i<Host_Job.size())
       If (Host_Job[i].Volatility!=1 AND Host[i]. cloudlet.assign!=1)
           For each vm in Host[i]
             Bind(Host[i].cloudlet, vm)
             Host[i].cloudlet.assign=1
             Host[i]. cloudlet.remove()
           End For
      Elseif(Host_Job[i].Volatility==1ANDHost[i]. cloudlet.assign!=1)
             Bind(Host[i].cloudlet, vm)
             Host[i].cloudlet.assign=1
             Host[i+1].cloudlet.add(Host[i].cloudlet)
             Host[i]. cloudlet.remove()
      End If
   End While
End
```

4.2.5 Threshold Calculator

In the new model, threshold is used to generate server reconfiguration interrupts. The scheduler computes the threshold value based on current active, processed and overall jobs in the job queue. The reconfiguration interrupt generation inside the cloud scheduler is triggered by the turn around function which accepts, sums up and compares the total number of interrupts to the specified threshold set by threshold calculator.

As it is shown in Algorithm 4-9, the threshold value is compared after calculating the current status of the scheduler using the following equation.

$$T = \frac{\left(\sum_{i=1}^{n} \text{Jacc\_i}\right) + \left(\sum_{j=1}^{m} \text{Jass\_j}\right)}{\left(\sum_{i=1}^{n} \text{Jacc\_i}\right) + \left(\sum_{j=1}^{m} \text{Jass\_j}\right) + \left(\sum_{k=1}^{p} \text{Jt\_k}\right)} \quad (2)$$

where J_{acc} is jobs accomplished, J_{ass} stands for active jobs assigned, and finally J_t denotes the total amount of jobs left on the ready queue.

Moreover, Figure 4-6 shows how the turnaround function activates the reconfiguration interrupt and sends it to the group state manager whenever the threshold limits is exceeded.

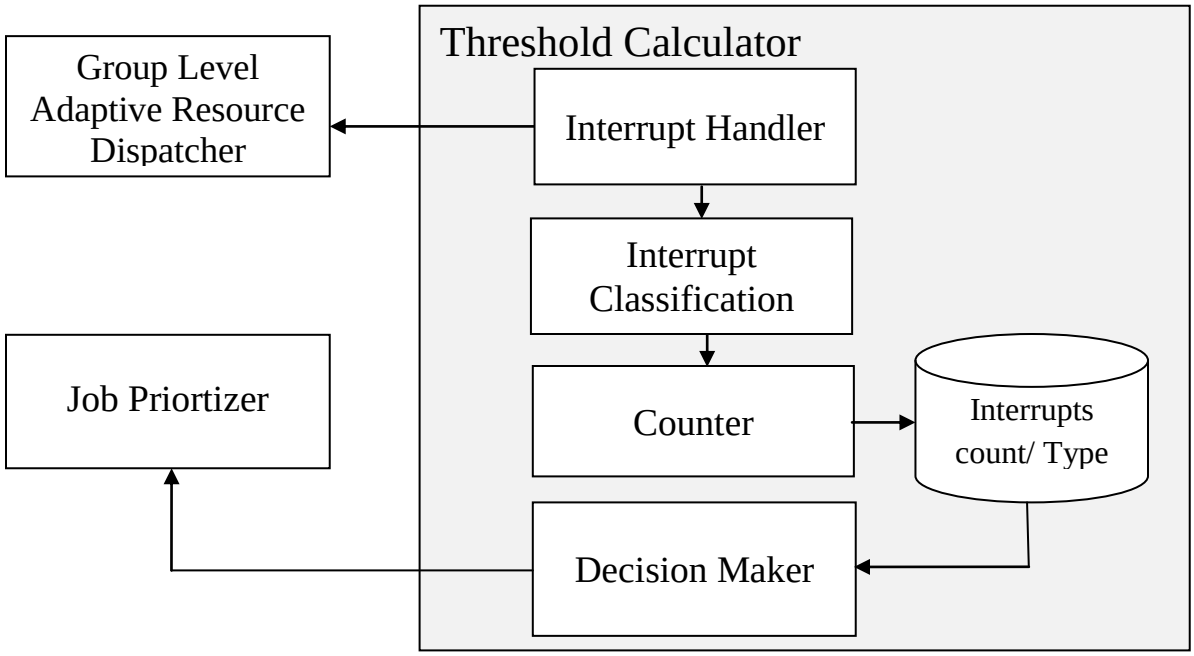

Figure 4-6: Architecture of Threshold Calculator

Algorithm 4-9: Threshold Calculator

```
INPUT: INTERRUPT_ATTRIB: LIST
OUTPUT: Reconfigure:Boolean
BEGIN:
     Initiate_Runnable(INTERRUPT_HANDLER)
     Handler=InterruptClassifier(Event e, Done, Error)
     If Handler ==Done
         Log(e, Count[1])
         If Turn_around<=Count[1]
           Reconfigure=1
           Group_state_manager()
           Return;
          End if
      End if
End
```

4.2.6 Job Prioritizor

Prioritization is the process of determining which of many jobs should have resources. This technique helps to improve resource utilization and reduce starvation of jobs. As it is shown in Algorithm 4-5, jobs which could not be addressed by the current capacity of machines are truncated and sent to "reconfigured_list". This list data structure is examined at the beginning of each reconfiguration interrupt. As it is shown in Algorithm 4-10 the priority of all cloudlets inside the "reconfigured_list" will be enhanced. The last operation in this component is merging incoming jobs with prioritized cloudlets.

Let us clarify Algorithm 4-10 using the following scenario. Assume we have jobs **$J=\{j_1,j_2,j_3,j_4,\ldots\ldots, j_n\}$.** Assuming reconfiguration interrupt is generated before job accomplishment interrupt for jobs **j_1, j_2**, and **j_{n-1},** then the "reconfigured_list" which is a collection type data structure to store remaining jobs will be fetched and the priority of jobs **j_1, j_2**, and **j_{n-1}** will be enhanced to the maximum priority. Consequently, **j_1, j_2**, and **j_{n-1}** will have a higher priority than the incoming jobs in the ready queue.

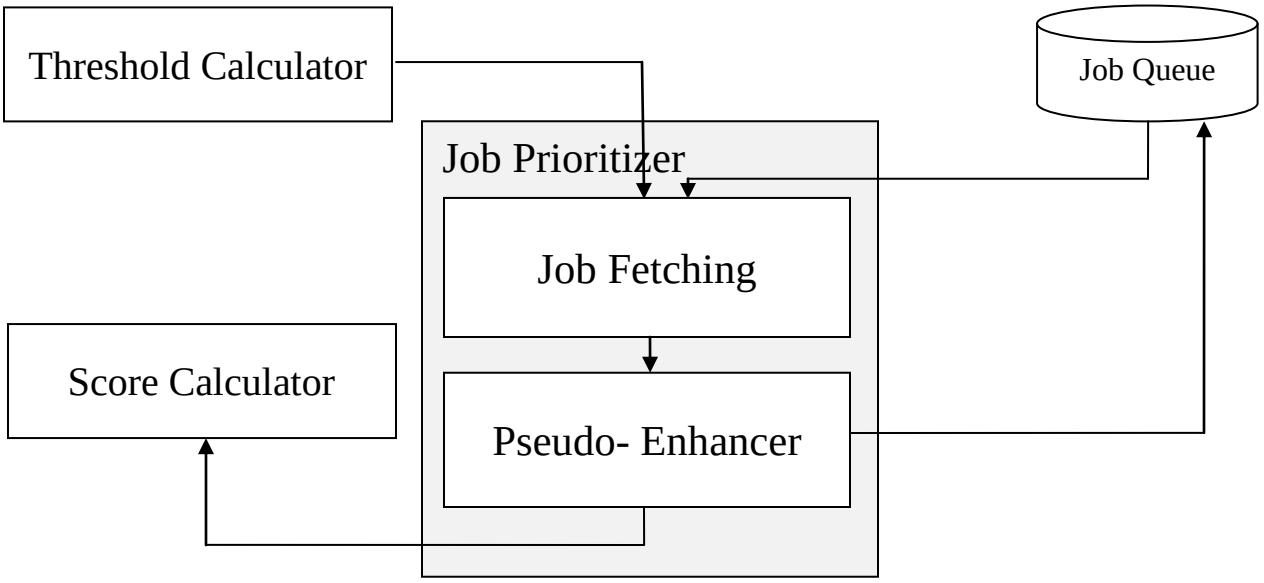

Figure 4-7: Architecture of Job Priortizer

Algorithm 4-10: Job Priortizer

```
INPUT: Cloudlet: LIST
       Reconfigured: List
OUTPUT: cloudlet_enhanced: LIST
BEGIN:
  FOR each cloudlets in reconfigured
   Cloudlet.priority= Max Priority
  END FOR
  Cloudlet.add(Reconfigured)
  Return Cloudlet
END
```

4.3 Summary

Job scheduling is one of the decisive components of cloud computing. A computing environment without efficient resource provisioning and job dispatching algorithm might not provide services that satisfy the service level agreement with the end user.

In DSBJS model, the most salient components are added in a way they can boost scheduling performance. All components enclosed in the new architecture are aimed at addressing computational time, load balancing and resource categorization.

The new architecture uses a scoring mechanism in order to manage the overall scheduling process. The score calculation takes various QoS parameters to calculate and rank the resource center. The score of each resource center in turn is used to process two important tasks namely: group state management and job consolidation. The simple mathematical technique named "rounding" is used to decide the number of groups and the QoS variation among various groups. Given a specified number of groups based on the aggregated score, the job consolidator, the most significant component of the architecture, will be triggered. Job consolidation is performed in three palletized levels. The group level job consolidator is the very first level job categorizer of the architecture. By this component, all jobs in the job queue will be tagged by the possible volunteer and appropriate resource group. The tagging process usually follows the job-resource reordering. Consecutively, the resource level and host level consolidation will be performed in the same way as the group level consolidator, though different in time of occurrence.

Once the host level consolidation is performed the group level adaptive resource dispatcher will take over the flow. This component is designed to let it impose different and appropriate

scheduling policy for various groups. Mostly, the degree of replication, server information checkup and the job to resource allocation policy is differentiated among groups. For instance, for groups with lower score, the scheduler will increase the degree of server replication, reconfiguration frequency and so on. On the other hand, for such groups composed of high score resource centers the replication degree and server state reconfiguration will be minimum. The adaptive job allocation component is directly influenced by the output of the threshold calculator. This component accepts every interrupt from resource centers, classifies them and aggregates their total result. The newly acquired value will then be used to either trigger the reconfiguration enabler component or bypass the reconfiguration interrupt generation phase. Besides, for those jobs which are yet to be executed during interrupt generation, the job priortizer component is used. This component gives high priority for jobs left on the job queue during the reconfiguration interrupt. Later in the reconfiguration, the old or remaining jobs will not starve because of the incoming jobs.

5. Experiment

5.1 Overview

This Chapter presents tools and methodologies that have been used to implement Distributed Score Based Job Scheduling Algorithm for Cloud Computing Environment. The prototype is developed based on the architecture depicted in Chapter Four. This Chapter also discusses programming approaches and techniques used to model and manipulate entities of the cloud computing environment. The last topic discussed in this Chapter, experiment and evaluation, presents the performance of DSBJSA that is evaluated by undertaking successive experiments and comparisons.

5.2 Scope of the Prototype

The prototype is designed to meet the basic requirements of the system architecture shown in Figure 4-1. Components such as score calculator, group state manager, cloudlet/job consolidator and group level adaptive job scheduler are implemented. However, components required to trigger reconfiguration interrupt and enhance rescheduled jobs are not implemented in this prototype as they are not mandatory for preliminary system checkup. Besides that, the prototype does not visualize data center creation and manipulations using pictorial view. Hence, the default input and output mode of NetBeans IDE, Text Based Mode, is used.

5.3 Development Tools

The following tools are mainly used to develop the prototype.

5.3.1 Java Platform, Standard Edition Development Kit (JDK) Version 8

The Java Development Kit is a software development environment used for developing Java applications and applets. It includes the Java Runtime Environment (JRE), an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (javadoc) and other tools needed in Java development [101]. This version of java development kit is chosen because of its major improvements mentioned below.

i. Java Programming Language

- Default methods enable new functionality to be added to the interfaces of libraries and ensure binary compatibility with code written for older versions of those interfaces.

- Repeating annotations provide the ability to apply the same annotation type more than once to the same declaration or type use.
- Type annotations provide the ability to apply an annotation anywhere a type is used, not just on a declaration.

ii. Collection Type: Classes in the new java.util.stream package provide a Stream API to support functional-style operations on streams of elements. The Stream API is integrated into the Collections API, which enables bulk operations on collections, such as sequential or parallel map-reduce transformations.

iii. Date-Time Package: A new set of packages that provide a comprehensive date-time model.

iv. java.lang and java.util Packages

- Parallel Arrays Sorting

v. Java DB:

Java DB is Oracle's supported distribution of the Apache Derby open source database. It supports standard ANSI/ISO SQL through the JDBC and Java EE APIs. Java DB is included in the JDK. This tool mainly helps the prototype development process by providing efficient Data Definition and Data manipulation Techniques. Specifically, features of this tool includes

- Transaction-protected and crash-recoverable
- Embeddable in applications
- Pure Java and portable and across CDC FP 1.1, Java 5, Java 6, and Java 7 (everywhere from tablets to mainframes)
- Included in the JDK
- Compact (2.6 MB)

vi. Concurrency

- Classes and interfaces have been added to the `java.util.concurrent` package.
- Methods have been added to the `java.util.concurrent.ConcurrentHashMap` class to support aggregate operations based on the newly added streams facility and lambda expressions.

- Classes have been added to the `java.util.concurrent.atomic` package to support scalable updatable variables.
- Methods have been added to the `java.util.concurrent.ForkJoinPool` class to support a common pool.

5.3.2 NetBeans Integrated Development Environment (IDE) Version 8.1

The **NetBeans IDE** is an official open-source Integrated Development Environment for Java 8. It allows applications to be developed from a set of modular software components called modules. Applications based on the NetBeans Platform, including the NetBeans integrated development environment (IDE), can be extended by third party developers. NetBeans is cross-platform and runs on Microsoft Windows, Mac OS X, Linux, Solaris and other platforms supporting a compatible Java Virtual Machine (JVM) [102].

The salient features that make this version of Netbeans IDE the best and suitable for this prototype development are:

i. Java Enhancements

- Enhanced Code Completion("intellisense")
- More expressive navigator shows overridden & implemented methods
- Performance improvements for Java navigation tools

ii. Java EE Enhancements

- Support for remote Oracle WebLogic Server
- Community contributed support for WildFly 9 and WildFly 10

5.3.3 CloudSim 3.0.2

CloudSim is an extensible simulation toolkit that enables modeling and simulation of cloud computing systems and application provisioning environments. The cloudSim toolkit supports both system and behavior modeling of cloud system components such as data centers, virtual machines (VMs) and resource provisioning policies. It implements generic application provisioning techniques that can be extended with ease and limited effort. Currently, it supports modeling and simulation of cloud computing environments consisting of both single and inter-

networked clouds (federation of clouds). Moreover, it exposes custom interfaces for implementing policies and provisioning techniques for allocation of VMs under inter-networked cloud computing scenarios.

Cloudsim Architecture

Figure 5-1 [103] shows a multi-layered design of cloudSim software framework and its architectural components. The current releases of cloudSim uses SimJava as the discrete event simulation engine that supports several core functionalities, such as queuing and processing of events, creation of cloud system entities (services, host, data center, broker, VMs), communication between components, and management of the simulation clock.

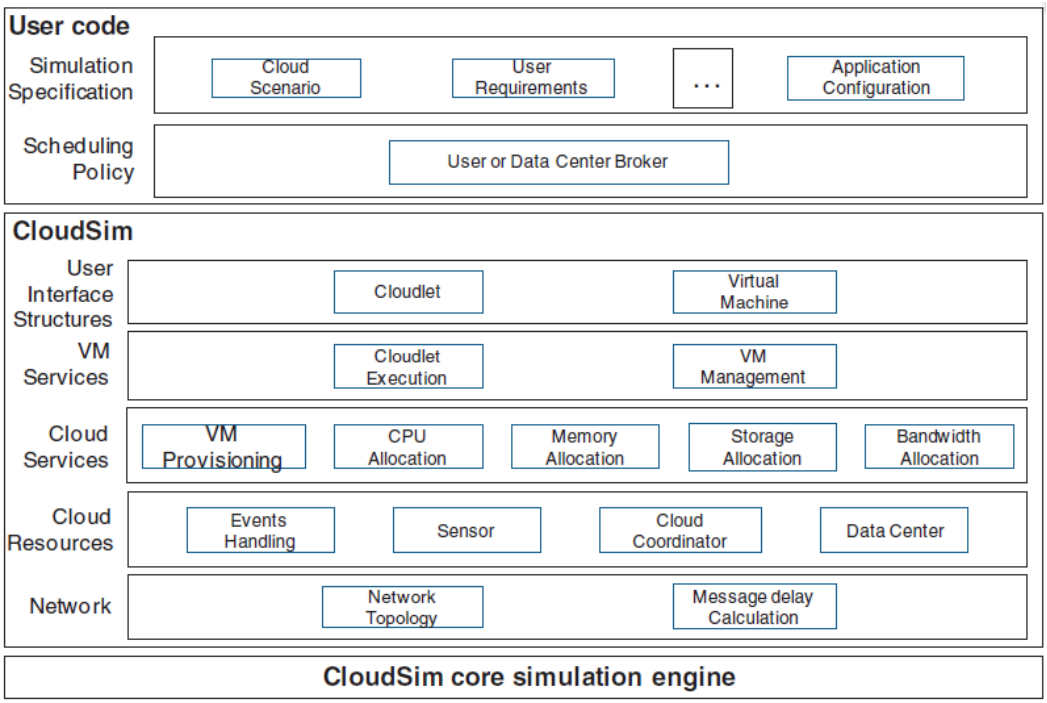

Figure 5-1: Layered CloudSim Architecture

- **Simulation Layer**

The layer shown at the bottom of Figure 5-1 provides support for modeling and simulation of virtualized cloud-based data center environments including dedicated management interfaces for VMs, memory, storage, and bandwidth. The fundamental issues, such as provisioning of hosts to VMs, managing application execution, and monitoring dynamic system state, are handled by this layer.

A Cloud provider, who wants to study the efficiency of different policies in allocating its hosts to VMs (VM provisioning), would need to implement the strategies at this layer. Such implementation can be done by programmatically extending the core VM provisioning functionality. There is a clear distinction at this layer related to provisioning of hosts to VMs. A Cloud host can be concurrently allocated to a set of VMs that execute applications based on SaaS provider's defined QoS levels. This layer also exposes the functionalities that a cloud application developer can extend to perform complex workload profiling and application performance study [103].

- **User Code Layer**

This is the top-most layer of cloudsim stack shown in Figure 5-1. It exposes basic entities for hosts (number of machines, their specification, and so on), applications (number of tasks and their requirements), VMs, number of users and their application types, and broker scheduling policies. By extending the basic entities given at this layer, a cloud application developer can perform the following activities: (i) generate a mix of workload request distributions, application configurations; (ii) model Cloud availability scenarios and perform robust tests based on the custom configurations; and (iii) implement custom application provisioning techniques for clouds and their federation [103].

5.4 Prototype Development

A prototype is developed to implement the proposed scheduling algorithm. As it is mentioned in the first Chapter, this prototype implements only the job/cloudlet scheduler. However, some entities of the cloud are implemented as a preliminary step for running the scheduler. For instance, the default virtual machine provisioning policy is adopted.

5.4.1 Data Center Modeling

In order to perform scheduling on the cloud one must perform some abstraction of the real cloud computing environment. Modeling the largest computing environment, the cloud, at least requires an efficient abstraction of cloud entities such as Data Center, Host, Data Center Broker, and Virtual Machine.

Moreover, core packages of the simulator namely, "org.cloudbus.clousim" and "org.cloudbus.clousim.core" and some of its classes are overridden to make the toolkit suitable to the proposed scheduling algorithm. Specifically, two cloud models with different resource

characteristics are designed using built-in classes such as “Datacenter”, “Host”,“Vm”, “Pe”, and “Cloudlet”. The purpose of using various cloud models is to evaluate the proposed job scheduler in various size cloud model; particularly in both small scale and large scale cloud computing environments. Table 5-1 describes the characteristics of the first cloud model used in this experiment.

Table 5-1: Cloud Model I - Resource Constrained Cloud Model

Datacenter	**Host**					
DSBJSA_0	**Host**	**RAM (MB)**	**MIPS**	**BW**	**Secondary Memory (MB)**	**Cores.**
	Host I	2048	550	50	1000	1
	Host II	3072	1100	75	1560	1
	Host III	4096	1100	100	2650	2

This model is categorized as small scale cloud model whose resources are located in a site. It contains a data center which has three hosts with different resource characteristics. This model uses space shared virtual machine scheduling method in which one virtual machine is allowed to use a core at a time.

The other model used to test the proposed job scheduler is shown in Table 5-2. This model is designed to imitate resourceful cloud model. The number of hosts and cores this cloud model have surpasses the first cloud model. It is designed and deployed with the purpose of showing the performance of the proposed algorithm in resource unconstrained cloud environment.

Table 5-2: Model II: Resource Unconstrained Cloud Model

Datacenter	**Host**					
	Host Name	**RAM (MB)**	**MIPS**	**BW**	**Secondary Memory(MB)**	**Cores**
DSBJSA_0	Host I	2048	6200	500	500000	1
	Host II	1024	7500	500	1000000	2
	Host III	4096	8000	500	1000000	2
DSBJSA_1	Host I	4096	4200	500	500000	1
	Host II	8192	5000	500	1500000	4
DSBJSA_2	Host I	2048	7100	500	1500000	4
	Host II	2048	9495	500	1500000	4
	Host III	8192	8500	500	1500000	4
	Host IV	2048	11900	500	1500000	4
	Host V	4096	12100	500	1000000	3

As it is shown in Table 5-2**,** the resources in this cloud model are located in three different sites. Hence, this model is used to show the performance of the proposed scheduler given

geographically distributed resources. In addition, this model is used to show how well the scheduler utilizes resources even if there are various alternative resources that could execute cloudlets.

```
private static Datacenter createDatacenter(String name){
                List<Host> hostList = new ArrayList<Host>();
                List<Pe> peList = new ArrayList<Pe>();
                int mips = 2000;
              // creating processing cores - quad core modeling
              peList.add(new Pe(0, new PeProvisionerDSBJSA(mips)));
              peList.add(new Pe(0, new PeProvisionerDSBJSA (mips)));
              peList.add(new Pe(0, new PeProvisionerDSBJSA (mips)));
              peList.add(new Pe(0, new PeProvisionerDSBJSA (mips)));
                int ram = 4096;
                long storage = 1000000;
                int bw = 10000;
                hostList.add(
                        new Host(
                                getHostidgenerator(),
                                new RamProvisionerDSBJSA (ram),
                                new BwProvisionerDSBJSA (bw),
                                storage,
                                peList,
                                new VmSchedulerSpaceShared(peList)
                        ));
          hostList.add(
                        new Host(
                                getHostidgenerator(),
                                new RamProvisionerSimple(ram),
                                new BwProvisionerSimple(bw),
                                storage,
                                peList,
                                new VmSchedulerSpaceShared(peList)
                        ) );
                // Datacenter characterstics
                String arch = "x86";
                String os = "Linux";
                String vmm = "Xen";
                double time_zone = 10.0;
                double cost = 3.0;
                double costPerMem = 0.05;
                double costPerStorage = 0.001;
                double costPerBw = 0.0;
LinkedList<Storage> storageList = new LinkedList<Storage>();
DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
                    arch, os, vmm, hostList, time_zone, cost, costPerMem,
costPerStorage, costPerBw);
                Datacenter datacenter = null;
                try {
```

Figure 5-2: Data Center Creation

The aforementioned data center models are realized by using classes in the cloudsim simulator. Furthermore, “Datacenter” is among the major classes used to model the cloud. It is a cloud resource whose host list are virtualized. It deals with processing of VM queries instead of processing cloudlet related queries. The other type of class used in both models is Processing Element (Pe). This class represents CPU unit, defined in terms of Millions Instructions Per Second (MIPS) rating. Figure 5-2 shows a code snippet that creates data center with more than

one host and processing cores in each host. This code creates a complete cloud model with machines having different processing, storage, bandwidth and other salient entities. The other important class used in this model is “Host”. This class is mainly used as a substitute for machine in the cloud computing environment.

Moreover, “Host” executes actions related to management of virtual machines and it has a defined policy for provisioning memory and BW, as well as an allocation policy for Pes to virtual machines. Hence, a host is directly associated to a datacenter and it can host virtual machines. Figure 5-3 shows the output generated after the successful creation of cloud model. It mainly shows the type of data centers and their characteristics in the model.

```
run:
Initializing Datacenter...
Datacenter Characteristics:
Name: DSBJSA_0   Sec Storage:83886080 MIPS:100.0 Total RAM:52428 BW:10240
Name: DSBJSA_1   Sec Storage:70086100 MIPS:32.0  Total RAM:11530 BW:10300
      .
      .
DATACENTER_0 is starting...
DATACENTER_1 is starting...
      .
      .
Broker is starting...
Entities started.
0.0: Broker: Cloud Resource List received with 3 resource(s)
```

Figure 5-3: Message on Successful Creation of Data Centers

5.4.2 Server Score Calculation

As it is mentioned in the first Chapter and also shown in Figure 4-1, the proposed system architecture mainly bases on the score of each server. Each server is scored based on QoS criteria stated in Chapter Four and the overall score will be stored using a dedicated database server.

The first task in this phase is to get the values of each parameter relative to the maximum possible value in a specific cloud environment. Figure 5-4 shows the result of resource level score calculator. This result is acquired by calculating the relative position of resource attributes relative to the maximum possible value of attributes in that cloud environment. For instance, if the MIPS of a given data center is k and the maximum available data center MIPS value in the

specific cloud environment is n, then the relative position of a given data center from the perspective of its MIPS is calculated as a percent of k/n.

```
Initializing Score Calculator
Acquired Available MIPS of DATACENTER_0
Acquired Available Secondary Memory of DATACENTER_0
Acquired Available RAM Size of DATACENTER_0
Acquired Available Bandwidth of DATACENTER_0
<< All Parameters are Acquired for >> DATACENTER_0
<< Logging Score  >>DATACENTER_0 as
 Identifier_: DATACENTER_0
 Secondary Memory: 80.381774902
 MIPS: 1.0
 RAM: 71.0
 Bandwidth: 80.3125
<< Logged server score as 58
Acquired Available MIPS of DATACENTER_1
Acquired Available Secondary Memory of DATACENTER_1
Acquired Available RAM Size of DATACENTER_1
Acquired Available Bandwidth of DATACENTER_1
<< All Parameters are Acquired for >> DATACENTER_1
<< Logging Score of Score of >>DATACENTER_1 as
 Identifier_: DATACENTER_1
 Secondary Memory: 10.38177490234375
 MIPS: 30.0
 RAM: 25.0
 Bandwidth: 20.3125
<< Logged server score as 14
            .
            .
```

Figure 5-4: Score Calculation

5.4.3 Group State Manager

In this phase a list which contains the score of available data centers is received as a return value. Following that, data centers will be categorized into groups. For ease of implementation this prototype formulates a maximum of 10 predefined groups to categorize resources. As it is shown in Figure 4-1, the group state manager is highly dependent on the score of available servers. Specifically, the group state manager uses a “Half Down” rounding mode to exploit the category of each data center.

As it is mentioned in Chapter Four, group formulation and score calculation tasks go hand in hand. Hence, one of the behaviors of this class is to identify the belongingness of resources to a group. The final output of this operation is shown in Figure 5-5. In this phase, resources with

similar score are categorized together as a group. Therefore, the maximum possible number of groups that can be formulated in a given reconfiguration interrupt is decided here.

```
System checking the group state...
DATACENTER_0 is registered as a member of Group ... 6
.
.
.
System checking the group state...
DATACENTER_1 is registered as a member of Group ... 2
```

Figure 5-5: Group State Management

5.4.4 Job Creation

This component is responsible to implement user actions by creating various size jobs throughout the scheduling period. As it is known, executing such type of operations must have a multi-threaded component in order to run concurrently. Hence, this prototype implements a multi-threaded component to concurrently generate and send jobs to the cloud environment. Basically, a code snippet that is callable by the thread body and used to concurrently generate various size jobs is shown in Figure 5-6. For ease of demonstration, we used a random number generator to differentiate the length of each job/cloudlet.

```
public static List<Cloudlet> createCloudlet(int userId, int
cloudlets, List attributes){
LinkedList<Cloudlet> jobQueue = new LinkedList<Cloudlet>();
//Creating object type cloudlet

int jobLength []={60000, 63000, 69000,75000, 77000, 81000,
86000, 89000, 92000, 98000, 105000, 107000, 110000, 113000,
119000};

//job creation
Cloudlet[] cloudlet = new Cloudlet[cloudets];
for(int i=0;i<= cloudets;i++)
                    {
cloudlet[i] = new Cloudlet(i, jobLength[i], attributes[0],
attributes[1], attributes[2], attributes[3], attributes[4],
attributes[5]);
// setting the owner of these Cloudlets
                cloudlet[i].setUserId(brokerId);
                jobQueue.add(cloudlet[i]);

}
           }
           return jobQueue;
     }
```

Figure 5-6: Job/Cloudlet Generator

This component concurrently and continuously generates jobs with different characteristics and logs it to the buffer named job queue. Figure 5-7 shows a part of the system output generated while creating cloudlets/jobs for the very first time. For the purpose of this prototyping we managed this component to generate jobs stated in Tables 5-1 and 5-2.

```
Looking for incoming jobs...
System Accepted a Cloudlet with the following requirement
 Job with id 0 have the following features
Length: 100 Core: 1 filesize: 300 outputsize: 300

 Job with id 1 have the following features
Length: 500 Core: 1 filesize: 1500 outputsize: 1500

 Job with id 2 have the following features
Length: 700 Core: 1 filesize: 2100 outputsize: 2100

 Job with id 3 have the following features
Length: 500 Core: 1 filesize: 1500 outputsize: 1500    . . .
```

Figure 5-7: Concurrent Cloudlet Creation

As it is depicted in Chapter One this thesis is not dedicated to invent the wheel for virtual machine creation and scheduling. However, in order to support the dependency between cloudlets and virtual machines we managed this model to include cloudlet based virtual machine creation and implemented spaceshared virtual machine provisioning mechanism. There are two major scenarios here. The first scenario is when the system creates virtual machines for the first time and the second scenario is when there are either occupied or free virtual machines already created in the cloud environment. In the first scenario, the system scans the attributes of existing cloudlets and creates a virtual machine for each cloudlet keeping the distribution of virtual machines across data centers.

```
>> Checking the availability of resources for the incoming cloudlets …

Cloudlet to Host Binding confirmed ….

Submitting cloudlet list to the job consolidator
```

Figure 5-8: Cloudlet Resource Bind Confirmation

Figure 5-8 shows the output of this component. The later scenario depicts the cloud environment in which virtual machines are in some extent occupied. Following reconfiguration interrupt, the

system checks the capacity of released virtual machines and assigns incoming cloudlets to them. Figure 5-9 shows a snippet of output generated by this component.

```
>> Reconfiguration Interrupt is triggered ….
System checking reusability of existing virtual machines
00:00 Found 23 reusable virtual machiness
Destroying used virtual machines …
Searching for backlogs . . .
Pseudo cloudlet Enhancement is in progress …
11 cloudlets were prioritized
Broker is sending the list to job consolidator …
```

Figure 5-9: Cloudlet to Resource Binding

5.4.5 Job Consolidator

Job consolidation is the major component of the proposed algorithm. In this phase, cloudlets fed from the user are categorized into their respective groups. In order to perform consolidation, the total serving capacity of data centers per group must be known. Figure 5-10 shows the process of exploiting the capacity of every available group in the cloud computing environment. Besides that, the system checks whether the overall request can be addressed by the existing resources or not.

```
Gathering the group capacity in the cloud...
                            ... ...
                                  ... ...
System acquired the capacity of each Group
+----------------------------- Group 1 -------------------------------+
+    Storage: 25511135    MIPS: 210    RAM: 1020    Bandwidth: 5000 +
+----------------------------- Group 2 -------------------------------+
+    Storage: 54430400    MIPS: 500    RAM: 2560    Bandwidth: 5200 +
```

Figure 5-10: Group Level Capacity

In addition, Figure 5-11 shows the process of normalizing job requests by resource capacity. Given an array of values to normalize cloudlets, this component decides on how many cloudlets to send to each data center.

```
>> Checking cloudlet addressability ….
Confirmation: Creating cloudlet to resource normalization …
42 cloudlets are sent to 3 Groups
Sending consolidation information to group level Scheduler
```

Figure 5-11: Job Consolidation

5.4.6 Group Level Adaptive Scheduling

As it is briefly discussed in Section 4.1, the group level adaptive job scheduler is the last component of the system which decides on how to bind cloudlets to virtual machines. Besides that, this component is responsible for tweaking the "submitcloudletlist()" and "CreateVirtualMachine()" methods of the simulator. Given previously defined parameters for contextualizing the scheduling policy, this component fetches data from the persistent model for group level policy alteration.

5.4.7 Threshold Calculation

Threshold calculation is one of the most important features of the proposed model. This component persists the feasibility of the proposed algorithm by keeping track of interrupts during the assignment and reallocation of cloudlets. Specifically, this component is implemented using a multi-threaded method that conquers various interrupts, especially virtual machine release and cloudlet submit interrupts, and decides whether to initiate or not to initiate the group state manager.

5.4.8 Job Prioritization

This component is implemented in a way to show how the proposed algorithm eliminates problems related to job starvation. As it is mentioned in Chapter Four the default priority of any job in its very first creation is 5. However, if the reconfiguration interrupt is generated, the cloudlet priority will be set to 10 which is the highest one. As it is briefly discussed in the previous Chapters, this could help to overcome starvation among cloudlets.

5.5 Test Results

A prototype is developed on a machine which has Intel(R) core(TM) i3 Processor 2.6 GHz, Windows 7 platform and using CloudSim 3.0.3 simulator. The CloudSim toolkit supports modeling of cloud system components such as data centers, host, virtual machines, resource provisioning policies, and mainly cloudlet scheduling platforms.

The experiment is held on the two cloud models shown in Tables 5-1 and 5-2. The two major cloud models are used to run various types of cloudlets to show the performance of the proposed scheduler in both resource constrained and resource unconstrained cloud environments. In addition, two major group of cloudlets are fed to the scheduler.

On the other hand, the performance of the proposed scheduling algorithm is presented in comparison with First Come First Served (FCFS) and Round Robin job scheduling algorithms. These algorithms are selected based on two major reasons. Primarily, we considered the popularity of the scheduling algorithm. FCFS and Round Robin scheduling algorithms are well known scheduling algorithms which are usually considered as a standard for comparing the proposed scheduling model with other existing models. Secondly, the capability of the simulation toolkit is considered. In CloudSim, the implementation of the aforementioned scheduling algorithms are likely to be found than other scheduling algorithms. The following Sections briefly describe the results gained after an intensive and various undertaking of experiments.

5.5.1 Running Few Resource Demanding Cloudlets on Model I

This Section presents how the proposed and other job scheduling algorithms perform where there is a need to allocate few resource demanding cloudlets to resource constrained cloud environment. Table 5-3 shows the number and demands of cloudlets used in this experiment. As it is known, most resource scheduling algorithms perform well when executing cloudlets with smaller demand of cloud resources. This is the scenario which could be mentioned as the best case scenario for most scheduling algorithms. However, most job schedulers may not have the same performance in resource constrained environment. In this Section, the performance of distributed score based job scheduling algorithm relative to other scheduling algorithms is presented.

Table 5-3: Few Resource Demanding Jobs

Job Id	Resource Requirement			
	Length (MI)	File Size (Byte)	Output Size	Processing Elements
0	5, 000	80	10	1
1	10, 000	120	30	1
2	15, 000	150	45	1
3	20, 000	170	60	1
4	25,000	230	80	1
5	30, 000	260	110	1
6	35, 000	290	130	1
7	40, 000	320	150	1
8	45, 000	350	180	2
9	50, 000	390	210	2

A. First Come First Served Scheduling Algorithm

The performance of FCFS scheduling algorithm is depicted in Figure 5-12. The minimum response time of the scheduler given a cloudlet with smaller resource requirement is 33.43 millisecond. The performance of this scheduler is highly compromised as the demand of incoming jobs in the resource constrained environment increase. Consequently, the maximum execution time of FCFS scheduling algorithm given a list of cloudlets with relatively small amount of resource demand is 158.63 millisecond. This result shows how the execution time of the cloudlets compromised as the number of cloudlets increase in resource constrained cloud environment.

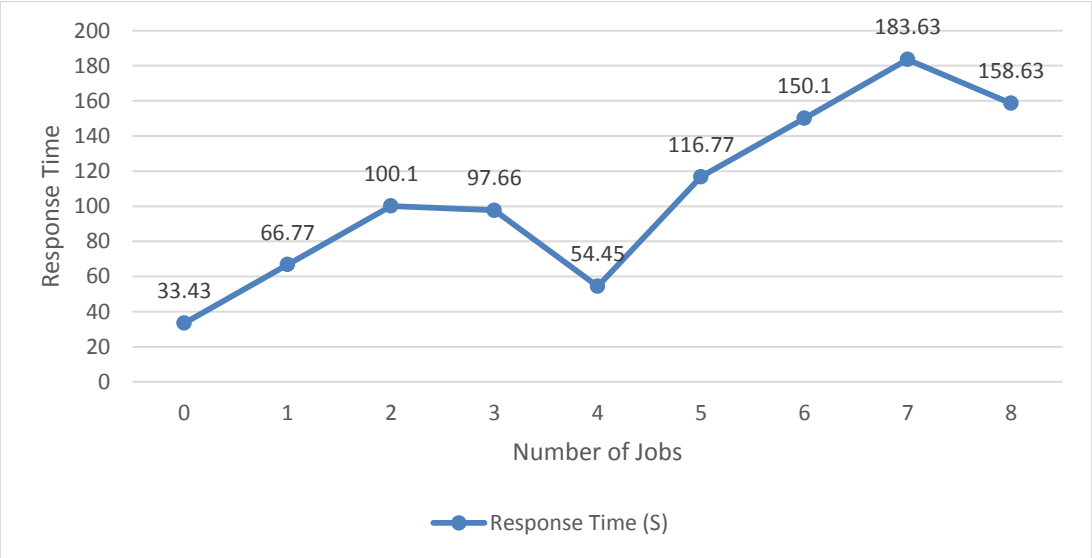

Figure 5-12: Results of FCFS - Scenario I

Generally, FCFS job scheduling algorithm happens to have a greater susceptibility to resource constraints size and number of jobs. This is because of the poor allocation policy of this scheduler and ill consideration of resources in the computing system.

B. Round Robin Job Scheduling Algorithm

This scheduling algorithm happens to have a better response time relative to FCFS scheduling algorithm. This is because of the characteristics of the scheduler that is a better insight on the overall characteristics of the cloud model. Moreover, as it is shown in Figure 5-13, the demand of cloudlets that do not exceed the capacity of the cloud model does not degrade the performance of this scheduler. However, with increased number of cloudlets and with cloudlet requirement exceeding the capacity of the cloud, the Round Robin scheduling algorithm happened to compromise its performance.

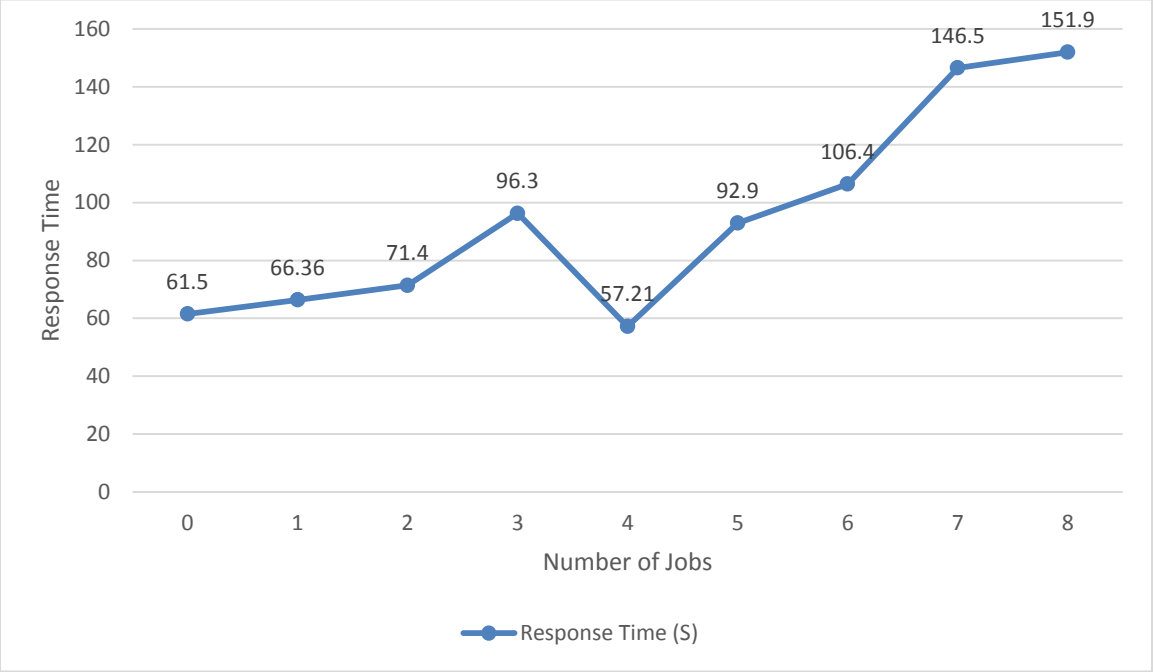

Figure 5-13: Results of Round Robin - Scenario I

Figure 5-13 also shows how the performance of this scheduler degrades when the amount of cloudlets exceeds the serving capacity of host in cloud model. However, time bounded job execution of this algorithm gives a better result over First Come First Served (FCFS).

C. Distributed Score Based Job Scheduling Algorithm

In this resource constrained computing environment, the proposed algorithm shows the best response time relative to other scheduling algorithms. Various factors contributed for this result. The first one is DSBJSA is not much affected by the serving capacity of the cloud model given a small demand of cloud resources. The second factor is that distributed score based job scheduling algorithm uses an important score collection and utilization mechanism that lets the scheduler know the state of each datacenter and host in the model. This in turn enabled a fair distribution of cloudlets among machines in the cloud. Moreover, the fair distribution of cloudlets on resource-aware manner helped the scheduler to consolidate and assign cloudlets to the right point of execution.

In addition, as it is shown in Figure 5-14, the rate of change in response time of DSBJSA given increased demand of resources is insignificant than other types of scheduling algorithms.

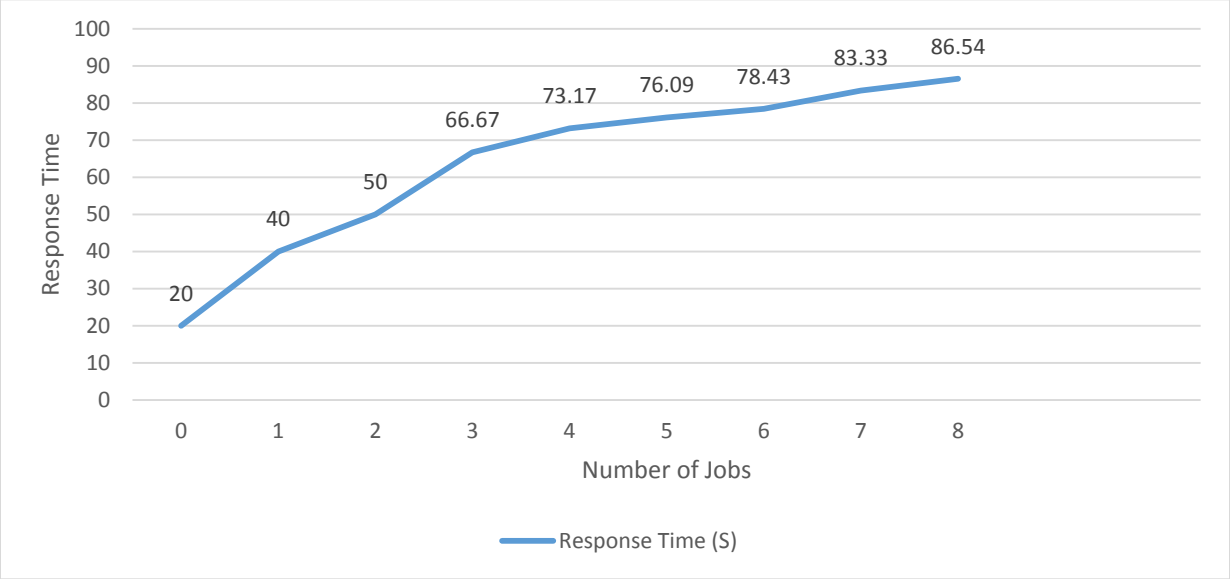

Figure 5-14: Results of DSBJSA - Scenario I

This outstanding performance is achieved because of two major components incorporated in the scheduling architecture namely: group state manager and group level adaptive scheduler. The group state manager contributes a lot to identify the variation in capacity among cloud resources and establishing a category of resources based on their serving capacity. Similarly, the group level adaptive job scheduler, taking the output from job consolidator, makes cloudlet to virtual machine binding efficient. Moreover, the characteristics of the aforementioned and other

components enabled efficient job assignment and also provided higher degree of resource utilization.

Figure 5-15 shows the performance of the proposed scheduling algorithm in relation to First Come First Served and Round Robin job scheduling algorithm given resource constrained cloud environment. The graph mainly shows two major strengths of the proposed algorithm over the counterparts. The first feature that could be observed from Figure 5-15 is the performance of the job scheduler from the response time point of view. The minimum response time of distributed score based job scheduling algorithm is 20 which is 13.43 and 37.4 less than the response time of FCFS and Round Robin scheduling algorithms respectively. Similarly, the maximum response time of the proposed scheduling algorithm outperforms FCFS and Round Robin scheduling algorithm by 97.09 and 64.46 less response time respectively. Furthermore, the average response time of DSBJSA is 63.8 which is an outstanding result relative to FCFS and Round Robin's average response time 106.84 and 94.49 respectively. The second salient feature that differentiate the scheduling algorithms is the degree of their susceptibility for radical change in resource requirements.

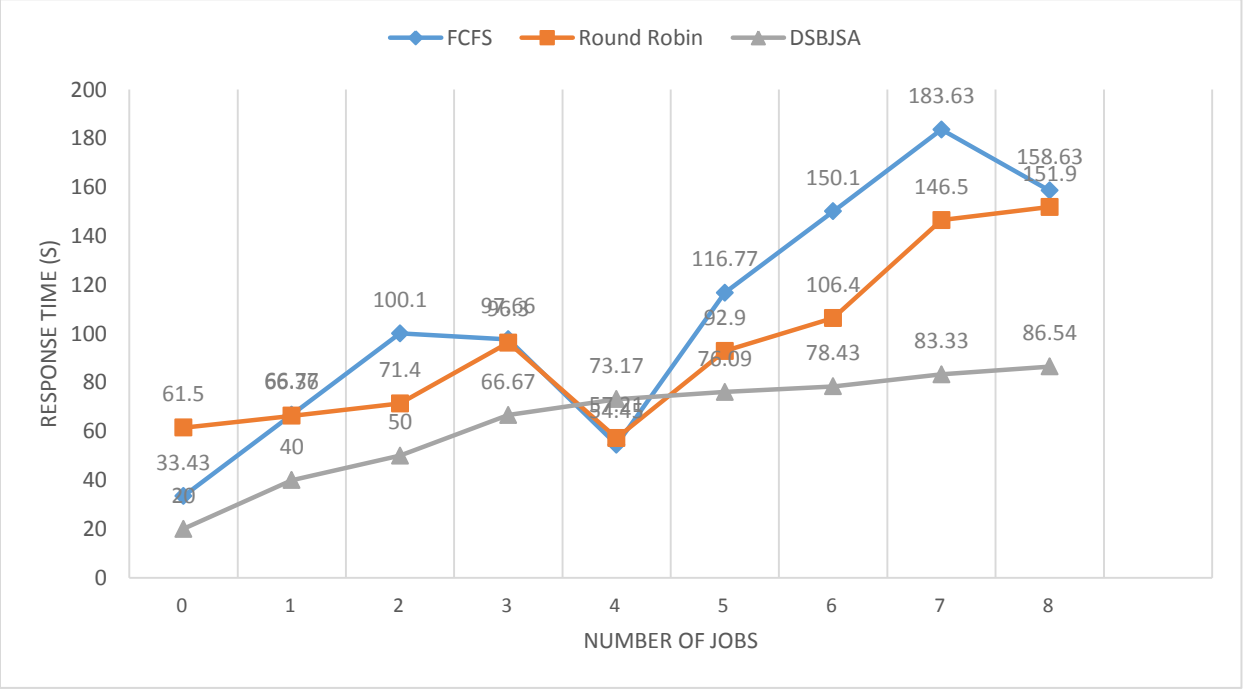

Figure 5-15: Comparison of FCFS, Round Robin and DSBJSA - Scenario I

This clearly shows how well the system balances load among various machines and especially it shows how the load balancing mechanism boosted the degree of resource utilization.

5.5.2 Running Larger Cloudlets on Model I

Running fewer cloudlets in resource constrained cloud model might not have a significant effect on any job scheduling algorithm. However, the impact of resource constrained cloud model is higher when jobs to be executed have greater demand of cloud resources. Hence, in this Section the performance of distributed score based job scheduling algorithm as well as other selected scheduling algorithms is presented. The type of cloudlets used during the experiment is shown in Table 5-4.

Table 5-4: Large Resource Demanding Jobs

Job Id	Resource Requirement			
	Length (MI)	File Size (Byte)	Output Size (Byte)	Processing Elements
0	60, 000	600	230	1
1	63, 000	630	300	1
2	69, 000	650	450	1
3	75, 000	680	460	1
4	77,000	710	475	2
5	81, 000	790	486	2
6	86, 000	800	505	2
7	89, 000	950	511	2
8	92, 000	960	520	3
9	98, 000	965	521	3
10	105, 000	970	526	3
11	107, 000	975	529	3
12	110, 000	975	531	4
13	113, 000	981	545	4
14	119, 000	991	547	4

A. First Come First Served Job Scheduling Algorithm

In this Section the performance of FCFS scheduling algorithm while provisioning resource demanding cloudlets on resource constrained cloud model is presented.

As it is shown in Figure 5-16, the performance of FCFS job scheduling algorithm is highly compromised and degraded when the resource requirement exceeds the available processing and storage capacity of the cloud.

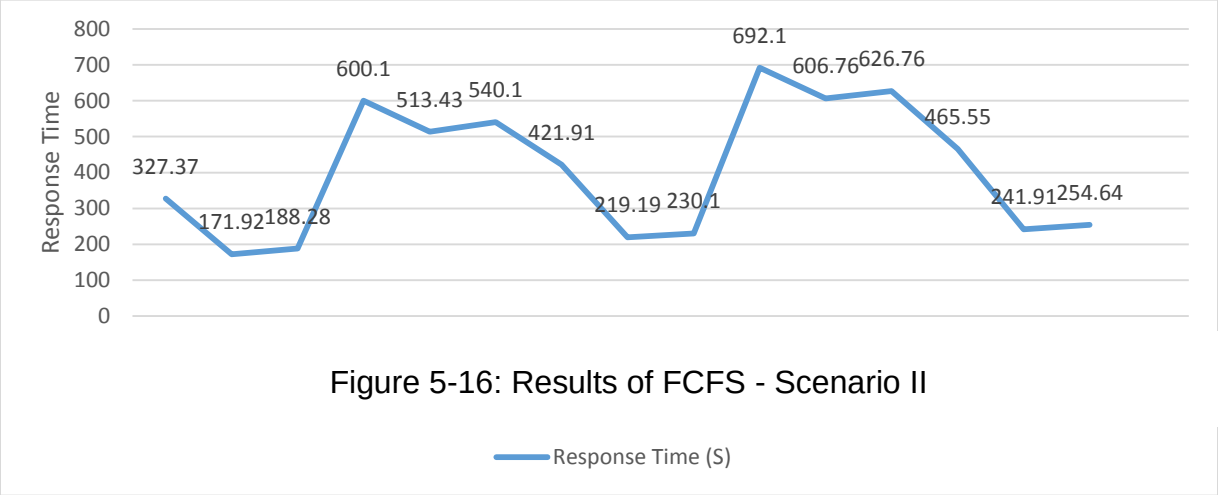

Figure 5-16: Results of FCFS - Scenario II

Figure 5-17: Results of FCFS - Scenario II

Moreover, the load balancing scheme of this algorithm is poor. It is observed that not more than thirty percent of the cloud resources are effectively used during an iterative experiment and evaluation of this algorithm. The percentage is calculated by taking utilized resources with respect to the total number of resources the algorithm should have considered. Figure 5-16 shows the performance of this algorithm from the response-time point of view.

B. Round Robin

Round Robin job scheduling algorithm exhibits degraded performance even relative to its own performance in scenario I. The minimum scheduling time used to schedule cloudlets shown in Figure 5-17 is 215.35 and the maximum time observed is 459.2. This clearly shows that the increment in size and number of cloudlets in resource constrained cloud environment highly affects its response time. The load balance made in this scheduling algorithm is relatively better than the FCFS algorithm which exhibited around thirty nine percent efficient resource utilization.

The visualized result of this algorithm shows relatively less stepper graph than FCFS. This could be used as the best way of depicting to what extent the Round Robin is affected by the change in request of cloud resources.

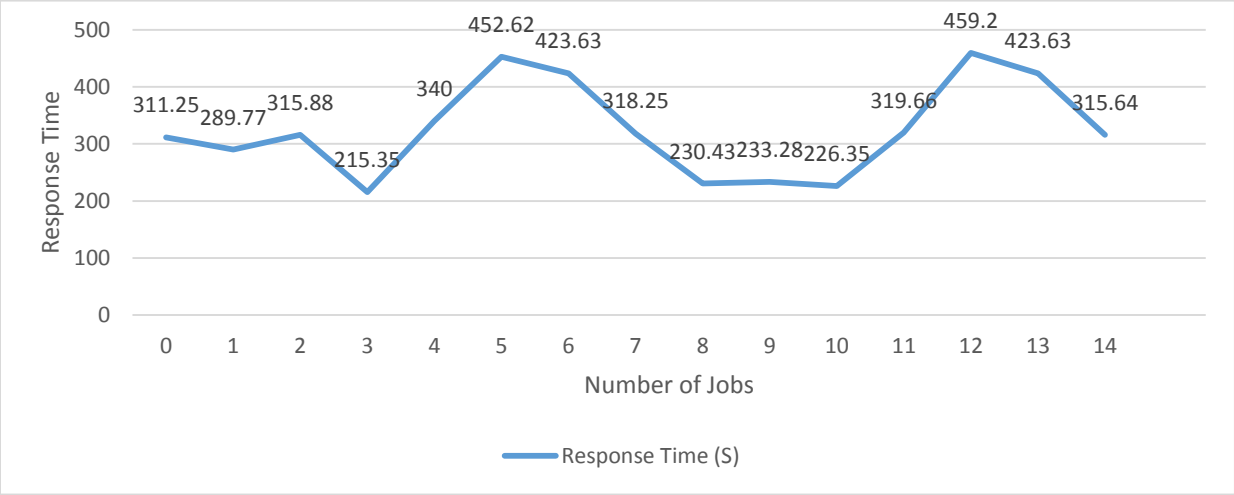

Figure 5-18: Results of Round Robin Scheduler -Scenario II

The more stepper the graph is the greater impact of the cloudlet size in a given cloud computing environment. Hence, in relation to FCFS algorithm, round robin happened to have a smaller rate of change in response time when there is a linearly growing resource demand.

C. Distributed Score Based Job Scheduling Model

This scenario could be seen as the best way of evaluating the performance of distributed score based job scheduling algorithm and the counter parts given the worst computing environment. It is also known that running resource demanding cloudlets in resource constrained cloud model could yield the worst possible performance of scheduling. The performance of Distributed Score Based Job Scheduler given cloudlets with larger resource demand is shown in Figure 5-18.

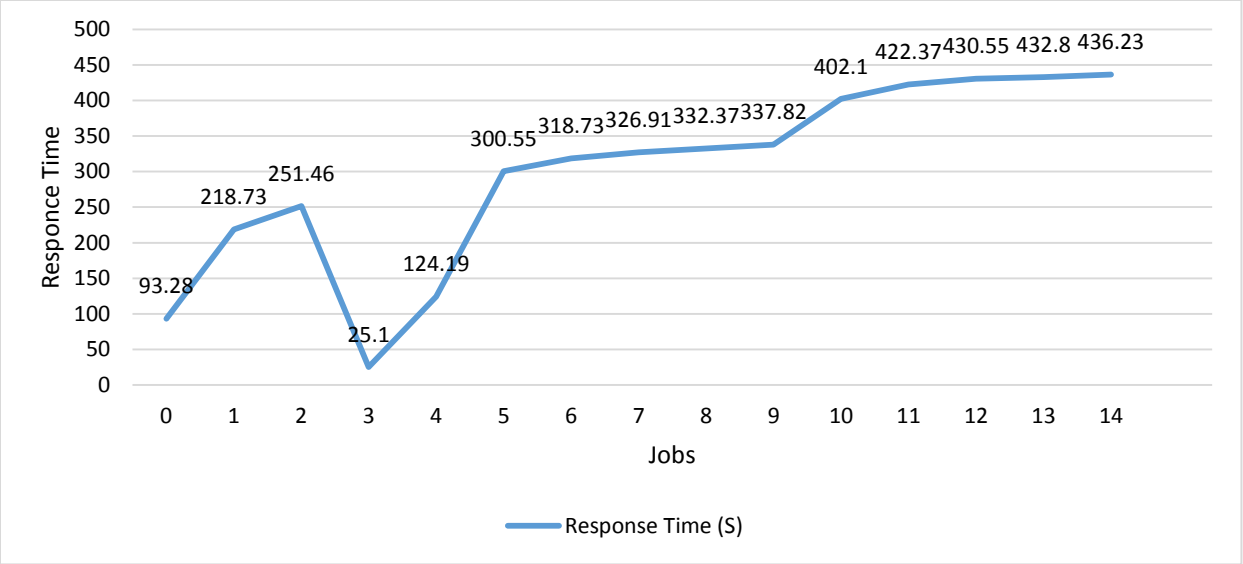

Figure 5-19: Results of DSBJSA - Scenario II

The performance of distributed score based job scheduling algorithm relative to its own performance shown in the first scenario is a bit degraded. However, DSBJSA outperforms FCFS and Round Robin scheduling algorithms given the same cloud model and larger resource demand. Furthermore, the minimum and maximum response time of cloudlets is 25.1 and 436.23 respectively. This result shows how well the proposed algorithm considers the cloud environment for efficient scheduling and provisioning of jobs and resources respectively.

On the other hand, larger cloudlets are run on resource constrained cloud model. Obviously, this scenario gives the worst performance for any job scheduling algorithm, so do for Distributed Score Based Job Scheduling Algorithm. Figure 5-19 depicts to what extent the proposed algorithm outperforms the counterparts in the worst computing environment.

Indeed, the proposed algorithm exhibits the worst performance in this scenario than the remaining other scenarios. However, the algorithm still outperforms FCFS and Round Robin. It is observed that the average response time of DSBJSA, 298.308, outperforms the FCFS and round robin by 108.36 and 26.24 respectively.

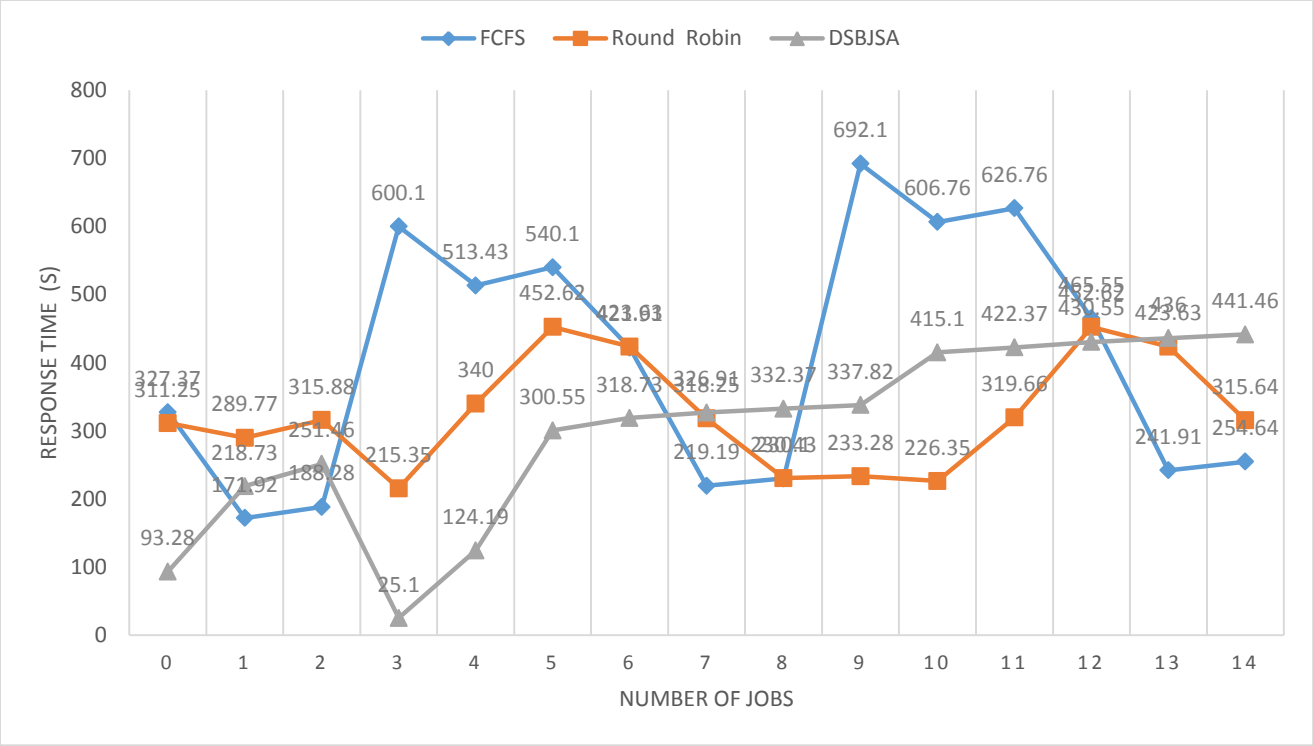

Figure 5-20: Comparison of FCFS, Round Robin and DSBJSA - Scenario II

5.5.3 Running Few Resource Demanding Cloudlets on Model II

This scenario presents how well the scheduling algorithms perform in few resource demanding cloudlets which are executed in resource unconstrained cloud computing environment. Obviously, a scheduler executing less resource demanding cloudlets in resourceful cloud environment exhibits an outstanding performance. This topic presents the performance of distributed score based job scheduling algorithm and the counterparts given the best case scenario both from the point of cloudlets and cloud resources. The type and number of cloudlets used in this experiment are shown in Table 5-3.

A. First Come First Served Scheduling Algorithm

A scheduling algorithm with an advanced resource utilization feature can obviously outperform in this kind of scenarios in which the cloud environment is not constrained by resources. Figure 5-20 presents the result of scheduling few resource demanding cloudlets in resource unconstrained cloud computing model. The result gained after successive experiments on a given cloud model is not surprising. This is mainly because of the resource unaware processing of

FCFS. The minimum and maximum response time of cloudlets scheduled in this scenario are 4.05 and 18.85 respectively.

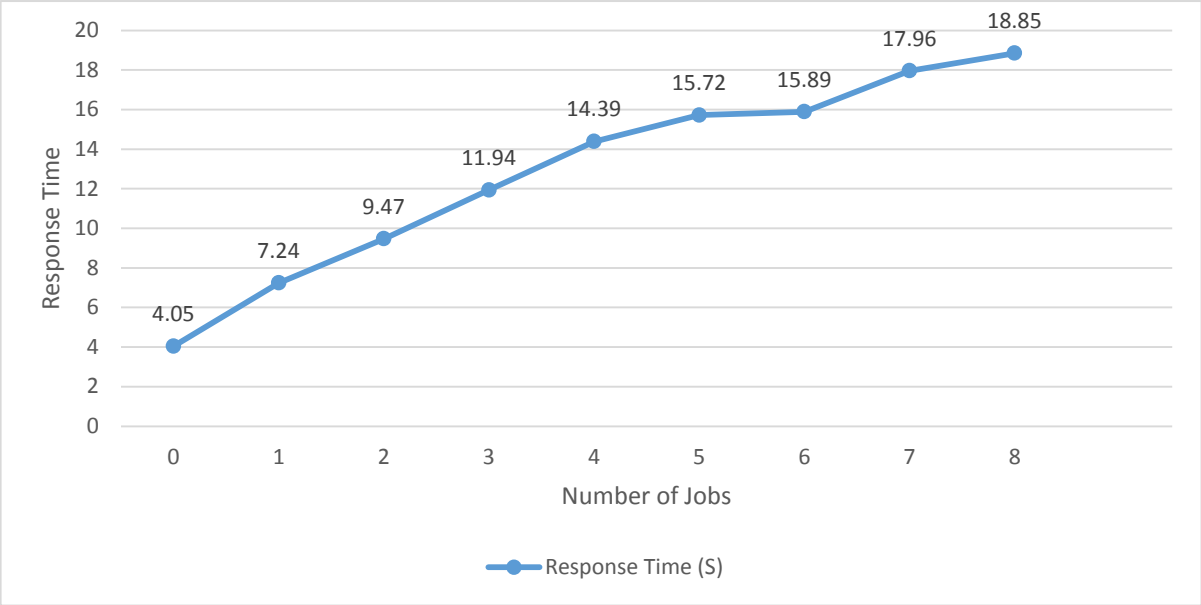

Figure 5-21: Results of FCFS Scheduling Algorithm: Scenario III

B. Round Robin Scheduling Algorithm

Round Robin scheduling algorithm improved its performance by taking the advantage of resources served in the given cloud environment. Hence, the performance of this scheduling algorithm is boosted relative to its previous performance in resource constrained environment. Figure 5-21 shows the performance of Round Robin in resourceful cloud environment.

The result of this scheduler is depicted in Figure 5-21. It is easily observable that the number of resources in cloud directly affects the performance of the Round Robin scheduling algorithm. Specifically, the minimum and maximum execution time of the cloudlets provisioned using round robin scheduling algorithm is 9.35 and 13.24 respectively.

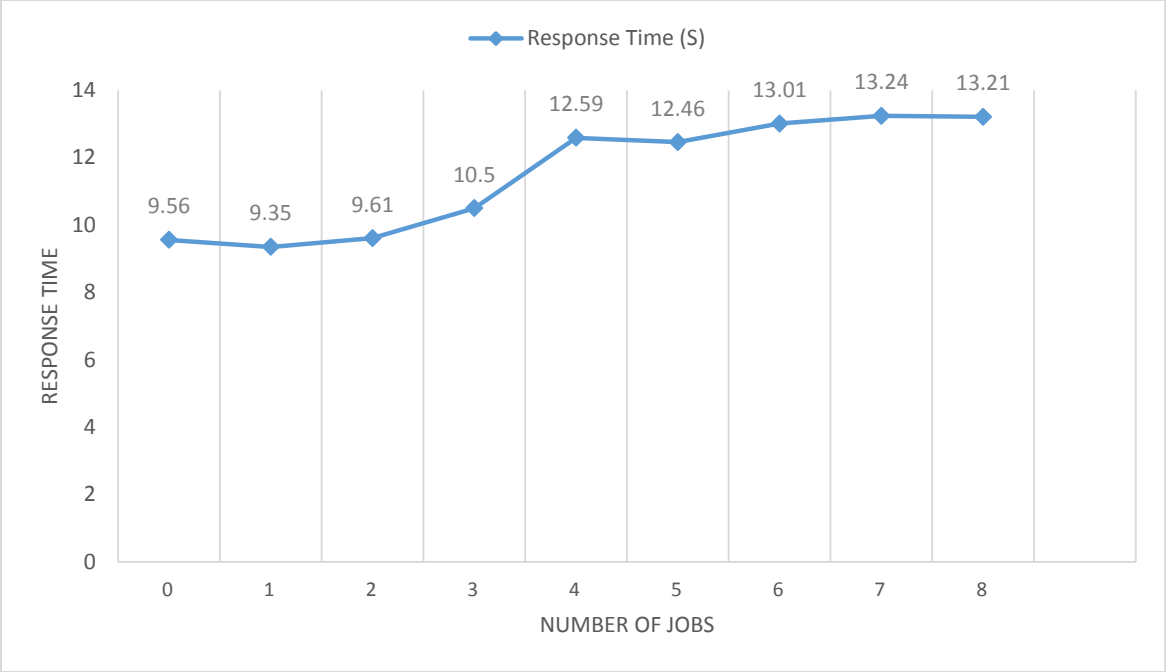

Figure 5-22: Results of Round Robin Scheduling Algorithm - Scenario III

C. Distributed Score Based Job Scheduling Model

This scenario could be stated as the best way to check the maximum achievable performance of Distributed Score Based Job Scheduling Algorithm. Generally, it could be used to evaluate the performance of resource aware scheduling algorithm given the best computing environment.

Distributed Score Based Job Scheduling Algorithm is among the few cloudlet scheduling algorithms which takes advantage of resource availability. Hence, the response time of this scheduling algorithm is by far smaller than its response time in resource constrained environment and others in resourceful computing environment. Figure 5-22, depicts the results acquired after executing few cloudlets in resourceful cloud environment.

It shows how Distributed Score Based Job Scheduling Algorithm takes advantage of well capacitated cloud model. Evaluating this scheduling algorithm from the point of load balancing, the maximum number of hosts as well as datacenters are used in efficient manner. Since the size of jobs does not exceed the processing capacity of the cloud, trying to address every available data center and host in each data center is controversial and unrealistic. However, the proposed algorithm yields better cloudlet distribution relative to other scheduling algorithms.

Moreover, Figure 5-22 also shows how well the algorithm performs in the given cloud environment. The graph specifically shows to what extent the performance of the scheduling algorithm is improved. The minimum and maximum response time of this scheduling algorithm are 3.43 and 7.39 respectively.

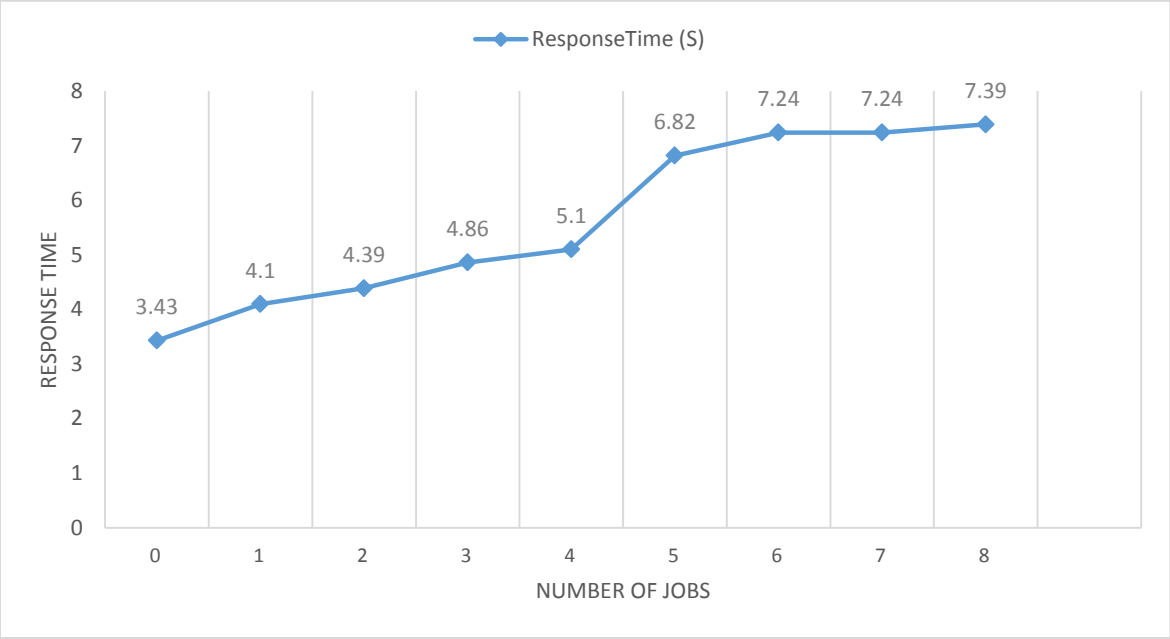

Figure 5-23: Results of DSBJSA - Scenario III

Generally, this scenario shows the performance of the scheduling algorithm running few resource demanding cloudlets in resourceful cloud model. It yields a better performance for both round robin and Distributed Score Based Job Scheduler. However, FCFS happened not to use the advantage of excessive resources in the cloud model. Hence, the proposed algorithm outperforms round robin algorithm from the response-time perspective. The comparison between the proposed scheduling algorithm and the best among the counterparts, Round Robin, yields an outstanding response time and resource utilization. Specifically, the average cloudlet execution time of DSBJSA, FCFS and Round Robin are 5.62, 11.51, and 12.83 respectively.

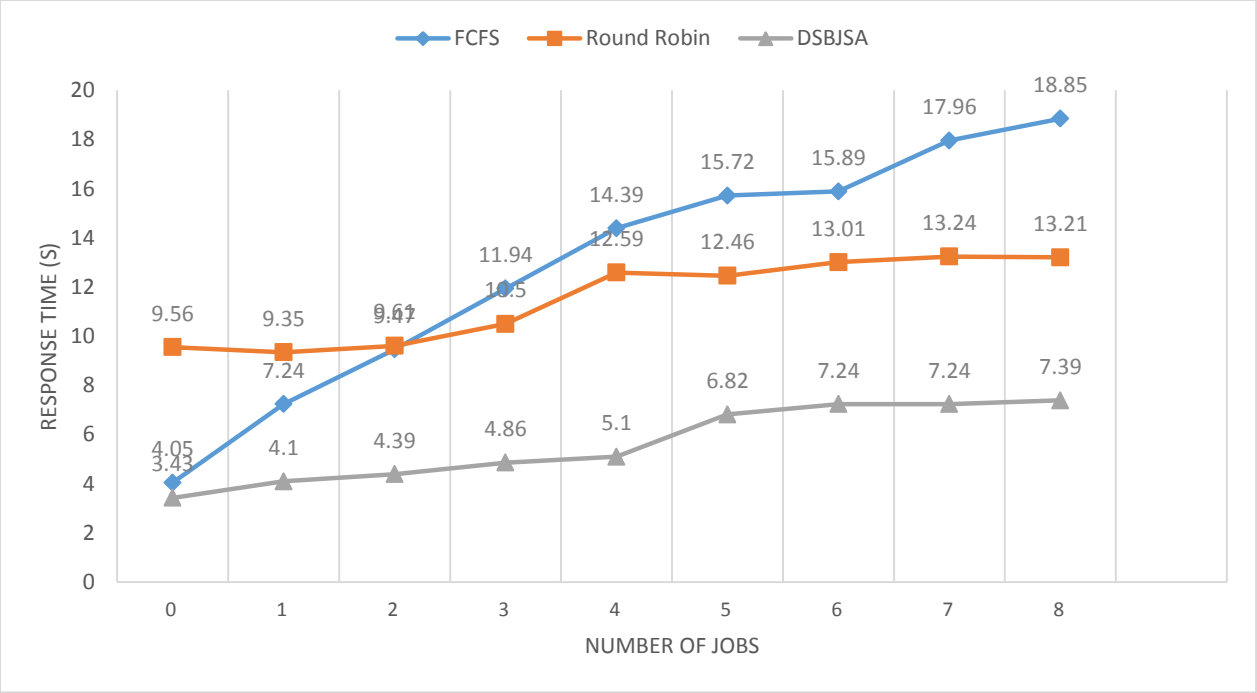

Figure 5-24: Comparison of FCFS, Round Robin and DSBJSA - Scenario III

5.5.4 Running Larger Cloudlets on Model II

In this Section the performance of the proposed scheduling algorithm as well as the counterparts while executing large cloudlets in resourceful cloud computing environment is presented.

A. First Come First Served Scheduling Algorithm

As it is discussed previously, FCFS scheduling algorithm does not take the advantage of resourceful cloud environment. It simply assigns the upcoming cloudlet for the first resource available in the list. Therefore, the performance of this scheduler is not usually impacted by the size and performance of resources in the cloud model.

Figure 5-24 shows the result gained after executing large cloudlets in resourceful cloud environment. It is observable that the existence of excess resources does not have much positive impact on this scheduler. The minimum and maximum response time of cloudlets in this scenario are 20.3 and 63.47 respectively.

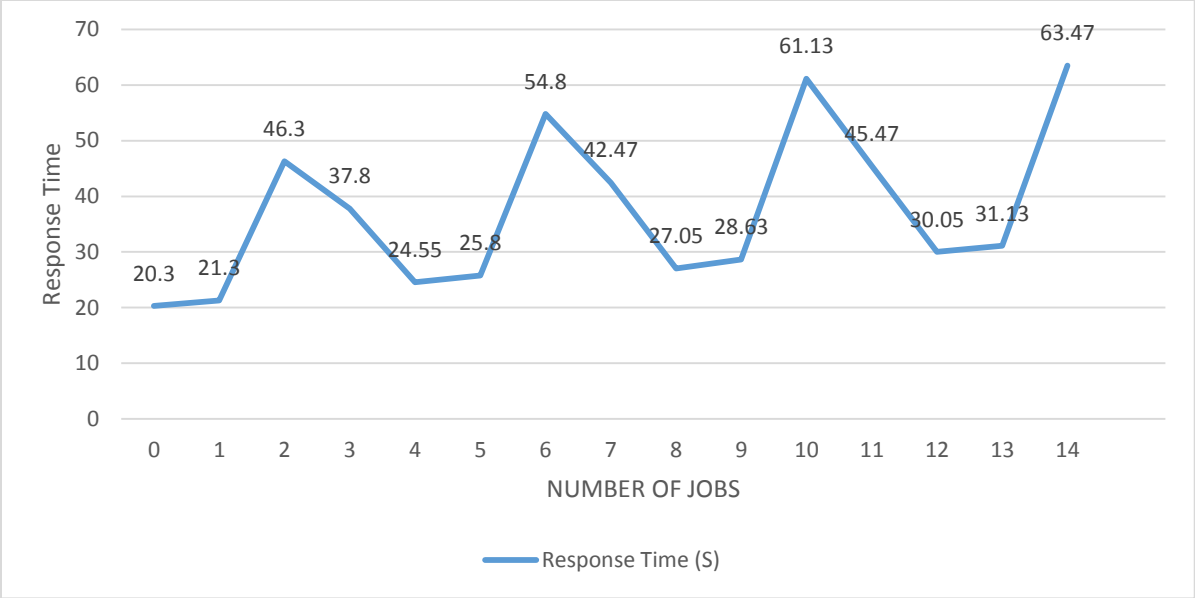

Figure 5-25: Results of FCFS Scheduling Algorithm: Scenario IV

B. Round Robin Scheduling Algorithm

Round robin as usual exhibits a better cloudlet scheduling performance relative to FCFS job scheduling algorithm. However, the disparity between the serving capacity of the cloud and cloudlets demand highly affected the performance of this scheduler.

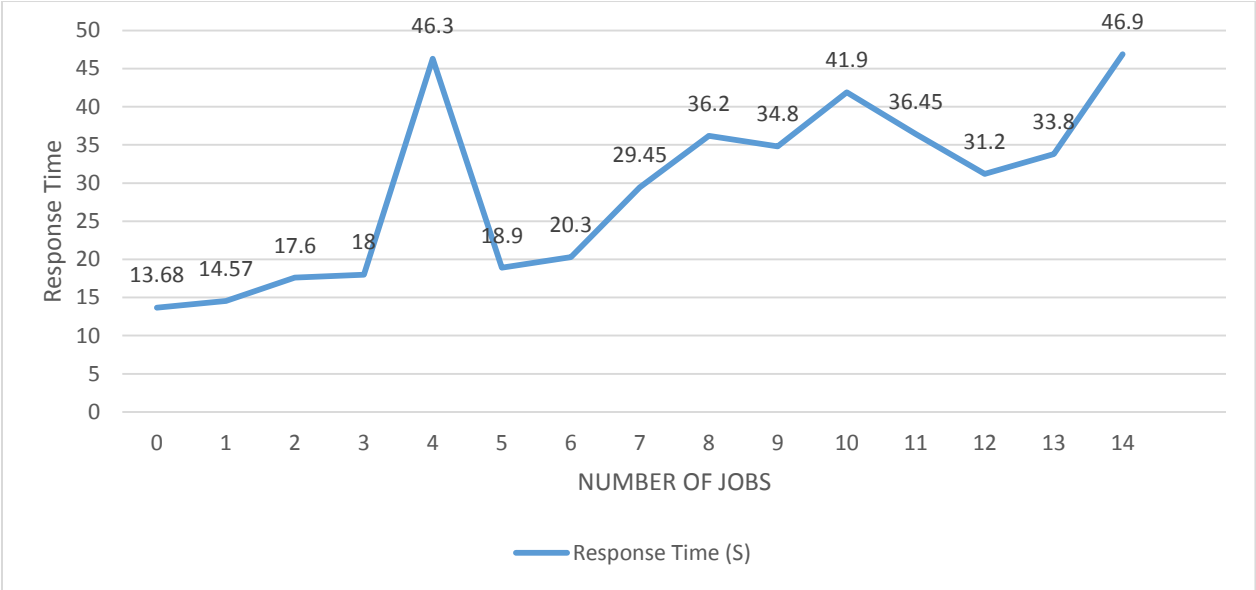

Figure 5-26: Results of Round Robin Scheduling Algorithm - Scenario IV

In addition, the result of the scheduling algorithm is shown in Figure 5-25. The graph shows the impact of increased size of cloudlets even in resource full cloud model. Generally, the minimum and maximum response time observed from in this scenario are 13.68 and 46.9 respectively.

C. Distributed Score Based Job Scheduling Model

As it is shown in the previous experiments, Distributed Score Based Job Scheduling Algorithm has a better degree of resource utilization. Figure 5-26 shows the performance of the proposed scheduling algorithm while running larger cloudlets in relatively resourceful cloud model.

The result clearly shows the efficiency of the proposed algorithm on utilizing available cloud resources. Hence, the minimum and maximum response time of cloudlets fed to the proposed scheduler are 8.67 and 17.63 respectively.

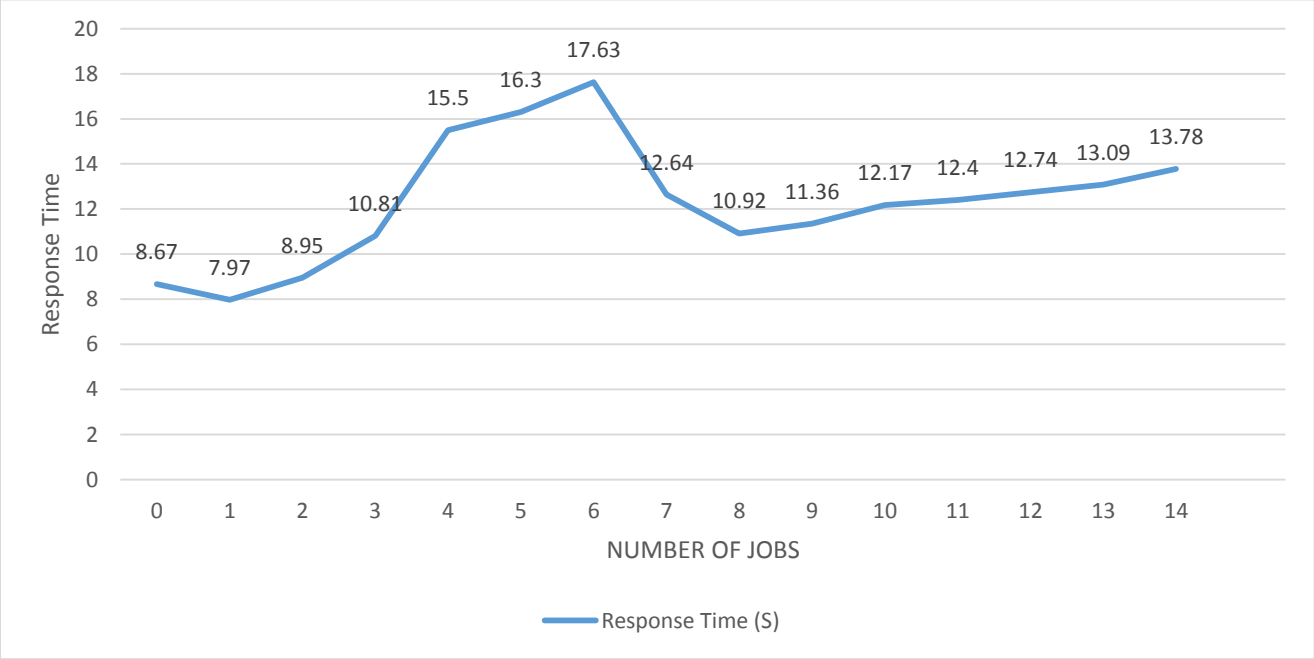

Figure 5-27: Results of DSBJSA - Scenario IV

In this scenario, the performance of the proposed scheduling algorithm is managed to run large resource demanding cloudlets in a well-equipped and resourceful cloud model. As it is shown in Figure 5-27, this scenario shows how well the proposed scheduler performs large cloudlet allocation in resourceful cloud computing environment. A job scheduler which has greater performance in such kind of computing environment could be mentioned as the one with greater degree of resource utilization.

Similarly, Distributed Score Based Job Scheduling Algorithm yields the best cloudlet scheduling performance relative to its own performance in the case of resource constrained model and its counterparts in resourceful cloud model. Hence, the minimum and maximum time consumed by the proposed algorithm are **7.97** and **12.74** which is better than the minimum and maximum response time of FCFS and Round Robin scheduling algorithm. Moreover, the greater performance acquired in the given scenario implies how efficient the proposed algorithm is from the perspective of resource utilization.

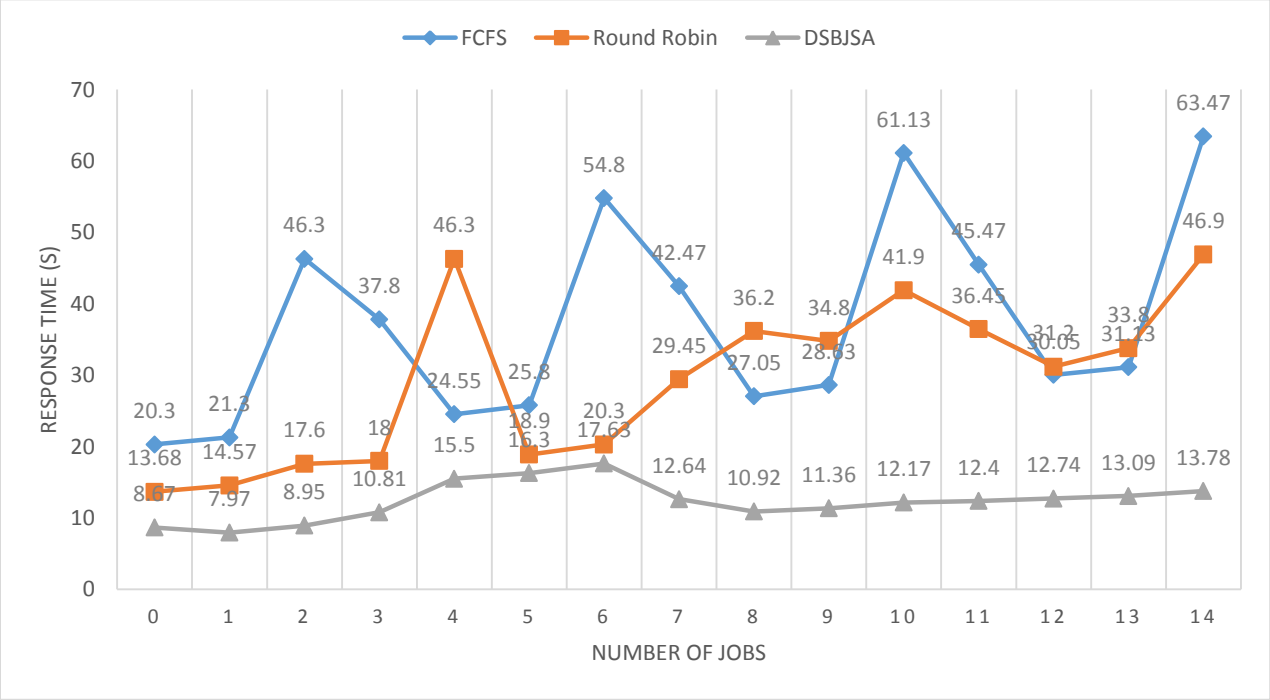

Figure 5-28: Comparison of FCFS, Round Robin and DSBJSA - Scenario IV

5.6 Summary

In this Chapter, a prototype for Distributed Score Based Job Scheduling Algorithm is developed using various system development tools including Java and cloudsim as a core simulation library. Various packages of the simulation toolkit as well as some built-in classes are overridden to create a conducive computing environment for Distributed Score Based Job Scheduling Algorithm.

Moreover, experimental results are conducted by creating two cloud models and running a variety of cloudlets in both models. We also presented performance of the proposed scheduling algorithm in comparison with FCFS and Round Robin scheduling algorithm. Specifically, the

outstanding performance of the proposed scheduling algorithm from the perspectives of response time, load balance and resource utilization is discussed in a much more detail.

The outstanding response time acquired while using Distributed Score Based Job Scheduling Algorithm is quantified and also presented in comparison with the other scheduling algorithms. On the other hand, a fair distribution of jobs among resource centers is presented as the major sign for how efficiently the proposed algorithm balances load among machines in the cloud computing environment. We also discussed the importance of every component conceived and implemented to acquire high degree of resource utilization.

Moreover, the result is presented based on four different scenarios. The selected scheduling algorithms are evaluated based on the time they take to allocate various jobs and how well it distributes incoming jobs to resource centers.

In the first scenario, all the three algorithms happen to have relatively better response time. As it is shown in the first column of Figure 5-28 (Scenario I), the average response time of the proposed scheduling algorithm excels FCFS and Round Robin by 43.1 and 30.7 respectively. Given a conducive cloud computing environment with small resource requiring jobs, exhibiting such an outstanding result could be mentioned as a salient feature of the proposed algorithm. The other important feature observed during this experiment is load balancing. In this specific experiment, FCFS scheduler happens to have poor load balancing capability. The degree of resource utilization exhibited by Round Robin relative to Distributed Job Scheduling Algorithm is not much. It is observed that the proposed algorithm takes advantage of resourceful cloud model.

The second Section of the experiment presents the performance of the proposed algorithm when large resource demanding jobs run on resource constrained cloud model. This experiment clearly shows how the proposed algorithm survives lack of resources even if there is large resource demand. Indeed, the proposed algorithm is a bit affected by the lack of resources in the cloud. However, the impact of lack of resources in FCFS and Round Robin scheduling algorithm is by 108.5s and 26.5s greater than DSBJSA. The result is also shown on the second column of Figure 5-28 (Scenario II). On the other hand, the degree of resource utilization of Round Robin and FCFS is by far less than the proposed scheduling algorithm.

The third case could be mentioned as the best case scenario for any kind of job scheduling algorithm. In this case the proposed scheduling algorithm as well as the counterparts happen to have better scheduling algorithm. However, the addition of new components enabled the proposed algorithm take advantage of resource unconstrained cloud computing environment. Hence, as it is shown in the third column of Figure 5-28 (Scenario III), the average response time of the proposed scheduling algorithm is by 65 and 53 less than FCFS and Round Robin algorithm. Besides that, regardless of Distributed Score Based Scheduling Algorithm the corresponding algorithms do not take the advantage of free resources.

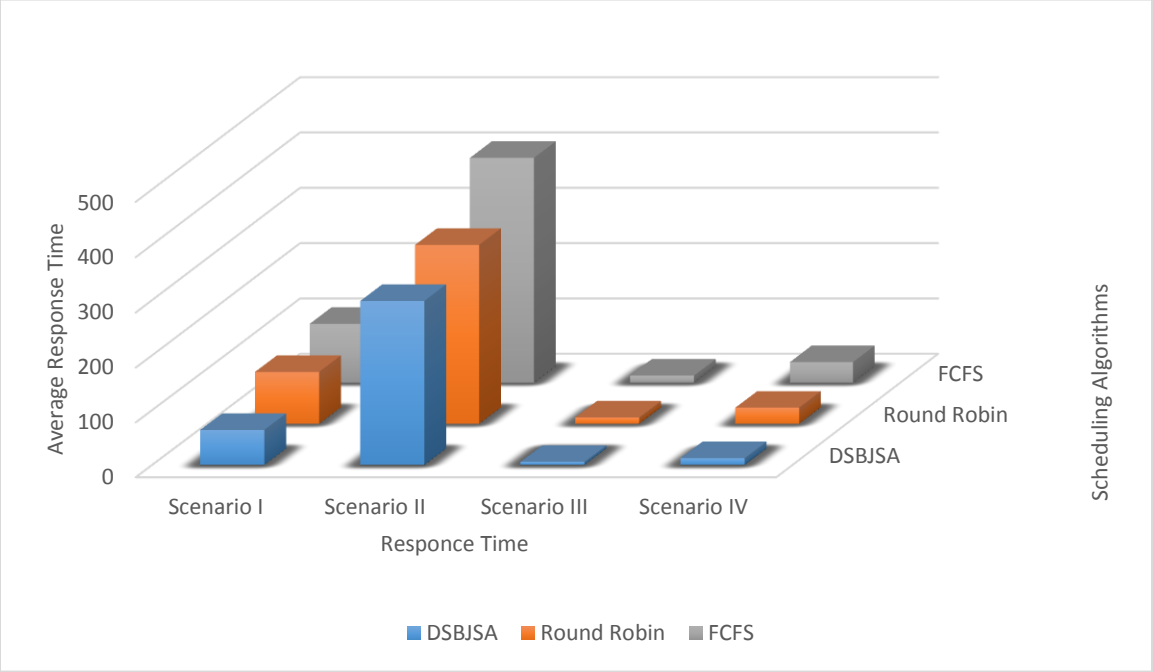

Figure 5-29: Average Response Time of FCFS, Round Robin, and DSBJSA in Four Different Scenarios

This clearly shows how the research conceived and implemented the best approach to address the problem of load balancing and resource utilization.

Finally, large resource demanding jobs are run on resource unconstrained environment. In this scenario, the significance of employing load balancing and resource utilization ingredients in the proposed scheduling algorithm is observed. The last column of the Figure 5-28 (Scenario IV), shows the average response time of the three algorithms. The average response time of the proposed scheduling algorithm is 18.7 which is by 375.5 and 255.5 less than FCFS and Round

Robin respectively. Moreover, Distributed Score Based Job Scheduling Algorithm exhibited better resource utilization and load balance.

Generally, use of the score of each resource centers in distributed manner and fitting in various conceived and adopted components contributed a lot for the outstanding performance of the proposed algorithm. Hence, this experiment clearly shows the outstanding performance of DSBJSA in relation to FCFS and Round Robin scheduling algorithms.

6. Conclusion and Future Work

This Chapter summarizes the major findings in this research work. Moreover, the salient contributions of the proposed job scheduling algorithm and future works are outlined.

6.1 Conclusion

Cloud computing environment is characterized by a collection of data centers that have high computational power than other technologies. It is true that having a collection of computing machines with higher processing and storage capacity contributes a lot for the overall performance of the computing system. However, the presence of well-equipped computing machineries by itself does not guarantee high performance.

This research work identified job scheduling approaches as the major contributing factors for the performance of the cloud. Moreover, absence of efficient job scheduling algorithm in resourceful cloud environment affects the performance of the computing system. On the other hand, a resource constrained cloud computing environment with relatively efficient scheduling mechanism has a better performance than those coupled with inefficient job scheduling algorithm.

Consequently, after an intense review of literatures and related works we have identified resource utilization, response time and load balancing as unresolved problems of cloud computing environment. Thus, we proposed efficient scheduling framework that could resolve the aforementioned three problems.

The research work solved the aforementioned problems by subsuming additional components to the default computing system architecture. The overall architecture of this work bases on a score calculating component which we call it “the corner stone” of the architecture. This component decides the score of each data center by considering its MIPS, RAM, Storage, PE, and BW in relation to the maximum available capacity in that computing environment. The value returned from this component is managed to be used by all other components. Besides that, the load balancing problem is resolved by introducing a “group state manager” into the architecture. This architecture contains various sub components that are responsible to categorize data centers and hosts based on their performance and job demands respectively. This component enabled a load aware resource provisioning and cloudlet scheduling. Furthermore, the highest cost of data center

reconfiguration is resolved by incorporating a threshold calculating component. This component enabled suppression of unwanted reconfiguration interrupts which are set to trigger based on some time interval. In the proposed model, reconfiguration interrupt is set to trigger based on the fulfillment of some threshold. The threshold is calculated by tracking the number and amount of cloudlets/jobs executed in a given computing environment. The other important component proposed in this work is group level adaptive job scheduler. Given the category of resources with different characteristics, we proposed an adaptive cloudlet scheduling algorithm that considers the capacity as well the volatility of the data centers. Resource and job starvation problem is also resolved by setting higher priorities for those jobs which are rescheduled.

The scheduling algorithm is examined in different scenarios. The variation of demand of cloudlets and serving capacity of the cloud are used to generate four different scenarios. The proposed scheduling algorithm outperforms the counterparts in all of the four scenarios. Specifically, the first and last scenarios are used to show how the proposed algorithm performs in situations where demand of cloudlets and available cloud resources are balanced. The second scenario is used to show how well the algorithm performs in the worst available scenario. Moreover, the third scenario shows the outstanding performance of the scheduler given conducive computational capability.

6.2 Contributions of the Research

The main contributions of this research work are summarized as follows.

- **Outstanding Scheduling Time**

The major goal of this research work is to improve response time. Hence, this research work conceived and implemented various components to achieve this goal. From an exhaustive experiment and evaluation it was possible to realize that the response time of the proposed scheduling algorithm is by far less than FCFS and Round Robin scheduling algorithms. Hence, we would say that DSBJSA has a substantial contribution for the major effort to solve the aforementioned problem in the cloud.

- **High Degree of Resource Utilization and Load Balancing**

This research work designed various components with the intention of solving the load balancing problem. Moreover, incorporation of group state managing component in a stepwise and layered manner eased an effort to achieve this goal. As it is known, balancing loads based on servers

score yields a better resource utilization. Hence, this research work also has a significant contribution in overcoming the major cloud computing problems namely: Load Balancing and Resource Utilization.

6.3 Future Work

The Distributed Score Based Job Scheduling Algorithm proposed in this research could be used in various cloud vendors after a bit of configuration undertakings. In addition, various research works could use the methodology and findings of this research work. We would say the proposed scheduling algorithm is fit to the current requirements of most cloud vendors though there are some issues that need further work. Hence, this research work presents different areas that can be further improved as well as some components that should be implemented and integrated for better functioning of the algorithm.

- **Threshold Calculation**

Incorporating machine learning approaches to calculate the threshold for both reconfiguration interrupt and load balancing yields a better result.

- **Score Calculation**

It would be better if the score calculating component takes the default capacity of the cloud automatically.

- **Service Level Agreement**

The proposed algorithm does not predict the characteristics and demand of cloud users. It just distributes cloudlets on arrival. However, it is observed that considering the trend of requests per user level could help to achieve high level of service level agreement satisfaction. Hence, it would be important to incorporate a component that learns from user's trends and handle further requests based on users past trend. The result of this component could also be used as an ingredient in group state manager.

- **Adaptive Job Scheduling**

Since we already got satisfactory result in the preceding components much attention has not been paid for group level adaptive job scheduling. However, incorporating additional feature vectors to decide the behavior of resource centers could improve the response time and resource utilization.

References

[1] Barrie Sosinsky, "Cloud Computing Bible", Wiley Publishing, Inc., Indianapolis, Indiana, 2011.

[2] Peter Mell and Timothy Grance, "Cloud Computing", National Institute of Standards and Technology - Computer, 2013.

[3] Yatendra Sahu and R.K. Pateriya, "Cloud Computing overview with Load Balancing Techniques", International Journal of Computer Applications, Vol. 65, No. 24, March 2013.

[4] Wenke Ji and Jiangbo Ma, "A Reference Model of Cloud Operating and Open Source Software Implementation Mapping", in the 18th IEEE International Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises, 2009.

[5] R. Gogulan, A. Kavitha, and U. Karthick Kumar, "A Multiple Pheromone Algorithm for Cloud Scheduling With Various QoS Requirements", International Journal of Computer Science Issues (IJCSI), Vol. 9, Issue 3, No. 1, May 2012.

[6] V. Venkatesa Kumar and K. Dinesh, "Job Scheduling Using Fuzzy Neural Network Algorithm in Cloud Environment", Bonfring International Journal of Man Machine Interface, Vol. 2, No. 1, March 2012.

[7] Mousumi Paul and Goutam Sanyal, "Task-scheduling in Cloud Computing using Credit Based Assignment Problem", International Journal of Computer Science and Engineering, 2013.

[8] S. Sindhu and Saswati Mukherjee, "Efficient Task Scheduling Algorithms for Cloud Computing Environment", High Performance Architecture and Grid Computing Communications in Computer and Information Science, pp 79-83, 2013.

[9] Miyuki Sato, "Creating Next Generation Cloud Computing Based Network Services and the Contribution of Social Cloud Operation Support System (OSS) to Society", 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, 2009.

[10] Ubramanian S., Nitish Krishna G., Kiran Kumar M., Sreesh P., and G. R. Karpagam, "An Adaptive Algorithm for Dynamic Priority Based Virtual Machine Scheduling in Cloud", International Journal of Computer Science Issues, Vol. 9, Issue 6, No. 2, November 2012.

[11] Yiqiu Fang, Fei Wang, and Junwei Ge, "Task Scheduling Algorithm Based on Load Balancing in Cloud Computing", WISM'10 Proceedings of the 2010 International Conference on Web Information Systems and Mining, 2010.

[12] Kiran Kumar Shakya1 and D. Singh Karaulia, "A Process Scheduling Algorithm Based on Threshold for the Cloud Computing Environment", International Journal of Computer Science and Mobile Computing (IJCSMC), Vol. 3, No. 4, April 2014.

[13] Q. Yin, A. Schüpbach, J. Cappos, A. Baumann, and T. Roscoe, "Rhizoma: A Runtime for Self deploying, Self-managing Overlays", Lecture Notes in Computer Science, pp. 184-204, 2013.

[14] Meng Xu, Lizhen Cui, Haiyang Wang, and Yanbing Bi, "A Multiple QoS Constrained Scheduling Strategy of Multiple Workflows for Cloud Computing", Parallel and Distributed Processing with Applications, IEEE International Symposium, Aug, 2009.

[15] O. M. Elzeki, M. Z. Reshad, and M. A. Elsoud, "Improved Max-Min Algorithm in Cloud Computing", International Journal of Computer Applications, July 2012.

[16] Kalpana Parsi and M. Laharika, "A Comparative Study of Different Deployment Models in a Cloud", International Journal of Advanced Research in Computer Science and Software Engineering, May 2013.

[17] Bill Williams, "The Economics of Cloud Computing", Cisco Press, Indianapolis, Indiana, USA, June 2012.

[18] Dave Malcolm, "Defining Characteristics of Cloud Computing", Available: www.zdnet.com/article/the-five-defining-characterstics-of-cloud-computing, Last Accessed: March, 2015.

[19] Jeevana Jammula, Swapna Goud N., and Vishnu Murthy G., "Simple Overview on Cloud Computing", Department of Computer Science and Engineering, CVSR College of Engineering, Anurag Group of Institutions, India.

[20] Intel IT, "Including Cloud Adoption by Giving Developers the Key to Cloud Aware Development", Intel IT Center, August 2013.

[21] Intel IT, "Extending Intel's Enterprise Private Cloud with Platform as a Service", Intel IT Center, June 2012.

[22] Peter Mell and Timothy Grance, "The NIST Definition of Cloud Computing", National Department of Standards and Technology, USA, September 2011.

[23] Wikipedia, "User Virtualization", Available at: http://en.wikipedia.org/wiki/User_virtual ization, Last Accessed: December, 9 2015.

[24] Charles David Graziano, "A Performance Analysis of Xen and KVM Hypervisors for Hosting the Xen Worlds Project", Iowa State University, 2011.

[25] Amit Singh, "An Introduction to Virtualization", Available at: http://www.kernelthread. com/publications/virtualization, Last Accessed: December, 20 2015.

[26] Lakshay Malhotra, Devyani Agarwal, and Arunima Jaiswal, "Virtualization in Cloud Computing", Aset Amity University Noida, India, January, 6 2015.

[27] Carpathia, "Characteristics of Virtualization", Available at: http://carpathia.com/blog/virt ualization-what-is-it-what-types-there-are-and-how-it-benefits-companies/, Last Accessed: January, 6 2015.

[28] VMware, "VMware ThinApp Agentless Application", Available at: http://www.vmware.co m, Last Accessed: January, 12 2010.

[29] Wepodia, "Desktop Virtualization", Available at: http:// www.webopedia.com/ TERM/D/desktop_v irtualization.html, Last Accessed: Feburary,9 2015.

[30] Margaret Rouse, “BYOD policy”, Available: http://searchconsumerization.techtar get.com/definition/ BYOD-policy, Last Accessed: February, 92015.

[31] Frunk Bunn, Nik Simpson, Robert Peglar, and Gene Nagle, "Storage Virtualization", Storage Networking Industry Association, 2010.

[32] Rob Peglar ,"Storage Virtualization", Storage Networking Industry Association, 2011.

[33] Andrew S. Tannenbaum, "Modern Operating Systems", 4th International Edition, March 2014.

[34] Z. Li, Q. Huang and Z. Gui, "Enabling Technologies, In Spatial Cloud Computing: A Practical Approach", CRC Press, pp. 31-46., February 2013.

[35] Ling Qian, Zhiguo Luo, Yujian Du, and Leitao Guo, "Cloud Computing: An Overview", Beijing, China, 2007.

[36] Thunder Emperor, "Cloud Computing: Designing A Private Cloud"**,** Available at: http://3-4-5-6.blogspot.com/2011/06/cloud-computing-designing-private-cloud.html,Last Accessed: February,26 2015.

[37] Yu Kwong Kwok and Ishfaq Ahmad, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors", The University of Hong Kong, December 1999.

[38] Wubin Li, Johan Tordsson, and Erik Elmroth, "Modeling for Dynamic Cloud Scheduling via Migration of Virtual Machines", 3rd IEEE International Conference on Cloud Computing Technology and Science, 2011.

[39] Qingjia Huang, Kai Shuang, Peng Xu, Jian Li, Xu Liu, and Sen Su, "Prediction Based Dynamic Resource Scheduling for Virtualized Cloud Systems", Journal of Networks, Vol. 9, No. 2, February 2014.

[40] Wikipedia, "Hypervisor", Available at: http://en.wikipedia.org/wiki/Hypervisor, Last Accessed: February, 13 2015.

[41] Juniper, "Securing Multi-Tenancy and Cloud Computing", Juniper Networks Inc., March 2012.

[42] Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner, "Elasticity in Cloud Computing: What It Is, and What It Is Not", Institute for Program Structures and Data Organization, 2014.

[43] Asmita Pandey and Pooja, "Cloud Computing – An on Demand Service Platform", International Conference on Advances in Management and Technology, 2013.

[44] Ajith Abraham, Jaime Lloret Mauri, John Buford, Junichi Suzuki, and Sabu M. Thampi, "Advances in Computing and Communications", First International Conference Proceedings Part III, Kochi, India, 2011.

[45] Feng Bao and Jian Weng, "Information Security Practice and Experience", 7th International Conference, ISPEC, 2011.

[46] Eduard Heindl and Bhupesh Saurabh Sardana, "Cloud Computing", Hochschule Furtwangen University, January 2011.

[47] Xuanhua Shi, Bo Xie, Song Wu, Hai Jin, and Hongqing Zhu, "Pervasive Computing and the Networked World: A Dynamic and Reusable Scheduler for Cloud Infrastructure Service", Joint International Conference, ICPCA/SWS, Istanbul, Turkey, 2012.

[48] Gronau, "Cloud Deployment and Service Models", Available at: http://vpierre.it/cloud-101-cloud-deployment-and-service-models/, Last Accessed: February, 29 2015.

[49] Association of Modern Technologies Professionals, "Cloud Computing", Available at: http://www.itinfo.am/eng/cloud-computing/, Last Accessed: February, 29 2015.

[50] Flavio Lombardi and Roberto Di Pietro, "Secure Virtualization for Cloud Computing", Journal of Network and Computer Applications, June 2010.

[51] Gartner, "Factory Scheduling", Available at: http://www.gartner.com/it-glossary/factory-scheduling, Last Accessed: March, 3 2015.

[52] M. Krishnan, T. Karthikeyan, and T. R. Chinnusamy, "Performance Study of Flexible Manufacturing System Scheduling Using Dispatching Rules in Dynamic Environment", International Conference on Modeling Optimization and Computing, Vol. 38, pp. 2793–2798, 2012.

[53] Gerhard Fohler, "How Different are Online and Online Scheduling?", University of Kaiserslautern, Germany, 2011.

[54] Jorge Magalhães-Mendes, "Active, Parameterized Active, and Non-Delay Schedules for Project Scheduling with Multi-Modes", Portugal, 2007.

[55] Fei Teng, "Resource Allocation and Scheduling Models for Cloud Computing", Central Paris, 2011.

[56] Yogita Chawla and Mansi Bhonsle, "A Study on Scheduling Methods in Cloud Computing", International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), Vol. 1, Issue 3, October 2012.

[57] Bernd Bruegge and Allen H. Dutoit, "Object-Oriented Software Engineering", Carnegie Mellon University, School of Computer Science, Pittsburgh, USA, 1999.

[58] David Villegas and S. Masoud Sadjadi, "Mapping Non-Functional Requirements to Cloud Applications", Florida International University, Miami, Florida, 2011.

[59] P. Liu, "Cloud Computing", Publishing House of Electronics Industry, 2011.

[60] Beni, G. and Wang, J., "Swarm Intelligence in Cellular Robotic Systems", Proceeding in NATO Advanced Workshop on Robots and Biological Systems, Tuscany, Italy, June 26, 1989.

[61] Fogel, D. B., "Evolutionary Computation: Toward a New Philosophy of Machine Intelligence", 2nd ed.; IEEE Press: New York, NY, USA, 2000.

[62] Hai Zhong, Kun Tao1, and Xuejie Zhang, "An Approach to Optimized Resource Scheduling Algorithm for Open-source Cloud Systems", in 5th Annual China Grid Conference, 2010.

[63] Metta, Haritha, "Adaptive, Multi-Objective Job Shop Scheduling Using Genetic Algorithms", Unpublished Masters Thesis, University of Kentucky, 2008.

[64] G. Guo-Ning and H. Ting-Lei, "Genetic Simulated Annealing Algorithm for Task Scheduling based on Cloud Computing Environment", In Proceedings of the International Conference on Intelligent Computing and Integrated Systems, 2010.

[65] Q. Cao, W. Gong, and Z. Wei, "An Optimized Algorithm for Task Scheduling Based on Activity Based Costing in Cloud Computing", In Proceedings of the Third International Conference on Bioinformatics and Biomedical Engineering, 2009.

[66] Y. Yang, Kelvin, J. chen, X. Lin, D.Yuan, and H. Jin, "An Algorithm in Swin DeW-C for Scheduling Transaction Intensive Cost Constrained Cloud Workflow", In Proceedings of the Fourth IEEE International Conference on eScience, 2008.

[67] Pardeep Kumar and Amandeep Verma, "Independent Task Scheduling in Cloud Computing by Improved Genetic Algorithm", Panjab University, Chandigarh, India, 2012.

[68] Sung Ho Jang, Tae Young Kim, Jae Kwon Kim, and Jong Sik Lee, "The Study of Genetic Algorithm-Based Task Scheduling for Cloud Computing", School of Information Engineering, Inha University, 2012.

[69] A. Kaleeswaran, V. Ramasamy, and P. Vivekanandan, "Dynamic Scheduling of Data Using Genetic Algorithm in Cloud Computing", International Journal of Advances in Engineering & Technology, India, Jan 2013.

[70] Li L. and Liu F., "Group Search Optimization for Application in Structural Design", Springer, Berlin, Germany, 2011.

[71] Qinghai Bai, "Analysis of Particle Swarm Optimization Algorithm", College of Computer Science and Technology, Inner Mongolia University for Nationalities, Vol. 3, No. 1, Feb 2013.

[72] Kennedy, J. and Ebernhart, R., "Swarm Intelligence", Morgan Kaufmann Publishers: San Francisco, CA, USA, 2001.

[73] L. Zhang, Y.H. Chen, R.Y Sun, S. Jing, and B. Yang. " Genetic Simulated Annealing Algorithm for Task Scheduling based on Cloud Computing Environment", In Proceedings of the International Conference on Intelligent Computing and Integrated Systems, 2010. 2008.

[74] Suraj Pandey, LinlinWu, Siddeswara Mayura Guru, and Rajkumar Buyya, "A Particle Swarm Optimization Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments", Cloud Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, 2010.

[75] Ayed Salman, Imtiaz Ahmad, and Sabah Al-Madani, "Particle Swarm Optimization for Task Assignment Problem", Microprocessors and Microsystems, November 2002.

[76] Clerc M. and Kennedy J., "The Particle Swarm-Explosion Stability and Convergence in a Multidimensional Complex Space", IEEE Transactions on Evolutionary Computation, 2002.

[77] Chen Yonggang, Yang Fengjie, and Sun Jigui, "A New Particle Swam Optimization Algorithm", Journal of Jilin University, 2006.

[78] Xin Lu and Zilong Gu, "A Load Adapative Cloud Resource Scheduling Model Based on Ant Colony Algorithm", Proceedings of IEEE CCIS2011, 2011.

[79] Gao Y., Guan H., Qi Z., Hou Y., and Liu L., "A Multi-Objective Ant Colony System Algorithm for Virtual Machine Placement in Cloud Computing," Journal of Computer and System Sciences, Vol. 79, No. 8, pp. 1230-1242, 2013.

[80] Medhat A. Tawfeek, Ashraf El-Sisi, Arabi E. Keshk, and Fawzy A. Torkey, "Cloud Task Scheduling Based on Ant Colony Optimization", In Proceedings of the IEEE International Conference on Computer Engineering & Systems (ICCES), 2013.

[81] Pawar, C. S. and Wagh, R. B., "Priority Based Dynamic Resource Allocation in Cloud Computing", International Symposium on Cloud and Services Computing, IEEE, 2012.

[82] Hazem Ahmed and Janice Glasgow, "Swarm Intelligence: Concepts, Models and Applications", School of Computing, Queen's University, February 2012.

[83] Teodorović D. and Dell'Orco M., "Bee Colony Optimization - A Cooperative Learning Approach to Complex Transportation Problems", Proceedings of the 10th EWGT Meeting and 16th Mini-EURO Conference, Poznan, Poland, 2005.

[84] D. T. Pham, E. Kog, A. Ghanbarzadeh, S. Otri, S. Rahim, and M. Zaidi, "The Bees Algorithm – A Novel Tool for Complex Optimization Problems", 2nd International Virtual Conference on Intelligent Production Machines and Systems IPROMS, Oxford, Elsevier, 2006.

[85] H. A. Abbass, "A Single Queen Single Worker Honey-Bees Approach to 3-SAT", The Genetic and Evolutionary Computation Conference GECCO2001, San Francisco, USA, 2001.

[86] Chee and Brian J. S., "Cloud Computing: Technologies and Strategies of the Ubiquitous Data Center", CRC Press, 2010.

[87] Danieal Kunkle and Jiri Schindler, "A Load Balancing Framework for Clustered Storage Systems", High Performance Computing, 15th International Conference, Bangalore, India, December 17-20, 2008.

[88] Sudha Sadhasivam, R. Jayarani, N. Nagaveni, and R. Vasanth Ram, "Design and Implementation of an Efficient Two Level Scheduler for Cloud Computing Environment", In Proceedings of the International Conference on Advances in Recent Technologies in Communication and Computing, 2009.

[89] Jasmin James and Bhupendra Verma, "Efficient VM Load Balancing Algorithm for a Cloud Computing Environment", International Journal on Computer Science and Engineering, Vol. 4, No. 09, September 2012.

[90] A. Verma, P. Ahuja, and A. Neogi, "pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems", LNCS, pp. 243–264, Springer, 2008.

[91] Topcuoglu, H. Hariri S., and Wu M. Y., "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing", Parallel and Distributed Systems, IEEE, 2002.

[92] R. Alonso-Calvo, J. Crespo, M. Garcia-Remesal, A. Anguita, and V. Maojo, “On Distributing Load in Cloud Computing: A Real Application for Very-Large Image Datasets”, Procedia Computer Science, Elsevier, pp. 2669-2677, 2010.

[93] Saeed Parsa and Reza Entezari-Maleki, "RASA: A New Task Scheduling Algorithm in Grid Environment", Parallel Processing and Concurrent Systems Laboratory, Department of Computer Engineering, Iran University of Science and Technology (IUST), Tehran, Iran, 2009.

[94] G. W. You, S. W. Hwang, and N. Jain, “Scalable Load Balancing in Cluster Storage Systems”, LNCS Vol. 7049, Springer, pp. 101–122, 2011.

[95] M. Dakshayini and H. S. Guruprasad, “An Optimal Model for Priority Based Service Scheduling Policy for Cloud Computing Environment”, International Journal of Computer Applications, Vol. 32, No. 9, October 2011.

[96] Y. C. Lee and A. Y. Zomaya, "Rescheduling for Reliable Job Completion with the Support of Clouds", Future Generation Computer Systems, 2010.

[97] Zhao L., Ren Y., Xiang Y., and Sakurai K., “Fault-Tolerant Scheduling with Dynamic Number of Replicas in Heterogeneous Systems", In Proceedings of the IEEE International Conference on High Performance Computing and Communications, Washington, USA, pp. 434-441, 2010.

[98] Arash Ghorbannia Delavar, Mahdi Javanmard, Mehrdad Barzegar Shabestari, and Marjan Khosravi Talebi, “RSDC: Reliable Scheduling Distributed in Cloud Computing”, In International Journal of Computer Science, Engineering and Applications, Vol. 2, No. 3, June 2012.

[99] Shalmali Ambike, Dipti Bhansali, Jaee Kshirsagar, and Juhi Bansiwal, "An Optimistic Differentiated Job Scheduling System for Cloud Computing”, International Journal of Engineering Research and Applications, Vol. 2, Issue 2, Mar-Apr 2012.

[100] A. Ganapathi, Y. Chen, A. Fox, R. Katz, and D. Patterson, “Statistics-Driven Workload Modeling for the Cloud”, University of California, Berkeley, 2009.

[101] Oracle, “What is new in Java Development Kit 8.1”, Available at: http://www.oracle .com/technetwork/java/javase/8-whats-new-2157071.html, Last Accessed: February 2016.

[102] NetBeans Org.,“What is new in NetBeans IDE 8.1”, Available at: https://netbeans.org/community /releases/81/, Last Accessed: February 2016.

[103] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, C´esar A. F. De Rose, and Rajkumar Buyya, “CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms”, Wiley Online Library, August 2010.

Declaration

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other university, and that all source of materials used for the thesis have been duly acknowledged.

Declared by:

Name: _____Natnael Argaw_____

Signature: _____

Date: _____

Confirmed by advisor:

Name: _____

Signature: _____

Date: _____