DISTRIBUTION OF VIRTUAL MACHINES USING STATIC AND DYNAMIC LOAD BALANCING IN CLOUD COMPUTING

MISBAH LIAQAT

University of Malaya

FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2017

DISTRIBUTION OF VIRTUAL MACHINES USING STATIC AND DYNAMIC LOAD BALANCING IN CLOUD COMPUTING

MISBAH LIAQAT

THESIS SUBMITTED IN FULFILMENT
OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2017

University of Malaya

UNIVERSITI MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate:Misbah Liaqat

Registration/Matrix No.:WHA130023

Name of Degree:Doctorate of Philosphy

Title of Thesis: Distribution of Virtual Machines using Static and Dynamic Load Balancing in Cloud Computing

Field of Study: Computer Science (Cloud Computing)

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date

Subscribed and solemnly declared before,

Witness's Signature Date

Name:

Designation:

University of Malaya

ABSTRACT

Cloud computing, a user-centric computational model, is flexible paradigm of deploying and sharing distributed services and resources with the pay-per-use model. With virtual machine (VM) technology and data centers (DCs), computational resources, such as memory, central processing unit (CPU), and storage, are dynamically reassembled and partitioned to meet the specific requirements of end users. The demand's growth for cloud services is presenting considerable challenges for cloud providers to meet the requirements and satisfaction of end users. Virtualization technology reduces cloud operational cost by increasing cloud resource utilization level. In addition, the ever growing computational demands of users call for efficient cloud resource management to avoid SLA violation. Virtualization co-locates multiple virtual machines (VM) on a single physical server to share the underlying resources for efficient resource management. However, the decision about ”what” and ”where” to place workloads significantly impacts performance of hosted workloads. Load balancing between physical servers is important to avoid dangerous hot spots in the Cloud; in fact, overload situations are dangerous since they can easily lead to resource shortage and, at the same time, they can affect hardware lifetime, thus undermining data center reliability. Existing cloud schedulers consider a single resource (RAM) to co-locate workloads that as a result lead to SLA violation due to non-optimal VM placement. In addition, allocation of VMs based on traditional scheduler inefficiently balance the workload distribution that leads to extended the application execution time. Furthermore, exiting studies incorporates the migration technique in order to balance the load after the initial placement of workload, which leads to the maximum numbers of migrations. Therefore, to overcome these issues, this study propose the efficient load balancing solutions to uniformly distribute the workload among the physical servers. The initial VM placement method called Static Multi Resource based Sched-

University of Malaya

uler (SMRS), is designed to enhance the application execution time while balancing the CPU utilization without VM migrations. In addition, the Dynamic Multi Resource based Scheduler (DMRS) method is proposed to minimize the number of migrations after the initial placement of workload. We performed the real time experiments using the OpenStack cloud to highlight the efficiency of SMRS and DMRS solutions. Moreover, this study proposed the mathematical model for SMRS and DMRS method. To validate the correctness of the mathematical model, the empirical results and mathematical results are compared based on the CPU utilization, application execution time, and numbers of VM migrations as a performance metrics. The effectiveness of the proposed solution is evaluated by comparing their empirical results with well-known standard OpenStack nova scheduler. Experimentally, we have shown that our proposed method has lessened application execution time by 50% when compared with standard OpenStack cloud in static environment. In dynamic environment, the performance gain is reported up to 85% and 94.4% based on application execution time and CPU utilization. The improvement in application execution time increases the usability of cloud data centers.

University of Malaya

ABSTRAK

Cloud computing, model pengiraan user-centric, adalah paradigma fleksibel melaksana dan perkongsian perkhidmatan diedarkan dan sumber dengan model bayar-per-use. Dengan teknologi mesin maya (VM) dan pusat-pusat data (DC), sumber pengiraan, seperti memori, unit pemprosesan pusat (CPU) dan penyimpanan, secara dinamik dipasang semula dan dibahagikan untuk memenuhi keperluan khusus pengguna akhir. Pertumbuhan permintaan bagi perkhidmatan awan membentangkan cabaran yang besar untuk pembekal awan untuk memenuhi keperluan dan kepuasan pengguna. teknologi virtualisasi mengurangkan awan kos operasi dengan meningkatkan tahap penggunaan sumber awan. Di samping itu, permintaan pengiraan semakin meningkat dari pengguna menggesa pengurusan sumber awan berkesan untuk mengelakkan SLA pelanggaran. Virtualization bersama menempatkan pelbagai mesin maya (VM) pada pelayan fizikal tunggal untuk berkongsi sumber asas untuk pengurusan sumber yang cekap. Walau bagaimanapun, keputusan mengenai apa dan di mana untuk meletakkan beban kerja dengan ketara prestasi kesan beban kerja menjadi tuan rumah. Beban mengimbangi antara pelayan fizikal adalah penting untuk mengelakkan tempat-tempat berbahaya panas dalam Awan; sebenarnya, keadaan beban adalah berbahaya kerana mereka dengan mudah boleh membawa kepada kekurangan sumber dan, pada masa yang sama, mereka boleh mempengaruhi hidup perkakasan, dengan itu melemahkan kebolehpercayaan pusat data. penjadual awan sedia mempertimbangkan sumber tunggal (RAM) untuk bekerjasama mengesan beban kerja yang akibat membawa kepada SLA pelanggaran kerana tidak optimum penempatan VM. Walau bagaimanapun, peruntukan VMS berdasarkan penjadual tradisional tidak cekap mengimbangi pengagihan beban kerja yang membawa kepada memanjangkan masa pelaksanaan permohonan. Walau bagaimanapun, keluar kajian menggabungkan teknik penghijrahan bagi mengimbangi beban selepas penempatan awal beban kerja, yang mem-

University of Malaya

bawa kepada nombor maksimum migrasi. Oleh itu, untuk mengatasi isu-isu ini, kajian ini mencadangkan penyelesaian pengimbangan beban berkesan untuk seragam mengagihkan beban kerja di kalangan pelayan fizikal di penempatan awal VMS. Kaedah yang dicadangkan awal VM penempatan, yang dipanggil Content Multi Sumber Scheduler berdasarkan (SMRS), direka untuk meningkatkan masa pelaksanaan permohonan manakala mengimbangi penggunaan CPU tanpa migrasi VM. Di samping itu, kajian ini mencadangkan Scheduler Dynamic Multi Berasaskan sumber (DMRS), untuk mengurangkan jumlah migrasi selepas penempatan beban kerja. Kami melakukan eksperimen masa nyata menggunakan awan OpenStack untuk menyerlahkan kecekapan penyelesaian SMRS dan DMRS. Selain itu, kajian ini mencadangkan model matematik untuk SMRS dan kaedah DMRS. Untuk mengesahkan ketepatan model matematik, keputusan empirikal dan keputusan matematik dibandingkan berdasarkan penggunaan CPU, masa pelaksanaan aplikasi, dan bilangan migrasi VM sebagai metrik prestasi. Keberkesanan penyelesaian yang dicadangkan itu dinilai dengan membandingkan keputusan empirikal mereka dengan terkenal ditanda aras OpenStack nova penjadual. Uji kaji, kita telah menunjukkan bahawa kaedah yang dicadangkan kami telah berkurangan masa pelaksanaan permohonan sebanyak 50 % berbanding dengan penanda aras dalam persekitaran statik. Dalam persekitaran yang dinamik, keuntungan prestasi dilaporkan up-to 85 % dan 94.4 % berdasarkan masa pelaksanaan aplikasi dan penggunaan CPU. Peningkatan dalam masa pelaksanaan permohonan meningkatkan kebolehgunaan pusat data awan.

University of Malaya

ACKNOWLEDGEMENTS

First of all, I am thankful to Almighty Allah for endowing me the strength, wisdom, and endless blessing to study. It has been a great privilege to spend three years in the department of Computer Science and IT at University of Malaya, and its members will always remain dear to me. However, It would have not been possible to write this doctoral thesis without the help and support of the kind people around me, to only some of whom I can give particular thanks here. My hearty thanks must go to my advisors, Prof. Dr. Abdullah Gani, and Assoc. Prof. Dr. Siti Hafizah ab Hamid, who have patiently provided the vision, encouragement and advice necessary for me to proceed through this doctoral program and complete my thesis. Their continuous support and guidance helped me producing a valuable piece of research reported in this thesis. Thereafter, i am deeply indebted and grateful to Dr. Anjum Naveed, Dr. Rana Liaqat Ali, Dr. Raja Wasim Ahmed, and Muhammad Taqi Jawad for their extensive technical guidance and personal involvement in this research.

I would like to sincerely thank my dearest and loveliest parents for their faith in me and allowing me to be as ambitious as I wanted. My father always is a tremendous mentor for me. I would also like to gratefully express my special appreciation and thanks to my beloved husband, Rana Muhammad Haseeb-ur-Rehman, and my daughter Ansa for their great support, encouragement and unwavering and unconditional love. Words cannot express how grateful I am to my loveliest parents and dearest siblings (Sidra, Areeba, Rimsha, and Sami) and my daughter for all of the sacrifices that they have made on my behalf. I owe them everything and I hope that this work makes them proud. I dedicate this thesis to my beloved daughter Ansa and loveliest parents who scarifies a lot in this journey. Last but not the least, I am thankful to the BrightSpark Program for financially supporting me.

University of Malaya

TABLE OF CONTENTS

University of Malaya

University of Malaya

University of Malaya

University of Malaya

University of Malaya

LIST OF FIGURES

University of Malaya

University of Malaya

LIST OF TABLES

University of Malaya

University of Malaya

CHAPTER 1: INTRODUCTION

Cloud computing is evolving and increasing as an embryonic computing paradigm. By design, it is assembled with diverse computing technologies such as grid and utility computing, high performance computing, networking, virtualization, storage, distributed systems, and security (Buyya, Yeo, Venugopal, Broberg, & Brandic, 2009), (Stawski, 2015). In addition, virtualization offers the potential for on-the-fly and on demand configuration of physical machines to run various tasks and virtual machines (Espadas et al., 2013). Clouds adopt the virtualization concept for many reasons, such as (a) server consolidation, (b) applications' adaptive and dynamic configuration, (c) high availability, (d) and responsiveness. All virtualization features provide clouds a basis to meet service level (SLA) requirements. Moreover, virtualization is the crucial solution to reduce the energy consumption, cost of ownership and to attain better resource utilization in data centers (Barham et al., 2003). By providing physical resource sharing, live migration and fault isolation multiple virtual machines (VMs) can share resources on a single physical machine.

To ensure the network performance along with dynamic resource provisioning, virtualization tries to balance the load of the whole system dynamically (Espadas et al., 2013) there is always a chance of over utilization or under utilization of resources. Overloaded servers lead to performance degradation whereas under loaded servers cause poor utilization of resources. Due to inefficient distribution of load more heat will be generated by the overloaded servers which in turn increase the cost of cooling system and substantial emission of CO2 contributing to greenhouse effect (Shaw & Singh, 2014). Therefore, there is a need to provide right amount of resource dynamically to the applications running in VMs and develop an energy-efficient schemes.

Load balancing is commonly deployed function, which plays a vital role in cloud

University of Malaya

and virtualized data centers realization (Singh, Korupolu, & Mohapatra, 2008), (Chaczko, Mahadevan, Aslanzadeh, & Mcdermid, 2011). Load balancing is the evenly distribution of the resources among users or requests in a uniform manner so that no node become overloaded or idle in the cloud. In the absence of load balancing establishment, efficiency of overloaded nodes abruptly reduced with time and leads to SLA violation (Singh, Juneja, & Malhotra, 2015). In addition, load balancing is an essential aspect in Internet based all others distributed computing tasks (Foster, Zhao, Raicu, & Lu, 2008). Besides, Cloud Service Provider (CSPs) also provide the efficient load balancing solutions to the users in their own cloud computing platforms (Mondal, Dasgupta, & Dutta, 2012). Toward this goal numerous load balancing schemes such as Minimum Execution Time (MET) (Armstrong, Hensgen, & Kidd, 1998), Min-Min scheduling (Etminani & Naghibzadeh, 2007), Cloud Analyst (Sefraoui, Aissaoui, & Eleuldj, 2012) exists in literature and a comprehensive study is also done with and First Come First Serve (FCFS) and Round-robin. The main goal of job scheduling is to achieve a high performance computing and the best system throughput. Traditional job scheduling algorithms are not able to provide scheduling in the cloud environments.

The rest of the chapter is organized as follows. Section 1.1 presents the preliminary background related to the research field. Section 1.2 explain the research motivations to carry out proposed research presented in this thesis. The research aim and objectives are described in section 1.3. Further, in section 1.5 research methodology is proposed in order to address the research problem. The scope of the research is explained in section 1.6. Finally, the thesis layout is stated in section 1.7.

University of Malaya

1.1 Background

Cloud computing is a computing model that vigorously extends virtual resources including computing power, network resources, and storage, which helps users to approach resources on demand using pay-as-you-go service models through web services. Cloud computing comprises of a compilation of service models, functioning as Platform as a Service (PaaS) (Dash, Sahoo, Mohapatra, & Pati, 2012), Infrastructure as a Service (IaaS) (Erdogmus, 2009), and Software as a Service (SaaS) (Grossman, 2009), which are reachable over cloud layers (e.g client layer, application layer, infrastructure layer, server layer and platform layer). Though, Cloud service model results in reduction of delivery time and costs as well as enhances flexibility and efficiency. Cloud computing adopts the virtualization concept as an indispensable feature. Virtualization enables physical resources to utilize IT infrastructure on cloud computing platforms as a virtual resource. Furthermore, virtualization is dynamic in nature, whereby cloud computing services are automatically provisioned as and when needed by users (Jararweh et al., 2014).

During the last one decade, due to extensively increasing demand for high-end computational servers, efficient cloud resource management has become a must to meet requirements for cloud providers (Buyya et al., 2009) (Jararweh et al., 2014). Virtualization configures and runs numerous workloads on a single physical server to attain high resource utilization for effective cloud resource management (Barham et al., 2003) (Espadas et al., 2013). However, aggressive workload co-location leads to resource over utilization that significantly impacts application performance in terms of SLA violation. Therefore, the decision about what and where to place workloads is very important as efficient workload distribution surges in application performance due to diminishing hotspots with data centers (DC). Alternatively, cloud resource under utilization significantly impacts Return On Investment (ROI) for cloud operators. Load balancing within cloud data centers fairly

University of Malaya

distributes a workload onto a set of physical servers to, (i) increase ROI, (ii) minimize the number of hotspots, (iii) reduce SLA violation, and (iv) minimize cloud operational cost.

Load balancing guarantees that all physical resources within cloud DC have a uniform workload. The VM placement schemes are divided into static and dynamic load balancing based on the migration criteria. In static load balancing VMs are not migrated to other servers whereas in dynamic load balancing VMs are migrated other hosts in order to balance the system load. Numerous existing load balancing schemes such as, Round Robin (Sidhu & Kinger, 2013), Min-Min scheduling (H. Chen, Wang, Helian, & Akanmu, 2013) (S.-C. Wang, Yan, Liao, & Wang, 2010) (Etminani & Naghibzadeh, 2007), Max-Min Algorithm (Elzeki, Reshad, & Elsoud, 2012), OpenStack Scheduler (Litvinski & Gherbi, 2013), Min-min Algorithm (Patel, Mehta, & Bhoi, 2015), and Improved Max-min (Elzeki et al., 2012), have considered static load balancing (single-resource) to co-locate VMs. All aforementioned schemes opted VM placement scheduler that overlooks queuing user requests, and a fair-share algorithm enabled resources provisioning within data centers.

In VM migration process, number of VMs are migrated from one host to another in order to balance the load of over-provisioned or under-provisioned hosts. Migration technique keep the same performance of the VM as it is expected. Therefore, based on migration impact, numerous studies have been presented dynamic load balancing strategies based on VM migration for the sake of efficient load balancing because cloud load balancing is an extremely important topic due to economic issues (Corradi, Fanelli, & Foschini, 2014)and (Liaqat, Ninoriya, Shuja, Ahmad, & Gani, 2016), (Calcavecchia, Biran, Hadad, & Moatti, 2012), (Wuhib, Stadler, & Lindgren, 2012) , (Wuhib et al., 2012), (Beloglazov & Buyya, 2012), (Beloglazov, Abawajy, & Buyya, 2012), and (Bobroff, Kochut, & Beaty, 2007).

A dynamic resource allocation architecture is presented for the sake of load balanc-

University of Malaya

ing and minimizing the energy consumption among the different services of cloud (Wuhib et al., 2012). Based on management objectives, the architecture focus on the three components in terms of (i) admission control policy of VMs, (ii) Placement controller for VMs, and finally (iii) the implemented is done as an extension of OpsnStack cloud (Corradi et al., 2014). The shortfall of this architecture is that it only balance the load based on migration concept and after the allocation of VMs. For the migration of VM, black box and gray box algorithms are proposed in (Wood, Shenoy, Venkataramani, & Yousif, 2007). The black box algorithm decides when to migrate, while the decision based on what and where to migrate is determined from gray box algorithm. To move the VM from overloaded to least overloaded host (Wood et al., 2007) uses a greedy heuristic. The VectorDot scheme is proposed in (Singh et al., 2008), which considers the load on the communication paths that connect servers to shared network storage. Moreover, Entropy approach reduce migration time as well as the number of nodes acquiring low performance overhead.

University of Malaya

Based on cloud load balancing schemes it is observed that serval studies have been conducted based on the static load balancing. Static load balancing do not efficiently balance the workload. The shortcoming of static load balancing is that it only consider the number of CPUs and memory while the placement of VM without considering the effect of load on physical host. Therefore, dynamic load balancing is used to address the issues of static load balancing. The shortfall of this method is that it only balance the load based on migration concept and after the allocation of VMs. However, in order to balance the load based on migration techniques number of migration should be minimized and controlled because it effects the performance of other VMs running on source and destination hosts. In aforementioned studies, load is balanced after the deployment of VMs using the migration concept. Therefore, there is a need to balance the load at the deployment time of the VM.

1.2 Motivation

According to the forester research the business of cloud computing will grow upto $241 billion in 2020 whereas it is represented as $40.7 billion in 2010 (Truong, 2010). Besides, several open-source IaaS-cloud management platform have been developed in order to facilitate the creation of private clouds such as OpenNebula, Nimbus, OpenStack (Sefraoui et al., 2012), and Eucalyptus (Nurmi et al., 2009). These IaaS platforms have huge impact the adoption of cloud computing technology. OpenStack cloud is one of the most widely used open-source cloud among all and revenue will grow to $3.3 billion in 2018 (Fig. 1.1) whereas it will be $5.3 billion in 2020 as presented in Fig. 1.2. So, in order to undertake our research, we are using OpenStack cloud. According to crisp research 190,000 individuals and 114 showing the worldwide support to the OpenStack.

University of Malaya

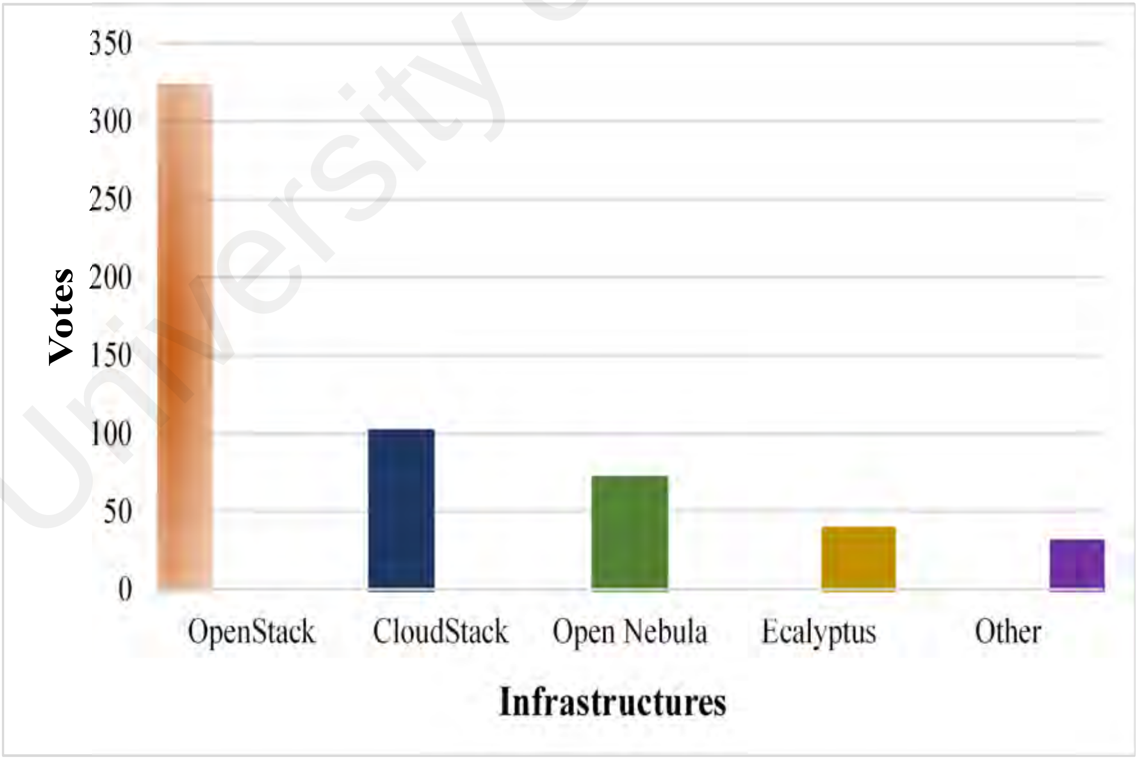


Figure 1.1: Top Cloud Infrastructures as a Service Projects

Latest research shows that cloud data centers consumes 70 billion kilowatt-hours of energy, examined in recent year, which is the 2% of the total energy consumption of the country (NRDC, 2014). Furthermore, the cloud infrastructures will consume more energy up-to 1,963.74 TWh in 2020 (Greenpeace, 2010). In last five years, the growth rate of DCs is doubled. According to the (gignet, 2011) 52% of the cloud resources remains underutilized. DC resources in underutilized or idle state consume 70% to 80% of the energy, which is consumed during the peak utilization (Heller et al., 2010). Therefore; to handle the issues related to underutilization and overutilization of the resources an efficient workload distribution is required to fairly distributes resources in cloud.

Cloud load balancing efficiently distributes the cloud resources. Several studies presents that cloud load balancing address the resource utilization issues with the allocation of the VMs to the physical hosts based on static and dynamic load balancing. Majority of the study discuss the static load balancing including (Samal & Mishra, 2013), (Gautam & Bansal, 2014), (Kaur & Kaur, 2015), and (Domanal & Reddy, 2013). In contrast, number of dynamic load balancing schemes have been presented to address the inefficient workload distribution issues (Ghribi, Hadji, & Zeghlache, 2013), and (Guo et al., 2010). The static load balancing schemes highlights that while the initial deployment of workload CPU utilization behavior is not studied. The dynamic load balancing mitigates the shortfall of static load balancing and balanced the load after the migration of VMs. The limitation of the dynamic load balancing is that it only consider the workload using the migration technique and maximum number of migrations minimize the cloud performance in terms of energy consumption. Therefore; an efficient load balancing solution is required address the limitations of static load balancing as well as the shortfalls of dynamic load balancing in order to enhance the cloud resource utilization.

University of Malaya

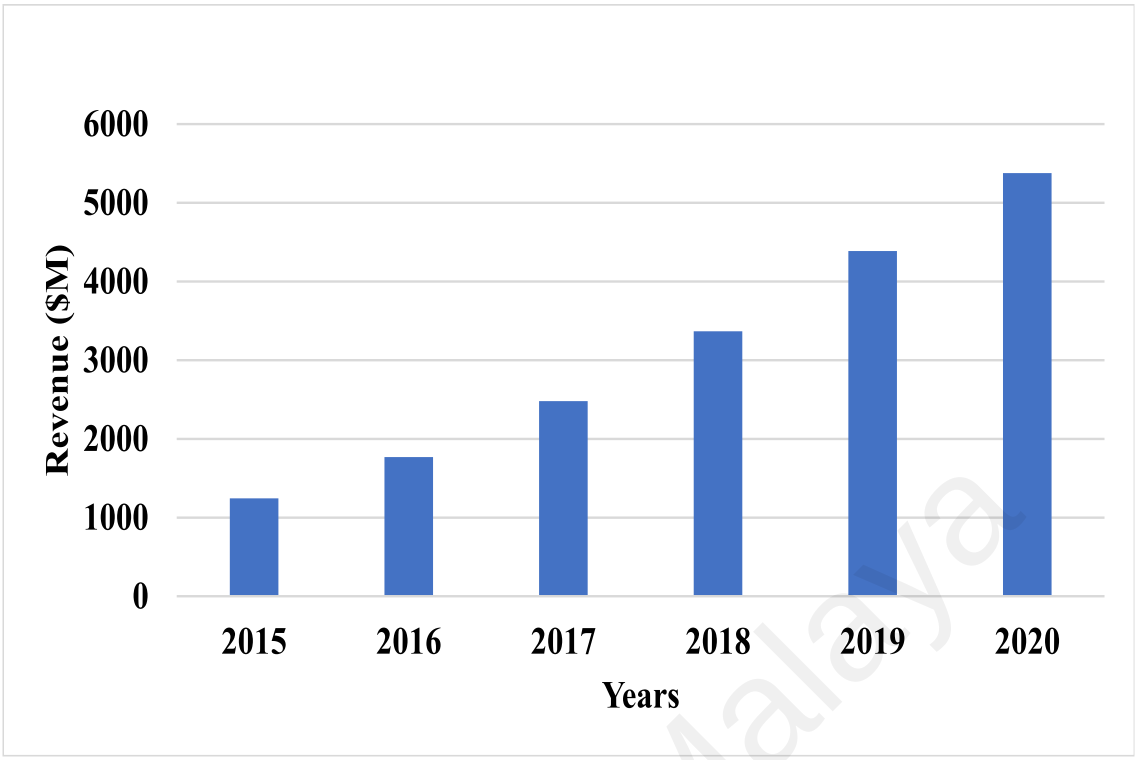


Figure 1.2: Worldwide Support to OpenStack Cloud

1.3 Statement of Problem

Load balancing is considered as a vital feature in Internet based distributing tasks as well as in cloud computing. Regarding the static load balancing, researchers (Mills, Filliben, & Dabrowski, 2011), (Mills et al., 2011), (Kousiouris, Cucinotta, & Varvarigou, 2011), considered VM placement at the time of creation and investigate different factors such as, deployment scenarios, real-time scheduler's decisions, and types of workload, to identify the parameters influencing VMs co-location. On the other hand, researchers (Q. Wang & Varela, 2011) considered the relationship among three categories of workload based on several virtual network configuration strategies in terms of the number of VMs, vCPUs usage per VM, and memory size for each VM. However, these techniques overlooked the impact of CPU utilization while the initial placement and only distribute the workload based on the number of CPU used and the available amount of RAM.

Numerous studies have been conducted based on the dynamic VM placement to provide the efficient solutions in order to assign the clients requests to available cloud

University of Malaya

nodes.Various aspects of cloud load balancing schedulers are extensively presented in (Kllapi, Sitaridi, Tsangaris, & Ioannidis, 2011), (Phan, Zhang, Zheng, Loo, & Lee, 2011), and (Sotiriadis, Bessis, Xhafa, & Antonopoulos, 2012). For the apprehension of resource reservation based on VMs, VM migration is the fundamental scheme. Authors in (Zhao & Figueiredo, 2007) projected migration cost in order to find the accurate estimation of migration time for preparing resource reservation.

Based on literature review it is perceived that in static load balancing cloud schedulers do not consider the CPU utilization at the deployment of VMs. Besides, most of the current state-of-the-art load balancing schemes considered dynamic load balancing to handle the static load balancing issues. In aforementioned dynamic load balancing studies, load is balanced after the deployment of VMs using the migration concept. Maximin number of migrations degrade the cloud performance. Therefore, there is a need to balance the load at the deployment time of the VM. Based on the aforementioned research gap the problem statement of this thesis is stated as.

Majority of the cloud load balancing algorithms incorporate the static parameters while the initial placement of resources. The static load balancing does not consider the CPU utilization on physical hosts. Moreover, in order to address the limitations of static load balancing the dynamic load balancing handle the over-utilization and under-utilization of the hosts by adapting the migration solution when the resources are initially deployed. However, in static load balancing the absence of dynamic state of CPU load, during the scheduler's decision making leads to inappropriate workload distribution on physical hosts and when the workload is balanced through the migration it leads to the high migration overhead. As a result, the inappropriate workload distribution surges application execution time. Therefore, there is a need of a solution that should efficiently manage the CPU utilization (CPU load) for initial placement to minimize number of VM migrations.

University of Malaya

1.4 Statement of Objectives

In this section the problem of inefficient distribution of workload while the allocation and reallocation of VM is addressed. The aim of the research is to minimize the inefficient distribution of load during the placement of VM and to enhance the execution time of VMs. The objective of the research are as follows.

- To critically review the current state-of-the-art cloud load balancing schemes while the placement of VMs to gain insight to the performance limitations.
- To investigate the workload distribution of cloud load balancing schedulers to reveal inefficiencies in existing schemes without considering the CPU utilization and current load while the placement of VMs.
- To design and propose a multi resource-based scheduler to minimize the deficiencies of current cloud schedulers based on CPU utilization, and application execution time while the placement and migration of VMs.
- To evaluate the performance of proposed multi resource based algorithms and compare it with the state-of-the-art current VM placement cloud scheduler, and to validate the developed mathematical model.

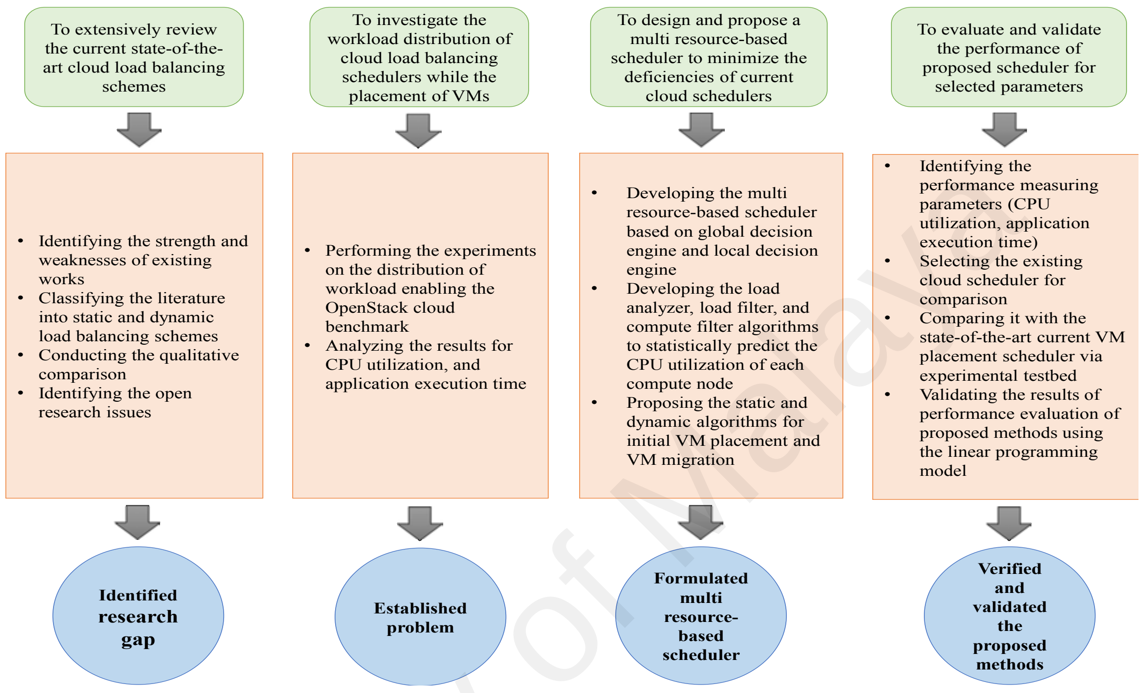
1.5 Proposed Methodology

The proposed methodology is highlighted in Fig. 1.3, which is followed to conduct this research. The research is divided into four objectives such as literature review, problem analysis, design of static and dynamic cloud load balancing scheduler, evaluation of the proposed scheduler, and validation of the proposed solution.

In the first phase, existing literature work is extensively reviewed in order to highlight the strength and weaknesses of state-of-the-art cloud load balancing schemes. The load balancing schemes are categorized based on the proposed thematic taxonomies. Based on

University of Malaya

the proposed taxonomies the existing schemes are compared with respect to the objective functions to highlight the commonalities and differences among them. Moreover, the issues that affects the performance of existing schemes are also discussed.



To extensively review the current state-of-the-art cloud load balancing schemes
To investigate the workload distribution of cloud load balancing schedulers while the placement of VMs
To design and propose a multi resource-based scheduler to minimize the deficiencies of current cloud schedulers
To evaluate and validate the performance of proposed scheduler for selected parameters
• Identifying the strength and weaknesses of existing works
• Classifying the literature into static and dynamic load balancing schemes
• Conducting the qualitative comparison
• Identifying the open research issues
• Performing the experiments on the distribution of workload enabling the OpenStack cloud benchmark
• Analyzing the results for CPU utilization, and application execution time
• Developing the multi resource-based scheduler based on global decision engine and local decision engine
• Developing the load analyzer, load filter, and compute filter algorithms to statistically predict the CPU utilization of each compute node
• Proposing the static and dynamic algorithms for initial VM placement and VM migration
• Identifying the performance measuring parameters (CPU utilization, application execution time)
• Selecting the existing cloud scheduler for comparison
• Comparing it with the state-of-the-art current VM placement scheduler via experimental testbed
• Validating the results of performance evaluation of proposed methods using the linear programming model
Identified research gap
Established problem
Formulated multi resource-based scheduler
Verified and validated the proposed methods

Figure 1.3: Research Methodology

The second phase is described as the problem establishment phase. In this phase, research problem is investigated by analyzing the performance of current cloud scheduler while the placement of VMs. The benchmark OpenStack filter scheduler along with the set of filters including Ram filter, Availability zone filter, Core filter, All-Hosts Filter filter is used to investigate the CPU utilization and application execution time when the VM are deployed using the 100% CPU resources. The performance parameters are examined based on the static and random load distribution factors while the placement of VMs to PMs. Moreover, in order to fully utilize the CPU resources a CPU intensive algorithm is designed. Besides, the analysis is exercised to reveal the performance of current cloud scheduler based on CPU utilization and application execution time.

University of Malaya

In the third phase of the research multi resource-based scheduler is proposed to minimize the deficiencies of the current cloud scheduler. The basic objective of the proposed scheduler is to minimize the application execution time and efficiently utilize the CPU resources. The proposed multi resource-based scheduler is comprised with two modules named as global decision engine and local decision engines. The local decision engine is consists of two algorithms including load analyzer and compute load. These algorithms are implemented at compute nodes where another algorithm is designed for global engine and represented as load filter. To reduce the application execution time, the proposed scheduler is designed for the static (initial VM placement) and dynamic (and VM migration) scenarios. Based on the static algorithm the proposed scheduler balance the CPU utilization and minimize the application execution time. In contrast, for dynamic algorithm the CPU utilization is balanced at the deployment time of VM and the CPU capacity is increased and affects the application execution time at time interval t the VM is migrated to the least loaded host to efficiently utilize the CPU resources.

In the fourth phase, the significance of the proposed scheduler is evaluated while conducting the real experiments using the benchmark application. The behavior of the proposed multi resource-based scheduler is verified based on the benchmark application. The performance of the proposed static scheduler is analyzed based on its CPU utilization, application execution time, and the sequence if the VM deployments when the resources are homogeneous and heterogeneous based on the RAM availability and number of vCPUs associated to each VM. Further, the behavior of dynamic resource based scheduler is evaluated based on the number of migrations, CPU utilization, and application execution time. In addition, the empirical date is conducted while varying the number of vCPUs and workload associated to each VM. In this last phase the findings of the proposed scheduler algorithms are compared with existing solution to validate the efficiency of proposed

University of Malaya

scheduler. Moreover, the scheduler is validated using the statistical based analysis by conducting the average, mean, and standard deviations results. In contrast, a mathematical model based on the linear programming of static and dynamic algorithms is proposed in order to efficiently balance the CPU utilization among physical servers. Furthermore, the effectiveness of mathematical model is validated by experimentations.

1.6 Scope

Virtualization technology assists to achieve computing-as-a-service vision of cloud-based solutions. VM process helps to achieve various management goals, such as, load balancing, fault tolerance, and green cloud computing. It transfers system state from one server to another to offer uninterrupted services. However, VM migration is not free and consumes a significant amount of sender and receiver resources in terms of power to carry out migration process successfully. This study does not consider the power consumption constraints when the VMs are allocated on the physical hosts. In addition, this research consider the effect of optimal combination of multi resources such as number of vCPUs, RAM and CPU load through VM provisioning for the maximum resource utilization. This study does not reflecting the effects of I/O resources and network communication pattern. Therefore, the scope of this research is limited to the VM efficient distributions of VMs while satisfying the load balancing constraints.

University of Malaya

1.7 Thesis Structure

The thesis entitled as “Distribution of Virtual Machines using Static and Dynamic Load Balancing in Cloud Computing” comprises a detailed study of this research. Therefore, for the better understanding to readers this research is structured based on seven chapters. The thesis outline is presented in Fig. 1.4.

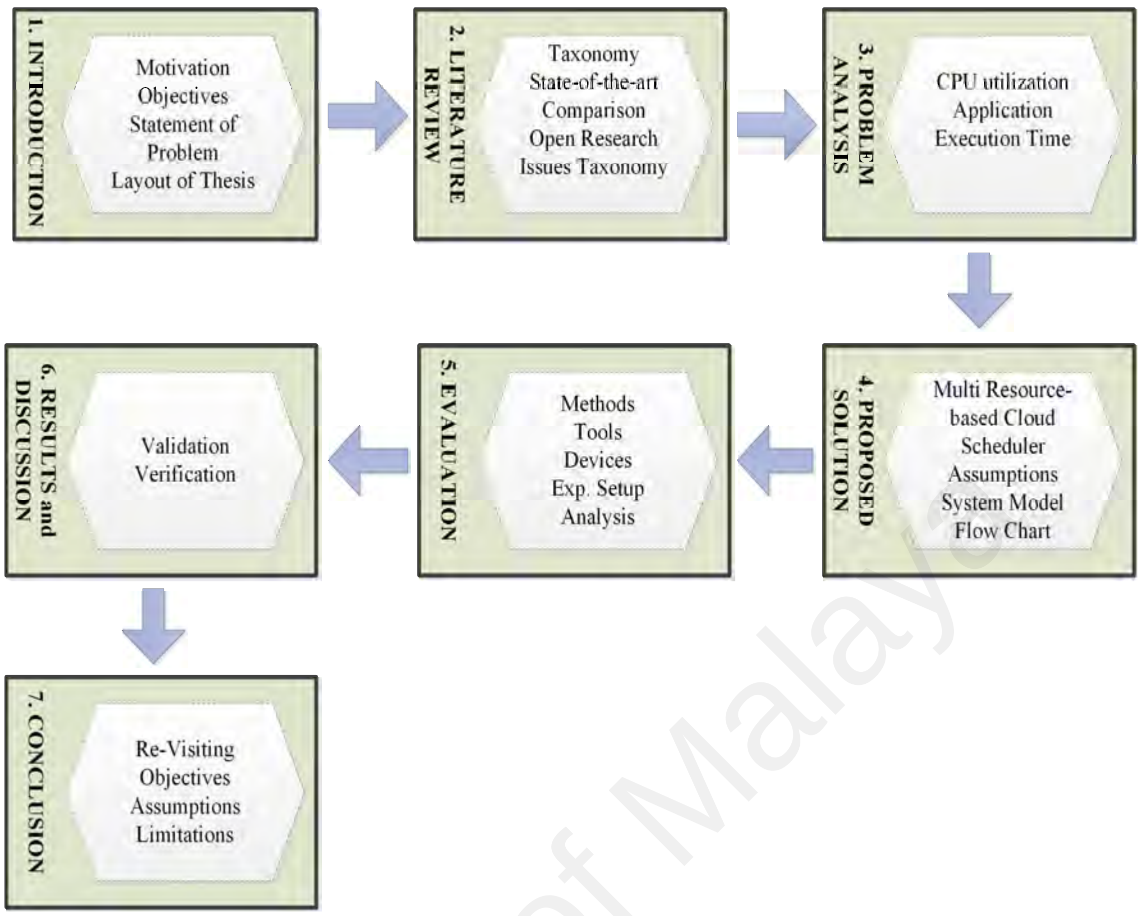


Figure 1.4: Thesis Organization

Chapter 2 presents the state-of-the-art cloud load balancing schemes. This chapter presents the detailed discussion and review on the current load balancing techniques and classify these techniques into static and dynamic load balancing categories based on the characteristics presents in the thematic taxonomy. The strength and weaknesses of each scheme is highlighted based on the objective functions stated in the taxonomy. Furthermore, a critically qualitative analysis is presented for the static and dynamic load balancing schemes. This chapter compose the details taxonomy based on the issues and challenges regarding to the cloud resource management. This chapter highlights the several open research issues as well as the discuss issues that are addressed in this research work.

Chapter 3 analyze the issues of legacy load balancing method. The OpenStack cloud is used as a benchmark for the distribution of the resources in the cloud. The performance

University of Malaya

overhead in terms of CPU utilization and application execution time is analyzed to gather insight the computational limitations while varying the number of resources including RAM, vCPUs, and disk space associated to VMs. The analysis presents that the existing study report the inefficient distribution of resources while the deployment of VMs. Further, the performance analysis study reveal the need for an efficient distribution of the resources within cloud.

Chapter 4 presents the proposed multi resource-based VM deployment solution to address the issues of the legacy VM deployment methods. In this chapter the proposed solutions static load balancing solution is described in detailed based on three algorithms including load filter, compute load, and load analyzer. Based on the dynamic load balancing the dynamic multi resource based scheduler is proposed. The working of the dynamic load balancing method is presented with the system flow diagrams. Further, the assumptions and constraints in terms of resource constraints, operational constraints, and load balancing constraints are described in this study as a equation for the static and dynamic load balancing solutions.

Chapter 5 shows the data collection methodology for the proposed solutions. The study describes the detailed experimental setup accompanying with the benchmark and devices. Further, the data collection method is also explained for empirical results for static and dynamic solution. The validation model and its equation are implemented using A Mathematical Programming Language (AMPL) tool by using the gurobi solver. Besides, the tools, and hardware level specifications are described to in order to perform the experiments.

Chapter 6 validates the proposed solutions by comparing the results of empirical evaluations with the results of mathematical model. This study presents the effectiveness of proposed solution based on (i) CPU utilization, (ii) number of migrations, and (iii) application execution time. In this study, the performance of proposed multi resource based

University of Malaya

schedulers is also compared with the legacy benchmark application.

Chapter 7 concludes this thesis by revising the research objectives. This, study summarizes the contributions of this thesis and highlights significance and limitations. Moreover, the future research directions are also provided in this chapter.

University of Malaya

CHAPTER 2: LOAD BALANCING SCHEMES FOR CLOUD DATA CENTERS

This chapter briefly reviews the importance of cloud load balancing, and cloud resource management. In this chapter, critical aspects and significant features of existing frameworks are investigated through qualitative analysis. Moreover, this chapter critically review the state-of-the-art of cloud load balancing schemes and present the a detailed thematic taxonomy, which covers the load balancing characteristics and classifies the existing literature. In order to compare the performance of existing load balancing schemas critical parameters are considered from the literature. The comparison parameters are: (a) reliability, (b) availability, (c) energy efficiency, (d) delay time optimization, (e) scalability, and (f) latency. In addition, this survey provides a detailed taxonomy and discussion of various open research issues and challenges based on selected functions in terms of (a) resource selection, (b) resource allocation, (c) resource monitoring, (d) resource discovery, (e) resource prizing, and (f) disaster management to pave the way for future research directions.

This chapter is comprised of eight main sections. The essential background and basic terminologies are presented in section 2.1 in order to describe fundamental concepts of cloud computing and virtualization. Section 2.2 discusses the importance of cloud load balancing and presents a thematic taxonomy for load balancing characteristics. This section compares and reviews existing state-of-the-art load balancing schemes based on static and dynamic load balancing schemes to address the commonalities and differences among them. Section 2.3 presents the necessary background of cloud resource management and a review of substantial resource management techniques covering resource management functions in terms of resource selection and allocation. Furthermore, the main features of cloud deployment tools are explained in section 2.4. Section 2.5 presents a detailed the-

University of Malaya

matic taxonomy of open research issues and challenges that hinder designing optimized resource management technologies for cloud environment. Section 2.6 discusses a brief summary on load balancing schemes based on initial VM provisioning and migration to analyze the most significant issues in cloud load balancing research domain. Finally, an overview on findings and the complete chapter conclusion is presented in section 2.7.

2.1 Background

This section discuss the concept of the cloud computing and virtualization.

2.1.1 Cloud Computing

Information Technology (IT) industry has evolved from its birth in the last century into one of the most prominent industry in today's world. Along with its rapid growth, IT is changing daily lifestyles and is becoming a technology enabler for many veteran industries and businesses (Liaqat et al., 2016). Cloud computing technologies have emerged as a backbone of all IT services. Cloud computing is assembled with diverse computing technologies such as grid and utility computing, high performance computing, networking, virtualization, storage, distributed systems, automation and security, etc (Buyya et al., 2009). With the assistance of cloud computing definition, elasticity is defined as the creation of numbers of virtual machines instances depending on user's demand. Therefore, in order to fulfill the user demands cloud must provide the high performance gain and at the same time must be beneficial for the CSP (Shaw & Singh, 2014).

Cloud computing provides a holistic storage solution for data storage in a remote location using a third-party server. For instance, Google File System (GFS), BigTable, Amazon's Simple Storage Service (S3), Simple DB, Hadoop Distributed File System (HDFS), and OpenStack, are available cloud storage applications that are accessed by numerous types of clients (Xu, Zheng, Wu, Huang, & Xu, 2010). With the assistance of

University of Malaya

cloud computing definition, elasticity is defined as the creation of numbers of virtual sensor instances depending on user's demand. Therefore, in order to fulfill the user demands cloud must provide the high performance gain and at the same time must be beneficial for the Cloud Service Provider (CSP) (Shaw & Singh, 2014). Major characteristics of cloud computing which are important in data analysis and processing, can be defined as follows:

- On-demand self-service: each cloud user can deploy and configure the cloud servers and services himself, no interactions with service providers are necessary
- Multitenancy: the resources and costs are shared across a large community of cloud users
- Scalability: an efficient and low-cost configuration and assignment of the system resources according to consumer demand
- Easy system access: cloud resources and services are accessed through standard (Internet) protocols, using standard web browsers regardless of their location and platform, e.g., smart phones.

2.1.1.1 Cloud services

Cloud computing consists of a collection of service models (Mell & Grance, 2009), such as PaaS (Dash et al., 2012), IaaS (Erdogmus, 2009), and SaaS (Grossman, 2009), which are available over cloud layers (e.g client layer, infrastructure layer, application layer, platform layer, and server layer) as presented in Fig. 2.1. At the infrastructure level, CC provides a service to an end user by provisioning the servers, networks, storage and fundamental computing resources, and the user can deploy and run the software that includes the applications and operating systems. Cloud computing offers a number of advantages by allowing users to utilize platforms (e.g., middleware services and operating systems)

University of Malaya

for the deployment of user-created applications using the cloud providers' supported languages, libraries and tools (e.g., Amazon, Google, and Salesforce) at low cost. At the platform level, users have control of the deployed applications and configuration settings of the environment hosted by the applications. In addition, cloud computing facilitates the elastic utilization of resources in on-demand manner. However, the cloud service model reduces delivery time and costs as well as improves flexibility and efficiency.

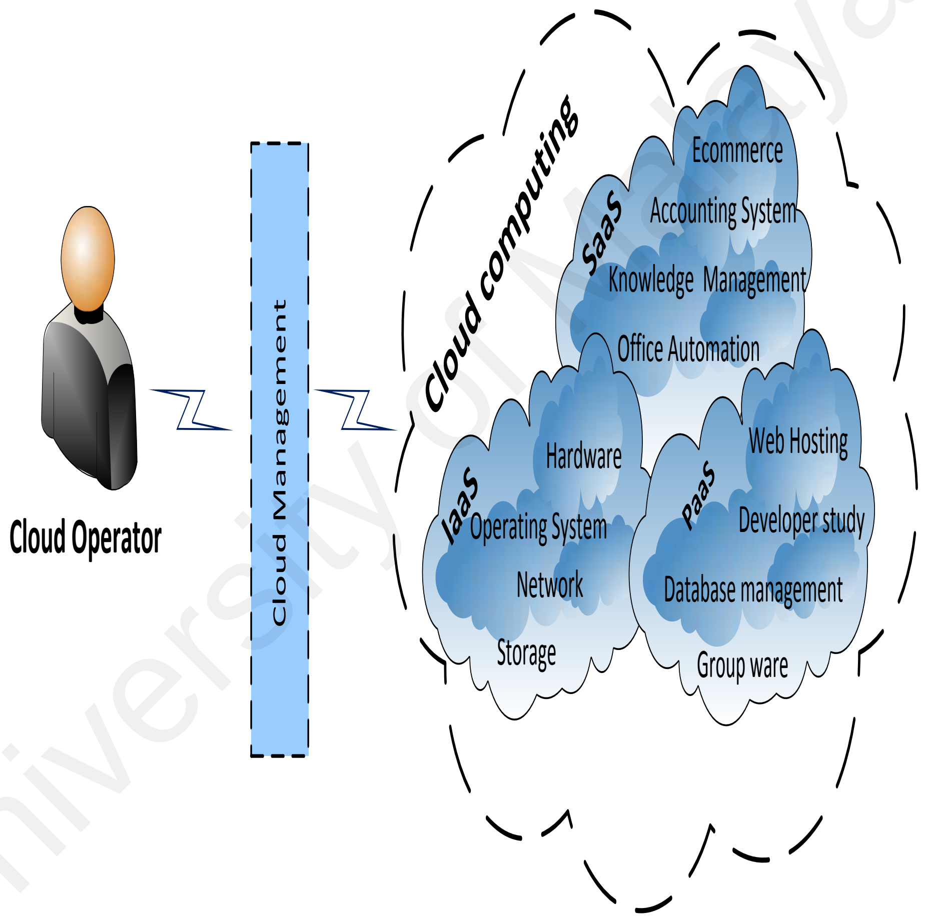


Figure 2.1: Cloud computing service model

2.1.1.2 Cloud Types

Cloud computing have four types of deployment models such as community, public, private, and hybrid cloud (Dash et al., 2012), (Erdogmus, 2009). In a public cloud, resources are dynamically provided to the general public on fine-grained and general third party pay

University of Malaya

the bill based on computing. Moreover, the popular services are public clouds, including EC2, Amazon, Google App-Engine, S3 and Force.com. The publicly availability of these services often called public cloud. In contrast, the infrastructure used for the particular organization is known as a private cloud (Tolba & Ghoneim, 2015). It is hosted internally or externally by the third party where the cost is divided on the several users not on the general users. For example, EUCALYPTUS is a software environment which is used for the deployment of the private cloud and has compatibility related concern with Amazon's EC2. Furthermore, it represents the extensible and modularized policy for the allocation of resources. Currently, EUCALYPTUS supports the two simple types of polices such as, round robin and greedy (Sotomayor, Montero, Llorente, & Foster, 2009). Hybrid cloud is made up from two or more clouds like public, community or private cloud, which provides the advantages of numerous deployment models. Private cloud is made for the single organization. It is hosted internally and externally or managed by the third party.

2.1.2 Virtualization

Virtualization was devised as a resource management and optimization technique for mainframes having scaleless computing capabilities. Virtualization in mainframes results in efficient management of coarse-grained resources with limited overhead (Liaqat et al., 2016). However, virtualization techniques have been able to make their way to multi-core (Petrides, Nicolaides, & Trancoso, 2012) and commodity server designs (Egi et al., 2010). The multi-core processor, blade server, and System on Chip (SoC) designs also provide virtualization techniques opportunity for resource consolidation and optimization where fine-grained resources are assembled to provide a virtual scalable platform for cloud applications. Virtualization techniques benefits data centers in many ways, such as:

University of Malaya

- Scalability: cloud users and applications view heterogeneous hardware and software resources as a single scalable platform. Virtual devices are scalable in terms of hardware resources. Server, memory, and I/O power can be added to a virtual machine when its resource utilization nears 100%.

- Consolidation and utilization: virtual resources can be easily consolidated over few physical resources that results in higher resource utilization levels and energy efficiency (Younge et al., 2011).

- Isolation: performance and faults are isolated between applications of the same resource (Uhlig et al., 2005).

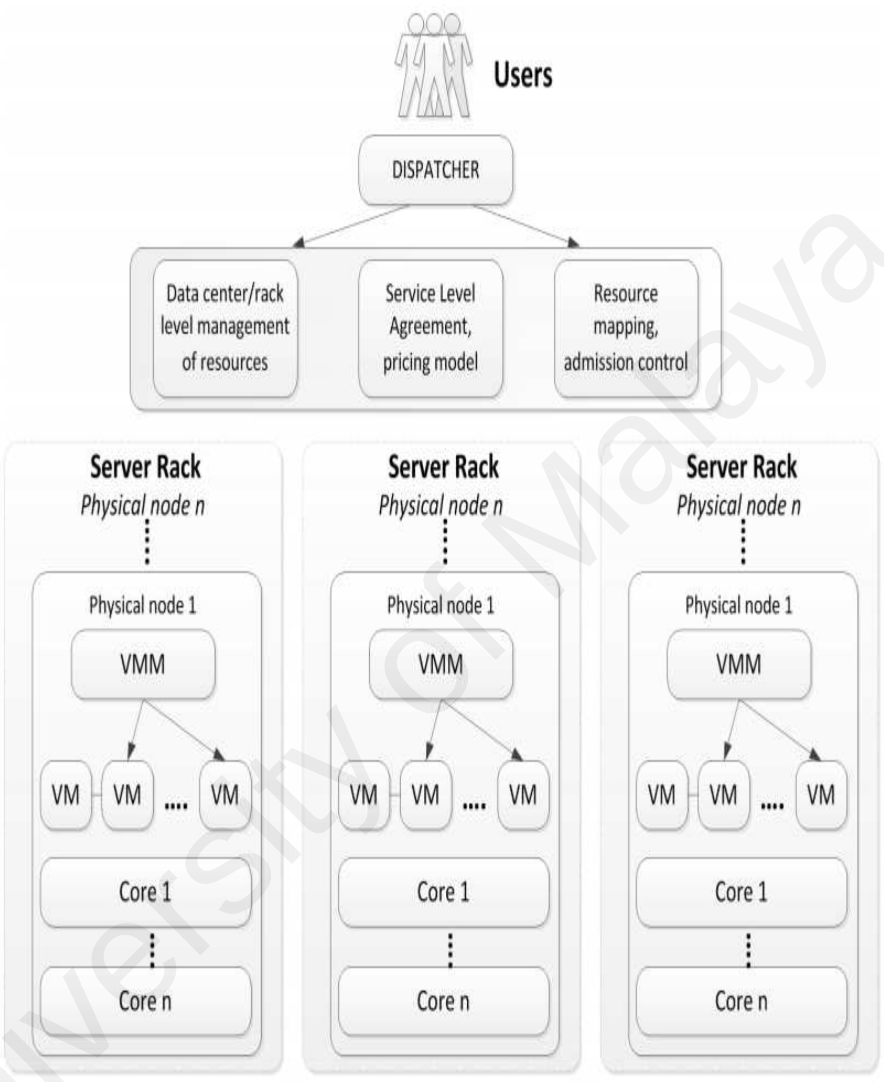
- Manageability: virtualization offers variety of resource management options such as VM creation, deletion, and migration.

- Robustness: virtualization leads to system robustness as clients spread across multiple VMs.

Due to the aforementioned benefits, virtualization is globally adopted in cloud data center environments (Goiri et al., 2012), (Ahmed, Gani, Khan, Buyya, & Khan, 2015). In a virtualized data center architecture, each cloud client (application or user) is assigned a chunk of data center resources. The data center resources close to the hardware platform can be categorized into physical resource set and virtual resource set (Lenk, Klems, Nimis, Tai, & Sandholm, 2009). The virtual resource set works as a management platform over the physical resource set to provide the illusion of a single scalable platform to all cloud clients. A hypervisor or Virtual Machine Monitor (VMM) is hardware independent technology that manages virtual machines over heterogeneous hardware platforms. The hypervisor is a set of computer hardware, firmware, and software that lies between the hardware and the Operating System (OS). The hypervisor has the ability to initiate one or

University of Malaya

more than one OSes over a single hardware resource set (Younge et al., 2011). Inside a virtualized data center, clients reside over a pool of virtual resource sets.



Users
DISPATCHER
Data center/rack level management of resources
Service Level Agreement, pricing model
Resource mapping, admission control
Server Rack
Physical node n
Physical node 1
VMM
VM
VM
....
VM
Core 1
Core n

Figure 2.2: Architecture of a virtual data center

When a new client request arrives at the cloud data center, it is forwarded by the dispatcher to corresponding VMM. The dispatcher requests physical resources according to the client SLA, pricing model, and application QoS requirements (Buyya, Garg, & Calheiros, 2011), (Shiraz, Gani, Shamim, Khan, & Ahmad, 2015). Fig. 2.2 illustrates the architecture of a virtualized data center consisting of multi-core servers. In a virtualized data center architecture, multiple clients often share same hardware resources with the

University of Malaya

help of virtualization techniques. Moreover, hardware resources provisioned for a data center client can be scaled dynamically according to varying workload. The resource scaling can be done with a variety of virtualization methods such as VM creation, deletion, and migration. A workload can be consolidated or migrated onto a lesser number of resources using VM migration. The resultant resource set provides energy efficiency and higher resource utilization. The CPU power along with other computing resources such as memory and I/O can be scaled gracefully with the help of virtualization technologies (Younge et al., 2011). When a hardware resource is underutilized due to lesser client requests, it represents and opportunity for resource consolidation. The workload of underutilized hardware is transferred to another suitable hardware with the help of hypervisor. The workload consolidation and migration technique is depicted in Fig. 2.4.

University of Malaya

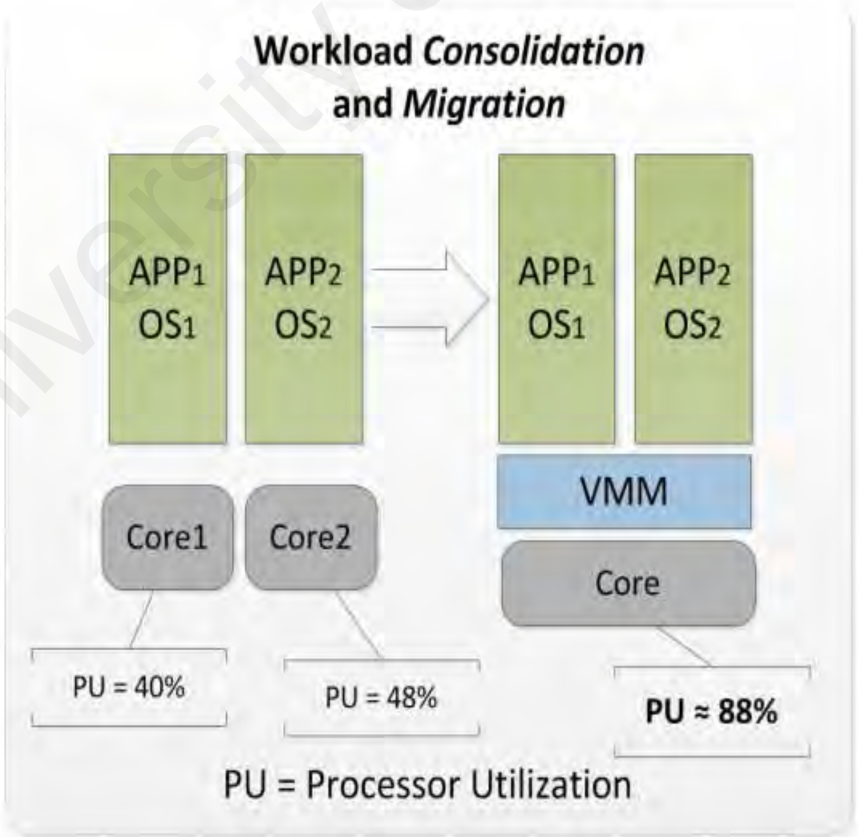


Figure 2.3: Workload Consolidation and Migration

Clouds adopt the virtualization concept for many reasons, such as a) server consolidation, b) applications' adaptive and dynamic configuration, c) high availability, d) and responsiveness (Foster et al., 2008). All virtualization features provide clouds a basis to meet SLA requirements (Ahmad et al., 2015). To ensure the network performance along with dynamic resource provisioning, virtualization tries to balance the load of the whole system dynamically (Espadas et al., 2013) there is always a chance of over utilization or underutilization of resources. Overloaded servers lead to performance degradation whereas under loaded servers cause poor utilization of resources. Due to inefficient distribution of load more heat will be generated by the overloaded servers which in turn increase the cost of cooling system and substantial emission of CO2 contributing to greenhouse effect (Shaw & Singh, 2014). Therefore, there is a need to provide right amount of resource dynamically to the applications running in virtual machines and develop a efficient load balancing technologies not only for reducing operational cost but also for decreasing its influence on system's reliability in order to meet the QoS requirement.

University of Malaya

2.2 Cloud Load Balancing

Load balancing, a deployed function, plays its vital role in cloud and cloud data center domains for efficient resource management (Chaczko et al., 2011). Load balancing ensures even distribution of resources among a set of users in a uniform way such that underlying servers do not become overloaded and idle at any time within cloud operation time line. Overlooking load balancing establishment abruptly decreases system throughput due to overloaded servers and ultimately leads to SLA violation. It has become an integral part of all distributed internet based systems as distributed computing comes with the challenges of high resource demands that overload servers. Load balancer increases the capacity and reliability of applications by decreasing the burden on a server. In addition, load balancing is an essential aspect in Internet based all others distributed computing tasks (Foster et al., 2008). Cloud service providers (CSP) also provide the efficient load balancing solutions to the users in their own cloud computing platforms. Furthermore, an inter CSP load balancing algorithm is required to build the low cost and infinite resource pool for the customers (Mondal et al., 2012). Toward this goal numerous load balancing schemes such as Minimum Execution Time (MET) (Armstrong et al., 1998), Min-Min scheduling (Etminani & Naghibzadeh, 2007), Cloud Analyst (Sefraoui et al., 2012) exists in literature and a comprehensive study is also done with and First Come First Serve (FCFS) and Round-robin. The main goal of job scheduling is to achieve a high performance computing and the best system throughput. Traditional job scheduling algorithms are not able to provide scheduling in the cloud environments.

2.2.1 Taxonomy of State-of-the-art Cloud Load Balancing Schemes

This section define the thematic taxonomy of the classification and characterization of cloud load balancing algorithms used to attain the various objectives such as, fair allocation, efficient utilization of resources, cost effectiveness, scalability and flexibility, and resource prioritization. Load balancing algorithms are characterized based on ten characteristics, namely (a) environment, (b) system topology, (c) VM placement function, (d) task scheduling, (e) resource allocation, (f) provisioning decision, (g) cloud type, (h) cloud resource type, (i) load balancing policies, (j) objective function.

2.2.1.1 Environment

The environment of the load balancing algorithms states that weather the nature of algorithms is static or dynamic. In load balancing algorithms, environment is either mobile or immobile based on run time state information to make efficient decision in order to

University of Malaya

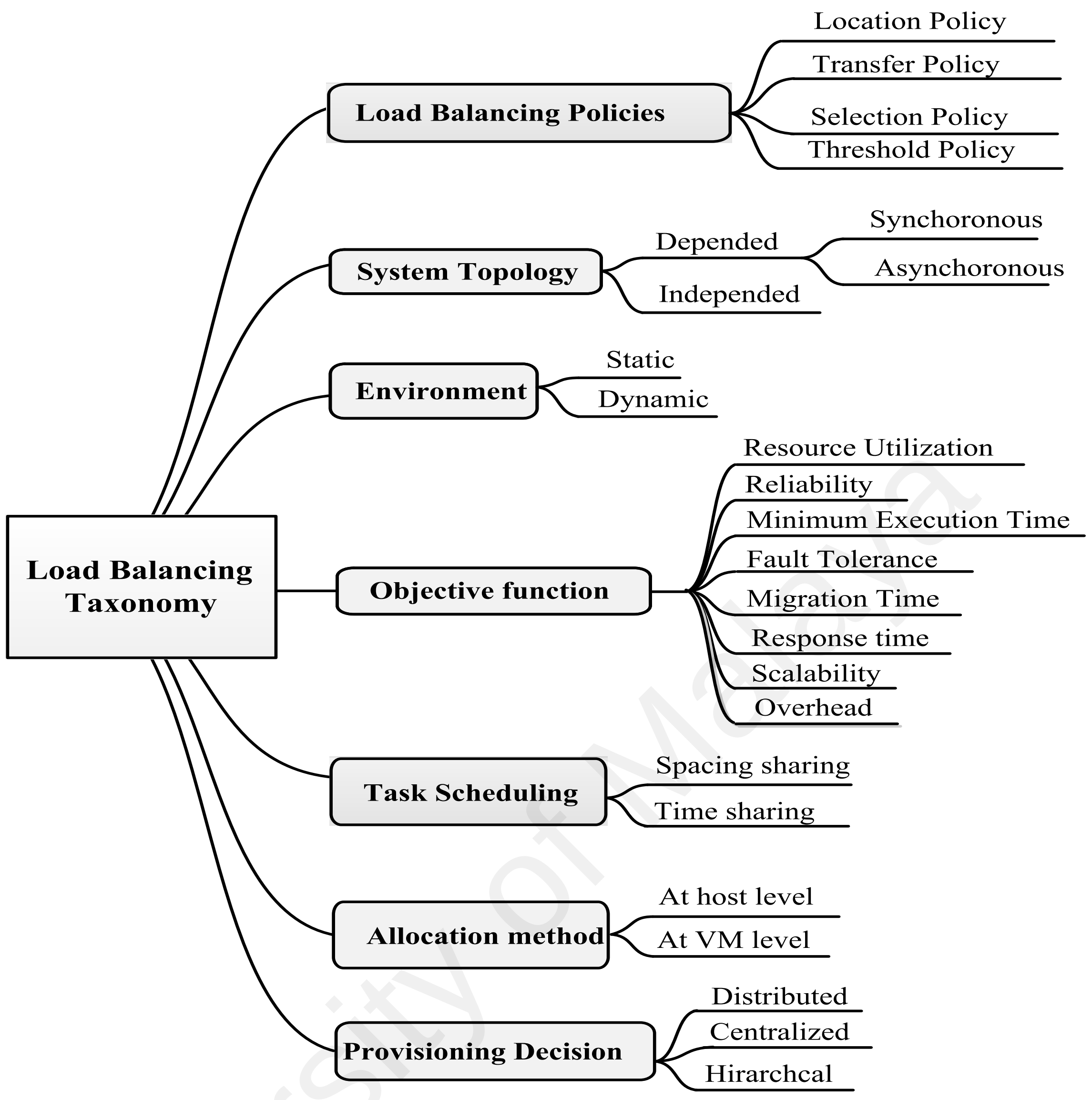


Figure 2.4: Taxonomy of Cloud Load Balancing Schemes

share the system load. In static load balancing, algorithms assumes that all the prior knowledge of the nodes related to network resources, computing resources, memory, processing power, and storage capacity are known and provided. Static algorithms assign the tasks to the nodes based on their new request's acceptance capabilities. The attractiveness of the static algorithms is that it offers minimum execution time. However, it has two major limitations. Firstly, in static approach load balancing decisions are taken into account at compile time probabilistically or deterministically and cannot respond at run-time. Secondly, it assumes all the prior information of nodes will remain same. Therefore, such an assumption may not suitable for distributed environments. In contrast, dynamic

University of Malaya

approaches consider the network bandwidth, and nodes properties at runtime. Most of the dynamic load balancing algorithms rely on the combination of information based on prior gathered knowledge in the cloud and runtime properties of the nodes. These algorithms can allocate and dynamically reallocate the tasks based on gathered and collected attributes of the system. The limitation of this approach is that continuous monitoring of the nodes and task progress is required which is hard to implement. However, dynamic schemes are more accurate in order to provide the efficient load balancing. Therefore, dynamic scheme is used in modern load balancing techniques because of its flexibility and robustness.

University of Malaya

2.2.1.2 Allocation method

Allocation method is defined as the mapping of the tasks to the cloud resources based on demands. In cloud, resources must be allocated in such a manner that no node/ host become under loaded and over loaded and all the available resources do not endure any kind of wastage in terms of core, bandwidth, and memory etc. Therefore, mapping is further classified into two categories such as (a) VM mapping on host, and (b) task's mapping on VMs. VMs are deployed on physical hosts. Based on the physical host capabilities and availabilities several VMs can mapped on a single host. Host is accountable in order to allocate the number of core to VMs. Algorithms with the VM mapping on hosts are provider oriented and as well as customer oriented. In allocation policy to ensure that characteristics of host and VM are not mismatched is challenging task. Moreover, better allocation policy provide the efficient utilization of resources and minimize the makespan time of the resources. Cloud users can execute their VM efficiently with limited number of physical machines. Hence, this approach will lead to an efficient utilization of resources available to facilitate maximum computing with minimum physical data centers infrastructures. Besides, based on task's mapping on VMs applications are executed on VMs. For the completion of task each application requires a certain amount of energy. So, VM must offer the required amount of energy in order to accomplish the mapped tasks. Moreover, based on VM availability and configuration tasks should be mapped on appropriate VMs. The objective of that policy is to achieve the minimum execution time with high performance.

2.2.1.3 Task Scheduling

Task scheduling is done when the resources are allocated to the cloud. Task scheduling is described as a method in which allocated resources are offered to end users (weather the resources are available based on sharing or fully available until task is complete). In cloud environment, it provides the multiprogramming capabilities. Moreover, task scheduling is further classifies in two modes such as, a) space sharing, and b) time sharing. Both hosts and VM provisioned to users based on time shared mode or either in space shared mode. In space shared scheduling one task is scheduled to the VM at a given instance of time and after its accomplishment another task is assigned to VM. Moreover same strategy is used to schedule the VM onto hosts. This policy behave same as the first come first serve algorithm (FCFS). This allocation policy enables the task units to be scheduled at an earlier time, but significantly affecting the completion time of task units that are ahead the queue. In Time-Shared scheduling policy it schedule all tasks on virtual machine at the same time. It shared the time among all tasks and schedule simultaneously on the virtual machine. This policy is also used to schedule the virtual machine on the host. The concept of round-robin (RR) scheduling algorithm is used in this policy. Space shared scheduling policy shows better results as compared to Time-shared scheduling policy when number of tasks are increased.

University of Malaya

2.2.1.4 Provisioning decision

An algorithm is centralized if the parameters necessary for making the load balancing decision are collected at, and used by, a single resource i.e. only one resource acts as the central controller and all the remaining resources act as slaves. The centralized approach is more beneficial when the communication cost is less significant e.g. in the shared-memory multi-processor environment. Its limitation is single point of failure and non-scalable. However, in decentralized approach all the resources are involved in making the load balancing decision. In Distributed approach, nodes individually forms their particular load vector by gathering the load data of other nodes. Based on the local load vectors the conclusions are accomplished locally. In addition, for the widely used distributed systems such as cloud computing this method is more appropriate. Decentralized algorithms are more scalable and have better fault tolerance. Hierarchical load balancing is presented with the multiple number of tiers of cloud while the load balancing decision. Besides, these schemes works based on the modes of master and slave where master node is responsible for all the slaves nodes in order to collect their data by using the light weight agent processes.

University of Malaya

2.2.1.5 Load balancing policies

An algorithm for the load balancing problem can be broadly categorized in terms of four policies in terms of (i) location policy, (ii) transfer policy, (iii) threshold policy, and (iv) information policy. Moreover, the location policy it is the policy that affects the finding of a suitable node for migration. The common technique followed here is polling, on a broadcast, random, nearest-neighbor or roster basis. Transfer policy it is that which determine whether a node is suitable for participating in a process migration. One common technique followed is the threshold policy, where a node participates in a negotiation only when its load is less than (in destination-initiated algorithm) or greater than (in sender-initiated algorithm) a threshold value. Selection policy it is the policy that deals with the selection of the process to be migrated. The common factors which must be considered are the cost of migration (communication time, memory, computational requirement of the process, etc.) and the expected gain of migration (overall speedup of the system, etc.). Information policy it is that component of the algorithm that decides what, how and when the information regarding the state of the other nodes in the system in gathered and managed. They can be grouped under demand-driven, periodic, or state-change-driven policies.

2.2.1.6 System topology

In order to understand the functionality of load balancing the schemes are classified into the topology depended and independent categories. Topology depended algorithm is described as the mechanism which is designed for the specific topology where its functionalities and logics are predefined. The topology depended algorithms are further classified into two groups names as synchronous and asynchronous algorithms. Synchronous algorithms are appropriate for exceedingly parallel schemes whereas the asynchronous are suitable for parallel systems because of their local behavior. In contrast, the independent algorithm are not designed for the specific topologies and their functionalities are also not defined in advance. Moreover, the independent algorithm does not face the compatibility issues based on topology designed parameters. Furthermore, the drawback of topology depended algorithm is that it leads to high communication overhead as compare to independent algorithms.

University of Malaya

2.2.1.7 Objective function

The objective function defines the aim of state-of-the-art SC frameworks. The metrics on which the existing load balancing techniques have been measured are discussed below:

- Reliability: It can be defined as the efficiency of the system. This has to be improved at a reasonable cost, e.g., reducing the response time though keeping the acceptable delays.

- Resource Utilization: It is used to ensure the proper utilization of all those resources, which comprised the whole system. This factor must be optimized to have an efficient load balancing algorithm. It should be maximum for an efficient load balancing system.

- Scalability: The quality of service should be same if the number of users increases. The more number of nodes can be added without affecting the service. It is the ability of an algorithm to perform uniform load balancing in a system with the increase in the number of nodes, according to the requirements. Algorithm with higher scalability is preferred.

- Execution Time: This metric is used to estimate the total time required to execute the specific application. Minimum execution time is necessary for overall system performance.

- Overhead: Overhead associated with any load balancing algorithm indicates the extra cost involved in implementing the algorithm. Overhead Associated determines the amount of overhead involved while implementing a load-balancing algorithm. It includes overhead due to movement of tasks, inter-processor and inter-process communication. It should be as low as possible.

University of Malaya

- Fault Tolerance: It measures the capability of an algorithm to perform uniform load balancing in case of any failure. A good load balancing algorithm must be highly fault tolerable.

- Migration Time: It is defined as, the total time required in migrating the jobs or resources from one node to another. It should be minimized.

- Response Time: It can be measured as, the time interval between sending a request and receiving its response. It should be minimized to boost the overall performance.

2.2.2 Review of load balancing schemes for cloud environments

This section briefly describes the state-of-the-art load balancing schemes. Load balancing algorithms are further categorized as static and dynamic load balancing techniques.

2.2.2.1 Static load balancing schemes

This section addresses state-of-the-art load balancing algorithms based on static load balancing. Moreover, Table 2.1 represents the load balancing schemes along with their objectives, strengths, and weaknesses as a future directions. Besides, based on the taxonomy the comparison of static load balancing algorithms is explained in Table 2.2.

Author in (Samal & Mishra, 2013) presented the static load balancing algorithm. This study uses the round robin algorithm for the allocation of VMs on PMs. The main objective of this scheme is to equally distribute the load to each PM. The process of round robin scheduling in cloud is very similar to the round robin scheduling in a process scheduling. While the deployment of VMs, the scheduler randomly selects the first node and then allocate the VMs to PMs in a circular motion until one VM is allocated to each node and then the scheduler return to the first node again. The advantage of this scheme

University of Malaya

Table 2.1: Summary of Static VM placement schemes in a cloud environment.

Schemes	Description/ Objective	Strength	Weakness
(Kaur & Kaur, 2015)	The states of VMs are recorded in record table in terms of idle or busy states. The -1 is returned when the record is not matched in table and request is queued until value is not 1	TLB attempts to equally distributes the load among VMs	Do not consider the current load of VMs
(Domanal & Reddy, 2014)	Requests are assigned to the VM which available resources in terms of RAM and VM is not selected for the allocation of previous request	VMs are utilized completely and properly	Overlooks the experimental setup along with its parameters
(Ashwin, Domanal, & Guddeti, 2014)	Any available VM is selected for the assignment of request. This algorithm handles the overloading while the placement requested to the VMs	Job processing time and response time is enhanced	Not a fault tolerant solution when single node is failed
(Shaw & Singh, 2014)	Based on existing processing power weight is allotted to each VM. Task are allocated to more powerful and least loaded VMs	Consider the energy consumption and load of available VMs	Weight consignment maximize the complexity of algorithm
(Gautam & Bansal, 2014)	Weight is allotted to each VM based on its power capacity. Maximum number of requests are assign to the VM which is associated with high power	Efficient resource utilization	Overlooks the application processing time
(Samal & Mishra, 2013)	Only first request is assigned to the randomly selected VM. The rest of the requests are allocated in circular order	Equally distribute workload in circular order	Execution time is not considered
(Adhikari & Patil, 2013)	This algorithm is based on two rules : i) Retiring State of VM and ii) Retirement Threshold and migration of VMs	Reduces the power consumption cost	Does not scale up for large data centers
(Domanal & Reddy, 2013)	For the request allocation the record table is searched from the next to already allocated VM	Enhance the response time	While the placement of the request state of index table may change
(H. Chen et al., 2013)	Similar to the min-min algorithm the smallest job's completion time is calculated through the maximum loaded resources on each PM	Minimize the application completion time and balanced the load	While scheduling does not consider the priority
(James & Verma, 2012)	VM allocation is like Throttled but based on priority which calculated using CPU speed and memory capacity of VM	Appropriate for heterogeneous cloud environment	Priorities are predefined
(Elzeki et al., 2012)	Similar to min-min algorithm where are jobs with the maximum execution time are completed first	Decreases the makespan	Smallest tasks have to wait for long time
(Kokilavani & Amalarethinam, 2011)	The smallest job is allocated to the fastest resources. Once the allocation process is complete the job is detached from the list and again the same process is repeated	Efficient algorithm	Does not consider the existing load
(Bramson, Lu, & Prabhakar, 2010)	Jobs are randomly assigned to available VMs	Efficient algorithm	Not considered the current load of VMs
(S.-C. Wang et al., 2010)	The selects jobs are assigned to the available VMs based on random selection criteria	Each VM is reserved into busy state	Overlooks the execution time of tasks
(Hu, Gu, Sun, & Zhao, 2010)	The proposed model is divided into three sub models where at first stage the tasks are received at request manager and sent it to service manager in second stage. The service manager allocate them to the service nodes for their execution as sub tasks	Availability of VM and RAM is considered, enhance the task execution time	Overlooks the node selection for the complicated tasks
(Rahmeh, Johnson, & Taleb-Bendiab, 2008)	TAt a specific node the sampling walk is started and move forward by selecting the random neighbors. Moreover for the distribution of load the last node elected	Proposed a suitable decentralized scheme which is suitable for large scale networks	Not appropriate for dynamic environments

University of Malaya

is that it allocates the equal number of VMs to each PM, which ensures the fairness. In contrast, this algorithm overlooks the resource exhaustion of a specific node based on the circular deployment and before moving to the next node. In addition, this algorithm performs well when the workload nature is uniform on each VM. This algorithm is not suitable when the workload nature is non-uniform in this situation some nodes get lightly loaded and some get heavily loaded.

Authors in (S.-C. Wang et al., 2010) proposed the static load balancing algorithm. For the resource deployment, OLB algorithms tries to keep every singles node busy and deals in random order with the non-executed tasks to the current available nodes. Without providing the satisfactory results this algorithm deals with the balanced load scheduling technique. Moreover, the positive aspect of OLB algorithm is that it provides the work-load to the nodes in free order. In contrast, the drawback of that technique is that does not compute the current execution time for every single node.

The proposed algorithm computes the completion and execution time of unassigned tasks thats waits in queue (Kokilavani & Amalarethinam, 2011). Authors focused on the static load balancing algorithm, therefore, resources related to the tasks are known in advance. Min-Min algorithm algorithm first deals with the tasks which have the minimum execution time by allocating them to the processor according to the capability of task completion time. In this algorithm the task with minimum time value is scheduled to the corresponding machine. After task assignment the execution time of the remaining resources is updated for further allocation. In addition, completed tasks are removed from the list after their completion when assigned to machine. The advantage of that algorithm is that it performs well when each task has the minimum execution. The negative point of that algorithm is that the tasks with the maximum execution time have to wait for unspecified period of time until the small tasks are not completely executed. The major drawback of this approach it leads to starvation when deals with tasks having the maximum execution time.

Max-min algorithm is vise-versa to the min-min algorithm. Min-min algorithm first deals with the task which has minimum execution time. In contrast, Max-min algorithm first handles the job which has maximum execution time (Elzeki et al., 2012). Moreover, in that algorithm the task execution time is known in advance. The methodology of the task selection and assignment is same as presented in (Kokilavani & Amalarethinam, 2011). In Max-min after the execution of jobs with higher time are completed first and removed

University of Malaya

from the task list. Moreover, this algorithm presents the enhanced version of min-min algorithm. In order to reduce the execution time of meta-tasks contains homogeneous tasks are allocated on same hosts. This approach improves the efficiency of the scheme by adapting the opportunity of concurrent execution of tasks on resources.The drawback of algorithms is that the small tasks have to wait for long time.

Weighted round robin is the extended version of round robin algorithm (Gautam & Bansal, 2014). I weighted round robin weight is allocated to each VM based on their capacity. Based on the associated weightage higher capacity VMs are allocated with multiple number of tasks. The weighted round robin algorithm performs well based on the processing capability of higher capacity VMs. Though, this algorithm leads to imbalance load among servers if the loads on VMs vary highly. Therefore, there is a possibility that the largest request with maximum execution time may be allocated to same VM. The shortcoming of that technique is that it does not consider the length of the task in order to select the appropriate VM.

The proposed algorithm scans the VMs and jobs which are listed in queue for execution. The objective of the algorithm is to assigned the queued up job to the available VMs (Domanal & Reddy, 2013). Proposed load balancer frequently examine the overloaded situation of the VM and based on the load conditions distributes the some of its jobs to another VM which is least loaded in order to make the equal distribution of load. This load balancer manage the list of allocated VMs which helps to identify that VMs are free and available to host the new jobs. The performance analysis of the proposed algorithm is done based on the cloud analyst simulations. The advantage of this algorithm is that it tries to improve the processing time and response time of task by picking it when there is a match available.

Authors in (Kaur & Kaur, 2015) proposed the throttled Load Balancer algorithm. This algorithm manage record of idle and busy states in an index table of VMs. In order to

University of Malaya

allocate the request to suitable VM, server and clients make request to the data center to perform recommended task. The data center send requests to load balancer regarding the distribution of VMs. The load balancer identifies the VM id in the index table from top until the requested VM is matched and after the identification process load balancer update to the data center for the requested VM deployment. Moreover, the load balancer sent -1 If the suitable VM is not matched. In addition, the acknowledgement is sent to data center after the completion of the allocated task to VM. The data center apprised to load balancer in order to de-allocate the same VM which has completed its assigned tasks and reallocate it for the next jobs.

Proposed algorithm estimates the total execution tine in three phases based on (i) VM deployment, (ii) task allocation to VM's, and (iii) VM reallocation process once the assigned task is completed (Shah, Kariyani, & Agrawal, 2013). This algorithms estimate the throughput based on the total number of assigned tasks to VMs within the required time-span without considering the third phase of VM reallocation. The positive point of this algorithm is that it enhance the performance by providing the on-demand resources. Moreover, it minimize the rejections rate of the submitted requests. The disadvantage of this algorithm is that is overloaded the initial deployed VMs and under utilizes the VMs which are deployed at the end.

This study presents the load balancing approach with the consideration of uniform load balancing among VMs and the availability of VMs based on requests (Domanal & Reddy, 2013). Proposed study focus on the two main objective. First one is depicted as a response time which is required for the task allocation and second one is described as the load distribution among the existing VMS. Based on throttled algorithm, modified throttled algorithm also manage the index table with the records of VM with their states. The first VM selection procedure is same as described in (Shah et al., 2013). Afterward, when next request is arrived, VM to the index next to the already allocated VM is selected based on

University of Malaya

the idle or busy states of VM. This algorithm maximize the response time as compared to throttled. The shortcoming of this algorithm is that in index table while the deallocation of VMs the states of VM may change. Therefore, it is not beneficial to allocate the new request using the next to assigned VM technique. There is a need to focus on the focus on the data structures while managing the index tables for the allocation of new tasks.

The enhanced version of throttled algorithm is represented in (Bhadani & Chaudhary, 2010). Based on the priority of VMs the state table is also maintained in this algorithm similar to throttled algorithm. The priority is computed based on the RAM capacity and speed of CPU. The VMs with high priorities will selected first by using the high priority based selection criteria. Moreover, if the selected VM is busy then it neighboring VM is selected and the process will continue until the VM is availability is not checked in whole table. Proposed algorithm only balanced the load in heterogeneous environments and leads to inefficient deployment when number of requests and increased because all request are entertained at central load balancer. In addition, in this scheme the priorities are predefined and calculated using a static method and the priorities are not modified after the allocation of jobs.

This scheme handles the allocation of requests at VM level (James & Verma, 2012). Information about each VM requests that are allocated to each VM is handled by this algorithm. While the allocation of requests the proposed algorithm selects the VM with the least load and if there are number of VMs are chosen with the same load the 1st VM is identified for the placement of request. Proposed algorithm address the problem of (Bhadani & Chaudhary, 2010) algorithm and update the load value after the allocation of requests to the VM. In contrast, proposed algorithm does not assign the priorities to the VMs. Moreover, in order to handle the load the id of specific VM where the request is allocated is send to the active load balancer. The proposed algorithm focuses on the current load value and overlooks the load current load at host level. Moreover, the energy

University of Malaya

consumption parameter is not considered while the allocation of requests at VM level.

Table 2.2: Comparison of state-of-the-art static load balancing schemes in a cloud environment

Algorithm	**Parameters used**	**Resources considered**	**Task scheduling**	**Objective functions**	**Allocation method**	**System topology**
(Kaur & Kaur, 2015)	Bin Packing	memory, network,	time-share	resource utilization	VM-level	dependent
(Gautam & Bansal, 2014)	Constraint Programming	memory and I-O	space-share	resource utilization, scalability	host-level	depended
(Ashwin et al., 2014)	Stochastic integer programming	memory as a constraint	time-share	response time	VM-level	independent
(Domanal & Reddy, 2014)	Bin Packing	memory	time-share	response time	VM-level	-
(Shaw & Singh, 2014)	Bin Packing	CPU	space-share	energy consumption	host-level	dependent
(Samal & Mishra, 2013)	Constraint Programming	memory	space-share	resource utilization	host-level	dependent
(Adhikari & Patil, 2013)	Constraint Programming	CPU, memory	time-share	energy consumption	host-level	dependent
(H. Chen et al., 2013)	Bin Packing	CPU, memory	time-share	application execution time	host-level	dependedt
(Domanal & Reddy, 2013)	Bin Packing	memory	time-share	resource utilization	VM-level	-
(James & Verma, 2012)	Bin Packing	memory	space-share	resource utilization	VM-level	dependent
(Elzeki et al., 2012)	Genetic Algorithm	memory, storage	space-share	overhead	host-level	-
(Kokilavani & Amalarethinam, 2011)	Genetic Algorithm	memory	space-share	response time	host-level	independent
(S.-C. Wang et al., 2010)	Bin Packing	memory	time-share	reliability	VM-level	independent
(Hu et al., 2010)	Constraint Programming	memory, storage	-	scalability	host-level	dependent
(Bramson et al., 2010)	Stochastic bin packing	memory	space-share	scalability	VM-level	independent
(Rahmeh et al., 2008)	Genetic Algorithm	memory	space-share	fault tolerant	host-level	dependent

The extended version of (James & Verma, 2012) is presented in (Domanal & Reddy, 2014). Upon the request placement to VM all the VMs availability is checked from the search table. If the VM is not available and others VM which are not selected in the previous assignment will be selected for that request and the least loaded VM is selected from table. Moreover, the proposed scheme not used the VM which is assigned to the request in preceding assignment. As compare to (James & Verma, 2012) the proposed algorithm efficiently utilize the VMs. In order to check the subsequent minimum loaded VM a task is evenly distributed to equally loaded VMs and the least loaded VM with the

University of Malaya

high processing power is selected regardless of the 1st VM. Moreover this algorithm not consider the statistic that whether it is used in the latest iteration or not. In contrast, using this algorithm most of the VMs are taking maximum time for the request allocation along with the greater response time. Moreover the proposed work not clearly defined how the least loaded VM selection policy works.

2.2.2.2 Dynamic load balancing schemes

This section discusses the state-of-the-art load balancing algorithms based on dynamic load balancing. Moreover, Table 2.3, and Table 2.4 compares the load balancing schemes based on their their objectives, strengths, and weaknesses.

Table 2.3: Summary of dynamic load balancing schemes in a cloud environment

Algorithm	**Objectives**	**Strength**	**Weakness**
(Ghribi et al., 2013)	To reduced the energy consumption while the scheduling of VMs in cloud data centers	Reduced energy consumption	Need for extension to multiple resources
(Nicolae & Cappello, 2012)	To efficiently manage the energy usage in cloud data centers	Reduced the number of APMs, Improved CPU utilization	Overhead of large searching spaces
(M. Chen et al., 2011)	To control the size of VM placement in cloud	Reduced the number of APMs & O (1) approximation	Need for extension to multiple resources
(Mishra & Sahoo, 2011)	To efficiently place the VMs and to mitigate the overhead using the vector based approach	Considered the migration overhead	Overlooks the details of experimentations
(Guo et al., 2010)	To guarantee the efficient bandwidth rate for the virtualized data centers	Reduced energy consumption and network traffic	More VM migration cost
(Beloglazov & Buyya, 2010)	To efficiently utilize the energy while the allocation of resources	Reduced cost of relocation	Uses slightly more number of bins
(Wood, Shenoy, Venkataramani, & Yousif, 2009)	To detect and identify the hotspots, and reconfiguring/ remapping VMs when required	Hot-spot detection & mitigation, Load balancing	VM resizing & Migration overhead
(Stage & Setzer, 2009)	To efficiently manage the network utilization by adapting the migration concept	Meets SLA targets	Need for extension to multiple resources
(Bobroff et al., 2007)	To dynamically balance the workload to manage the SLA violation	Reduced the number of APMs, Reduced migration cost	Slightly slow execution
(Kang & Park, 2003)	To address the problem of variable size bin packing	Hot-spot mitigation, Load balancing	SLA violation

Authors in (Nicolae & Cappello, 2012) have used a testbed of a head node and 5 VM hosts to study the effects of VM live migrations in a data center hosting Web 2.0 applications. The VM hosts run olio that defines a simple Web 2.0 application while Faban load generator is used for workload generation. A downtime of 3 seconds was observed near the end of a 44 second migration. Although no request was dropped during the downtime,

University of Malaya

the delay does affect the service level agreement (SLA). While each VM had a 2GB of memory allocation in the experimental setup, in real data center environments the size of a VM can scale up to hundreds of GB. Therefore the effects of migrating large VM's can be more severe. Furthermore the results showed that 2 VMs migrations occurring in close time proximity lead to sever SLA violations. Hence the modeling of live migration as a queuing system comes under consideration.

(Stage & Setzer, 2009) discusses the impact of VM live migration on the network resources. The proposed architecture consists of VM workload classifier, an allocation planner, a nonconformance detector, and a live migration scheduler. The VM workload classifier assigns the workload to a relevant cluster class based on the attributes of the workload. The allocation planner determines the resource bottlenecks that can occur after an allocation. The non-conformance detector classifies the bottlenecks detected on the basis of pre-defined performance thresholds. The migration requests are made by allocation planner and non-conformance detector to the migration scheduler. A migration scheduler determines the optimal schedule for the migrations, based on the knowledge of their duration, starting time and deadline. The optimal scheduler schedules the live migrations in such a way that the network is not congested by the VM live migration load. The live migrations are also fulfilled in time. The following diagram shows the difference between an uncontrolled and controlled migration scheduling algorithm. The lower timeline depicts a controlled migration in which three migration requests are executed as compared to two requests in an uncontrolled environment.

Authors in (Beloglazov & Buyya, 2010) have proposed that live migration of VMs can be used to concentrate the jobs on a few physical nodes so that the rest of the nodes can be put in a power saving mode. The allocation of VMs is divided into two sub-problems: (a) the admission of new requests and (b) optimization of current VM allocations. The allocation, of new requests for VMs, is done by sorting all the VMs in a Modified Best

University of Malaya

First Decreasing (MBFD) order with respect to the current utilization. The VM is then allocated to a host based on the least deterioration in the power consumption among the hosts. The current allocation of VMs is optimized by selecting the VMs to be migrated on the basis of heuristics related to utilization thresholds. If the current utilization of a host is below a threshold, then all the VMs from that host should be migrated and the host is put in the power saving mode. Again the allocation of VMs to hosts is done by MBFD algorithm.

A similar approach achieves energy efficiency with the help of Limited Look Ahead Control (LLC) (Kusic, Kephart, Hanson, Kandasamy, & Jiang, 2009). The LLC predicts the next state of the system by a behavioral model that depends on the current state, environment input and control input. A profit maximization problem, based on the non-violation of SLA and the energy conservation, is formulated to calculate the maximum number of physical hosts that can be powered off. The optimization problem suffers the curse of dimensionality as more control options and longer look ahead horizon are considered during formulation. To avoid the curse of dimensionality, the problem is decomposed into two sub-problems with respective sub-controllers. Although this approach caters for most of the virtualized environment dynamics, such as SLA and energy efficiency, it does not consider the effects of live migration on network dynamics.

In SecondNet (Guo et al., 2010), a central Virtual Data Center (VDC) manager controls all the resources and VM requests. When the VDC manager creates a VM for the VDC, it assigns the VM, a VDC ID and a VDC IP address, reserves the VM-to-VM and VM-to-core bandwidths, as mentioned in the Service Level Agreement (SLA) for the application using the VM. The inputs to the VM allocation algorithm are the m VMs and the m $*$ m bandwidth matrix R^9. The output is m physical server and the 4 paths corresponding to the bandwidth matrix R^9. Cluster of servers are formed based on the number of hops from one cluster to another. A cluster is chosen, Ck, such that: (a) it has more ingress and

University of Malaya

egress bandwidth then that specified in R9 and (b) the number of servers in the cluster is larger than the number of VMs i.e. m. A bipartite graph is formed from the VMs (m) and physical servers in the cluster Ck. Mapping from VMs (m) are made to physical hosts in Ck based on individual VMs memory, CPU and bandwidth requirements. A bandwidth defragmentation algorithm is also devised to reduce inter-cluster bandwidth and improve network utilization. A VM migration is scheduled if meets the following criteria (a) it increases the residual bandwidth of the data center and (b) the bandwidth requirements can be met by the cluster where VMs are reallocated. Simulations demonstrate that the system provides a guaranteed bandwidth and high network utilization. This approach does consider the residual bandwidth for VM allocation optimization, but it does not consider the bandwidth required during the process of reallocation.

A study to measure the impact of virtualization on network parameters, such as throughput, packet delay, and packet loss has been conducted by (G. Wang & Ng, 2010). The study is carried out on the Amazon EC2 data center where each instance of the data center is a Xen VM. Processor utilization and TCP/UDP throughput are measured by CPUTest and TCP/UDPTest programs, respectively. The packet loss is measured by the Badabing tool (Sommers, Barford, Duffield, & Ron, 2005). The results show an unstable TCP/UDP throughput and a very high packet delay among EC2 instances. It is concluded that these results are obtained due to virtualization and sharing of drivers among several VMs.

The VectorDot scheme as discussed in (Mishra & Sahoo, 2011) has considered the current load on the communication paths connecting physical servers and network attached storage. Furthermore, VectorDot has addressed the overloaded servers, switches, and storage entities while meeting the desired objective function. Moreover, using constraint programming paradigm, tasks are migrated within nodes located in a cluster and has proved that consolidation overhead is indomitable while choosing a new configuration and also it is affected from the total migration time with that configuration (Hermenier, Lorca,

University of Malaya

Menaud, Muller, & Lawall, 2009). Furthermore, employed Entropy has significantly reduced total VM migration duration in addition to the total number of nodes acquiring low performance overhead. Consequently, the authors of (Zhao & Figueiredo, 2007) has accurately projected the total migration cost in order to have an accurate estimation guess of migration time, so that sufficient resource can be prepared and reserved on the basis of VMs count and the performance degradation period instigated by VM migration, that is higher than actual total migration duration. Moreover, the proposed scheme has also presented the migration cost based on the migrating VM configuration and size.

Table 2.4: Comparison of state-of-the-art dynamic load balancing schemes

Algorithm	**Parameters used**	**Resources considered**	**Task scheduling**	**Objective functions**	**Allocation method**	**Performance Better Than**
(Ghribi et al., 2013)	Stochastic bin packing	CPU, bandwidth	space-share	energy consumption, fault tolerant	VM-level	First-fit, FFD & Harmonic algorithm
(Nicolae & Cappello, 2012)	Genetic Algorithm	CPU, request forecasting and Reconfiguration searching module	time-share	energy consumption	-	TSSP Approach
(M. Chen et al., 2011)	Stochastic integer programming	CPU, memory, server overflow probability '*p*'	-	energy consumption	VM-level	FFD algorithm
(Mishra & Sahoo, 2011)	Bin Packing	CPU, memory, network, i/o bandwidth, storage	space-share	migration overhead	VM-level	Best-fit, First fit, Worst-fit
(Beloglazov & Buyya, 2010)	Bin Packing	CPU, upper bound on *cost* of VM relocation	time-share	reource utilization	VM-level	Best-fit, First-fit
(Guo et al., 2010)	Constraint Programming	CPU, network bandwidth	time-share	energy consumption and network traffic	VM-level	Random algorithms
(Stage & Setzer, 2009)	Stochastic integer programming	CPU, time interval of length 'IJ'	time-share	-	reliability	Static algorithm
(Wood et al., 2009)	Bin Packing	CPU, memory, network, VSR (Volume to Size ratio)	time-share	resource utilization	VM-level	-
(Bobroff et al., 2007)	Constraint Programming	CPU	space-share	execution time	host-level	Best-fit heuristic
(Kang & Park, 2003)	Bin Packing	memory	space-share	hot-spot mitigation	host-level	-

An efficient energy scheduling algorithm is presented in (Ghribi et al., 2013) for the exact provisioning and consolidation of VMs in cloud DC based on two algorithms.

University of Malaya

Proposed algorithm decrease the consumption of energy and leads us to the optimal consolidation solution by adapting the migration concept at the service departure of VM. The first algorithm is designed as a bin packing problem with the aims of minimal power depletion. Moreover the performance of the proposed allocation based algorithm is compared with the best fit algorithm based on the energy parameter. Furthermore, the second algorithm based on the exact migration concept results based on the interwar formulation and linear programming in order to adjust the placement of VMs when resources are unconstrained. Proposed algorithm handles the number of migrations as well as the energy consumption based on the number of constrained and the set of inequalities. Moreover, this algorithm reports the minimum coverage time with the comparison of best fit heuristic. Besides, it does consider the CPU utilization while the allocation and consolidation of VMs.

The authors in (Bobroff et al., 2007) proposed the dynamic algorithm for the allocation of resources in the cloud. The objective of this algorithm is to minimize the running cost of DC. The cost is stated as the capacity of servers which is represented with the performance parameters named as overutilization and underutilization. The cost directly effects the performance of the application and leads to SLA violations. SLA is denoted with the response time factor of the applications and as well as the CPUs associated to that application. Moreover, with the perspective of business process spanning number of VMs are assigned with the assurance of CPUs associates to that VMs. Besides, the algorithm is characterized as measure forest remap (MFR). The MFR algorithm is the divided into three modules based on (i) historical data measurements, (ii) future demands anticipation, and (iii) VMs re-mapping to Physical machines (PMs). These three modules works in repetitive manners with the time interval t with the same sequence as represented with MFR. In addition, the proposed algorithm used the bin packing and time sequence predicting procedures in order to handle the workload to PMs. The performance gain of

University of Malaya

proposed algorithm is upto 50% when compared with the static algorithms by adapting the minimum SLA violations. In contrast, the limitation of the algorithm is that it works based on the probabilistic SLA assurances.

In this paper, the authors handle issue of server combination in virtualized DCs with respect to estimation schemes (M. Chen et al., 2011). The proposed framework is presented as a stochastic bin packing, where the servers limit and a permitted probability p is assigned based on the overflow criteria of servers. The objective of the proposed protocols is to allocate VMs to a number of PMs based on the associated load capacity to the servers. The proposed framework consider the effective VM sizing approach based on the stochastic optimization by correlating the dynamic load of VM based on the stable demands. Based on the principles of multiplexing the resource demand of VMs is decided using the proposed effective sizing algorithm. The proposed algorithm impacts on the aggregation of resource demands considering the multiple factors of hosts where the VM might be allocated. While considering the effective sizing, an algorithm is designed with the time parameter T for the VM allocation for the migration cost-aware and VM migration cost-obvious situations. The proposed algorithm showing the 24% better results when compared with the optimal solutions and enhance the energy saving upto 23%.

The authors in (Wood et al., 2009) presented the Sandpiper framework for detecting and identifying the hotspots, and reconfiguring/ remapping VMs when required. Sandpiper is explained as a framework that systematizes the undertaking of checking and distinguishing hotspots, deciding alternative representation of physical to virtual resources, resizing VMs to their new assignments to PMs, and starting any essential migrations. Sandpiper executes a black method that is completely OS-and application-rationalist and a gray box method that adventures OS-and application-level insights. Authors in proposed framework executes the schemes in Xen and lead a point by point assessment utilizing a network, memory and CPU intensive requests. The outcomes of Sandpiper demonstrate

University of Malaya

that it can determine single machine hotspots inside 20 s and scales well to large size DCs. The proposed work additionally demonstrate that the gray box method can assistance Sandpiper settle on more educated choices, especially with respect to the memory weight. While selecting that which VMs to relocate, Sandpiper migrate them utilizing a volume-to-size-ration (VSR) based on memory, CPU, and network load. Sandpiper relocate the most loaded VM from an over-burdened PM to one with suitable capacity.

2.3 Cloud Resource Management

A cloud computing infrastructure, whether single or federated cloud, is a complex distributed system composed of a multitude of computational resources. These resources handle the unpredictable client requests and the effects of external events beyond user and system administrator control. Cloud resource management significantly affects the performance, functionality, and cost factors of system evaluation. Cloud resource management also involves complex decisions and policies for multi-objective optimization. This task is challenging because of the complexity, geographical span, and unceasing and unpredictable interactions with the system, thereby making a precise global information state impossible.

Cloud resource management strategies related to the three delivery models of cloud, namely, PaaS, IaaS and SaaS, differ from one another. In all cases, the CSPs are faced with fluctuating, large workloads that challenge the claim of cloud elasticity. In some cases, when they can predict a workload spike, they can provide resources through advance reservation, e.g., seasonal web application may be subject to spikes.

For an unplanned spike, the situation is complicated. Auto scaling can be used for unplanned spike loads, provided that a monitoring system that justifies the decision to allocate, reallocate, or release resources on demand in real time exists. Auto scaling services

University of Malaya

are given by PaaS providers, such as Google App Engine . Auto scaling for IaaS is complex because of the lack of and the deficiencies in the available standards.

In cloud computing, whether single or federated, variation is unpredictable and frequent, and centralized management and control may be unable to provide uninterrupted services and functional guarantees. Thus, centralized management cannot support adequate solutions to cloud resource management policies.

Several problems should be considered while managing the resources in a federated cloud computing environment. In this section, we present a review of significant resource management techniques covering federated resource management functions, selection, and allocation.

University of Malaya

2.3.1 Resource Selection

The resource selection process finds a configuration that fulfills all user requirements and optimization of the infrastructure. The cornerstone of the resource or service selection is an optimization algorithm that considers all variables influencing the allocation. A general survey on selection solutions for federated infrastructures is presented in next section. Moreover, the summary of the reviewed resource selection schemes in a federated cloud environment is given in Table 2.5.

Table 2.5: Summary of resource selection schemes in a cloud environment.

Schemes	Objective	Strength	Weakness
(Jaikar & Noh, 2015)	To support the dynamic load while selecting the best position for allocating the request to attain the better performance.	Minimize the cost with acceptable performance	The failure index and energy consumption index of data centers are not included in decision making.
(Fan, Yang, Perros, & Pei, 2015)	To select trustworthy cloud services for cloud users.	Feedback driven trust basis.	The granularity of the historical data for decision making is not considered so that outdated history does not impact in decision making.
(Farokhi, Jrad, Brandic, & Streit, 2014)	To automatically select infrastructure services for SaaS provider such that the SLA claims of the SaaS provider for their customers are captured.	Cover functional and non-functional parameters of Inter-Cloud SLAs.	No SLA violation detection and penalty in case of violations.
(Gutierrez-Garcia & Sim, 2013)	Automating the service selection in the presence of incomplete information about cloud providers and their services.	Constantly changing consumer's needs are captured.	In the case of service migrations maintaining the agents and can be a difficult problem.
(Son, 2013)	To select the best provider in term of cost and requirements of the user.	The system components are pluggable with other programs and not depended on each other	The systems need human interaction to populate the database of the candidate CSPs and their characteristics.
(Vilutis, Butkiene, Lagzdinyte Budnike, Sandonavicius, & Paulikas, 2013)	Selection of suitable cloud in inter-cloud of computing services when there are no relevant resources available in the public and private cloud.	Minimizing the number of test tasks for quantifying the CSPs.	Networking factors are not used in the selection of a CSP.
(Sundareswaran, Squicciarini, & Lin, 2012)	To simplify and increase the speed of searching the CSP database for the best vendor selection.	100 times faster than brute-force search algorithm.	No opportunity for users to negotiate some terms of the SLAs.
(Jrad, Tao, & Streit, 2012)	To find the most worthy CSPs in order to fulfill the user's service requirements non-functional and functional SLA parameters.	Handling the interoperability and heterogeneity.	No Experimental evaluation to show its efficacy compared to existing schemes.
(Sim, 2012)	How software agents are employed for cloud resource selection in a multi-cloud marketplace where consumer selects the best cloud provider based on their utility function.	Concurrent negotiating agents for the best SP selection.	Changing user requirements are not captured.

University of Malaya

2.3.1.1 Review of resource selection based strategies for cloud environments

An index structure was designed by Sundareswaran et al. in (Sundareswaran et al., 2012) based on B+-tree (Bayer & Unterauer, 1977) to simplify a process of information insertion and retrieval for CSPs. In the proposed structure, different properties, such as security, service type, pricing units and measurement, and QoS had precise position to be considered and stored. Service vendors with the same characteristics should be stored together in adjacent rows to increase the speed at which the information management operators are executed and appropriate vendor queries could be found. The researchers also proposed a query algorithm based on a designed structure to search the provider database for the best vendors. The proposed architecture was compared with a brute-force search algorithm and showed almost 100 times better execution speed for solving the cloud computing service composition problem with 10,000 service providers.

A high-level generic brokerage architecture to find the most worthy CSP fulfilling the service requirements of the user in terms of non-functional and functional SLA parameters was proposed in (Jrad et al., 2012). The proposed architecture integrated a brokerage-based technology for assisting the user in SLA negotiation and finding the best provider for his service needs with respect to specified SLA.

The way software agents are employed for cloud resource/service selection in a multi-cloud marketplace, where the consumer selects the best cloud provider based on their utility function, was explored in (Sim, 2012), (Sim, 2008), (Sim, 2006). In their work, they proposed a negotiation protocol based on Rubinstein's alternating offer protocol (Rubinstein, 1982) and a negotiation strategy based on the functions of time, opportunity, and competitiveness for multiple consumer–broker agents negotiating simultaneously. Furthermore, they proposed service capability tables (SCTs) to store their services and the cloud agent's list. The coordination of self-organizing participants in a multi-cloud environment for automating a service selection in the presence of incomplete information about CSPs and their resources was investigated in (Gutierrez-Garcia & Sim, 2013). To handle this problem, a collection of two agent-based distributed problem-solving techniques, namely, SCTs and a semi-recursive contract net protocol, was integrated and devised into agent behavior to cope with (i) service selection based on dynamic services fees and (ii) incomplete knowledge about the existence and location of service providers and the cloud resources they offer. An agent-based cloud service composition testbed was implemented to

University of Malaya

support persistent, one-time, vertical, and horizontal cloud service compositions. Mechanisms to update and create service compositions based on constantly changing consumer needs were designed using self-organizing agents as building blocks.

In (Son, 2013), a resource selection decision maker (RSDM) was presented. The proposed decision maker listed the suggested resource providers and their resources by analyzing user demands. Users initially provided their requirements for the cloud service. According to these requirements, the RSDM retrieved all resources that match the requirements from a cloud information database. Once all candidate resource providers and their service types were retrieved, the estimated price was calculated. After the calculation of prices for each provider and service, the provider list was recorded by price and given to the user. Each item of the recommended list comprised information, including the resources to be allocated, name of the cloud provider, contract period, service type, and expected price. The selection of a suitable CSP in the inter-cloud of computing services for fulfilling the user task when no relevant resources are available in the public and private clouds was analyzed in (Vilutis et al., 2013). The Quality of Grid Services QoGS (Wickremasinghe, Calheiros, & Buyya, 2010) method was selected to determine the appropriate CSP. However, the QoGS method works appropriately only if the correct set of weighted coefficients (SoWC) is elected. Therefore, an algorithm was designed for selecting the best SoWC. Experimental results showed that the proposed methodology minimized the workload of inter-cloud by test tasks significantly.

The selection of cloud resource in a federated cloud environment was divided into two subproblems by (Jaikar & Noh, 2015), namely, DC selection and physical machine selection. The DC collection played a vital role in improving the performance and reducing the cost. An algorithm for selecting a DC in a federated cloud computing environment was presented. The approach was validated using a Cloud Analyst toolkit (Wickremasinghe et al., 2010). Results described that the DC selection algorithm offered considerable per-

University of Malaya

formance gains with respect to throughput, cost, and response time.

A SLA-based hierarchical service selection was presented for multi-cloud environments in (Farokhi et al., 2014). In their efforts, the authors adopted the idea of the algorithm presented in (Yau & Yin, 2011) and developed it to support service selection for a cloud composite service and to cover all the functional and non-functional parameters of inter-cloud SLAs. The architecture and phases involved in the selection process were based on prospect theory to evaluate the infrastructure services on the basis of the given SLAs and the degree of user satisfaction. The evaluation and a comparison of the utility-based matching algorithm showed that the approach effectively selected a set of services for the composition that satisfied SLA parameters.

Based on multi-attribute trust value evaluation cloud service selection was studied by Wenjuan et al. in (Fan et al., 2015). Their trust value estimation was based on two trust's characteristics, namely, reputation-based trust and perception-based trust, in which the trust facts were recorded on the trust reputation base and value base. Users could obtain the trust facts from the two bases and then apply the evidential reasoning approach to achieve the final trust results. After the service users used the service, they would give their feedback evaluation to the cloud system, which would be stored in the trust value base and reputation value base for other users to generate the indirect trust evidence. In the proposed framework, the trust value was produced from both the personalized indirect trust evidence and direct trust evidence, which was reliable with the service's requirement of users.

Table. 2.6 lists the extracted performance metrics from the federated resource selection schemes.

University of Malaya

Table 2.6: Performance Metrics for resource selection in federated cloud

Scheme	Cost	QoS	Load-aware	Energy-aware	Feedback-Driven	Selection Method	Experimental Platform
(Jaikar & Noh, 2015)	Yes	Yes	Yes	No	No	Matrix Based	Cloud Analyst (Wickremasinghe et al., 2010)
(Fan et al., 2015)	No	Yes	No	No	Yes	Evidential Reasoning	Formal Analysis
(Farokhi et al., 2014)	No	Yes	No	No	Yes	Prospect Based	Custom Java Based
(Sundareswaran et al., 2012)	No	Yes	No	No	No	Search based	Custom C Based
(Gutierrez-Garcia & Sim, 2013)	Yes	No	Yes	No	No	Negotiation	JADE (Bellifemine, Bergenti, Caire, & Poggi, 2005)
(Son, 2013)	Yes	No	No	No	No	Search based	CloudSim (Calheiros, Ranjan, Beloglazov, De Rose, & Buyya, 2011)
(Vilutis et al., 2013)	No	Yes	Yes	No	No	Weighted Rank	Not mentioned
(Jrad et al., 2012)	No	Yes	No	No	Yes	Simple Match Making	CloudSim (Calheiros et al., 2011) + OCCI4Java (Liaqat et al., 2017)
(Sim, 2012)	Yes	No	No	No	No	Negotiation	JADE (Bellifemine et al., 2005)

University of Malaya

2.3.2 Resource Allocation

Resource allocation is integral for obliging unpredictable resource requirements and capital return in cloud federation. In the context of federated clouds, application developers can lease resources in a pay-per-use manner from multiple geographically distributed CSPs to minimize the cost and SLA violation, and to enhance the application availability and fault tolerance. In addition, a summary of several resource allocation schemes is presented in Table 2.7 and Table 2.8 lists the performance metrics extracted from the federated resource allocation schemes. Besides, a general review of several resource allocation and scheduling strategies for federated cloud environments is presented in next section.

Table 2.7: Summary of resource allocation schemes in a cloud environment.

Scheme	Objective	Strength	Weakness
(Hassan, Hossain, Sarkar, & Huh, 2014) (Hassan, Song, & Huh, 2011)	To meet end-user QoS and economies of scale without increasing and enhancing a number of physical resources.	Maximizing the total utility of the federation.	The problem with this approach is that they model the resources as a single type.
(Woo & Mirkovic, 2014)	To study the benefits of allocating components of a distributed application on multiple public clouds.	Search based technique to meets the set of SLA constraints and achieve the performance and cost constraints.	The search time in case of a very large federation of cloud and large workflows.
(HoseinyFarahabady, Lee, & Zomaya, 2014)	To handle the scaling out of cloud resources while executing CPU-intensive applications with the non-proportional cost to performance ratios multi-cloud environment.	Fully polynomial-time randomized approximation algorithms for the task with known and unknown running time.	The scheduling decision does not take into consideration the fault tolerance.
(Zuo, Zhang, & Tan, 2014)	To provision the user's tasks in order to enhance the revenue of IaaS provider while satisfying QoS.	A high-quality scheduling solution by adaptively updating strategies.	The computational complexity of the proposed technique.
(Papagianni et al., 2013) (Papagianni et al., 2013)	Networking and computing resources are jointly optimized and treated for dynamically allocating virtual resources to physical resources within cloud network.	Handling both resource mapping and link mapping.	Overlooks the dynamic heterogeneous infrastructures and environments.
(Palmieri, Buonanno, Venticinque, Aversa, & Di Martino, 2013)	To take into account the possible contradiction between the interests of service provide r and client in the cloud.	This scheme had great benefits in a huge cloud organization which have maximum number of nodes with a wide varaity of tasks to be served.	Unpredicted situations and deviations in the environment.
(Di, Wang, & Chen, 2013)	To manage the social competition relations among resource contributors and consumers where everyone satisfied with its payoff.	Polynomial time complexity to find the best solution and ex-post incentive compatibility.	The fault tolerance and security mechanism is not discussed.
(Ardagna, Casolari, Colajanni, & Panicucci, 2012)	To handle the unpredictable workload fluctuation while guaranteeing SLA constraints by the coordination of multiple geographically distributed clouds.	Capacity allocation and load redirection distributed algorithms acting upon two angles, time scale, and workload prediction.	They did not study the capacity of the running instances while decision making and also overlooked the network latency.
(Ai, Tang, & Fidge, 2011)	To handle the deadline-constrained scheduling and resource allocation problem for multiple web services.	The algorithm depends on the credit assignment and the collaborator selection strategy method making it highly adaptable.	Only one process can be assigned to a machine at a time.

University of Malaya

2.3.2.1 Review of resource allocation and scheduling strategies for cloud environments

In (Malet and Pietzuch, 2010), middleware for cloud management was presented to migrate part of user's services (represented by number of VMs) among DC in order to manage the workload at the DC and to minimize total response times. Based on the monitoring of workload in DC, the middleware initiated VM migration to shift the components of application closer to the customer. The proposed approach was mainly designed for multiple DCs under a single cloud provider, but the formulation was still worthwhile for multiple cloud providers.

Authors in (Ai et al., 2011) considered the deadline-constrained scheduling and resource allocation in a hybrid cloud environment for various composite web services. The authors also took into account the running cost because in a hybrid cloud, the in-house private cloud resources were cheaper than their counterparts from public clouds. They proposed a cooperative coevolutionary genetic algorithm (CCGA). In the proposed CCGA, the co-operation among populations occurred while evaluating the individual's fitness (resource component) in a subpopulation. In population, the fitness value of specific entity was an estimate of how well it cooperated with different classes to generate good results. The populations worked cooperatively in order to explain the crises which are guided by the fitness value. This communication among the populations involved a selection of greedy collaborator and the credit assignment based on the fitness value. The performance of their algorithm depended on the credit assignment and the collaborator selection strategy method. However, only one process could be assigned to a machine at a time, thereby preventing the system from using many configurations.

The game-theoretic resource allocation scenario was studied in a federated cloud environment in (Hassan et al., 2014) (Hassan et al., 2011). The authors considered a horizontal dynamic cloud federation (HDCF), in which various CSPs cooperated dynamically for expanding their infrastructure capacity to meet end user QoS to gain economies of scale and to handle heterogeneous cloud resource demands without increasing and enhancing a number of physical resources. In this prospect, the authors proposed game-theoretic cooperative/non-cooperative price-based centralized and distributed resource allocation strategies to ensure that this horizontal cooperation was beneficial for each CSP. The authors formulated two resource provisioning games (i.e., cooperative and non-cooperative) in HDCF platforms to enhance the utility of the federation, specified as the total of the

University of Malaya

CSP utilities of the buyers. The authors applied a direct search method with multiple startup guesses to determine the best price (Kolda, Lewis, & Torczon, 2003). In both games, a buyer CSP publicized and defined the total amount of virtual resources provided. On the basis of this information, each seller CSP then updated its own strategy to maximize its utility. They concluded the existence of non-unique equilibrium states that yield an undesirable outcome under a non-cooperative game through experimental and theoretical analysis. Under the cooperative setting for resource allocation, the game was scalable and cost effective. The problem with this approach was that it modeled the resources as a single type.

The unpredictable workload fluctuation in a cloud computing environment to reduce the allocation costs in terms of VMs to meet the SLA constraints was studied in (Ardagna et al., 2012). They applied the open queuing model for modeling the coordination of multiple cloud providers operating in geographically distributed sites and presented a solution based on capacity allocation and load redirection distributed algorithms acting upon two angles, namely, time scale and workload prediction employing a closed-loop queuing technique with nonlinear optimization models. Furthermore, they investigated and demonstrated their solutions to be close to the solutions found by an oracle with perfect information about future workload. The workload prediction improved content locality. Aside from the comprehensiveness and rigorous evaluation of the work, the proposed method considered only the response time of server and did not take network latency into account. They also did not study the capacity of the running instances while making the decision.

(Di et al., 2013) considered the allocation of resources in a fully distributed self-organizing cloud (SoC), such that both resource consumers and contributors were satisfied with their prior results based on the declaration of their resource. In SoC, each host was deployed with an autonomous VM monitor and a resource state collector to act both as resource

University of Malaya

contributor and task scheduler. They constructed a VM with resource distribution concept from the execution nodes to optimize the efficiency of task execution and proposed a novel next-price bidding double-sided strategy based on the traditional second-price bidding to achieve ex-post incentive compatibility. For the resource query protocol, a random index diffusion strategy was adopted to reduce network traffic on query-message propagation. In this study the resource provisioning problem is devised as a convex optimization problem considering task characteristic, user budget demand, and resource availability. A polynomial time algorithm, local optimal VM resource allocation, was also designed to locate the optimal solution. Moreover, the basic idea was to temporarily remove the resource availability constraint and then recursively tune the solution with the constraint of resource availability until a distribution that satisfies this constraint was found. However, the system might be compromised because of the fully distributed and self-organizing nature, given that the fault tolerance and security mechanism were not discussed but were suggested as prominent future directions.

Networking and computing resources were jointly optimized and treated for dynamic allocation of virtual resources to the physical resources inside networked clouds in (Papagianni et al., 2013) and (Papagianni et al., 2013). To manifest the joint resource allocation solution for a federation of cloud network, the federated cloud resource mapping problem was formulated as a mixed integer programming (IP), taking into account the cost efficiency objective, such that QoS for user requests was met. Link mapping was formulated as the corresponding multi-commodity flow problem. Subsequently, a heuristic methodology for efficient mapping of networked cloud resources on shared substrate was proposed. The proposed framework did not consider dynamic heterogeneous infrastructures and environment beyond the conventional Internet (e.g., wireless), which presented further issues owing to wireless environment (e.g., uniqueness of nodes, isolation, and coherence), the stochastic environment of the corresponding resources, and the challenges related

University of Malaya

with the existence of mobile nodes.

A fully distributed scheduling scheme was proposed in (Palmieri et al., 2013) for uncoordinated federated environment of cloud. The scheduling framework was based on self-organized and independent agents, which did not depend on any centralized control that covered NE solution, with the potential contradiction between the interests of service provider and client in the cloud environment taken into account. A high performance was gained based on the completion time. This study had reliable results in the huge organizations where the wide variety of complex tasks can be observed by using the efficient partitioning strategy.

The effect of multi-cloud over a single provider for allocating components of distributed application for a variety of realistic scenarios was studied in (Woo & Mirkovic, 2014). The distributed cloud application workflow was modeled as a sequence of transactions composed of micro-tasks. The resource allocation for a given application workflow was formulated as the problem of determining a set of resources from the multiple clouds that met the SLA, cost, and performance constraints. Subsequently, an algorithm for resource allocation was proposed to find the best allocation for components of the distributed application over the multi-cloud environment. The algorithm worked by considering all possible allocations for each transaction and selected the one that met the respective SLA, cost, and performance constraints. The algorithm exhaustively searched from the cheapest to the expensive solution until a viable allocation was found. This exhaustive search could lead to a scalability problem in case of a very large environment.

The issue of scaling out cloud resources while executing CPU-intensive applications with non-proportional cost to performance ratios was resolved for the cost-effective deployment of an application into multiple cloud environments by (HoseinyFarahabady et al., 2014). The authors indicated that the degree of performance gain had no strong correlation with the usage cost of cloud resources. They presented fully polynomial-time ran-

University of Malaya

domized approximation algorithms to enable the execution of bag-of-tasks with known and unknown running time spanning beyond the private system/cloud (i.e., hybrid cloud) by explicitly taking into account the cost efficiency, that is, the cost-to-performance ratio. Meeting the peak demand while preserving QoS proactive machine purchasing, cloud federation resolves the problem of achieving economies of scale for IaaS. However, the former is not economic, and the latter is difficult in practice. In (Zuo et al., 2014), an allocation framework of resource was proposed where the providers of IaaS could outsource their tasks to ECs when their own resources were not adequate to fulfill the user's demand. This framework did not require any inter-cloud formal agreement for federation of cloud. The key challenge was how to assign user tasks to enhance the revenue of IaaS provider while satisfying the QoS. In this study the problem was devised as an IP model and solved by a self-adaptive learning particle swarm optimization (SLPSO)-based scheduling mechanism for scheduling the inter-cloud resources. In SLPSO, each aspect of a particle represented a particle, and task as a whole represented the priorities of all tasks. This scheme could acquire a high-quality scheduling by adaptively selecting velocity updating strategies to update each particle. The scheduling approach could find the suboptimal or optimal allocation scheme of external and internal resources to greatly improve the profit of IaaS providers and maximize the quality of scheduling solution.

Table 2.8: Performance Metrics for resource allocation in the federated cloud.

Scheme	**Cost**	**Scalability**	**Load Balancing**	**Energy-aware**	**Makespan**	**Availability**	**Fault Tolerance**
(Hassan et al., 2014) (Hassan et al., 2011)	Yes	Yes	No	No	No	No	No
(HoseinyFarahabady et al., 2014)	Yes	Yes	No	No	Yes	No	No
(Woo & Mirkovic, 2014)	Yes	No	Yes	No	Yes	No	Yes
(Zuo et al., 2014)	Yes	No	No	No	No	No	No
(Papagianni et al., 2013) (Papagianni et al., 2013)	Yes	No	No	No	No	No	No
(Palmieri et al., 2013)	Yes	Yes	Yes	No	No	Yes	No
(Di et al., 2013)	Yes	No	No	No	Yes	Yes	No
(Ardagna et al., 2012)	Yes	No	Yes	No	Yes	No	No
(Ai et al., 2011)	Yes	No	No	No	Yes	No	No

University of Malaya

2.4 Performance Analysis Tool for Cloud Deployment

This section explain the OpenStack cloud architecture, and VM provisioning in cloud.

2.4.1 OpenStack Architecture

OpenStack is feature rich, simple to implement and has all types of characteristics like the private and public cloud. Based on interconnected services OpenStack is deployed as an IaaS solution. By using the API and web-based dashboard, and command line tools users can manage the network resources, storage, and control pool of resources throughout the datacenters (Corradi et al., 2014). OpenStack maintains the number of components installed independently with distinct APIs that controls the both computations and storage resources in order to facilitate the dynamic allocation of VMs. In addition, according to the need of cloud, installed components communicate and work with each other through the RabbitMQ protocol and RPC. OpenStack includes the number of services such as, Networking, Compute, Identity Service, Object Storage, Image Service, Orchestration, Block Storage, Telemetry, and Database. Further, Network is used to create the topologies of virtual network whereas Image Service known as Glance is designed for the image management. In addition, Dashboard represented s web-based GUI service is used to manage the starting/stopping of the VMs and for the configuration of tenants while Identity Service is installed for authentication. Moreover, Compute service provides the IaaS characteristics comparable to Amazon EC2. Compute service is represented as Nova and considered as a primary module of OpenStack. In addition, while assessing the most suitable hypervisor Compute service deals with the resource provisioning and life-cycle management of VMs as shown in Fig 3.1.

University of Malaya

2.4.2 Request Flow for Provisioning VM in Openstack

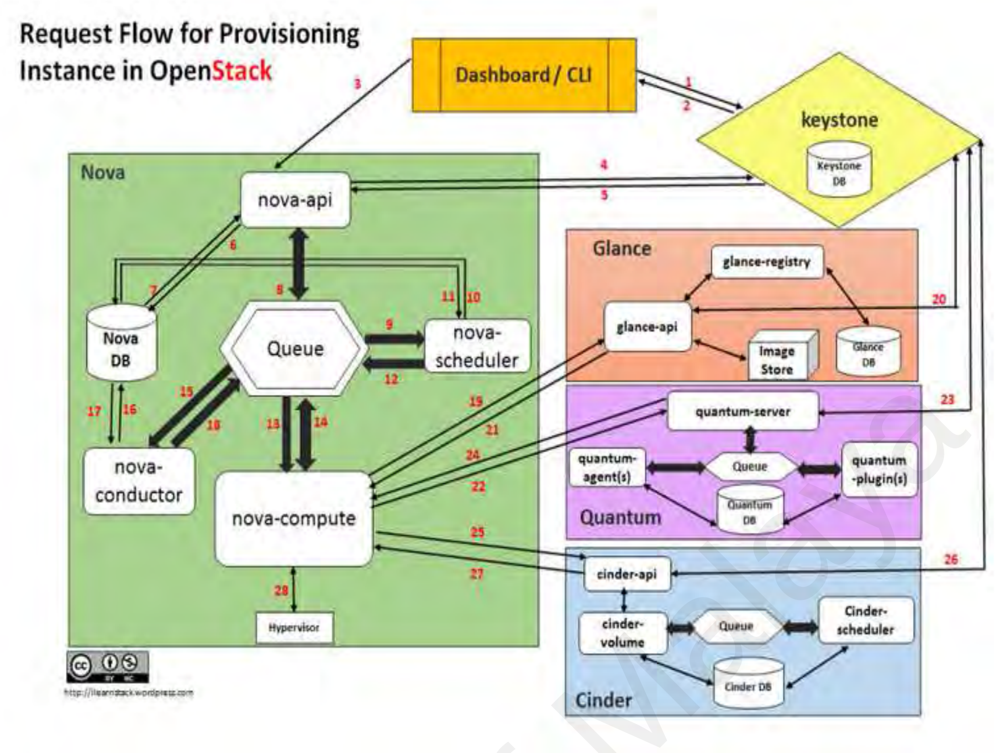


Figure 2.5: Request Flow for Provisioning of VM in Openstack

A provisioning of VM in any cloud is considered as one of the most essential use-case. This section discuss about the provisioning of VM in an OpenStack based cloud (Rosado & Bernardino, 2014). Moreover, this section describes the interaction of components and the request flow in order to boot a new VM under OpenStack. Based on section 2.4.1 the basic interaction of the services is defined whereas the current section elaborates the request flow for the deployment of VM as shown in Fig 3.1. A REST call is forwarded to the Keystone for the authentication process when the user credential is received at CLI. Upon the credential's authentication the Keystone sent back the auth-token which is further used for the communication with other components and for sending the requests using the REST-call. A new instance request is converted in the nova-boot or launch-instance form to REST API and forwarded to nova-api whereas it forward the request to keystone for access permission and validation of auth-token. Moreover, the updated headers with the permissions and roles are transferred by the keystone. Besides, nova-

University of Malaya

database and nova-api communicates with each other for the creation of initial record for new instance. Moreover, an rpc.call request is sent by the nova-api to nova-scheduler excepting to acquire updated instance record with specification of host ID (Wuhib et al., 2012).

A request from the queue is picked from the nova-scheduler whereas is communicates with the nova-database in order to choose the appropriate host by using the filtering and weighing methods. After going through the filtering and weighing process nova-scheduler return the updated information regarding the host Id and direct the rpc.cast application to nova-compute in order to launch the new instance on the appropriate host. In order to get the information regarding the flavor and host ID (CPU, RAM, Disk space) nova conductor receive rpc.call request from nova-compute and picks that request from queue. Furthermore, after picking up the request from queue nova-conductor coordinates with nova-database and sent back the instance record to it. In addition, nova-compute picks the information of instance from queue and send the REST call in order to attain the image URI with the ID of image from glance by passing the auth-token to glance-api and upload the image from image storage. Further, auth-token is validated by the glance-api with the keystone. Nova-compute get the metadata of an image and send REST-call to Network API by passing the auth-token in order to configure and allocate the instance with the private/ public IP address whereas with the keystone the auth-token is validated by the quantum-server. Nova-compute get the information of the network and send the REST call to Vloume API for the attachment of volume to the instance. In addition, auth-token with keystone is validated by cinder-api and the block storage information is provided to the nova-compute where it creates the data for hypervisors and execute request on hypervisors via api or libvirt.

In cloud, VMs are arranges and launched within physical machines. The libvirt library is used by the OpenStack in order to establish the contact with virtual operating

University of Malaya

systems those are deployed and running on physical nodes. In addition, a set of hypervisor independent APIs is represented by the libvert which is used to manage, enumerate, and monitor the VMs that are executing on physical machines. Moreover, physical server utilization is achieved by using the interface of an operating system (/proc) whereas through the libvirt library the VM utilization is obtained. OpenStack Nova assists administrators to deploy one or multiple hypervisors in virtual environment so that VM initiation and termination is facilitated based on different quires and performance metrics to VM load indicators. OpenStack support the XenServer, KVM, QEMU, ARM, and different types of hardware architectures (Ammal, Kumar, Alka, & Renjith, 2015). The XenServer and KVM are the most suitable choices for most of the use cases although KVM provides the solutions of full virtualization on x86 architectures. In addition, LXC technology is also used in order to offer the efficient solution with minimum overhead of virtualization. In addition, Openstack offers the characteristics of live migration with the condition when NFS is used for network storage and KVM is used as a hypervisor. Besides, in case of QEMU hypervisor CPU resources are multiplexed, which is currently used in our implementation. Further, through QEMU the differentiation among VM is achieved by assigning the different priorities which are assigned to the process that launches the VMs. Moreover, based on OpenStack scheduler, the controller node decides which compute node is suitable for the deployment of specific VM based on its associated resources.

University of Malaya

2.5 Issues and Challenges

Several problems should be considered while managing the resources in a cloud computing environment. In this section, we present a taxonomy of significant resource management issues covering resource management functions, such as pricing, discovery, selection, monitoring, and allocation as presented in Fig 2.6.

2.5.1 Resource Selection

Selecting worthy resources from a federated resource set is difficult because of the different requirements relevant to the provider, the high algorithm complexity, and dynamicity. The selection process should also consider behavioral aspects to maintain a user satisfaction level.

The dynamic changes in the resource utilization of the resources in a federated environment and the changes in the workload characteristics turn the resource selection into an iterative repetitive task considering user-specified functional and non-functional constraints. However, the task is difficult and has the following open issues, which should be addressed to make the resource selection in a multi-cloud environment feasible:

1. Monitoring data should be integrated as historical feedback to judge the credibility and effectiveness of the resources to be considered for selection.

2. Workflow modeling, breakdown, and mapping should be enhanced to utilize the full potential of the federated environment by selecting resources from the federated resource pool for individual components of the workflow according to some constraints.

3. Resource selection mechanism should consider the combination of networking factors and failure and energy indexes.

4. The effect of the selected resources on the utility of the CSP in the federation should be evaluated to verify whether this selection really improves the utility in the long run.

University of Malaya

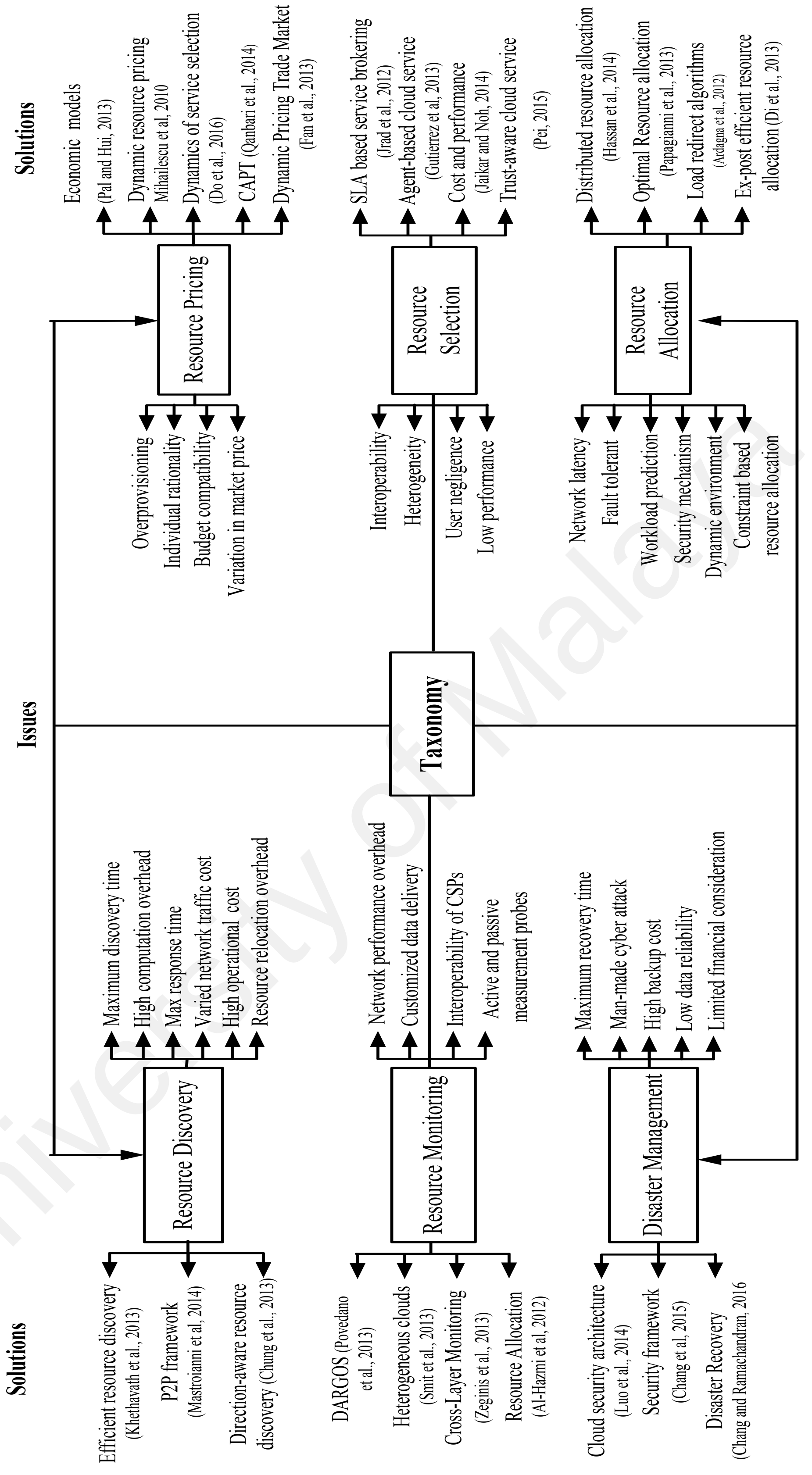

Taxonomy
Issues
Solutions
Resource Discovery
Maximum discovery time
High computation overhead
Max response time
Varied network traffic cost
High operational cost
Resource relocation overhead
Efficient resource discovery (Khethavath et al., 2013)
P2P framework (Mastroianni et al., 2014)
Direction-aware resource discovery (Chung et al., 2013)
Resource Monitoring
Network performance overhead
Customized data delivery
Interoperability of CSPs
Active and passive measurement probes
DARGOS (Povedano et al., 2013)
Heterogeneous clouds (Smit et al., 2013)
Cross-Layer Monitoring (Zeginis et al., 2013)
Resource Allocation (Al-Hazmi et al, 2012)
Disaster Management
Maximum recovery time
Man-made cyber attack
High backup cost
Low data reliability
Limited financial consideration
Cloud security architecture (Luo et al., 2014)
Security framework (Chang et al, 2015)
Disaster Recovery (Chang and Ramachandran, 2016
Resource Pricing
Overprovisioning
Individual rationality
Budget compatibility
Variation in market price
Economic models (Pal and Hui, 2013)
Dynamic resource pricing Mihailescu et al, 2010
Dynamics of service selection (Do et al., 2016)
CAPT (Qanbari et al., 2014)
Dynamic Pricing Trade Market (Fan et al., 2013)
Resource Selection
Interoperability
Heterogeneity
User negligence
Low performance
SLA based service brokering (Jrad et al., 2012)
Agent-based cloud service (Gutierrez et al, 2013)
Cost and performance (Jaikar and Noh, 2014)
Trust-aware cloud service (Pei, 2015)
Resource Allocation
Network latency
Fault tolerant
Workload prediction
Security mechanism
Dynamic environment
Constraint based resource allocation
Distributed resource allocation (Hassan et al., 2014)
Optimal Resource allocation (Papagianni et al., 2013)
Load redirect algorithms (Ardagna et al., 2012)
Ex-post efficient resource allocation (Di et al., 2013)

Figure 2.6: Taxonomy of Cloud Resource Management Issues

University of Malaya

2.5.2 Resource Allocation

Resource allocation is integral for obliging unpredictable resource requirements and capital return in cloud federation. Considerable work on resource allocation mechanisms for federated clouds is being conducted, but these mechanisms still need improvements. From analysis, we find that controlling the effect of reconfiguration cost over the federation and cloud utility is not studied. Another shortcoming is the lack of generality to make the allocation scheme visible for all type of service provisioning. Several open issues required to be addressed to achieve efficient resource allocation mechanisms.

1. Interoperability among virtualization engines, such as VMware ESX (Zhang, Denniston, Baskakov, & Garthwaite, 2013), KVM, and Xen should be investigated to realize seamless flow of data between their local applications and across clouds.
2. VM migration across subnets and realization of maintaining network flows by decoupling should be analyzed (Kalim, Gardner, Brown, & Feng, 2013).
3. VM behavior modeling and workload of a federated cloud environment should be evaluated to realize the peculiarities of the workload and VM.
4. Precise forecasting in a distributed and heterogeneous federated environment, where the common information among the entities is lacking because of the different administrative policies, is needed.
5. Security: Inter-VM attacks when a malicious VM is being migrated from a malicious provider.

2.5.3 Resource Monitoring

The function of cloud resource monitoring is to provide wide monitoring information data about service management and infrastructure, such as access control, service elas-

University of Malaya

ticity, service billing, and SLA management (Bernsmed, Jaatun, Meland, & Undheim, 2011) (Carlini, Coppola, Dazzi, Ricci, & Righetti, 2011). Monitoring data about their running services deployed in federated clouds are provided to customers. In the federated environment, resource monitoring is important for CSPs to maintain the federation and fairness in the distribution of revenue generated by the cloud clients.

From our analysis of the literature on monitoring solutions for federated cloud environments, the capability of monitoring cross-domain services has been regarded as a privacy and security risk and monitoring as an attack tool (Ristenpart, Tromer, Shacham, & Savage, 2009). This fact leads to a no-production-level monitoring solution with a federation of cloud members having different business objectives and enterprise policies. The expensiveness of the monitoring solutions and their effects on the application QoS are not explored for most of the solutions. Open issues pertaining to the efficacy of resource monitoring in federated environments are listed below.

1. Standardization is lacking when logical or physical domain boundaries are crossed, and monitoring activities is a challenge because of vendor lock-ins and heterogeneous infrastructures and architecture.

2. Architectural standardization efforts should be made to standardize APIs for gathering monitoring data from CSPs.

3. Energy monitoring of federated CSPs should be conducted to encourage efficient, green cloud computing by scheduling and rescheduling an application according to the energy consumption index monitored to reduce the energy consumption.

4. Cross-domain data leakage of applications and the internal configurations of a cloud member need to be addressed to enable third-party monitoring and unified moni-

University of Malaya

toring of federated environments.

5. An autonomous monitoring tool for validation and performance measuring of heterogeneous application sets deployed in a federated cloud environment is required.

6. No monitoring data of single or federated cloud environment are publicly available, and no workload traces of the monitoring solutions themselves exist to analyze the data by statistical tools to acquire more insight into the monitoring process.

2.5.4 Resource Discovery

The resource discovery function describes how a CSP exposes its resources and service to enable other CSPs in the federation to find these resources and services for automating the resource selection process and ensuring easy use of services, thereby complying with requests. The responsibility of resource discovery in the federated environment is extended to handle physical and geographical proximities and the costly inter-domain traffic of resources.

The literature has indicated that the resource discovery solutions for federated cloud are mostly inspired by P2P systems given the autonomic nature of the federation members. Moreover, several of the discovery solutions employ a central brokerage for the discovery process. In addition to the issues of P2P resource discovery mechanisms discussed in (Meshkova, Riihijärvi, Petrova, & Mähönen, 2008), a few open issues pertaining to the efficiency of federated cloud resource discovery are listed below.

1. Controllable resource advertising protocols taking the peculiarities of the federated clouds should be designed.

2. The semantic description of resources should be standardized to enhance cross-domain discovery and interoperability.

University of Malaya

3. Runtime SLA negotiation in ad hoc federation should be empowered by first discovering and then negotiating, but this could be turned into an attack tool. Thus, a trusted third party can be involved, thereby leading to the use of resource discovery as a service.

4. QoS-differentiated resource discovery is an interesting aspect in a large federation with numerous users.

2.5.5 Resource Prizing

In federated cloud computing, consumers leverage various types of computational and storage resources and services from one or more than one resources or service providers using a fixed or variable (pay-per-use) pricing schemes. In federated clouds, consumers and suppliers of cloud resources are rational players and inclined toward maximizing their own benefits when contributing and utilizing shared resources and services (Nielson, Crosby, & Wallach, 2005). In a federated cloud environment, resource supply and demand fluctuate as consumers and providers join and leave the federation. Pricing function is subsequently used to manage the individual rationality of the consumers and providers. The oscillation in the workload and resource availability in the federation is confined to the need of dynamic pricing strategies in federated cloud based on the principles of demand and supply.

From analysis of the different federated resource pricing schemes, the association between SLA and pricing model is not clear, and incomplete information exists about the resources of the federated environment. We present the most recent available pricing models to qualitatively measure their applicability and relevance. However, most of them have a bias and do not work in improving the overall utility of the federation. In addition, an individual model does not fit all potential scenarios because of the varying nature of the business objectives and enterprise policies of the federation members. Our analysis

University of Malaya

also indicates that the functions and features of auditing and accounting are insufficiently included in these models because of the distributed administration of the federation to follow up legal requirements. Several open issues listed below still need to be resolved.

1. The effect and overhead of pricing model on the multi-tier hierarchy of the federation of CSPs should be evaluated.
2. QoS-differentiated pricing schemes are not yet considered for federated setup.
3. In case of ad hoc federation with no prior agreement among the federation members, a malicious member that reveals untruthful resource prices to the marketplace can comprise the efficiency of the pricing model.
4. Pricing model should consider the workflow characteristics of composite web services.
5. The effect of utilizing business intelligence and integration services should be investigated to analyze the federated cloud marketplace and price predictions for handling misreporting resource bidding functions (Chang, 2014) (Chang, Walters, & Wills, 2012).

2.5.6 Disaster Management

Disaster management and fault tolerance play an important role in restoring organizational data in case of natural hazards or man-made disasters. Disaster management functions enable a system or component to continue normal operation despite hardware/software failures or compromises. In scenarios of federated cloud environment disaster, management functions should be distributed and coordinated among each node of the federated setup to enable a micro-level disaster-aware federated cloud infrastructure, which ensures the QoS level required by members of the federations.

University of Malaya

Considerable literature is available on handling the DR issue in federated cloud environments. The primary site becomes unavailable when a disaster happens, and the secondary site has to be activated. In this case, in a backup site no sync or async replication ability exists, but system and data states can only be locally stored. This phenomenon is a serious threat to the system, yet it is temporary and will be removed after recovery of the primary site. However, all risky situations should be considered to attain the best DR solutions, especially in high-availability services (such as business data storage). Several open research issues are briefly stated below.

1. In order to provide true business continuity for a DR service, it must assist seamless reconfiguration of the network for an application once it is brought online in the backup site.
2. Synchronizing the in-memory intermediate states is part of the DR process to save computation rather than data.
3. The cost of DR mechanism should be analyzed to identify which DR mechanism is more suitable and does not reduce the net system utility.
4. The time required to detect a failure strongly affects the service downtime and initiation of a DR process. However, while replicating across multiple mirror sites, the problem is how to differentiate between network failure and component failure.
5. As mentioned before, DR can be be human made or it can formed by nature. A cyberterrorism attack is a human-made disaster that is accomplished for many reasons. In this case, recovery and protection of essential data will be the major objective in DR plans aside from system restoration.

University of Malaya

2.6 Discussion on Cloud Load Balancing

Virtualization is the most adopted power management and resource allocation technique used in cloud computing infrastructure and data centers. Virtualization in network domain does not provide for energy efficiency. In effect, network resources are burdened by the virtualization techniques. Live migration of VMs in the data center is an active area of research as data has to be transferred from one physical host to another, generating a significant amount of traffic (Voorsluys, Broberg, Venugopal, & Buyya, 2009). The live migration of a VM essentially requires the copying of VM memory pages from the current location to a new location across the network while the VM does not stop its services at the current location. The pages that are modified during the process of live migration are marked as dirty and have to be re-transferred after the first iteration of the copy.

Nowadays, power consumption within cloud data centers is a big challenge due to high power consumption by DC equipment because of hosting and deploying high resource demanding applications within data centers. Server consolidation is a mechanism that packs maximum possible VMs on a single server so that rest of the servers can be switched off to minimize power consumption budget. Moreover, applying dynamic voltage frequency scaling (DVFS) also helps to minimize power consumption budget. However, decisions about where to place services while keeping in mind customer's location and needs is a big challenge that needs significant attention to surge data center performance.

Lightweight VM migration design will help to minimize the resource usage of VM migration process as lightweight design uses low system resources. Incorporating lightweight design feature in current VM migration schemes will help to accelerate DC performance. Moreover, proposing three dimensional queuing modeling based approach can be proposed to effectively highlight the objective and constraints of VM migration technology.

In virtual data center infrastructures, the goals of energy efficiency and resource utiliza-

University of Malaya

tion arise along with the problem of non-optimized placement of VMs on different physical hosts (Chernicoff, 2009). The non-optimal placement of VMs results in two VMs with large mutual (VM-to-VM) traffic being placed in different network domains with multiple-hop distances. The VM-to-VM traffic consumes a significant part of available network bandwidth. Energy-efficiency, higher resource utilization and optimal VM placement can be achieved by VM live migrations. The main disadvantage of live migration is that it can consume significant network bandwidth during the process of VM memory image transfer from one physical host to another.

Transferring large sized data over the shared network link is a big challenge, especially when several goals in terms of SLA violation avoidance, minimum end to end delay, high throughput, and high service quality has to meet. To improve energy efficiency, network management policies employ visualization technology to fully utilize the peak capacity of existing resources. Server consolidation methods collocate the most potential VMs based on (a) memory similarity ratio, (b) network communication pattern, (c) degree of workload on the server, and (d) flexibility to security concerns, to switch off the idle servers such that the total power budget is minimized. State-of-the-art VM migration schemes suppress VM contents using de-duplication, compression, write throttling, and various innovative ways (workload enabled compression) to efficiently utilize bandwidth capacity. However, applying all these optimizations consumes significant amount of system resources which ultimately affects co-hosted application performance in terms of SLA violation. For effective resource utilization, VMs are packed on a few servers. However, the decision about co-location is affected by the type of workload hosted within VMs, CPU capacity, memory availability, and communication pattern of VMs. Degree of SLA violation is increased if infeasible VMs are co-located. Therefore, it is must to decide which VMs should be co-located. Parameters, such as, application profiling and statistical analysis helps in identifying the most suitable VMs.

University of Malaya

Overlooking load balancing establishment abruptly decreases system throughput due to overloaded servers and ultimately leads to SLA violation. Within the cloud, effective resource management is very important, as cloud resources are not infinite in reality. To manage resources within the cloud, underlying resources are shared fairly among a set of users. VM migration is one of the processes that effectively manage cloud resources by ensuring load balancing, fault tolerance, server consolidation, and in-time service maintenance provisioning. However, VM migration itself is a time and resource consuming activity and it impacts the performance of co-hosted applications. Co-hosted applications share the underlying resources to optimally utilize existing resources. However, in co-hosting multiple VMs, the challenge with isolating performance is very big. Also, co-hosting affects SLA if resources are not fairly distributed among legitimate users based on their requirements. Moreover, privacy and security concerns are also present due to VM co-location.

University of Malaya

2.7 Conclusion

Virtualization is a technique that allows the sharing of one physical host among multiple VMs, where each VM can serve different applications. The CPU and memory resources can be dynamically provisioned for a VM according to the current performance requirements. This makes virtualization perfectly fit for the requirements of resource allocation and management in data centers. This chapter extensively review the resource allocation schemes based on load balancing techniques. Load balancing schemes are further classified into static and dynamic load balancing. A detailed taxonomy based on load balancing characteristics is derived in this chapter in order to attain the various objectives such as, efficient utilization of resources, cost effectiveness, fair allocation, resource prioritization, scalability and flexibility. Moreover, this chapter discusses the issues related to the cloud resource management based on, resource selection, resource allocation, resource monitoring, disaster management, resource discovery parameters. In addition, based on the literature review of existing cloud load balancing schemes the number of open research issues are discussed in detailed and thematic taxonomy is proposed with remarks in order to handle the issues.

Load balancing has become an integral part of all distributed internet based systems as distributed computing comes with the challenges of high resource demands that overload servers. Load balancer increases the capacity and reliability of applications by decreasing the burden on a server. Load balancer starts with identification of hot spot, an overloaded server, and start migrating its load on a server which has sufficient resources such that the resources are evenly distributed.

Based on the static load balancing schemas it is observed that majority of the algorithms do not incorporate the application execution time parameter and overlooks the CPU utilization at the time of VM deployment. In contrast, the CPU utilization is considered in dynamic algorithms based on migration techniques after the deployment of the workload on underutilized physical hosts. In order to stable the load the VMs are migrated after their placement which leads to the migration overhead. Based on analysis of existing static and dynamic schemes it is concluded that inefficient load balancing schemes maximize the application execution time when the CPU is overutilized and ultimately leads to performance degradation. However, the criterion of where, which, and how to migrate workloads from the physical servers pose challenges that cloud operator has to consider during all these decision makings. Therefore, based on analysis a solution is required that efficiently balance the load at the time of initial deployment of VMs and migrate the load from highly utilized server to under underutilized server to controls the total number of migrations.

University of Malaya

CHAPTER 3: PROBLEM ANALYSIS OF DYNAMIC LOAD BALANCING IN CLOUD THROUGH VIRTUAL MACHINE PLACEMENT

This chapter analysis the performance of existing static load balancing schemes. It discusses the methods, experimentation tools, and the test programs designed for the problem analysis for traditional load balancing methods. The objective of this chapter is to establish the problem, which is highlighted in Chapter 1. In order to show the severity of the problem we performed the in depth investigation of the problem. Therefore, in this chapter we conducted the set of experiments under the existing VM deployment conditions.

This chapter is divided into four main sections. Section 3.1 shows the experimental methodology, evaluation method, and test program design in order to conduct the experiments. Section 3.2 shows the performance analysis of existing load balancing scheme based on static and random based load distribution and illustrates the VM deployment in OpenStack cloud, load distribution analysis, and load distribution behavior study based on performance parameters in terms of core utilization, CPU utilization, and application execution time. Section 3.3 conclude the findings of this chapter.

3.1 Experimental Methodology

This section briefly discusses tools, test program designs, and highlights experimentation equipments for performing the experiments on the traditional load balancing methods. In this study, the experimentation is conducted on real hardware equipments to analyze the performance of VM deployment in cloud in terms of average CPU utilization and execution time.

The proposed study has considered a small in house data center to conduct the experimentation for problem analysis in traditional load balancing methods. We have selected the OpenStack cloud for experiments. During the experiments, a control environment is

University of Malaya

modeled to deploy VM using OpenStack scheduler. Moreover, it has used the traditional computers rather than expensive and powerful machines (small industrial cloud replica). For the analysis, we deployed our small OpenStack cloud infrastructure comprising of four physical machines connected through a flat-DHCP networking module. The configured servers are heterogeneous in terms of their resource capacity such as RAM, System cache, bus speed, and CPU. One of the servers is set responsible to act as a controller node with distinguished resource specifications in terms of, Xeon(R), Intel(R), CPU E5620 with 2.40GHz, and QEMU hypervisor as presented in Table 3.1. In first set of experiment the controller node installed using 32 GB RAM. Besides, for the rest of three servers called compute nodes carries 16 GB RAM capacity. Moreover, experiments are also conducted using the homogeneous RAM with the configuration of 16GB for each compute. However, the forth compute node is configured on the controller node.

During experiments, to analyze the behavior of traditional load balancing methods, the workload is generated for analyzing the CPU resource consumption rate. In the current study, to analyze the traditional static load balancing methods, CPU bound applications are designed to fully (100%) utilize the peak capacity of CPU within deployed VM. In order to analyze the CPU utilization behavior, all the unnecessary applications including users and background system were turned off prior to experiments. To analyze accurate CPU utilization all experiments are conducted with the 100% utilization of each deployed VM.

This study has performed each experiment 15 times to report the average of data for suppressing the noise due to uncontrollable system background activities such as garbage collection, dynamic voltage frequency scaling, and frequent context switching. The average of 15 times run is discussed in this chapter.

University of Malaya

Table 3.1: Physical Server Specification Profile

Capacity	Physical Server(s)
CPU type	Xeon(R)
Thread(s) per socket	2 with flavor id 1
Thread(s) per socket	4 with flavor id 2
Hypervisor	QEMU
CPU(s)	8
CPU Freq, (GHz)	2.40
Architecture	$x86-64$
Kernel	3.11.0-26-generic
L1 Cache (KB)	128
L2 Cache (MB)	1
L3 Cache (MB)	12

3.1.1 Evaluation Method

This study has conducted experimentation on a small in house cloud data center using the open source cloud platform (OpenStack) to analyze the impact of load distribution. To analyze load distribution among physical machines, the load is characterized as static or dynamic. To generate static and random load, it has considered a CPU bound application that executes the multi-core Python application to increase CPU load. In order to conduct the behavior of VM distribution in cloud, VMs are deployed using the 2 different flavors as presented in Table 3.2. For static load generation case, VMs are created using flavor ID 1 with the specification of 512MB RAM, 1GB Disk Space, and 2 vCPUs, 32 GB controller node (Edge1) and 16GB compute nodes names as, Edge2, Edge3, and Edge4, respectively.

For static analysis equal numbers of VMs are deployed on each physical server. Moreover, it has considered same CPU bound application (synthetic test program) for each vCPU to generate the workload. In contrast, for random load based VM distribution scenario, the VM deployment is considered using the flavor ID 1 and 2. While, using the flavor ID 1 all experiments are conducted using the same specifications as used in static

University of Malaya

scenario, but for the second time the experiments are conducted using the homogeneous RAM (16GB) for each compute and controller node using 2vCPUs. Moreover, random load based VM deployment behavior is also conducted using the 4vCPUs per VM with the 16RAM for each compute node. For the random load distribution, test program has developed an application that randomly generates workload of different capacity based on flavor ID. For simplicity and randomness, load generator's generated load is mapped between 0 and 2 random values when the VM is deployed with flavor ID 1. The number 0 specify that no CPU intensive application is executing on VM; whereas, values, including 1 and 2 states that only 1 and 2 cores are fully utilized. In addition, when the VMs are deployed with 4 Vcpus using the flavor ID 2 the random load generator function generates the value between 0 and 4. The value 0 shows that no application is executed on this VM; whereas, values 1,2,3, and 4, specify that 1, 2, 3, and 4 number of vCPUs are fully loaded for that VM.

In order to examine the CPU resource consumption rate, the design of proposed synthetic benchmark program is presented in the next section of this chapter. Moreover, to capture the system CPU consumption rate during synthetic benchmark execution within a VM, the proposed study has used top Linux utility. Top shows the CPU load that is average based on average of, current CPU load, last five minutes load, and last fifteen minutes load. In addition, load average is measured based on the number of processes that are waiting for their CPU turn for execution. The proposed study has used awk command to acquire the average load as captured by the top Linux utility.

Table 3.2: VM Configuration Profile

Flavor Type	**Flavor ID**	**Disk Space(GB)**	**RAM(MB)**	**vCPUs**
2Core_based_flavor	1	1.0	512	2
4Core_based_flavor	2	1.0	512	4

University of Malaya

3.1.2 Test Program Design

In order to investigate the performance of CPU utilization, a CPU bound test program is designed comprising the different size of loops performs the basic arithmetic operations. For instance, CPU bound test program is comprised of multiple nested loops along with the set of statements performing the arithmetic operation on the unsigned integers values. The name of the test program is chosen while considering the size of test program in terms of its loops count. Test program generates the workload to fully utilize the capacity of CPUs allotted to a particular VM. The test program is designed such that the VM utilizes 100% capacity of the CPU resources until the test program complete its execution. In order to generate the 100% load test program is executed based on the number of core of specific VM. For example, if the VM is deployed with the 2 cores the two test programs are executed for the fully utilization of CPU resources. Moreover, the size of the CPU-bound-test is associated based on the loop size within the program. For instance, in the 2000000K test program the body of the loop is executed for 2.0×10^8 number of times. We have designed the test program because there is no benchmark program available in order to check out the performance of OpenStack cloud.

University of Malaya

3.2 Performance Analysis of Existing Load Balancing Schemes

The core motivation of this study is to analyze the behavior of scheduler at the time of VM deployment or when it is shielded from any external influence. By using the initial set of experiments, the performance of each compute node is evaluated based on the launched VMs to analyze the CPU utilization rate based on the load factors.

3.2.1 Illustration of VM deployment in Open Stack

In OpenStack Nova scheduler (standard scheduler) is used for scheduling VMs to map the nova-API calls to suitable components. Schedulers took the decisions based on multiple factors such as memory, number of vCPUs, distance to the availability zone, and CPU architecture, etc. Moreover, OpenStack Nova is composed of three types of schedulers including, Filter, Chance, and Simple. Chance scheduler randomly chooses an available node regardless of its characteristics whereas Simple scheduler identifies the available node with the least load to deploy first VM. In contrast, filter scheduler is used for VM scheduling purpose, which maps the nova-api calls to appropriate component. Number of different filters are supported by the nova scheduler including the core filter, RAM filter, availability zone filter, disk filter, image-based filter, host-based filter, and net-based filter. In the resource based filter the decisions are taken based on the available resources in terms of memory, disk space, and number of CPU cores. The image-based filter decisions depends on the properties of image while the properties are describes based on hypervisor, architecture of CPU, and mode of VM including suspend, unsuspended, running, terminated, and created. In contract, in host-based filter hosts are selected using the grouping criteria while considering the availabilityzone, location, and host aggregate properties.

$$W = weight1\_multipl * norm(weight1) + weight2\_multipl * norm(weight2) + \ldots \tag{3.1}$$

In addition, the default scheduler used for the deployment in OpenStack cloud is known as the filter scheduler. At the deployment of VM filter scheduler creates a disk image for the VM and calls the hypervisor to boot the VM. The parameters to this call may include the type of the (virtual) vCPUs, the number of cores, the amount of memory, the hard disk image to boot from, and the local CPU allocation policy.

University of Malaya

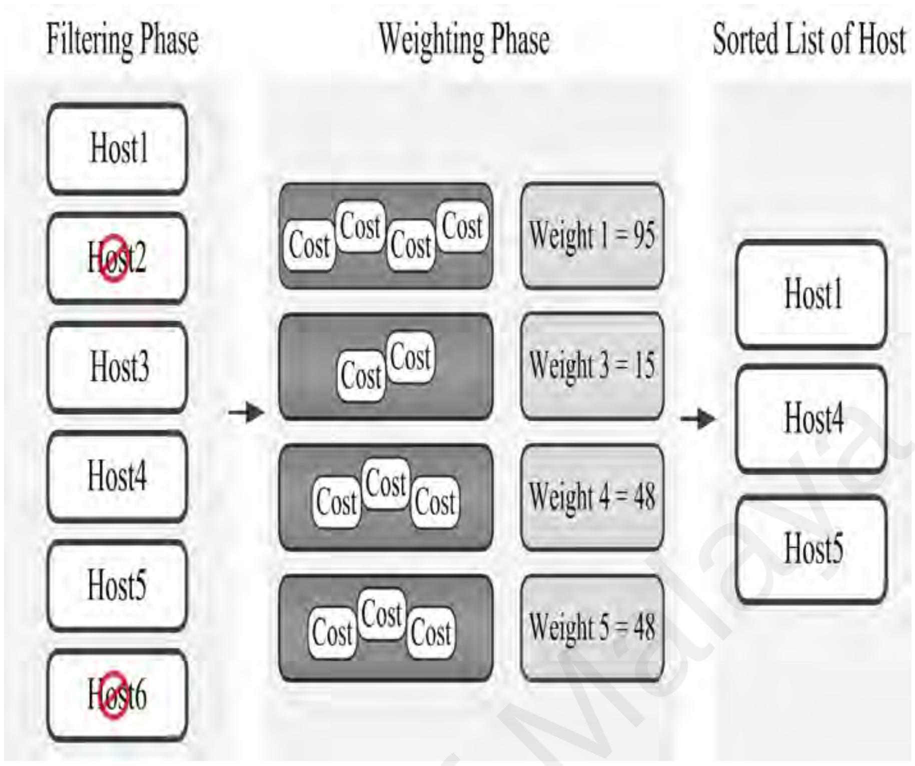


Figure 3.1: Selection of Hosts based on OpenStack Cloud Standard Scheduler

Filter scheduler contains two generic functions filtering and weighting as discussed in Fig. 3.1. In the filtering phase the capacity of the hosts is determined based on the requested resource. Filter function selects the set of compute servers capable of running a given VM (Litvinski & Gherbi, 2013). In Fig. 3.1 host2 and host6 are not considered for the deployment of next VMs based on their available resources. In the second phase, the weights are associated to each hosts that are selected using the filtering phase in order to choose the best one host. Alternatively, a cost function ranks the filtered set of servers according to their suitability as shown in Equ. 3.1. By default the weight is assigned to the hosts based on the available RAM capacity using the RamWeigher.

University of Malaya

Algorithm 1 Legacy Compute Scheduler Based on RAM Filter Algorithm

```
Require: self, $StateofHosts$, properties, InstanceType$_i$, Rar,
1: procedure HostPasses(self, $StateofHosts$, properties)
2:     Type$_i$ $\leftarrow$ p.get(InstanceType$_i$)
3:     $ReqRAM \leftarrow$ Type$_i$[memory]
4:     $FreeRAM \leftarrow getFreeRAM(InstanceType_i)$
5:     Rar $\leftarrow self.getRAMllocationRatio(StateofHosts, properties)$
6:     TotalUsableRAM $\leftarrow StateofHosts.TotalUsableRAM$
7:     UsedRAM $\leftarrow TotalUsableRAM - FreeRAM$
8:     return $TotalUsableRAM * FLAGS.Rar - UsedRAM >\doteq ReqRAM$
9: end procedure
```

When the VMs are launched using the RamWeigher it balanced the usage of the RAM among all hosts by deploying the other VMs as represented in Fig. 3.1. Moreover, the selection procedure of RamWeigher is presented in Algorithm 1.

3.2.2 Load Distribution Analysis

This section demonstrates the scheduling flow in open stack. To analyze the scheduling flow behavior, it appropriately places the VMs on the physical server based on CPU usage level.

Fig. 3.2 has presented the scheduling flow of VMs based on CPU utilization for each physical machine. In VM deployment model for the existing OpenStack compute scheduler, VMs are differentiated using distinct color schemes. The light-gray color shows that first eight VMs are deployed on Edge 1. The VMs with dark gray-color represents that VM is fully loaded when CPU bound application is executed. Alternatively, white-color based VMs represents that VMs are deployed without load. Based on CPU utilization fac-

University of Malaya

tor, Edge 2 represents the current CPU usage capacity, which is 97.9% while four VMs are hosted on it at time "t". Alternatively, Edge 3 and 4 represent 75.8% and 52.3% CPU usage statistics at time "t".

Considering Fig. 3.2, for legacy non-optimized OpenStack scheduler, when 21st VM is deployed (edge 2) it is placed on the highly loaded physical host based on spread technique criteria to balance the RAM.

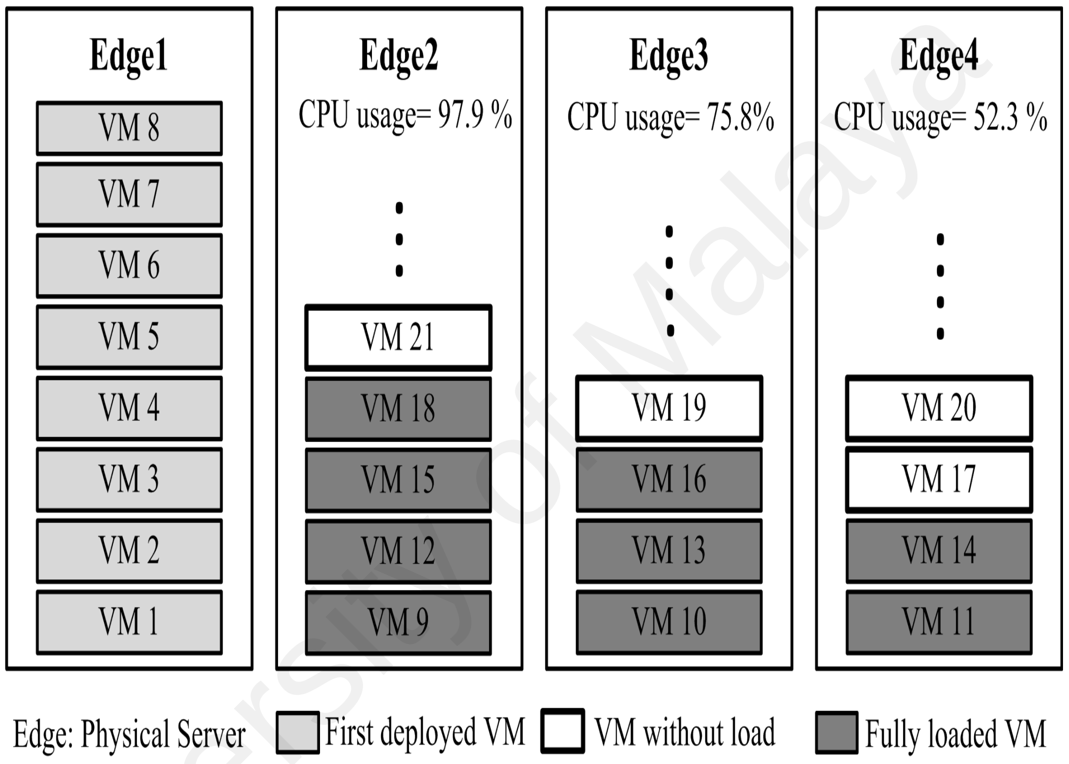


Figure 3.2: VM Deployment Using OpenStack Cloud Scheduler

3.2.3 Load distribution Behavior Study

This section discusses the load distribution behavior for the legacy cloud scheduler based on the static and random load distributions.

3.2.3.1 Static Load Based VM distribution

In this section a discussion on analyzing the relation between number of core's utilization and VM deployment sequence for four physical hosts (Edge1, Edge2, Edge3, and Edge4) is presented. Fig. 3.3 has presented the aforementioned study. In the said figure,

University of Malaya

x-axis shows the utilization of number of cores whereas the VM deployment is represented on the Y-axis. As can be seen from the Fig. 3.3, the function of total number of cores utilization is highly dependent on the number of VMs deployed. Also, it is shown that different cores are utilized on different physical machines. In order to highlight the deployment of VM on a specific physical machine for the clarity, the compute nodes are denoted with different types of shapes. For instance, Edge 1, Edge 2, Edge 3, and Edge 4, are represented with triangle, circle, diamond, and rectangle, respectively. Also, count of shapes for each particular instance of a shape is same because of the fact that every single physical carries equal 8 VMs.

Physical machines are configured to host 8 VMs as the number of cores allocated to each physical machine are 8. Therefore, in total there are 32 VMs deployed on compute nodes. In the here mentioned figure, the first plot has shown eight consecutive triangles. The consecutive 8 triangles plotted in Fig. 3.3 has shown that first 8 VMs are deployed on Edge 1 compute node. Every single VM is deployed to use 2 vCPUs as mentioned in the experimental details. In addition, it was noticed that once VMs has utilized 8 cores, the deployment sequence repeat its behavior for Edge 2, Edge 3, and Edge 4 nodes, respectively. Based on the behavior of Fig. 3.3 it was observed that compute node Edge 1 being having the highest RAM capacity is selected first for the deployment of VMs. Further, the repetitive sequence shows that the Edge 2, Edge 3, and Edge 4 have the same RAM 16 GB; as a result, the compute scheduler balanced the RAM utilization for each compute node.

University of Malaya

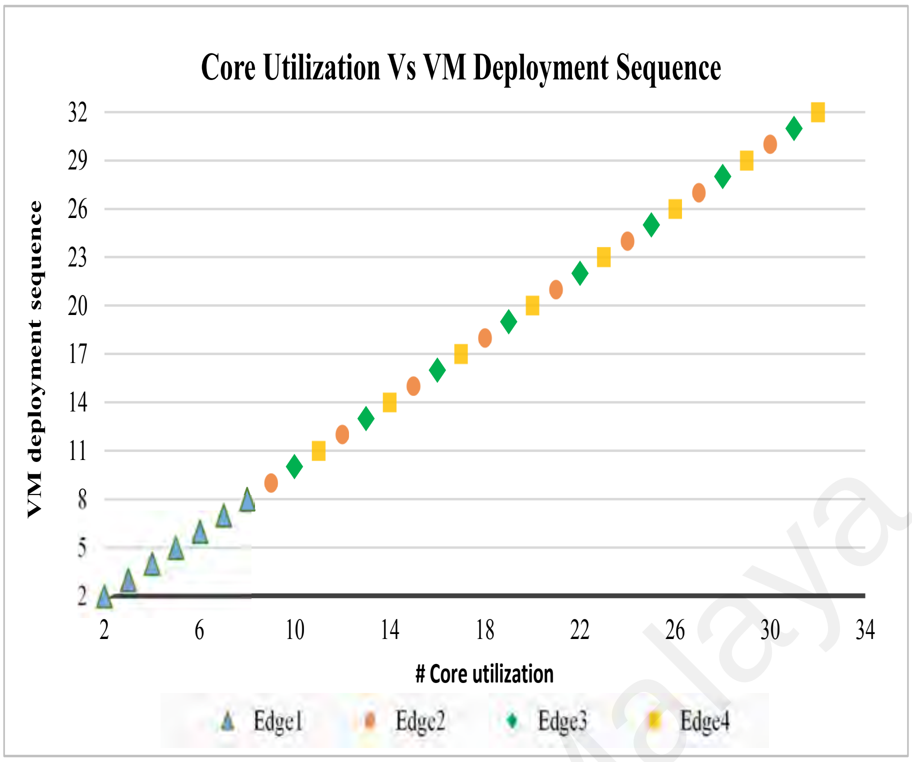


Figure 3.3: Core Utilization vs VM Deployment Sequence based on Static Load

The load or utilization of a physical server measures the extend to which a VM has used its resources. In terms of resource utilization, the most essential resource is represented as CPU. Performance degradation is highlighted in Fig. 3.3 when multiple co-located VMs compiles extensive computational tasks. In addition, Fig. 3.4 presents the number of VMs deployed on x-axis and CPU utilization across that deployment on y-axis. The performance of each single VM is measures by running a CPU intensive application to impose load upon a particular vCPUs. In order to deploy the same load, same program is executed for each VM. When the first VM is deployed and load is generated, the CPU utilization ratio is increased up-to 25% which is represented as 0.2% in the aforementioned figure. When the second VM is deployed and the same CPU intensive application is executes on same physical machines, the load value is surges to 28%. Moreover, the graph shows that the Edge 1 is selected again and again until it reaches to the 99% CPU utilization even though the other physical machines have minimum CPU utilization as compared to Edge 1. Furthermore, if the CPU utilization is too high, it

University of Malaya

seems that the VMs that are deployed on the given physical server are not receiving the required capacity of CPU resources and waits in order to accomplish their tasks. It is observed that during launching VMs, first eight VMs are created on Edge1 due to availability of sufficient RAM capacity. From ninth to onward, all VMs are launched on Edge 2, 3, and 4 (physical hosts/ compute nodes) in a sequence because of spread technique (nova-scheduler) that evenly distributes the VMs in order to manage the RAM. It was also observed that vCPUs load surges to 25%, 50%, 75%, and 99% when the number of VMs are increased on each single physical server. Moreover, the value from 9 to 32 at x-axis shows that there is a specific deployment sequence which is followed for other compute nodes with the increased CPU load value.

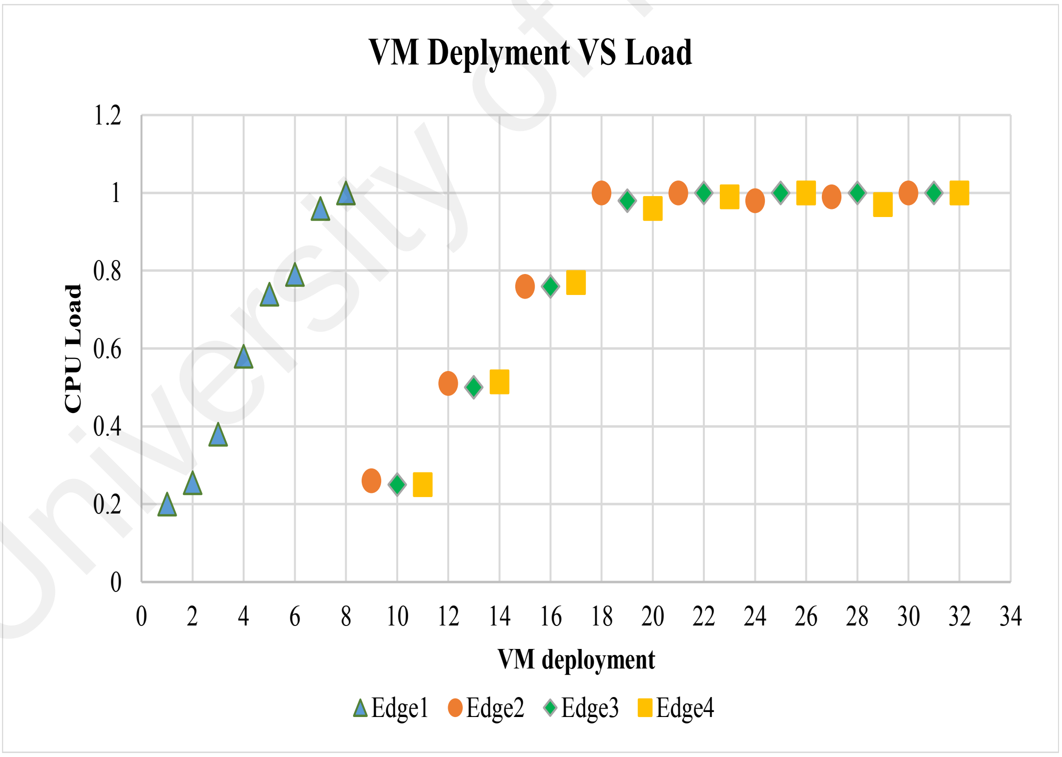


Figure 3.4: Analysis of VM deployment vs CPU Load based on Static Load

3.2.3.2 Random Load Based VM distribution

This section analyzes the deployment behavior of VMs using OpenStack while considering its legacy scheduling method. In order to differentiate the deployment from the static

University of Malaya

load based scenario, this study deploys VMs based on random load generator function. In this scenario, a random program is generated to predict the load. For the better understanding, the name of the program is designated as the random load generator function. At the deployment time, the specification of the VM shows that each VM is deployed using the 2 vCPUs as shown in Fig. 3.1. Therefore, based on number of vCPUs, the random load generator function generates the discrete numbers between range 0-2 for each VM for generating the load on each VM as described in section 3.1.1. Fig. 3.5 presents the sequence of VMs deployed on the physical servers. The physical servers are highlighted with different shapes to differentiate them. In order to present the distribution of VMs on the physical servers based on dynamic load balancing, the graph is plotted between core utilization and virtual machine deployment. In the said figure, the VM deployment sequence presents the true deployment of VMs on the physical servers. During VM deployment, firstly, VMs are deployed on Edge 1 node as it carries the maximum RAM capacity compared to the remaining three physical servers. Moreover, in the said figure, the scheduler followed the same deployment sequence as presented in Fig. 3.3 except the decision for node selection for VM deployment. For instance, in Fig. 3.3, the scheduler choose the sequence of Edge2, Edge3, and Edge4 for VM deployment. In contrast, in the current scenario, the order of physical server selection changes due to random function based scheduling that has chosen Edge4 and Edge2 for the initial set of VMs deployment. The physical machines repetition is again followed for the random load based on the RAM available RAM balancing criteria.

University of Malaya

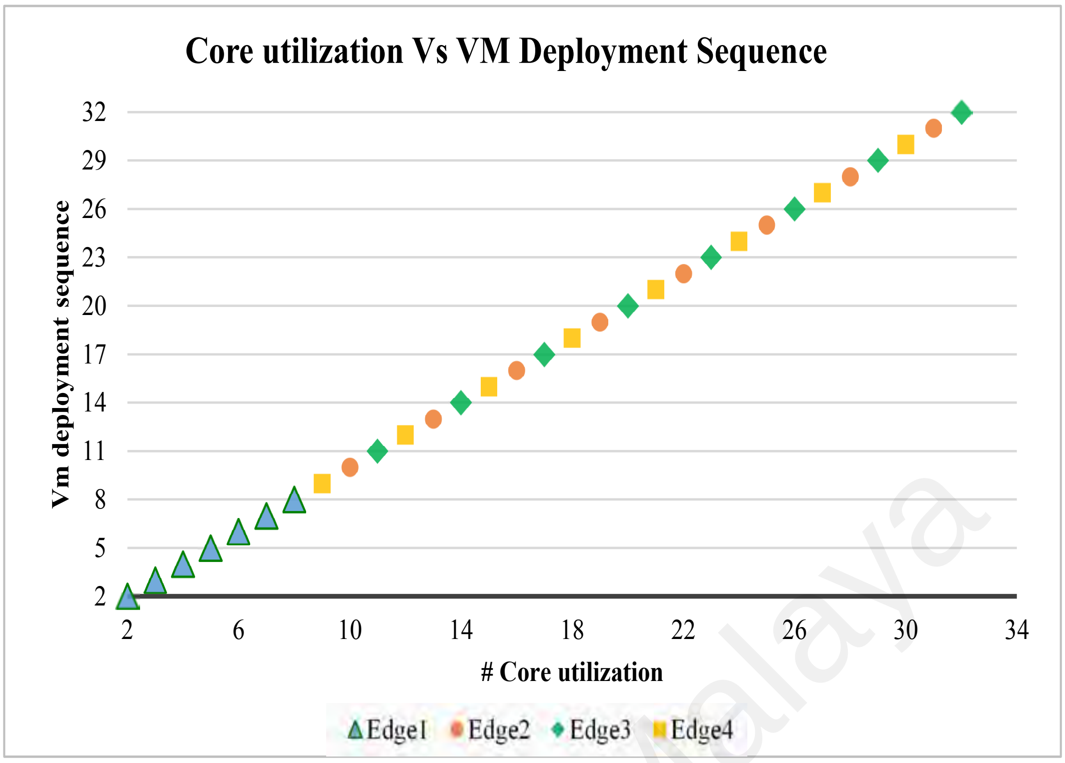


Figure 3.5: Core Utilization vs VM Deployment Sequence based on Random Load

Based on VM deployment sequence as presented in Fig. 3.5, Fig. 3.9 has shown the CPU utilization behavior. The CPU utilization rate is estimated by generating the load on VMs using the load generator function. It was noticed that when Edge 1 is selected for the first VM deployment, the CPU utilization was observed 0.2% for 2 test programs executing for each core. For static balancing case, for the second VM deployment, again Edge 1 is selected based on RAM filter. For the second VM, it only runs one program and it was noticed that the CPU consumption surge to 0.25% of total CPU capacity. Moreover, after deployment of 2nd, 3rd, and 4th VMs, the CPU utilization approaches to 0.6%. Besides, the 6th and 7th VMs are deployed and the generation of load on single core and two cores, respectively. As a result, the CPU utilization is reached to the 0.8%. In addition, for the deployment of 8th VM, the Edge 1 is decided by the compute scheduler even though it is showing the maximum utilization of CPU as compared to other PMs in cloud.

For better understanding, based on OpenStack scheduler, first eight VMs are deployed on Edge 1, which has increased the aggregate CPU usage capacity to 95.90%. At the time of

University of Malaya

deployment of the ninth VM, the load on the Edge2 is noticed reaching to 25.2%. Moreover, for the Edge 3 and Edge 4, the load rises to 20.1% and 25.6%, respectively when the tenth and eleventh VMs are deployed based on scheduler selection criteria. Moreover, 12th, 13th, and 14th VMs are deployed on Edge 2, 3, and 4, respectively. The Edge2, and Edge4 reaches to their highest CPU utilization as compared to Edge3. Based on random load generator function, the Edge3 has shown the minimum load compared to the other nodes because at the time of deployment of 14th VM, no core was utilized. At the deployment of fifteen VM based on placement criteria, VM is deployed on the maximum loaded host without considering the CPU utilization as shown in Fig. 3.5 and Fig. 3.9. The behavior of the deployment is showing the scheduler is not considering the CPU utilization except RAM and the repetitive selection criteria of PMs.

University of Malaya

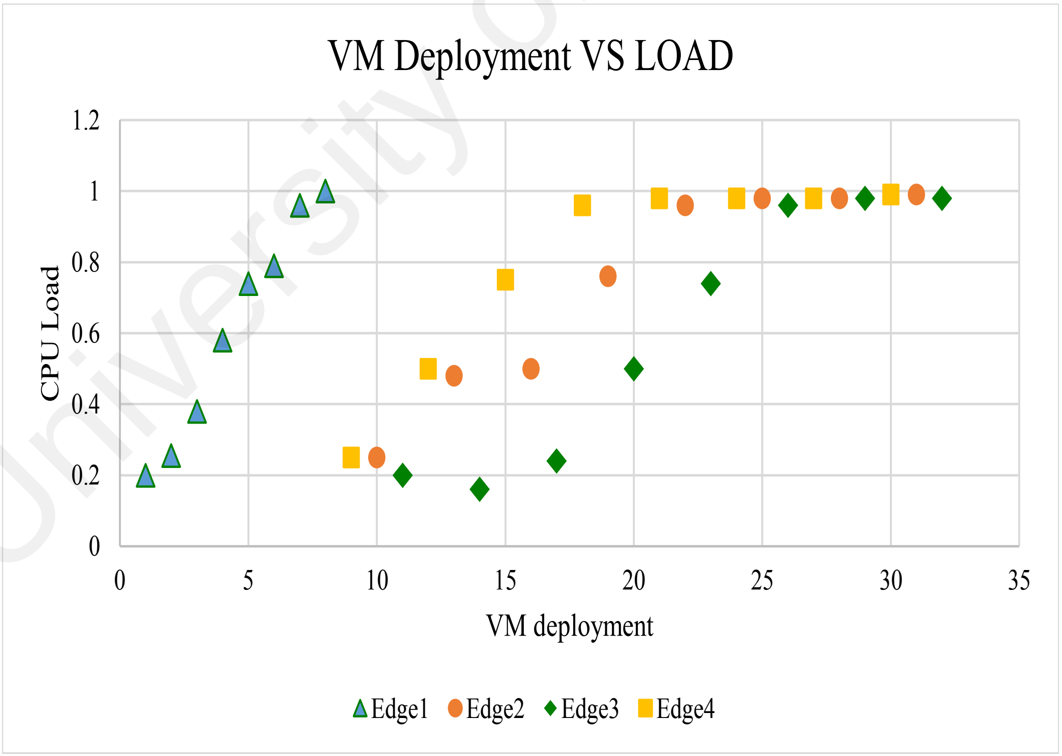


Figure 3.6: Analysis of VM deployment vs CPU Load based on Random Load

First two scenarios as presented in Fig. 3.3 and Fig. 3.5 shows the deployment sequence based on the deployment of VMs using the 2 vCPUs per VM. In order to highlight the deployment with the dynamic load, the VMs are deployed with the homogeneous RAM and with flavor ID 2. The behavior of VM distribution is presented in Fig. 3.7. For the first four deployed VMs, the random load generated function has generated the 2, 0, 2, and 1 values. Based on the above mentioned values, deployed VMs have executed the CPU intensive programs. For instance, first VM placed on Edge 1 fully loads the CPU by executing the CPU intensive program on 2 vCPUs. The second VM on Edge4 is showing 0.4% usage of CPU when there is no program is executed. Edge3 and Edge2 are showing the 25.4% and 15.2% of CPU usage with the execution of 2 and 1 programs, respectively. For 5th, 6th, 7th, and 8th VMs, the load generator function has generated values based on 1, 2, 2, and 1 random umber as shown in Fig. 3.7. Moreover, the Edge3 has shown the same load on the deployment of VM 6, 7, and 8. The figure shows that the CPU usage is fluctuating between 55.6% and 55.8%. The fluctuation shows that the load generator function has randomly generated the 0 value for the Edge3 using its sequence though no CPU intensive program was executing for these VMS. The overall deployment behavior with the random load based characteristics illustrates that the CPU load is not considered while the deployment.

University of Malaya

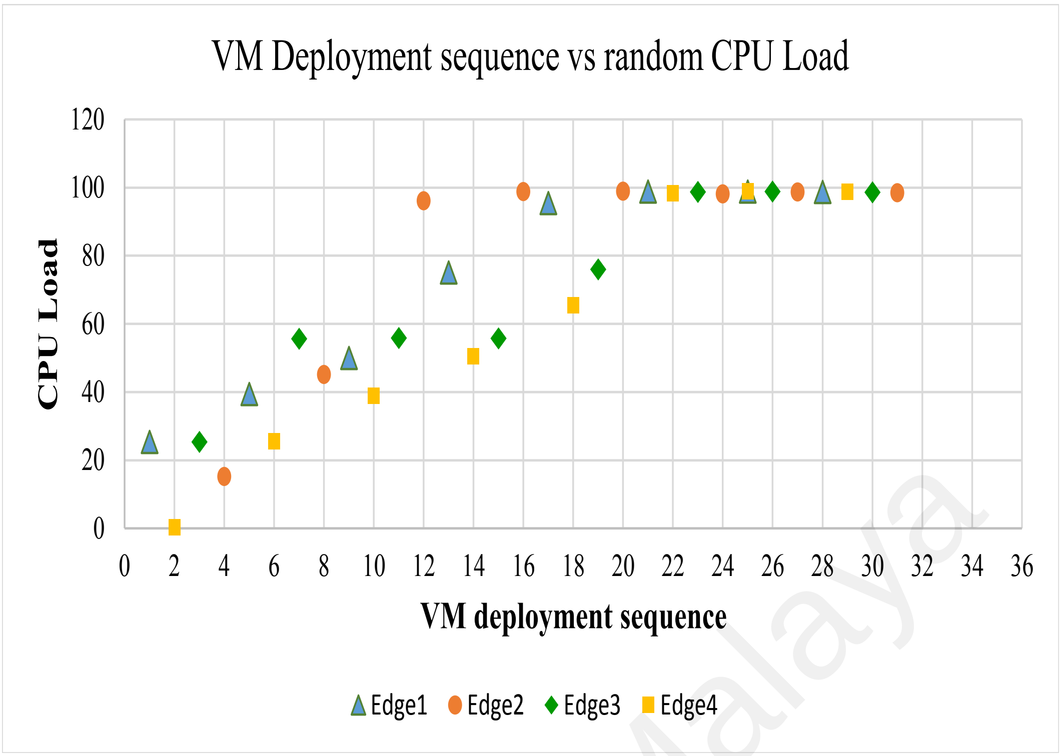


Figure 3.7: VM Distribution Behavior with Homogeneous RAM

University of Malaya

Fig. 3.3 and Fig. 3.5 shows that the first 8 VMs are continuously deployed on Edge1 node based on the maximum and RAM availability criteria. In contrast, Fig. 3.7 has shown that each compute node is selected cyclically with the same repetition. For the allocation of first VM, the Edge2 is selected where the VM is fully utilizing the 4 vCPUs and surges the CPU consumption up-to 50.5%. The second VM is deployed on Edge 1 and increases the CPU consumption rate to 50.5% of its total capacity. During the third VM placement on the Edge3, the random load generator function generates the 0 valu; therefore, no CPU intensive application is executed on it and the CPU utilization is 0.3% only. In addition, the forth VM is deployed on Edge4 and executes the CPU bound test programs based on random load generator function. Based on the distribution of first four VMs, the Edge3 node is showing the minimum CPU utilization. Among others, Edge3 has the least value. However, for the allocation of fifth VM, the CPU scheduler has selected Edge2 based on spread technique and overlooks the CPU utilization behavior. The figure has shown that rest of the VMs are deployed in a circular order.

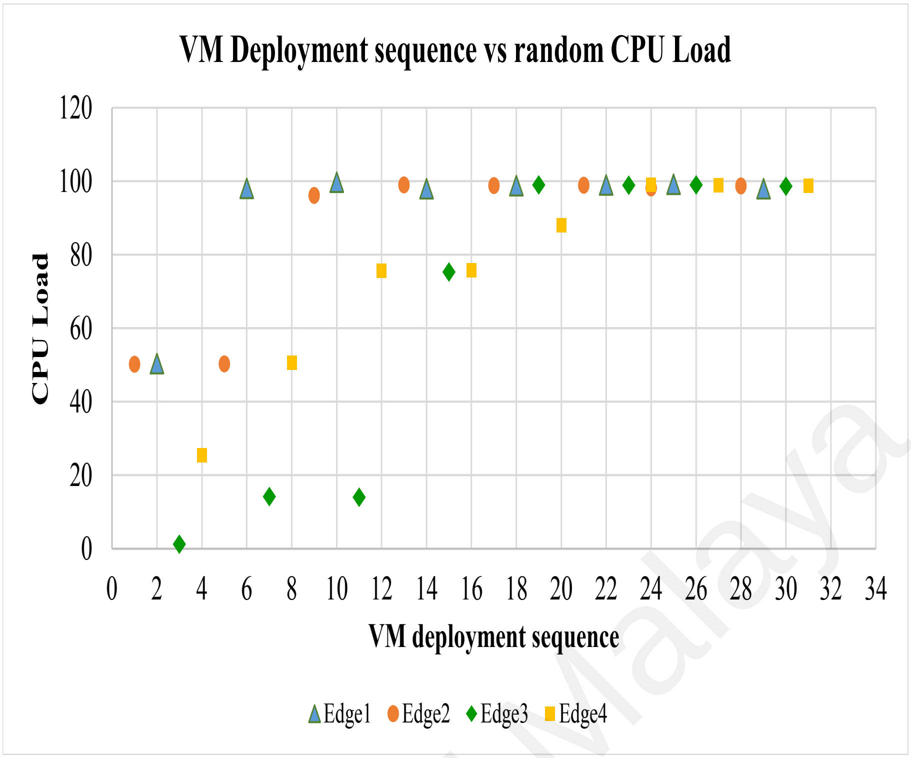


Figure 3.8: VM Distribution Behavior with Random Load and 4vCPUs

University of Malaya

3.2.4 Execution Time Analysis

This study analysis the application execution time for different CPU usage levels for one physical machine by varying the number of VMs as shown in Fig. 3.9. On compute node, firstly the program was executed without the deployment of VMs to acquire the actual time taken by the test program program. During second experiment, 1 VM was deployed using the 2 vCPUs on the single VM. In order to check its behavior, the VM was fully loaded while CPU intensive programs were executing for each core associated with VM. In order to check the behavior for 3 VMs based configuration, only three VMs were deployed with the CPU intensive programs. Moreover, as explained in experimental setup, physical machines have the 8 physical CPUs; therefore, we have checked the CPU behavior with the utilization of physical CPUs. It is observed from the said figure that when no VM is deployed, the program has finished its execution in 1400 rounds and it has utilized the CPU resources up-to 15%. Moreover, CPU usage reached up-to 25.5%, 51.0%, 75%, and 99.8%5, when the 1, 2, 3, and 4 VMs were deployed on physical machine. For 2 VMs, the same CPU bound test program was executed for 1675 rounds. In addition, when 3 VMs executed simultaneously, the execution time has increased with little bit change as compared to the time taken by 2 VMs. For 3VMs, the total completion time was completed in 1700 rounds. In addition, the execution time for 3 and 4 VMs was noticed approximately similar; For instance, the time was noticed 1715 and 1720, respectively.

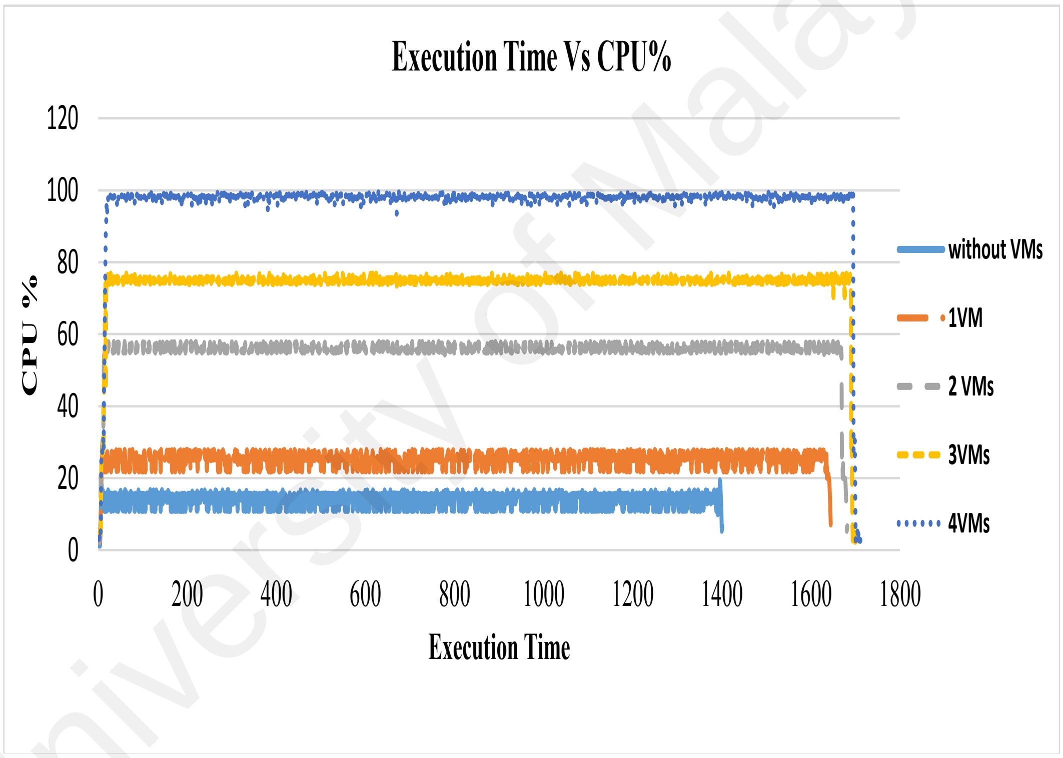


Figure 3.9: Application Execution Time vs CPU Usage with Different VMs

In second study, the experiment was conducted to relate execution time of an application to its CPU utilization for each compute nodes. The Fig. 3.10 shows the results based on application execution time on each VM for single server while Fig. 3.3 has shown the behavior for each compute node in the cloud. The analysis is conducted based on static load distribution, as shown in Fig. 3.3 in order to check the performance of compute scheduler. The figure based on Edge1 has shown the maximum execution time due

University of Malaya

to the availability of RAM as nodes deployed on Edge1 shares the CPU resources until it reaches to the 100% utilization. Rest of the compute nodes with the 16GB RAM has shown the same execution time. In order to compute the total execution time of each compute nodes, the programs for each VM were started, instantaneously. The graph is showing the 1910 sec execution time for Edge1 when it fully loaded with the distribution of 8 VMs whereas for the Edge2, Edge3, and Edge4 it has shows the same time when it was fully loaded with the execution time of 1675 rounds.

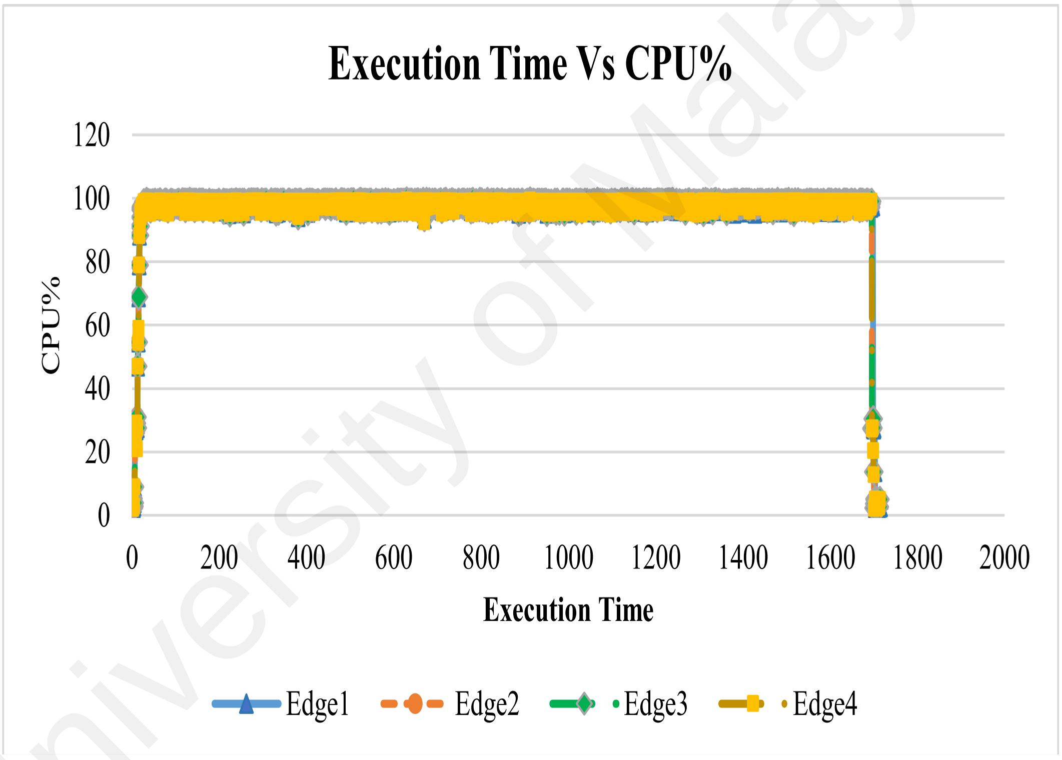


Figure 3.10: Execution Time Based On Static Load Based Distribution

University of Malaya

The experiment in Fig. 3.10 shows the execution time when physical machines are deployed when configured with the heterogeneous amount of RAM. Fig. 3.11 shows the execution time for each compute node based on the deployment sequence on the PMs as plotted in Fig. 3.7 which is measured in seconds. Based on the random load based distribution of VMs, Fig. 3.10 has depicted that each compute node has completed its processes on the same finishing time which were running inside the VMs. The behavior of the said figure has shown that each physical machine has same execution time in case of static and random load based scenarios when the RAM criteria is same for each compute node. For all the compute nodes the total execution time is calculated and was noticed up-to 1650 rounds.

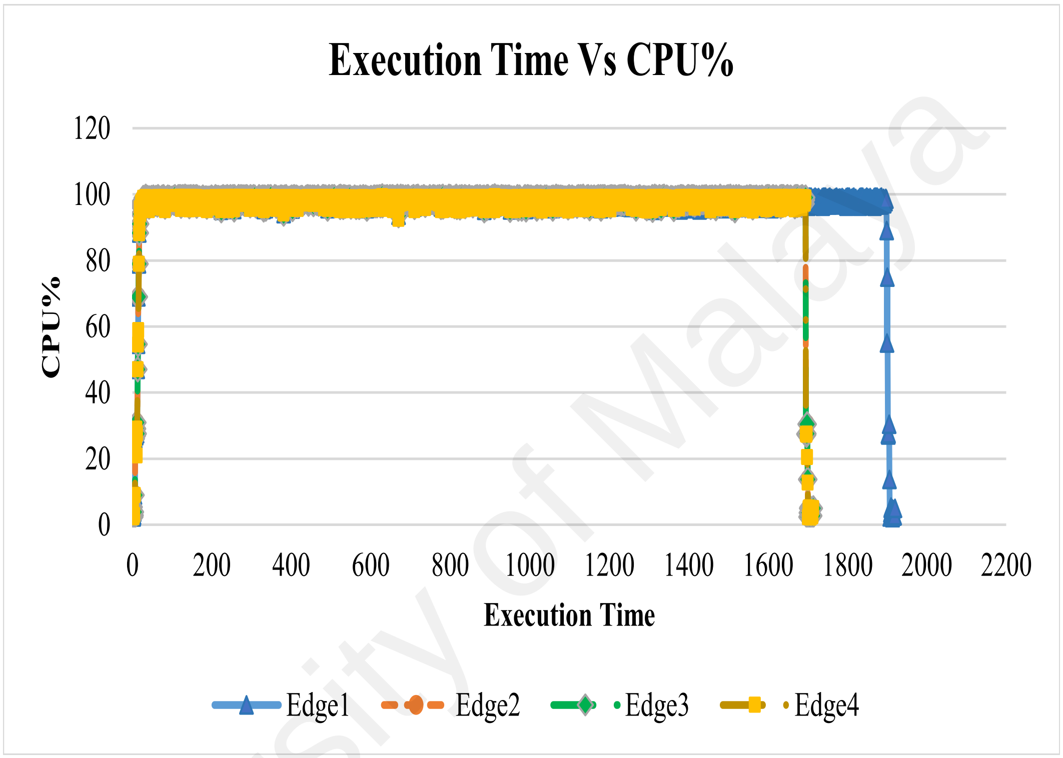


Figure 3.11: Execution Time Based On Dynamic Load Based Distribution

3.3 Conclusion

In this chapter, our focus is to critically analyze the OpenStack's scheduler for VM placement in controlled environment in perspective of small-scale private cloud using four compute and one controller node. In addition, the agency compute scheduler is an overall solution to allocate resources, and the multiple available filters give a comprehensive set of choices. However, filter only provides the reservation of requests based on vCPUs and memory parameters and generated the list of servers that are ready to use. Besides, they overlooks the performance maximization strategies. Based on the balanced load factor it is the weigher's responsibility of allocate the request to efficiently utilize the available

University of Malaya

resources.

The current strategy of only weighing against memory usage, by way of the Ram Weigher, is limited in effectiveness. This is because each VM has its own separate memory space, and random-access memory performance is not largely impacted simply by memory consumption. Further, the OpenStack OS base, namely Linux, always harnesses unused memory for caching and performance improvement. In contrast, if a VM is launched on a host with heavy CPU utilization, the VM performs poorly. On a compute heavy host, contention for CPU time slices results in the VMs on the host enduring a performance penalty. The key problem is that the current weighting strategy is weak and leads to inefficient usage of resources. Weighers should measure more than RAM usage. VM performance is largely affected by the host's computation ability and its usage. Those factors can be CPU utilization rate, vCPU usage, and processor characteristics including frequency and model. It is better to dispatch a VM to an idle host with powerful CPUs and less memory. In particular, if a VM requires more cores and is compute intensive, more attention should be paid to a host CPU utilization than its available memory to ensure better performance.

University of Malaya

CHAPTER 4: PROPOSED STATIC AND DYNAMIC LOAD BALANCING METHODS

The empirical study based on the load distribution behavior presented in the previous chapter established the problem of the impact of CPU utilization on the application execution time while the initial deployment of VMs. The purpose of this chapter is to propose the solution to solve the problem as highlighted in chapter 3. We proposed the Static Multi Resource based Scheduler (SMRS) and Dynamic multi resource based schedulers (DMRS) as a solution of the problem. In this chapter initial VM placement algorithm (static algorithm) and VM migration based algorithm (dynamic algorithm) is proposed while considering the CPU utilization of physical hosts. In static algorithm the load is balanced when the VM is newly launched. In contrast, in dynamic algorithm, load is also balanced after the deployment of VMs by adapting the migration solution. The proposed solutions are modeled mathematically in order to balance the load of the physical servers based on the CPU utilization and in dynamic method along with the balanced load distribution the number of migrations are minimized.

The organization of this chapter is as follows. Section 4.1 present cloud architecture based on the proposed algorithm Section 4.2 discuss the proposed SMRS method based on compute load, load analyzer, and load filter. Section 4.3 presents the proposed dynamic multi resource based scheduling algorithm. Section 4.4 presents the system flow diagram of proposed DMRS method while adapting the minimum migration objective. Section 4.5 mathematically model the proposed algorithms based on resource constraints, operational constraints, and migration constraint while considering the load balancing objective. It also discusses assumptions that are considered while balancing the load within a data center. Section 4.6 discusses the data design that is considered to evaluate the proposed methods based on the CPU utilization, application execution time, and number of VM migration parameters. In addition, section 4.7 highlights the distinguishing features of

University of Malaya

the proposed work. Finally, section 4.8 conclude the proposed methods.

4.1 Multi Resource Based Scheduler Cloud Architecture

Load balancing fairly distributes the workload on the physical machines for efficient resource utilization. In this section, an overview of cloud deployment architecture is presented that this study has considered for load balancing.

Fig. 4.1 presents an overview of the proposed VM deployment architecture in OpenStack cloud. In the said figure, OpenStack cloud controller represents the physical host that runs the API components, schedulers, and compute servers. Each compute server is deployed with the OpenStack compute component. The functionality of a request handler is apprehended in OpenStack scheduler and API components. The request handler accepts the VM placement requests, as long as CPU demand remains below the cloud capacity (i.e. Utilization of CPU does not exceed the defined threshold). Otherwise, it rejects the VM placement request due to low resource availability. The placement architecture is split in two engines including Global Decision Engine (GDE) and Local Monitoring Engine (LME). The Load Filter module of open stack runs inside the controller which is hosted within GDE. The GDE performs the initial placement of VM based on set of decision making parameters collected through the LME. LME shows the Compute Load and Load Analyzer components to perform computations and load analysis, respectively. In the non-optimized model of open stack, compute load component performs computations based on CPU states. In contrast, Load Analyzer collects the aggregated results of CPU utilization from each compute server and transfers it to the LME module.

University of Malaya

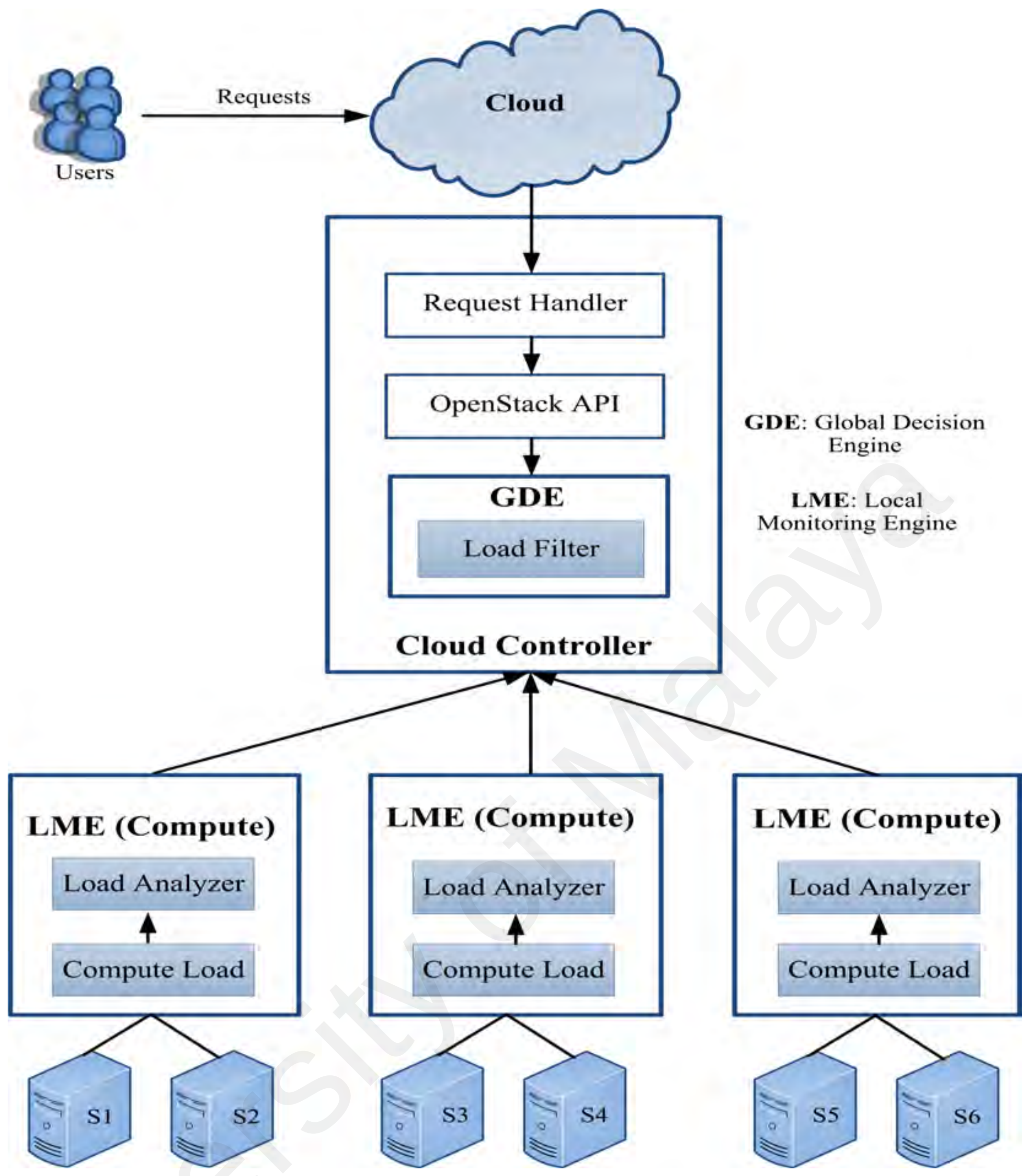


Figure 4.1: Proposed SMRS based VM Deployment Cloud Architecture

University of Malaya

4.2 Static Load Balancing Method for initial VM placement

In this section, discussion on the design of proposed static load balancing algorithms is provided. Table 4.1 represents a set of symbols that are used in the design of proposed algorithms. The responsibilities and flow of execution for compute load, load analyzer, and load filter is discussed below.

Table 4.1: Algorithm's Symbols and their Description

Symbol(s)	Description
ul	User Load
sl	System Load
nl	Nice state
ncl	Idle state
SP	System's previous load
SR	System's resent load
AvgLoad	Average Load
N	Total number of physical hosts
n	specific physical hist
s	Host state
p	Filter properties
TU_i	requested instance type from users
i	i is instance type
car	CPU allocation ratio
d	Database
TvCPUs	Total number of vCPUs

University of Malaya

4.2.1 Compute Load

Compute Load (CL) calculates the average load and updates it in a local database. It exploits current and previous CPU utilization states to compute average load on CPU. Initial phase in CL as highlighted at line 3-6 calculates the CPU utilization based on user state, system state, nice state, and idle state. Moreover, subsequent stages as highlighted at line 7-11 in algorithim 1 refers to average load estimation process. Average load is computed based on the ratio of CPU used to total CPU capacity as presented in Algorithm 2.

Algorithm 2 Compute Load

```
1:  tsecond ← 30
2:  while (1) do
3:      ul ← getUserProcessLoad()
4:      sl ← getSystemProcessLoad()
5:      nl ← getnice()
6:      ncl ← getNotUsedCPUpercentage()
7:      $SR_1$ ← {ul, sl, nl }
8:      $SR_2$ ← {ul, sl, nl, ncl}
9:      $SP_1$ ← {ul, sl, nl }
10:     $SP_2$ ← {ul, sl, nl, ncl}
11:
        $AvgLoad \leftarrow \frac{\sum_{I \epsilon SR_1}^{length(SR_1)} Cost\ (i) - \sum_{j \epsilon SP_1}^{length(SP_1)} Cost\ (j)}{\sum_{I \epsilon SR_2}^{length(SR_2)} Cost\ (i) - \sum_{I \epsilon SP_2}^{length(SP_2)} Cost\ (j)}$
12:     Wait(tsecond)
13:     Load-DB-update(AvgLoad)
14:     Goto step 2
15: end while
```

4.2.2 Load Analyzer

Load Analyzer (LA) computes system load based on the Algorithm 2 for all physical servers and share it with GDE as shown in Fig. 4.1. Later on, LA (Algorithm 3), transfers the CPU load by establishing a one-to-one communication link between GDE and Load Analyzer for every compute node (line 2-3).

Algorithm 3 Load Analyzer

```
1: for each node $n \in N$ do
2:     Loadinfo< $n$, val > ← ComputeLoad(n)
3:     send-LoadInfo-GDE(ComputeLoad(n))
4: end for
```

University of Malaya

4.2.3 Load Filter

Load Filter (LF) decides whether a particular compute server is feasible to host a VM or not based on its resource capacity. Load Filter follows four steps to decide feasibility of a physical server for VM placement. It starts with a verification process to see that the user requested instance fulfills the deployment criteria or not (see line 2-5). In the second step, it investigates available resources based on vCPUs's current utilization level (see line 6-11). During step 3, LF collects the load information across each physical server from GDE as gathered by Algorithm 2 (line 12). Moreover, it sets flag as TRUE if it finds a compute node with a minimum load or vCPUs used are maximum but the target server has minimum CPU utilization as discussed at line 14-18 in Algorithm 4.

Algorithm 4 Load Filter

```
Require: self, s, p, $TU_i$, i, car, d, N, threshold
 1: procedure HostPasses(self, s, p)
 2:     $Type_i$ ← p.get(i)                              ▹ i is instance type
 3:     if $Type_i \doteq TU_i$ then
 4:         return True
 5:     end if
 6:     vCPU ← $getVCPU()$
 7:     car ← $self.getCPUAllocationRatio(s,p)$
 8:     TvCPU ← $s.TvCPU * car$
 9:     if Tvcpu > 0 then
10:         vCPU ← TvCPU
11:     end if
12:     l ← $\forall_{n\in N}$ LoadInfo-GDE-DB(n)
13:     MinLoad = $\lceil(s.vcpus - used)\rceil * l$
14:     if l > threshold then
15:         return Tvcpu - $s.vcpus - used >\doteq TU_i$.reqvcpu
16:     else
17:         return Tvcpu - (s.vcpus$-used *$ MinLoad) $>\doteq TU_i$.reqvcpu)
18:     end if
19: end procedure
```

University of Malaya

4.3 Dynamic Load Balancing Method (DMRS)

Dynamic load balancing algorithm balanced the load based on the migration technique in cloud. Dynamic method initially place the VM while satisfying the minimum workload criteria and migrate the VMs after their deployment. This method controls the number of migrations to enhance the cloud performance. Proposed method presented as a Dynamic Multi Resource Based Scheduler (DMRS) is consists on number of algorithms including, (a) time slot, (b) allocation of VM to PM algorithm, (c) balanced load, and (d) load balanced algorithms. In the following the working of algorithms is explained in detail.

4.3.1 Time Slot

Time Slot algorithm as presented in Algorithm 5 placed the newly launched VMs on the physical servers based on the allocation criteria described in Algorithm 6. It also balanced the load after the placement of VMs using the Algorithm 7.

Algorithm 5 Time Slot

```
1: for each timeslot $t_i$ do
2:     for each newVM $v_i$ do
3:         $AllocatePhy(v_i)$
4:     end for
5:     BalancedLoad( );
6: end for
```

University of Malaya

4.3.2 Allocation of VM to PM

Algorithm 6 allocated the physical VMs to the PMs. It calculates the load of all physical machines as mentioned at line 2. It assign the VM to the physical host which has the minimum load in term of CPU utilization (line 4-5).

Algorithm 6 AllocatePhy(v_j)

```
1: for each Phy  $P_k$  do
2:     $Load_k \leftarrow getLoad(P_k)$
3: end for
4: $P_{min} \leftarrow min(P_{Load_k})$                ▷ for all k
5: $P(v_j) \leftarrow P_{min}$
```

4.3.3 Balanced Load

Balanced load algorithm 7 balanced the loads when the VMs are initially deployed using the Algorithm 6. This algorithm controls the status of the unbalanced state of servers and total numbers of migration. As presented at line 4, the load of each host is computed. Moreover, the under loaded and over loaded PMs are categorized by this algorithm (from line 6-9). Afterward, the load of each VM on the PMs is computed by this algorithm. Later, the one VM with the minimum load is migrated to the PMs which are showing the minimum load (see line 11-12). Based on this algorithm one number of migrations are taken place among the servers to balance the CPU utilization. The proposed algorithm minimized the numbers of migrations as initially the VMs are deployed using the balanced load criteria. This algorithms migrates the VMs based on minimum load factor because if the highly load VMs are migrated the host became overloaded; therefore; the migration criteria is set less then five time and greater than one in order to fairly distribute the workload.

University of Malaya

Algorithm 7 BalancedLoad( );

```
1: migrations ← 0
2: while (!LoadBalanced( ) && migrations<5) do
3:     for each Phy $P_k$ do
4:         $Load_k$ ← getLoad($P_k$)
5:     end for
6:     $P_{max}$ ← max($Load_k$)                        ▷ for all k
7:     $P_{min}$ ← min($Load_k$)                        ▷ for all k
8:     for each $v_j$ on $P_{max}$ do
9:         $Load_j$ ← $Load_{v_i}$
10:    end for
11:    $v_{min}$ ← min($v_{Load_j}$)                     ▷ for all j
12:    $P_{min}$ ← $v_{min}$
13:    migrations++;
14: end while
```

4.3.4 Load Balanced

Algorithm 8 check out the balanced load based on the difference of load calculated between the physical hosts. This algorithms shows that if the difference among the servers is greater than one physical core that load is not balanced and the algorithm will return false. In contrast, if the difference is less than to core the load is balanced and the algorithm will written true.

University of Malaya

Algorithm 8 LoadBalanced( );

```
1: for each Phy $P_k$ do
2:     $Load_k \leftarrow getLoad(P_k)$
3: end for
4: for i in $P_k$ do
5:     for j in i $\leftarrow P_k$ do
6:         $DiffLoad_{ij} \leftarrow abs(Load_{P_i} - Load_{P_j})$
7:     end for
8: end for
9: for each $DiffLoad_{ij}$ do
10:    if $DiffLoad_{ij} > (minPhy_{cores} - 1)/(minPhy_{cores})$ then
11:        return False
12:    end if
13: end for
14: return True
```

4.4 System Flow Diagram with Dynamic Load Balancing

A VM provisioning in cloud is an vital use-case as presented for existing placement solution in section 2.4.2. This section explains the VM placement and interaction flow of components of the proposed model. The flowchart describes that the load of every compute node is analyzed using the load analyzer algorithm and load aggregator collect the load of each PM and send the load information to the admission controller. As it is perceived when the VMs are deployed on PMs the load of each PM is varied based on the VMs and the programs associated to that VMs; therefore, admission controller checks the threshold based criteria in order to update the load value for each PM took the decision based on the threshold. The decision maker checks that if the value of threshold is less than the load update time the load value is not updated and at that time the request time is expired and again send to the admission controller otherwise the load value is aggregated again and transfer to the admission controller in a recursive order. In addition, at the same time admission controller receives the requests from the request handle for the

University of Malaya

provisioning of the VMs in cloud.

The compute node related information regarding the load parameter is stored in admission controller then based on this information the request is forwarded for further decision related to the PM availability. The decision maker checks if the PM is not available the request is put in the wait state and checks for the available PM through the PM availability decision maker in iterative mode. Otherwise, if the PMs are available the further decision is taken place on it. At this stage the decision maker with check the capacity of each PM and the requested resources in terms of CPU, RAM, etc. If the decision maker fulfills the criteria of requested VM it pass the request and chose the feasible PM for the placement of that VM. Moreover, if the request is not accomplished the migration will be taken place for that request. Based on migration criteria the request is again forwarded to PM availability decision maker and that process is followed is repetitive order until the feasible PM is not associated with requested VM.

University of Malaya

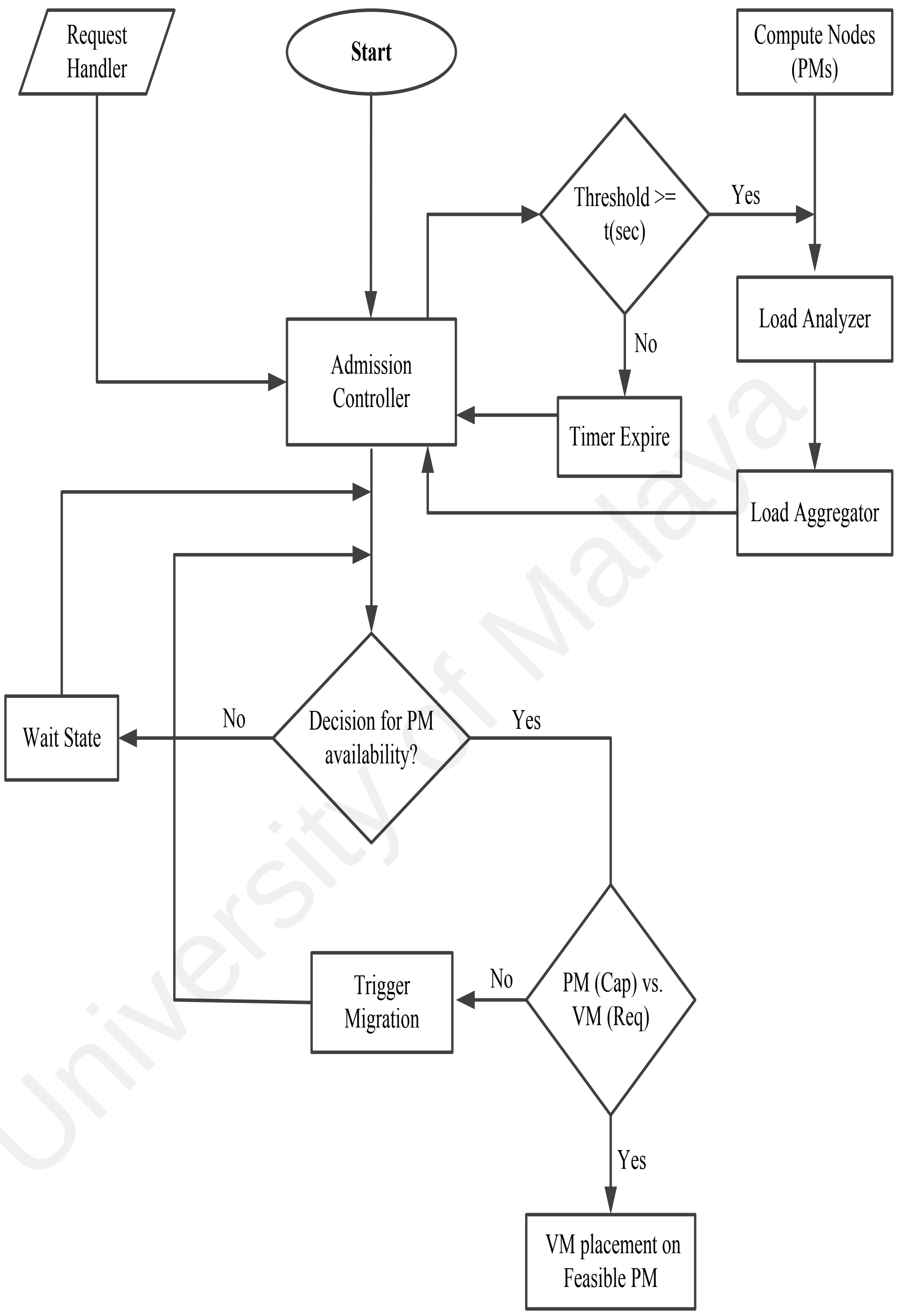


Figure 4.2: System Flow Diagram for VM Placement and Migration

University of Malaya

4.5 Optimization Model

This study has proposed a linear programming based optimization model that balances the workload on each physical server. The proposed optimization model computes bound on the load balancing of physical machine with the given computational load on VM as input for the optimal VM placement. For the better understating of the model, it proposes a set of symbols representing constants and variables as listed in Table 1.

Table 4.2: Notations and Description

Notation	Description
N	the request's size as a number of requested VMs
M	total number of physical machines (PM) in data center
$Vcpu_i$	shows the requested core (s) of VM_i
X_{ij}	bivalent variable which is representing the VM_i assignment to the PM j
$Vcpu_j$	indicates the maximum number of virtual cores of server j
Phy	denotes the physical servers
$Load_{max}$	denotes the maximum value of load
v_{min}	denotes the VM with the minimum load
P_{min}	describes the physical server with minimum load value
P_{max}	describes the physical server with maximum load value
P_{v_j}	represents the VM on the physical machine j

University of Malaya

4.5.1 Assumptions

For the VM placement, following assumptions and limitations are considered by the proposed optimization model for load balancing on physical server within a Data center.

1. Virtual machine load does not vary over time. Therefore, it is assumed that throughout the execution time window it will remain same.

2. When VM is running, the workload running inside it will surge CPU utilization to

its 100% before its execution time.

4.5.2 Linear Programming Formulation

This section discuses the decision variables, constraints, and objective function of the model.

4.5.2.1 Decision variables

The optimization model computes the deployment of VM_i on the physical server j in time slot t. This is explained by decision variableX_{ijt} as shown below. The value of X_{ijt}=1 when VM_i is deployed; otherwise, X_{ijt}= 0.

$$X_{ijt} = \begin{cases} 1, & if\ the\ VM_i\ is\ placed\ on\ server_j\ in\ time\ interval\ t \\ 0, & otherwise. \end{cases}$$
$$\text{where} \forall i = 1..,M, \forall j = 1..,N, \forall t = 1..,T.$$

$$C_{it} = \begin{cases} 1, & if\ the\ VM_i\ is\ not\ migrated\ on\ server_j\ in\ time\ interval\ t \\ 0, & otherwise. \end{cases}$$
$$\text{where} \forall i = 1..,M, \forall t = 1..,T.$$

Cost C_{it} is an integer variable. Based on each VM in every single time slot the value of C is represented as 0 or 1. It represents that the VM (i) is running on specific PM (j) in time slot (t), if it is running on same PM in next time slot (t+1) the cost value will be 0; otherwise; it will be denoted with 1 when the VM is migrated to other PM.

4.5.2.2 Load balancing constraint

The proposed static and dynamic load balancing algorithm ensures that the same workload is hosted on all physical machines. In the following model, the fair share load on the PM is presented.

University of Malaya

$$\sum_{i=1}^{M} X_{ijt} - \sum_{i=1}^{M} X_{ikt} \leq 1 \quad , \forall j = 1.., N, \forall k = 1.., N, where j \neq k$$

The load balancing constraint presents that the load is equally balanced when the difference between the serves is reported ≤ to 1 based on the numbers of VMs allocated to the servers.

4.5.3 Optimized Static Load Balancing Method

This section presents the resource constraints, operational constraints, and objective function of optimization static load balancing method. This model considers load factor while scheduling initial placement of VMs within a data center.

4.5.3.1 Resource constraints

The number of linear constraints based on the optimization concept reflects the capacity limit of PMs, which subjects the obvious facts that a PM can only host the number of VMs based on its remaining resources. Each server carries limited number of cores (vCPUs). Also, $Vcpu_j$ that cannot be exceeded when hosting or serving the VMs based on the remaining resources as presented in the following equation.

$$\sum_{i=1}^{M} Vcpu_i * X_{ijt} \leq Vcpu_j \quad , \forall j = 1.., N, \forall t = 1.., T$$

Similarly, during VM deployment, the capacity of RAM available should be higher than the one required by the hosting VM for each physical host. This behavior is modeled in below equation.

$$\sum_{i=1}^{M} R_i X_{ijt} \leq R_j \quad , \forall j = 1.., N, \forall t = 1.., T$$

University of Malaya

Each VM is scheduled for execution when its time slot comes. The following constraint represents that VM_i should not be scheduled before its start time.

$$\sum_{t=1}^{T_{Si}-1} X_{ijt} \leq 0 \quad ,\forall i = 1..,M,\forall j = 1..,N$$

4.5.3.2 Operational constraints

The cloud providers have to fulfill the request within the prescribed quota. A single VM can only be deployed to one PM at time "t" as modeled below.

$$\sum_{j=1}^{N} X_{ijt} \leq 1 where,\forall t = 1..,T, \quad \forall i = 1..,M$$

The following constraint present that execution time of every VM at all intervals should be less then or equal to its total execution time.

$$\sum_{t=1}^{T}\sum_{j=1}^{N} X_{ijt} = E_T \quad ,\forall i = 1..,M$$

4.5.3.3 Objective function

The objective function along with the complete set of constraints are listed in Equations 4.1, 4.2, 4.3, 4.4, 4.5, and 4.9.

$$Max \sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{t=1}^{T} X_{ijt} - \sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{t=T/2}^{T} X_{ijt}$$

Subject to:

$$\sum_{i=1}^{M} Vcpu_i * X_{ijt} \leq Vcpu_j \quad ,\forall j = 1..,N,\forall t = 1..,T \tag{4.1}$$

$$\sum_{i=1}^{M} R_i X_{ijt} \leq R_j \quad ,\forall j = 1..,N,\forall t = 1..,T \tag{4.2}$$

University of Malaya

$$\sum_{t=1}^{T_{Si}-1} X_{ijt} \leq 0 \quad ,\forall i = 1..,M, \forall j = 1..,N \tag{4.3}$$

$$\sum_{j=1}^{N} X_{ijt} \leq 1 where, \forall t = 1..,T \quad ,\forall i = 1..,M \tag{4.4}$$

$$\sum_{t=1}^{T} \sum_{j=1}^{N} X_{ijt} = E_T \quad ,\forall i = 1..,M \tag{4.5}$$

$$\sum_{i=1}^{M} X_{ijt} - \sum_{i=1}^{M} X_{ikt} \leq 1 \quad ,\forall j = 1..,N, \forall k = 1..,N, where j \neq k \tag{4.6}$$

4.5.4 Optimized Dynamic Load Balancing Method

This section presents the resource constraints, operational constraints, and objective function of optimization dynamic load balancing method.

4.5.4.1 Resource constraints

A VM must be scheduled during its scheduled slots between start and end slots. Moreover, in single time slot the VM should be scheduled on exactly 1 PM. It shows that when the value is one for one specific server for others its value will be 0 and VM will not scheduled to that servers. The following constrain ensures the continuous allocation of VM_i on the PM_j in each slot.

$$\sum_{j=1}^{N} X_{ijt} \leq 1, \forall t = 1..,T_S, \forall i = 1..,M$$

4.5.4.2 Operational constraints

The constraint represents that the VM is not scheduled to any PM until its start slot is not allocated to any PM before that interval the value of this VM will be 0. Moreover, in single time slot only 1 VM is assigned to 1 PM.

University of Malaya

$$\sum_{j=1}^{N} X_{ijt} \leq 0 \quad ,\forall i = 1..,M,\forall t = 1..,T\ and 1 \leq t \leq T_s$$

The VM no more scheduled when its value of end slot comes. The two constraints are presenting the VM is not scheduled before its start slot and after its end slots. That interval is showing VM has completed its task between start and end slots.

$$\sum_{t=1,t>T_E}^{T} \sum_{j=1}^{N} X_{ijt} \leq 0 \quad ,\forall i = 1..,M$$

4.5.4.3 Migration based constraint

Migration constraint is represented as the VM_i in t time interval should be scheduled on PM_i. For instance, if the VM is deployed on PM 1 and in next time slot the same VM is still placed on it the cost values is 0 to 0 which means VM is not migrated. The migration constraint shows the difference of two transmission. Therefore, if the value of C changes from 0 to 1 or 1 to 0 it means the migration is taken place and VM is transferred from one server to another in same time interval t. In contrast, if the values of C is 1 to 1 VM is not migrated to other server. The 2 following constraints shows the absolute value of migration constrains. In order to cancel out the absolute values the constraints are derived as,

$$X_{ijt} - X_{ijt+1} \leq -C_{it} \quad ,\forall t = 1..,T_{Sch} and t \neq T_E,\forall M ,\forall N$$

$$X_{ijt} - X_{ijt+1} \geq -C_{it} \quad ,\forall t = 1..,T_{Sch} and t \neq T_E,\forall M ,\forall N$$

University of Malaya

4.5.4.4 Objective function

The objective function along with the complete set of constraints are listed in Equations 4.7, 4.8, 4.9,4.10, 4.11, 4.12.

$$Max \sum_{i=1}^{M} \sum_{j=1}^{N} \sum_{t=1}^{T} X_{ijt} - \sum_{j=1}^{N} \sum_{t=1}^{T} C_{it}$$

Subject to:

$$\sum_{j=1}^{N} X_{ijt} \leq 0 \quad ,\forall i = 1..,M, \forall t = 1..,T \; and 1 \leq t \leq T_s \tag{4.7}$$

$$\sum_{t=1,t>T_E}^{T} \sum_{j=1}^{N} X_{ijt} \leq 0 \quad ,\forall i = 1..,M \tag{4.8}$$

$$\sum_{i=1}^{M} X_{ijt} - \sum_{i=1}^{M} X_{ikt} \leq 1 \quad ,\forall j = 1..,N, \forall k = 1..,N, where j \neq k \tag{4.9}$$

$$\sum_{j=1}^{N} X_{ijt} \leq 1, \forall t = 1..,T_S, \forall i = 1..,M \tag{4.10}$$

$$X_{ijt} - X_{ijt+1} \leq -C_{it} \quad ,\forall t = 1..,T_{Sch} and t \; \neq T_E, \forall M , \forall N \tag{4.11}$$

$$X_{ijt} - X_{ijt+1} \geq -C_{it} \quad ,\forall t = 1..,T_{Sch} and t \; \neq T_E, \forall M , \forall N \tag{4.12}$$

4.6 Data Design

This section explains the metrics in order to evaluate the proposed method.

4.6.1 CPU utilization

CPU utilization states the amount of the CPU capacity that a program required during its execution. CPU is a shared resource and its utilization varies based on the amount of resources handled by it. The utilization of the CPU is modeled based on the following Equ. 4.13

University of Malaya

$$CPU\ utilization = user + nice + system + idle \tag{4.13}$$

CPU utilization is represented as the sum of user state, system state, nice state, and idle state. User time states the amount of time taken by CPU in order to execute the program in user space. In addition, program executed in user state with high (nice) priority is presented in nice state. Moreover, system time is explained as the time in which the CPU is busy in order to execute the program in kernel space. Idle time is also measured based on the unused CPU capacity and it represents the CPU state when the CPU not executing any processes.

4.6.2 Execution Time

Application execution time states the system time that application takes to finish its execution. Total execution time is derived from the product of three parameters as discussed in terms of I, CPI, and C whereas it is measured in seconds.

$$ExecutionTime = \sum_{i=1}^{n} CPI_i \times C_i \tag{4.14}$$

Where the parameter I is explained as a program is consists of a multiple instructions that are executed, which are measured as number of instructions/ program. Moreover, each instruction acquire a number of cycles in order to accomplish the execution which is presented as, verage cycle / I and denoted with cycle per instruction (CPI) whereas CPI is also presented as Instructions per cycle IPC= 1/ CPI. In contrast, CPU takes the stable clock cycle time and explained as seconds/ cycle which is represented in the form of C = 1/f.

University of Malaya

4.6.3 Numbers of migration

Number of migrations shows the total migrations within the cloud in order to balance the load when the VMs are deployed and running. In the static algorithm VMs are initially placed to balance the CPU utilization among all physical servers. When the VMs start their execution the physical servers become overloaded again in order to balance the load after the deployment of VMs the number of migrations are considered. Overlooking load balancing establishment abruptly decreases system throughput due to overloaded servers and ultimately leads to SLA violation. It has become an integral part of all distributed internet based systems as distributed computing comes with the challenges of high resource demands that overload servers.

4.7 Distinguishing Features of Proposed Method

This section explains the distinguishing features in comparison of traditional VM placement algorithm.

4.7.1 Efficient Resource Utilization

Resources within a cloud data centers are usually over-provisioned to avoid SLA violations. Inside the cloud, the controller is responsible for managing the resources such as physical servers, virtual machines, and network connectivity equipment. The traditional static load balancing methods inefficiently utilize the resources of a data center owing to improper VM placements. The proposed algorithms efficiently utilize the underlying resources of a data center for better resource management. Therefore, the proposed study helps in resource scheduling policies.

University of Malaya

4.7.2 Ground for Energy Efficiency

Cloud data centers consume a significant amount of energy due to the provisioning of equipment to cool the data centers. Servers are switched on to provision the services on 24/7 basis. However, due to non-optimized VM placement, the majority of the server remained on while doing the limited activity. Existing non-optimal load balancing methods do not consider all the resources of a server such as RAM, CPU, network bandwidth, and degree of interaction among hosted VMs during placement. As a result, the energy consumption of cloud data center increases. The proposed research as it considers multiple resources at the time of VM placement creates a ground to switch off servers which are under loaded by efficiently placing the load on remaining servers.

4.7.3 Hot Spot Elimination

Over resource utilization usually, leads to hot spot within cloud data centers. Existing static load balancing methods lead to resource provisioning as it adds VMs on highly loaded physical servers (chapter 3). As a result, the performance of the whole network gets down. The proposed research isolates overloaded servers. It does not place a VM on a server that is already loaded. Rather, it wisely places it on most appropriate server devices. As a result, the host spots in the data centers are minimized. The cloud management policies migrate the VMs from the hot spot server to another server that increases the cost of the data center as VM migration is resource expensive process.

4.7.4 SLA Violation Avoidance

The performance of a VM is affected when it shares the resources with the tenant applications. Hosting so many applications on a single device lead to SLA violation. The SLA is a measure of reliability, availability, user service time, latency, and an end to end delay.

University of Malaya

Existing conventional load balancing schemes are inefficient in terms of handling the optimal resource placement for initial VM placement. The proposed research is effective as it optimally places VMs for minimizing the SLA violation.

4.8 Conclusion

In this chapter the static and dynamic load balancing algorithms are proposed in order to efficiently utilize the cloud resources. In order to enable this experiment, we presented the lightweight extension of OpenStack compute service as a multi resource scheduler, which is based on three modules: (a) Request Handler, (b) Global Decision Engine, and (c) Local Monitoring Engine. Static load balancing algorithm minimize the application execution time based on the fair distribution of the resources. Moreover, for dynamic load balancing a dynamic multi resource based scheduler method is proposed that incorporates the migration technique in order to balance the load after the placement of VMs. Dynamic method minimize the number of migrations and enhance the cloud performance while minimizing the running time of an application. This extension allowed us to compare our experimentation results with the standard OpenStack Nova scheduler according to load distribution principles. In addition, the mathematical models are proposed for the static and dynamic load balancing methods. The subsequent chapter presents the implementation details of the proposed methods, details of data design, evaluation, and validation of the mathematical model.

University of Malaya

CHAPTER 5: EVALUATION

The aim of this chapter is to discuss the data collection process adapted to evaluate the performance of proposed multi resource schedulers for static and dynamic cloud environment. The objective of the chapter is to discuss the experimental setup, tools, benchmark applications, data collection technique, which are used to test the performance of proposed algorithms. In addition, this chapter also explain the mathematical model parameters and statistical method in order to examine the correctness of collected data. The performance of proposed algorithms is explained based on different components including (a) application execution time, (b) CPU utilization as a load, and (c) numbers of VM migration.

This chapter is comprised based on seven sections. Section 5.2 discusses the evaluation setup, experimental devices, benchmark method, performance metrics, and data gathering and data processing. Section 5.3 present the data collected to evaluate the performance of SMRS algorithm based on initial VM deployment by comparing it with the benchmark method. Section 5.4 report the data collected for to validate the accuracy of developed mathematical model by comparing its obtained results with the results of experiments. Section 5.5 presents the collected data of proposed DMRS algorithm based on performance parameters in terms of CPU utilization, number of migration, and application execution time. Section 5.6 presents the data collected to analyze the impact of VM deployment with the execution time for the mathematical model, and proposed solution. Finally the section 5.7 highlight the conclusive remarks.

University of Malaya

5.1 Introduction

Static multi resource scheduler is proposed to initially place the VM on the physical server based on CPU utilization factor with the minimum application execution time. The significance of the SMRS analyzed by comparing it with the benchmark method as explained in chapter 3. The performance of SMRS is evaluated by varying the VM deployment parameters based on (a) number of vCPUs, (b) RAM, and (c) disk space criteria using the different flavors for different data traces. In order to statistically analyze the performance of SMRS the mean, standard deviation, and confidence intervals are also calculated which shows the significance of the proposed static solution.

Dynamic Multi resource scheduler is proposed to initially place the VMs using the SMRS deployment criteria and after the placement balance the load based on VM migration. The objective of this algorithm is to minimize the number of VM migration while considering the CPU load. The significance of the DMRS is evaluated with the mathematical model and statistical analysis. Moreover, the performance of DMRS is compared with the legacy benchmark method. The performance of the proposed algorithm is evaluated with respect to CPU utilization, application execution time, and the numbers of migration. The sample mean of the sample space is determined based on 30 values that shows the significance of results by finding the error estimate for 95% confidence interval.

5.2 Evaluation of Proposed Multi-resource based Scheduler

This section briefly discusses the evaluation methodology used to perform the evaluation of proposed algorithms.

5.2.1 Evaluation Setup

It states the tools, discusses test program designs, and highlights experimentation equipments for performing the experiments. In this study, the experimentation is conducted on real hardware equipments to analyze the performance of VM deployment in cloud in terms of average CPU utilization and execution time.

The proposed study has considered a small in house data center to conduct the ex-

University of Malaya

perimentation for problem analysis in traditional load balancing methods. It has selected the OpenStack cloud for experiments. During the experiments, a control environment is modeled to deploy VM using OpenStack scheduler. Moreover, it has used the traditional computers rather than expensive and powerful machines (small industrial cloud replica).

5.2.2 Experimental Devices

We implemented the static and dynamic multi resource algorithms on small-scale cloud data center using Linux operation system. In order to conduct the experiment, a private cloud setup is installed using the OpenStack Havana which is represented as OpenStack cloud. The cloud setup is comprised of 4 compute node. Each compute node is deployed with the homogeneous characteristics in terms of their resource capacity based on their vCPUs, CPUs, RAM, and system cache. In this setup each physical machine exactly have 16GB RAM capacity, CPU with 2.40GHZ, QEMU hypervisor as presented in chapter 3. Among the 4 servers one server is deployed with the specification of controller node as well as the compute node and it can manage the all compute nodes in the system. The compute nodes communicate with each other using the flat-DHCP networking module. Table 5.1 presents the system specifications of deployed OpenStack cloud based on their resource capacities.

University of Malaya

Table 5.1: System specifications

Specification	**Capacity**
Processor	2.40GHz
RAM	16GB
Compute Nodes	3
Controller Nodes	1
Operating System	Ubuntu 12.04 LTS
Over-provisioning frequency	2
Storage capacity	242GB

In order to check the performance of OpenStack cloud number of VMs are deployed. At the time of VM deployment the CPU utilization is captured while running the client server application that provides the continuous CPU utilization to the controller node using the load analyzer algorithm. During the evaluation, the behavior of proposed algorithms is analyzed based on the load and application execution time parameter. In order to consider load while running the VMs a CPU intensive application is executed inside each VM. For the SMRS known as static load balancing algorithm the behavior of CPU usage is observed with the static and random load distribution factors. For the static load analysis each VM is fully utilizing its number of vCPUs by executing the CPU bound test application. CPU bound test program is performing the simple arithmetic operation and it is designed including the nested loops of different sizes. CPU bound test program is executed to generate the workload inside the VM. Furthermore, the random distribution of loads inside the VMs the test program is executed with the VM based on random load generator function. The random load generator function produce the values between 0 and 2. The value 0 presents that no application is running for specific VM. In contrast the values 1 shows that only one CPU intensive program is running inside the VM whereas the two applications are running for value 2 as generated by the function.

In dynamic load balancing as represented with DMRS, VMs are executed at time interval t at the time of their creation. The VMs are migrated when the CPU load is varied at certain point and it fulfill the CPU overprovisioning criteria. In DMRS and SMRS workload is generated based on the number of VMs by executing the CPU intensive application. The CPU utilization value is captured using the top and gripped with the awk command. Moreover, the application execution time is calculated using the script which is designed using the shell script command.

University of Malaya

5.2.3 Test Program

To evaluate the performance of proposed algorithm the synthetic test program is designed that keep the VM busy and 100% utilize the capacity of CPU resources assign to VM. In order to generate the 100% load each VM executed the number of programs equal to its number of cores. The program is designed considering the factors that proposed algorithms targets the CPU load; therefore, it should measure the performance of CPU utilization and does not entail the human interaction while its execution. CPU intensive program is composed of the set of basic arithmetic operation within the nested loops. The program within the loops perform the operations on an array. The CPU intensive program is designed to fully utilize the CPU resources when executed inside the VM.

5.2.4 Performance metrics

To evaluate the performance of proposed load balancing algorithms cloud environment following performance measuring metrics are studied.

1. Application execution time
2. CPU utilization
3. Numbers of VM migration

Application execution time is computed based on how long applications are running inside the VMs and the time taken to complete the computations inside the cloud. The application execution time is capture for individual VMs running on each physical server. Moreover, it is also observed the overall system time when the numbers of VMs are running in order to show the effect of CPU load on the application execution time.

CPU utilization is presented as the capacity of CPU which is required to perform the computations. CPU utilization is measured based on the processes running inside the user state, system state, nice state, and idle state. The CPU utilization varies based on

University of Malaya

the number of resources handle by it. In this study CPU utilization is measured using the compute load filter. In the static environment we assumed that all VMs are running with the 100% utilization of load. Based on the number of cores allocated to each VM the test program is executed inside the VM and captured to analyze the CPU utilization.

Numbers of migration states the total migration required to balance the load when the physical servers are overloaded. Load balancing ensures even distribution of resources among a set of users in a uniform way such that underlying servers do not become overloaded and idle at any time within cloud operation time line. in our proposed DMRS algorithm load balancer increases the capacity and reliability of applications by decreasing the burden on a server. Load balancer starts with identification of hot spot, an overloaded server, and start migrating its load (VMs) on a server which has sufficient resources such that the resources are evenly distributed. However, the criterion of migration is set based on minimum CPU utilization.

5.2.5 Data gathering and data processing

The performance measuring parameters are investigated in diverse environments by varying the system variables such as number of vCPUs associated to each VM. The effects of system variables are analyzed on the performance measuring metrics. First of all we compared the performance of our static SMRS algorithm with the standard OpenStack based filter scheduler. In second phase, we compared the results of mathematical model with the results of our proposed DMRS algorithm to validate the correctness of mathematical model. Thereafter, we used the mathematical model to collect the results considering the measurement metrics. Finally, the performance of the propose algorithms is critically evaluated by comparing the proposed algorithms results with the standard algorithms and with the mathematical model.

The primary data is collected by testing the SMRS and DMRS algorithms in cloud

University of Malaya

environment in different scenarios based on statics, random, and dynamic load. The impact of numbers of VMs deployment on the CPU utilization is analyzed based on thirty values. The collected data for the application execution time and CPU utilization based on thirty data traces. The impact of CPU utilization on application execution time is analyzed by varying the CPU load. Moreover, the impact of application execution time for individual VM is evaluated based on the number of core utilization inside the VMs. The impact of numbers of VM migration are also analyzed when the load on physical server is not balanced. Furthermore, the data is collected for the comparison of the proposed solutions (SMRD and DMRS) with the state-of-the-art standard OpenStack scheduler for application execution time, CPU utilization, and numbers of VM migration.

5.3 Data Collection for Initial VM Deployment (SMRS)

In order to extensively analyze the behavior of number of VM deployment on the CPU utilization the VMs are launched considering the 2 vCPU per VM while considering the static environment. In this section CPU utilization behavior and application execution time is studied based on the static load factor and random load distribution factor.

5.3.1 Impact of Static and Random Load based VM Distribution on CPU utilization

In order to analyze the effect of VMs distribution on the CPU utilization two sceneries are analyzed while considering the static and random load on VMs. The deployment decision is taken by the SRMS based scheduler based on the distribution behavior of VMs and the load that is computed while the execution of VMs. The collected data presents the number of VMs placement on the physical host based on the static and random load. Each VM is placed using the 2 vCPU criteria.

University of Malaya

5.3.1.1 Static Load based VM Distribution

Table 5.2 represents the distribution behavior of VMs on the physical hosts along with the CPU utilization of that server. The data values are evaluated based on the average of thirty values. The table presents the VMs deployment in the cloud on Edge1, 2,3, and 4 compute nodes. Moreover, it shows that in order to balance the load number of VMs are deployed based on CPU utilization factor. Each VM is placed using the 2vCPUs criteria. The deployment sequence shows that considering the load factors first, second, third, and fourth VMs are deployed on Edge1, 2,4,and 3 with the CPU utilization of 5.3%, 25.4%, 25.6%, and 25.3%. The sixth and seventh VMs are deployed on same physical server in order to balance the load and showing the CPU usage up-to 12.8%, and 25.4%. At that point each physical server is showing the CPU usage up-to 25%. The CPU load is increased until the actual physical cores criteria is fulfilled. For VMs seventh, eighth, ninth, and tenth the load is reached at 50% of the CPU utilization.

The physical servers exactly have the 8 physical cores when two VMs are deployed on each physical server it shows that 4 physical cores are used by the VMs. Each VM has executed the CPU intensive program in order to fully utilize the CPU resources of that VM. Moreover, the fifteen, sixteen, seventeen, and eighteenth VMs are showing the CPU capacity upto 96.94%, 98.99%, 94.64%, and 97.59%, respectively. Theses values shows that physical servers have fully utilized its physical cores. As the overprovisioning criteria is set for each physical server the more VMs are divided on the Physical hosts.

5.3.1.2 Random Load based VM Distribution

Table 5.3 presents the collected data for the CPU utilization when each VMs is deployed using the random load values. The table is comprised with four columns. The first column shows the number of VMs on single physical host. The second column presets the CPU

University of Malaya

Table 5.2: CPU utilization Analysis with the static load based distribution of VMs

Deployment Sequence	Edge1 CPU%	Edge2 CPU%	Edge3 CPU%	Edge4 CPU%
Virtual Machine 1	5.3		0	0
Virtual Machine 2	0	25.4	0	0
Virtual Machine 3	0	0	0	25.6
Virtual Machine 4	0	0	25.3	0
Virtual Machine 5	12.5	0	0	0
Virtual Machine 6	25.4	0	0	0
Virtual Machine 7	0	0	0	50.8
Virtual Machine 8	0	51.2	0	0
Virtual Machine 9	0	0	49.6	0
Virtual Machine 10	50.4	0	0	0
Virtual Machine 11	0	0	0	75.89
Virtual Machine 12	0	75.1	0	0
Virtual Machine 13	0	0	74.63	0
Virtual Machine 14	72.24	0	0	0
Virtual Machine 15	0	0	0	96.94
Virtual Machine 16	0	98.99	0	0
Virtual Machine 17	0	0	94.64	0
Virtual Machine 18	97.59	0	0	0
Virtual Machine 19	0	0	0	98.18
Virtual Machine 20	0	97.59	0	0
Virtual Machine 21	0	0	99.68	0
Virtual Machine 22	96.12	0	0	0
Virtual Machine 23	0	0	0	99.68
Virtual Machine 24	0	99.96	0	0
Virtual Machine 24	0	0	0	0
Virtual Machine 25	0	0	99.51	0
Virtual Machine 26	0	0	0	98.37
Virtual Machine 27	0	98.96	0	0
Virtual Machine 28	0	0	99.64	0
Virtual Machine 29	99.64	0	0	0
Virtual Machine 30	0	0	0	98.96
Virtual Machine 31	0	99.50	0	0
Virtual Machine 32	0	0	99.52	0

utilization effected by the number of VMs. CPU utilization is presents the average value. Moreover, third and the fourth columns shows the statistical analysis based on the confidence interval and the standard deviation in order to proof that the results are significant or not.

University of Malaya

Table 5.3: CPU utilization Analysis with the random load based distribution of VMs

Number Of VMs	CPU utilization(%)	Standard Deviation	Confidence Interval
Virtual Machine 1	15.06	0.0230	±0.0127
Virtual Machine 2	25.4	0.2078	±0.1151
Virtual Machine 3	0.3	0.0404	±0.0223
Virtual Machine 4	25.4	0.1385	±0.0767
Virtual Machine 5	15.2	0.3002	±0.1662
Virtual Machine 6	39.4	0.0230	±0.0127
Virtual Machine 7	25.6	0.2540	±0.1406
Virtual Machine 8	55.6	0.3117	±0.1726
Virtual Machine 9	45.2	0.2424	±0.1342
Virtual Machine 10	50.05	0.0577	±0.0319
Virtual Machine 11	38.9	0.0519	±0.0287
Virtual Machine 12	55.8	0.1270	±0.0703
Virtual Machine 13	96.1	0.2944	±0.1630
Virtual Machine 14	75.07	0.3637	±0.2014
Virtual Machine 15	50.5	0.2598	±0.1438
Virtual Machine 16	55.7	0.3637	±0.2014
Virtual Machine 17	98.8	0.4156	±0.2302
Virtual Machine 18	95.5	0.2598	±0.1438
Virtual Machine 19	65.5	0.2598	±0.1438
Virtual Machine 20	75.99	0.0057	±0.0031
Virtual Machine 21	98.9	0.0577	±0.0319
Virtual Machine 22	98.8	0.4156	±0.2302
Virtual Machine 23	98.4	0.2078	±0.1151
Virtual Machine 24	98.7	0.3637	±0.2014
Virtual Machine 25	98.8	0.1847	±0.1023
Virtual Machine 26	98.8	0.1270	±0.0703
Virtual Machine 27	98.7	0.0173	±0.0095
Virtual Machine 28	98.7	0.3059	±0.1694
Virtual Machine 29	98.8	0.4156	±0.2302
Virtual Machine 30	98.6	0.2540	±0.1406
Virtual Machine 31	98.5	0.2598	±0.1438
Virtual Machine 32	97.2	0.1039	±0.0575

The table 5.3 shows that when the first VM is deployed it used 15.06% of the CPU capacity. The load is generated in each VMs based on the random number generated by the load generator function. The load generator function generated the values such as 1,2,0,2,1,0,0,1,2,2,0,2,2,1,2,0, respectively for the first 16 VMs deployed on four compute nodes. Using that values the CPU bound test program generated the load inside the VMs which shows the CPU utilization such as, 15.06%, 25.4%, 0.3%, 25.4%, 15.2%, 39.4%, 25.6%, 55.6%, 45.2%, 50.05%, 38.9%, 55.8%, 96.1%, 75.07%, 50.5%, and 55.7%, respectively. The first VM is deployed on Edge1 and based on the random load generator function it utilized the 15.06% of the CPU resources. The second VM is deployed on Edge4 and fully utilized the vCPUs with the workload value 25.4%. Based on the data collection the VMs are deployed using the random load based criteria and deployed on physical servers while satisfying the minimum load criteria. The deployment sequence of

University of Malaya

sixteen VM is reported as the third, fourth, fifth, and sixth VM is deployed on Edge2, 3, 1, and 4. Moreover, the next five VMs are deployed on servers Edge 2, 2, 3, 2, and 1. The deployment sequence is showing that four VMs are deployed on the Edge2 among the 11 VMs. This is because the load of that server is minimum as compare to other physical server and the random load generator function produced the 0 values for 2 VMs, which shows that these VMs are not execution any program.

Furthermore, the values generated for the next VMs from sixteen VMs from seventeen to thirty two are reported as 1, 2, 2, 2, 2, 1, 0, 2, 2, 2, 2, 2, 1, 1, 0, 2, and 1, respectively. Based on load generator function generated values, the VMs are deployed on a physical servers such as Edge 2, 3, 4, 4, 1, 2, 2, 3, 1, 3, 1, 2, 4, 3, 2, and, 1 while satisfying the balanced load criteria. According to the deployment, number of programs are executed inside the VMs and the CPU utilization is presented as 55.5%, 98.8%, 95.5%, 65.5%, 75.99%, 98.8%, 98.4%, 98.7%, 98.2%, 98.8%, 98.7%, 98.8%, 98.6%, 98.5%, and 97.2% along with the confidence interval presented in the collected data.

University of Malaya

5.3.2 Application Execution Time

Descriptive analysis of benchmarking results and proposed SMRS algorithm with the static and random load factors are summarized in table 5.4. Comparison tables shows the execution time values captured when the workload is deployed on physical nodes. Based on the static workload the proposed methods is showing the same results as computed with the default scheduler. When the VMs are deployed based on the CPU utilization and RAM based criteria using the SMRD and Nova scheduler each VM is running the 100% of the CPU capacity; therefore; the difference between the proposed and legacy methods is very small amount in seconds. In contrast, when the workload is deployed using the random load based distribution criteria it shows the successful results when analyzed based on SMRS method. The SMRS shows the 940, 800, 1160, and 1166 seconds in order to complete the computation of the tasks running inside the VM whereas the default nova scheduler is showing more computation such as 1667, 1664, 1668, and 1667 for Edge1, Edge2, Edge3, and Edge4, respectively. From the analysis it is observed that the behavior of SMRS is different based on the CPU utilization factor whereas the legacy scheduler is taking the same time in order to complete the task without considering the impact of CPU usage. As evaluation unveil that execution time increases as the workloads intensify on the physical hosts.

Table 5.4: Execution Time

Execution Time	Static		Random	
	Nova Scheduler	SMRS	Nova Scheduler	SMRS
Edge1	1669	1665	1667	940
Edge2	1667	1663	1664	800
Edge3	1668	1661	1668	1160
Edge4	1664	1662	1669	1179

5.4 Data Collection For Model Validation

The validation results via mathematical modeling are presented in table 5.5. Using the A Mathematical Programming Language (AMPL) the unbalanced load distribution problem is solved in order to achieve the balanced load objectives. Using the DMRS mathematical model as discussed in Chapter 3 the validation model presents the distribution o VMs and the three physical hosts. The terms mathematical model and validation model are used interchangeably.

Table 5.5 is comprised of VMs placement vs time slots. Based on time slot 1 the time in seconds is computed as t $\times$ 60 $\times$ 5 in seconds. Each time slot represents the 5 minuets time. At time slot 1 two VMs are deployed on Edge2, and Edge1. While studding the number of VMs deployment parameter first VM is continuously deployed on Edge2 while satisfying the operational constraint of validation model. The second Vm is deployed on Edge1 and complete its execution is first 5 slots. The VMs including third, fourth, fifth,

University of Malaya

sixth, and seventh are deployed on Edge3, Edge2, Edge3, Edge1, Edge1, and Edge2 as discussed in table with the representation of E1, E2, and E3. Based on the VM creation specification three VM launched using the 3vCPUs. Based on the deployment sequence the fourteenth, twenty third, twenty seventh VMs are deployed with 3vCPUs whereas other VMs are deployed by exactly using the of 1vCPU.

Table 5.5: Time Slots

	Number of Time Slots																							
# of VMs	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
VM 1	E2	E2	E2	E2	E2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
VM 2	E1	E1	E1	E1	E1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
VM 3	0	E3	E3	E3	E3	E3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
VM 4	0	E2	E2	E2	E2	E2	E2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
VM 5	0	E3	E3	E3	E3	E3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
VM 6	0	0	E1	E1	E1	E1	E1	E1	E1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
VM 7	0	0	0	E2	E2	E2	E2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
VM 8	0	0	0	E1	E1	E1	E1	E1	E1	E1	E1	E1	0	0	0	0	0	0	0	0	0	0	0	0	0
VM 9	0	0	0	0	E2	E2	E2	E2	E2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
VM 10	0	0	0	0	0	E1	E1	E1	E1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
VM 11	0	0	0	0	0	E2	E2	E2	E2	E2	E2	E2	0	0	0	0	0	0	0	0	0	0	0	0	0
VM 12	0	0	0	0	0	E1	E1	E1	E1	E1	E1	E1	E1	E1	0	0	0	0	0	0	0	0	0	0	0
VM 13	0	0	0	0	0	0	E2	E2	E2	E2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
VM 14	0	0	0	0	0	0	E3	E3	E3	E3	E3	E3	0	0	0	0	0	0	0	0	0	0	0	0	0
VM 15	0	0	0	0	0	0	0	E2	E2	E2	E2	E2	E2	E2	0	0	0	0	0	0	0	0	0	0	0
VM 16	0	0	0	0	0	0	0	0	E3	E3	E3	E3	E3	E3	E3	E3	0	0	0	0	0	0	0	0	0
VM 17	0	0	0	0	0	0	0	0	E1	E1	E1	E1	E1	E1	E1	E1	0	0	0	0	0	0	0	0	0
VM 18	0	0	0	0	0	0	0	0	E3	E3	E3	E3	E3	E3	E3	0	0	0	0	0	0	0	0	0	0
VM 19	0	0	0	0	0	0	0	0	0	E3	E3	E3	E3	E3	E3	E3	0	0	0	0	0	0	0	0	0
VM 20	0	0	0	0	0	0	0	0	0	E3	E3	E3	E3	E3	E3	E3	E3	0	0	0	0	0	0	0	0
VM 21	0	0	0	0	0	0	0	0	0	E1	E1	E1	E1	E1	E1	E1	E1	E1	0	0	0	0	0	0	0
VM 22	0	0	0	0	0	0	0	0	0	0	E2	E2	E2	E2	E2	E2	E2	E2	0	0	0	0	0	0	0
VM 23	0	0	0	0	0	0	0	0	0	0	E2	E2	E2	E2	E2	E2	E2	0	0	0	0	0	0	0	0
VM 24	0	0	0	0	0	0	0	0	0	E3	E3	E3	E3	E3	E3	E3	E3	E3	E3	0	0	0	0	0	0
VM 25	0	0	0	0	0	0	0	0	E2	E2	E2	***E2***	***E3***	E3	E3	E3	E3	0	0	0	0	0	0	0	0
VM 26	0	0	0	0	0	0	0	0	E3	E3	E3	E3	E3	E3	E3	E3	0	0	0	0	0	0	0	0	0
VM 27	0	0	0	0	0	0	0	0	0	E2	E2	E2	E2	E2	E2	E2	0	0	0	0	0	0	0	0	0
VM 28	0	0	0	0	0	0	0	0	0	E1	E1	E1	E1	E1	E1	E1	0	0	0	0	0	0	0	0	0
VM 29	0	0	0	0	0	0	0	0	0	E1	E1	E1	E1	E1	E1	E1	E1	0	0	0	0	0	0	0	0
VM 30	0	0	0	0	0	0	0	0	0	0	E3	E3	E3	E3	E3	E3	E3	0	0	0	0	0	0	0	0
VM 31	0	0	0	0	0	0	0	0	0	0	E1	E1	E1	E1	E1	E1	0	0	0	0	0	0	0	0	0
VM 32	0	0	0	0	0	0	0	0	0	0	E1	E1	E1	E1	E1	E1	E1	0	0	0	0	0	0	0	0
VM 33	0	0	0	0	0	0	0	0	0	E1	E1	E1	E1	E1	E1	E1	0	0	0	0	0	0	0	0	0
VM 34	0	0	0	0	0	0	0	0	0	0	E2	E2	E2	E2	E2	E2	E2	E2	0	0	0	0	0	0	0
VM 35	0	0	0	0	0	0	0	0	0	0	E3	E3	E3	E3	E3	E3	E3	0	0	0	0	0	0	0	0
VM 36	0	0	0	0	0	0	0	0	0	0	E1	E1	E1	E1	E1	E1	E1	0	0	0	0	0	0	0	0

The VMs with the 3 vCPUs are deployed on Edge3, Edge2, and Edge2. Total number of twelve VMs are deployed on Edge2 and Edge3 whereas Edge1 is showing the thirteen VMs in total. Based on the time slots, at time slot 1 VM is running at Edge1 while 2 VMs are allocated on Edge2, and Edge3. In addition, the equal number of VMs are deployed

University of Malaya

at time slot 4. At slot 5 and 6 there are four VMs are running on Edge2, three VMs are running on Egde3 whereas the Edge1 is showing three and four VMs, respectively. At slot 7 one more VM (thirteenth VM) is launched on Edge2 while for Edge3 one VM (fifth VM) has completed its execution. Edge3 is showing that nine VMs are deployed on Edge3. Moreover, time slots 11 and 12 are that servers resources based on number of vCPUs are fully utilized by the VMs.

Considering the number of VMs parameter in first column, each VM from eight to twenty four completed their execution on the single server showing that the load is balanced even if the new VMs are launched or the exiting VMs are terminated once their computations are completed. The VM twenty fifth shows that for the starting four slots (from time slot 9 to 12) when the VM is deployed it starts its executions on Edge2 and then it is migrated to Edge3 in order to balance the load on Egde2. After the migration, at time slot 13 and 14 the load of Edge2, Edge3, and Edge1 is balanced.

University of Malaya

5.5 Data Collection for Dynamic Multi Resource based Scheduler (DMRS)

In this study data is collected for the proposed Dynamic Multi Resource based Scheduler to evaluate its performance based on the CPU utilization, Application execution, and number of migrations. The VM deployment is same as presented in the previous section using the DMRS algorithm. This behavior and CPU utilization of this study validates the correctness of the mathematical model.

5.5.1 Impact of Dynamic Load based VM Distribution on CPU Utilization

This section presents the CPU utilization behavior of three compute nodes Edge1, Edg2, and Edge3. The evaluated data is collected based on the VM execution intervals when the VMs are deployed on the specific physical servers while satisfying the balanced load criteria. In DMRS VMs are deployed using the different numbers of vCPUs and each VM is running with the 100% utilization for its required vCPUs. The data collection for the three different compute nodes is discussed for the thirty data traces and average, standard deviation, and confidence interval is computed to show the significance of the conducted studies.

Table 5.6: Impact of VM execution interval on time CPU utilization for Edge

VM Execution Interval (seconds)	**120**	**240**	**360**	**480**	**600**	**720**	**840**	**960**	**1080**	**1200**	**1320**	**840**
Data Trace-1	18.7	40.7	55	67.7	95.6	94.8	95.8	94	95.2	96.4	94	94.1
Data Trace-2	20.7	42.8	55	66.6	96.6	95.6	95.8	94.8	94.9	96.4	93.9	94.6
Data Trace-3	17.2	46.1	55.7	62.2	96.8	96.1	96.2	95.3	94.4	96.1	92.9	96.2
Data Trace-4	16.6	40.9	54.7	66.6	94.4	95.7	95.3	92.7	92.4	95.6	95.6	96.4
Data Trace-5	20.7	39.8	54.5	60.6	95.4	95.5	95.3	94.8	96.2	95.5	95.6	88.9
Data Trace-6	18	43.6	55.7	62.7	95.7	96.2	95.9	94.6	95.3	95.7	96	87.9
Data Trace-7	19.4	49.4	58.5	59.2	95.8	96	95.8	95.1	95.2	95.3	94.6	88.1
Data Trace-8	17.7	40.6	58.1	64.9	96.6	96.9	95.7	95.4	93.5	95.8	94.4	89.4
Data Trace-9	28.5	43.3	53	64.9	96.1	96.4	95.4	94	91.2	95.6	94.1	87.1
Data Trace-10	16.5	40.9	57.9	61.8	96.3	95.8	95.5	94.4	90.3	96.3	94.8	88.2
Data Trace-11	20.7	46.5	55.8	57.6	96.5	96.3	95	94.9	89.2	96.3	94.1	87.4
Data Trace-12	19.5	46.9	55.5	56.8	96.3	96.5	95.6	95.6	89.7	96.6	94.2	87.3
Data Trace-13	16.3	44	50.5	56.8	96.6	96.3	96.7	95.6	89.2	95.9	95.3	84.2
Data Trace-14	19.9	46.8	57.1	66.8	96.1	96.4	94.5	95.9	89.1	95.4	94.7	87.6
Data Trace-15	17.7	45	50.5	64.6	95.6	96.1	96.2	95.4	89.9	95	93.1	86.8
Data Trace-16	21.4	43.3	53.8	64.1	96.3	96.1	95.8	96.1	90.9	95.4	93.2	89.8
Data Trace-17	21.4	42.4	54.3	62.9	96	96.4	95.7	95.6	89.8	95.5	93.9	86.5
Data Trace-18	18.3	40	54.4	60.7	96.3	96.1	95.6	97.3	89.3	95.9	94	86.9
Data Trace-19	17.3	42.3	55.1	62.9	95.8	94.8	96.6	96.9	86.3	96.5	91.4	82.4
Data Trace-20	22.1	42.6	57.9	66.6	96.5	94.8	95.8	96.3	85	95.9	93.6	86.7
Data Trace-21	18.3	40	54.4	60.7	96.3	96.1	95.6	97.3	89.3	95.9	94	86.9
Data Trace-22	17.3	42.3	55.1	62.9	95.8	94.8	96.6	96.9	86.3	96.5	91.4	82.4
Data Trace-23	22.5	43.8	55.5	62.6	95.9	94.9	95.4	95.5	87.7	94	93	85.7
Data Trace-24	22.9	44.4	54.9	63.6	96.4	93.9	96.1	95.9	95	95.7	94	85.1
Data Trace-25	17.2	40.4	55.5	64.6	96.7	94.3	94.6	95.5	94.1	96	94.6	83.6
Data Trace-26	16.7	43.8	55.7	62.1	96.2	95.4	94.8	96	93.6	95.4	94.7	84.5
Data Trace-27	19.9	45.2	50.4	66.9	95.8	95.6	96	94.8	94.8	95.5	95	86.7
Data Trace-28	20.8	39.8	55.8	66.4	94.7	95	95.2	94.6	95.1	94.1	95.5	84.9
Data Trace-29	18.3	39.6	55.8	62.9	96.3	94.4	95.9	95.1	95.2	93.8	94.7	84.3
Data Trace-30	20.8	39.8	55.8	66.4	94.7	95	95.2	94.6	95.1	94.1	95.5	84.9
Mean	19.39	43	55.03	63.09	96.04	95.62	95.66	95.38	91.65	95.65	94.14	87.60
Standard Deviation	2.61	2.56	2.03	3.02	0.55	0.77	0.54	1.01	3.25	0.71	1.09	3.67
Confidence Interval	±0.97	±0.95	±0.75	±1.12	±0.20	±0.28	±0.20	±0.38	±1.21	±0.26	±0.41	±1.37

Table 5.6is comprised of thirteen columns. Based on the VM execution intervals the CPU utilization values are captured. The load values changes at time interval t whenever the new instance is launched and load is generated on that VM for each compute node. At time interval 120 the average value of the CPU utilization is shown as 19.39%. It presents that at that interval one VM is deployed on the Edge one and the CI is represented for the VM is ± 0.97 based on the standard deviation 2.61. The estimated CI values

for the specific deployed VMs are presented as 0.95, ±0.75, ± 1.12, ± 0.20, ±0.28, ± 0.20, ± 0.38, ±1.21, ±0.26, ±0.41,and ±1.37. The Edge1 is showing maximum CPU utilization at time interval 720, 840, 960, 1080, 1200, and 1320. At that time VMs are utilizing the 100% of the CPU resources. And Edge1 is fully loaded while utilizing the overprovisioning criteria.

Table 5.7: Impact of VM execution interval on CPU utilization for Edge2

VM Execution Interval (seconds)	**120**	**240**	**360**	**480**	**600**	**720**	**840**	**960**	**1080**	**1200**
Data Trace-1	20.7	47.2	51.9	95.5	92.3	94.3	95.8	92.2	64.3	37.4
Data Trace-2	18.9	45.9	50.8	94	93.6	94.1	97	89.4	63.7	34.2
Data Trace-3	20.6	40.7	48	95.7	93.1	94.3	97.1	89.1	63.3	36.6
Data Trace-4	17	43.3	52	96	94.8	94.8	95.2	88.8	68.9	36.3
Data Trace-5	23.9	40.5	52.1	95.8	93.6	94.5	96.2	89.7	64.1	32.5
Data Trace-6	21.3	44.1	48	95.8	91.7	93.7	96.2	90	66.5	19.5
Data Trace-7	20.7	44.4	51.3	96.1	90.3	92.7	95.9	90	65.3	18.1
Data Trace-8	16.4	44.3	52	96.2	92.8	93.7	97.1	86	65.6	16.1
Data Trace-9	17.2	45.3	51.6	96.3	94.1	94	96.2	86.8	63.1	16.6
Data Trace-10	21.7	42.7	47.3	95.2	96.5	94.4	95.5	91	63.4	17.3
Data Trace-11	23.1	45.8	53.3	93.9	96.8	94.3	95.4	88	71.2	20.8
Data Trace-12	19.4	41.2	50.8	94.6	96.6	94.9	94.4	88.6	69.2	19.8
Data Trace-13	19.7	41.1	50.6	94.1	97.2	94.4	95.7	88.4	62.8	23.2
Data Trace-14	21.1	40.8	49.3	94.7	96.1	95.4	95.1	87.8	63.4	16.5
Data Trace-15	21.8	43.4	48.1	93.9	96.3	95.2	94.8	88.8	63.3	17.2
Data Trace-16	19.1	45.6	48.4	94.4	96.3	95.8	95.1	92.1	64.5	17.6
Data Trace-17	20.9	43.8	52.5	95.6	96.8	96.2	94.6	92.7	57.3	17.2
Data Trace-18	20.3	42.5	52.2	93.8	96	96.3	95.8	91.9	61.2	21.7
Data Trace-19	20.8	45	52.4	96	96.1	94.9	95.5	93.7	61.5	24.2
Data Trace-20	16.7	48.9	52.3	95.3	94.1	95.1	93	93.7	63.4	22.7
Data Trace-21	19	44.9	52.4	93.7	95.5	94.7	90.9	94.9	59.2	27.5
Data Trace-22	19.8	46.4	49.2	94.4	95.8	95.3	93.3	94.3	55.8	20.6
Data Trace-23	19.5	40.5	47.5	92.2	95.3	95.8	95	92.3	52.9	21.3
Data Trace-24	24.8	40.5	52.6	94.9	95.8	94.3	96.1	83.3	59	25.3
Data Trace-25	19.5	39.9	47.5	95.9	96.1	95.4	95.7	86.3	54	17.1
Data Trace-26	22.1	42.5	48.1	96.8	96.4	95.9	95.5	87.9	53.8	16.9
Data Trace-27	19.8	44.8	49.2	95.8	94.9	96.1	95.3	86.7	53.9	16.8
Data Trace-28	20.8	46.8	52	95.3	97.4	96.4	95.6	85	60.5	17.2
Data Trace-29	21.6	40.4	52.1	95.8	96.2	95.9	95.6	90.8	57	19.3
Data Trace-30	21.7	48.9	52.3	95.3	94.1	95.1	93	93.7	63.4	22.7
Mean	20.28	43.55	50.53	95.09	95.12	94.92	95.33	89.66	61.79	22.32
Standard Deviation	1.98	2.44	1.95	1.03	1.78	0.90	1.25	2.90	4.78	6.75
Confidence Interval	±0.73	±0.91	±0.72	±0.38	±0.66	±0.33	±0.46	±1.08	±1.78	±2.52

University of Malaya

Table 5.7 presents the distribution of VMs in the Edge2 while considering the impact of VM execution interval on CPU utilization. When the first VM is place on Edge2 the computed average is presented as 20.28% od the CPU usage. This table is only presents the VMs deployed on Edge2. However, the VMs deployed on that servers based on the

CPU utilization of the others servers to balance the load in cloud. The average load values is computed as 20.28%, 43.55%, 50.53%, 95.09%, 95.12%, 94.92%, 95.33%, 89.66%, 61.79%, 22.32% based on the time interval varying from 120 to 1200 with the different number of VMs. In addition, teh confidence interval plotted for the computed load average such as ±0.73, ±0.91, ±0.72, ±0.38, ±0.66, ±0.33, ±0.46, ±1.08, ±1.78, and ±2.52. The CI is showing less the 1 for the first seven time intervals. For the time inter eight, nine, and tenth the confidence CI value is reported > 1. Theses values shows that all VMs are started before the time interval 600; therefore, the CPU utilization is upto 99%. As the average and CI is computed for the 30 iterations the VMs have completed their executed little bit faster in these interval which shows that the CPU utilization values is varying from 85% to 93.7%, 54% to 71.2%, and 17.1% to 34.2% at time interval 900, 1080, and 1200.

Table 5.8 presents the values of VM execution at time t. The load is varies based on the CPU resources because number of other VMs are deployed on one server to perform their computations. The load values are changing when number of VMs are allocating to the cloud. At time inter 120 the load value varies from 16.8% to 30.2%, which shows that the data traces are recoded based on thirty value. When the first data trace is recorded the CPU usage low as compare to other data traces thats shows that at this time only one VM is deployed on Edge3. Moreover, for values from data traces 2 to 30 the 2 VMs are executed on that machine in first time slot. The difference is reported because top command save the values after every 3 seconds; therefore, the exact value is computed based on the average of 30 runs. Based on the average values of CPU utilization recorded as 29.05%, 29.88%, 28.16%, 17.85%, 70.78%, 95.7%, 95.09%, 96.36%, 96.59%, and 88.91%, the CI is presented with the significant results for that values ±1.13, ±0.83,

University of Malaya

Table 5.8: Impact of VM execution interval on time CPU utilization for Edge3

VM Execution Interval (seconds)	**120**	**240**	**360**	**480**	**600**	**720**	**840**	**960**	**1080**	**1200**
Data Trace-1	16.8	29.6	29.1	15.3	71.5	96.3	95.6	95.9	96.1	96.3
Data Trace-2	27.6	28.2	29.3	19.6	74.4	94.3	96.3	96.5	96.2	96.2
Data Trace-3	27.4	28.2	26.8	16.2	71.4	95.2	96.1	94.2	95.6	95.6
Data Trace-4	30.3	28.9	26.9	15.8	70.4	94.6	97.9	96.2	94.8	96.5
Data Trace-5	27.6	29	31.4	16.3	69	94.8	97.5	96.9	95.7	95.3
Data Trace-6	29.4	27.7	27.6	15.7	69.8	97	94.9	97.3	94.8	93.9
Data Trace-7	29.2	34.3	33.3	18.1	70	97.8	94.7	98.4	97.4	93.8
Data Trace-8	33.2	31.6	28.6	20.4	73.6	96.1	94.6	97.3	96.3	93.7
Data Trace-9	29.7	33.9	33.2	15	71.8	95.4	94.5	97.9	96.4	94.2
Data Trace-10	30	28.8	30.3	25.6	72.7	95.5	94.2	97.3	94.8	95.5
Data Trace-11	26.6	30.2	28.3	19.7	71.6	95.8	94.5	97	98.2	95.5
Data Trace-12	29.6	27.9	29.2	15.3	71	95	95.4	97.5	97.5	95.9
Data Trace-13	31.8	33.6	30.3	17.8	67.8	93.9	94.6	97.3	96.7	94.3
Data Trace-14	29.4	31.4	28.3	19.3	65.4	95.4	96.4	96.8	98	96.8
Data Trace-15	27.2	29.7	30.7	17.8	67.6	96.9	95.7	97.3	96.6	84.3
Data Trace-16	27.8	26.7	29.5	19.4	73.5	94.6	96	96	96.7	86.4
Data Trace-17	30.1	30.3	31	19.3	71.4	95.7	95.9	95.3	96.9	81.2
Data Trace-18	26.1	27.5	28.1	15.7	66.7	96	93.8	96	97.3	83.2
Data Trace-19	27.1	27.4	26.9	15.7	70.3	96.8	95.4	95.3	95.5	82.4
Data Trace-20	29.9	29.6	24.7	18.3	72.6	96.5	96.9	95	96.4	87.9
Data Trace-21	28	30.7	29.2	16.1	70.4	95.2	89.3	95.4	96.8	82.4
Data Trace-22	31.6	27.8	28.2	15.6	67.8	94.5	91.5	96.2	96.6	88
Data Trace-23	31.5	31.8	25.6	19	70.9	95	88.7	95.6	96.7	82.1
Data Trace-24	29.3	35.1	24.5	17.8	71.9	97.5	95.7	96.4	97.5	81.8
Data Trace-25	32.6	27.6	21.9	18.4	77	97.4	95.6	95.6	97.3	81.1
Data Trace-26	34	29.8	28.5	19.6	72.9	96.9	97.6	95.9	96.6	85
Data Trace-27	27.4	30.7	28.2	19.6	68.8	96.2	95.6	96	97.8	81.8
Data Trace-28	30.6	29	28	18.5	72.8	94.3	95.1	95.8	97.2	81.8
Data Trace-29	29.6	31.6	19.7	17.1	70.7	95	97.3	96.3	96.6	82.3
Data Trace-30	30.2	27.9		17.6	67.9	95.4	95.6	96.4	96.8	52.1
Mean	29.05	29.88	28.16	17.85	70.78	95.7	95.09	96.36	96.59	88.91
Standard Deviation	3.04	2.22	2.88	2.20	2.47	1.04	2.09	0.93	0.89	6.292
Confidence Interval	±1.13	± 0.83	±1.07	±0.82	±0.92	±0.39	±0.78	±0.34	±0.33	±2.34

±1.07, ±0.82, ±0.92, ±0.39, ±0.78, ±0.34, ±0.33, and ±2.34. Moreover, the CPU is fully utilized at time interval 720, 840, 960, 1080, and 1200. It shows that the VMs running inside the Edge3 using the CPU resource in order to complete their computations. The CPU utilization effects the application execution time because its resources are shared among the VMs. The impact of CPU utilization on execution time is studied in next section.

5.5.2 Application Execution Time

Table 5.9 presents the application execution time conducted for the legacy OpenStack nova scheduler and proposed DMRS method. The experiments environment is same in order to evaluate the CPU utilization behavior when VMs are running inside the physical

University of Malaya

servers to complete their execution. The table first column and first row of the table shows the number of times slots and CPU load of existing and proposed solution. The second, third, and forth column presents the data collected using the existing nova scheduler. Beside the last three columns shows the CPU usage for proposed DMRS methods. The time is captured based on the usage of CPU percentage which reflects the information regarding how many tasks are executing at time instance t. While considering the data of default nova scheduler the overall workload distribution of cloud is analyzed based on three physical server.

Table 5.9: Performance comparison of proposed DMRS model and Validation Model

Time Slots	**NovaSch Edge1**	**NovaSch Edge2**	**NovaSch Edge3**	**DMRS Edge1**	**DMRS Edge2**	**DMRS Edge3**
1	19.51	16.69	18.5	19.96	20	12.15
2	31.05	29.23	30.11	20.13	31	29.52
3	31.34	29.33	29.55	31.59	42	29.052
4	40.277	41.83	39.95	43.591	43	29
5	31.3	42.89	27.488	42.561	53	29
6	37.966	41.88	28.833	55.289	52	17
7	43.46	54.82	30.93	56	54	17
8	43.599	60.23	28.082	55	64	18
9	65.75	93.88	28.224	66	89	53
10	85.852	95.54	40.83	94	95	72
11	95.367	95.54	83.91	**95**	**95**	**92.84**
12	94.84	95.6	83.424	**95**	**95**	**95.45**
13	94.22	95.54	83.374	**95**	**95**	**95.18**
14	93.82	95.42	81.811	**94**	**94**	**95.29**
15	94.30	95.47	80.64	**94**	**95**	**95.36**
16	94.11	94.64	83.49	**94**	**94**	**95.57**
17	93.93	93.60	83.90	**94**	**92**	**95.97**
18	91.45	91.75	83.64	91	74	77.08
19	91.61	94.80	80.55	57	36	40.26
20	91.08	93.92	82.67	22	0	29
21	93.72	94.27	83.2	-	-	-
22	59.61	59.92	45.55	-	-	-
23	31.88	47.21	31.67	-	-	-
24	23.24	23.71	23.2	-	-	-
25	23.71	-	24.91	-	-	-
26	18.5	-	12.2	-	-	-
27	-	-	16.69	-	-	-
Total Execution Time (seconds)	**7800**	**8100**	**7200**	**6000**	**6000**	**5700**

For time slots 1 to 8, the load is continuously varying for each server. For time slots, 10 to 19 only two servers (Edge1 and Edge2) are showing the maximum CPU load with the load average of 95% whereas for remaining server (Edge3) the CPU usage is

University of Malaya

presented less then 83.90%. The deployment behavior of VMs shows that VMs are not fairly distributed. In contrast, proposed DMRS solution presents the stable CPU usage for each physical server from time slots 11 to 17. Moreover, it shows the minimum execution time such as 600, 600, and 5700 for Edge1, Edge2, and Edge3, respectively. Besides, the default scheduler shows the maximum execution time including 7800, 8100, and 7200 seconds for Edge1, Edge2, and Edge3, respectively. The analysis shows that proposed solution presents the minimum execution time and evenly distributes the workload when compared with the default solution. Moreover, from the analysis it is evaluated that inefficient CPU utilization affect the application execution time.

5.6 Data Collection For Performance Comparison of DMRS and Validation Model

The accuracy of validation model is evaluated by collecting the results obtained by mathematical model with the empirical results. The CPU utilization, numbers of migrations, application execution times metrics are used to validate the mathematical model.

5.6.1 Impact Of CPU Utilization on Application Execution Time

Table 5.10 represents the comparison of application execution time based on CPU utilization through experiments and validation model.The first column and first row of the table shows the time slots for which the CPU utilization is studied and the type of experiments, respectively. The application execution time is presented for different time slots for three physical servers. The VM deployment behavior of evaluation model is presented in table 5.5. Moreover, exactly the same deployment behavior is selected by our proposed DMRS algorithm.

The distribution behavior chosen by DMRS for every single server is presented in previous section. The sum of total time slots values is showing the overall execution time taken by the physical server for all VMs running inside that machine. The validation

University of Malaya

Table 5.10: Performance comparison of proposed DMRS model and Validation Model

Time Slots	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
Exp Edge1	19.96	20.13	31.59	43.59	42.56	55.28	56	55	66	94	95	95	95	94	94	94	94	91
Math Edge1	12.5	12.5	25	37.5	37.5	50	50	50	62.5	87.5	100	100	100	100	100	100	50	12.5
Exp Edge3	20	31	42	43	53	52	54	64	89	95	95	95	95	94	95	94	92	74
Math Edge3	12.5	25	25	37.5	50	50	62.4	50	62.5	87.5	100	100	100	100	100	100	75	12.5
Exp Edge2	2.15	29.05	29	29	17	17	18	53	72	92.84	95.45	95.18	95.29	95.36	95.57	95.97	77	40.2
Math Edge2	0	25	25	37.5	37.5	37.5	50	37.5	75	100	100	100	100	100	100	100	62.5	12.5

1. $Exp : Experiments, \quad Math : MathematicalModel$

model is showing vary small difference when compared with the experiments. The experiments are running for two more slots in order to complete the execution of running task. It is because in validation model there is no overhead is considered while running the number of processes while in real time experiments a top command is executed to continuously monitor the CPU behavior which shows the CPU usage from 0.9% to 4.5%. While considering the VMs distribution no VM is deployed and executed on Edge3 in time slot 1; therefore' the validation model Edge3 is showing 0 value whereas the experiments are showing 2.15%.

Moreover, based on the load balancing constraint in validation model the load is balanced if the difference among all the physical servers is reported less than or equal to 1 VM. The difference of 1 vCPU is reported in time slots 1 to 3 for Edge1 which shows the load is balanced as analyzed through experiments and validation model. In addition, when the all VMs are deployed and used the CPU resources the mathematical model presents the 100% CPU utilization while for experiments when the value is > 90 it shows that CPU is fully utilized. For time slots 4 to 16 the CPU load is same with the equal numbers of VMs for Edge1. With respect to the time slots the CPU utilization for validation model it is reported as 37.5%, 37.5%, 50%, 50%, 50%, 62.5%, 87.5%, 100%, 100%, 100%, 100%, 100%, and 100 while the experiments shows the CPU usage such as 43.59%, 42.56%, 55.2%, 89%, 56%, 55%, 66%, 94%, 95%, 95%, 95%, 94%, 94%, and

University of Malaya

94%.

At time slot 11 to 16, Edge1, Edge2, and Edge3 are fully loaded for both validation model and the experimental models. Moreover, the time slots with the balanced load factor and with the difference of 1 vCPU are highlighted. These results shows that the results conducted using the mathematical model are significant when compared with the experiments. The load behavior using the mathematical model is same as observed with the experiments which proves the validity of mathematical model.

5.6.2 Analysis of Application Execution Time for Individual VMs

Table 5.11 presents the comparison of empirical and validation model results based on execution time parameter for individual VM deployed inside the cloud on multiple servers. The first row and first column of the table depicts the numbers of VM in the cloud and application execution time. The application execution time is studied using the experiments and validation models results. Moreover, to validate the model results the percentile difference is also calculated. The VMs execution is reported based on their deployments as reported in table 5.5. In experiments the VMs runs the CPU bound intensive program to generate the load inside the VMs. In contrast, in order to execute the program inside the VMs script is written which produced the same load as generated by the experiments. While considering the Table 5.5, for first VM deployed on Edge2 it is running for the continues 5 slots. Each slots represents the 5 minutes time. When fist VM deployed there is no other resources are running; theretofore; the time computed by the experiments is presented as 1551 whereas the validation model shows the 1500 values for the same VM. The percentile difference of theses 2 values shows the minimum difference such as 3.28%. The percentile difference < 15% is accepted while satisfying the load balancing criteria because the load is computed while observing the load balancing constraint in DMRS validation model.

University of Malaya

Table 5.11: VM deployment vs Application /Execution Time

Number of VMs	**Execution Time**		**Percentile Difference**
	Experiment	Validation Model	
Virtual Machine 1	1551	1500	3.28
Virtual Machine 2	1529	1500	1.89
Virtual Machine 3	1954	1800	7.88
Virtual Machine 4	1324	1200	9.36
Virtual Machine 5	2419	2100	13.18
Virtual Machine 6	1339	1200	10.38
Virtual Machine 7	3168	3000	5.30
Virtual Machine 8	1805	1800	0.27
Virtual Machine 9	1531	1500	2.02
Virtual Machine 10	2415	2400	0.62
Virtual Machine 11	3475	3000	13.66
Virtual Machine 12	1647	1500	8.92
Virtual Machine 13	1359	1200	11.69
Virtual Machine 14	2631	2400	8.77
Virtual Machine 15	3533	3300	6.50
Virtual Machine 16	3145	2700	14.14
Virtual Machine 17	3239	3000	7.37
Virtual Machine 18	2789	2700	3.19
Virtual Machine 19	2829	2700	4.55
Virtual Machine 20	3549	3000	15.46
Virtual Machine 21	3040	2700	11.18
Virtual Machine 22	2540	2400	5.51
Virtual Machine 23	3203	3000	6.33
Virtual Machine 24	3416	3000	12.17
Virtual Machine 25	3330	3300	0.90
Virtual Machine 26	3364	3000	10.82
Virtual Machine 27	2776	2400	13.54
Virtual Machine 28	3111	2700	13.21
Virtual Machine 29	3261	3000	8.00
Virtual Machine 30	2331	2100	9.90
Virtual Machine 31	2714	2700	0.51
Virtual Machine 32	2783	2400	13.76
Virtual Machine 33	2711	2400	11.47
Virtual Machine 34	2724	2700	0.88
Virtual Machine 35	2775	2700	2.70
Virtual Machine 36	1908	1800	5.66

The execution time reported by the experiments for second VM to the tenth VM such as 1529, 1954, 1324, 2419, 1339, 3168, 1805, 1531, and 2415. In addition, for the same numbers of VMs the validation model presents the execution time based on 1500, 1800, 1200, 2100, 1200, 3000, 1800, 1500, and 2400 seconds. The percentile difference reported for theses values is discussed such as 1.89%, 7.88%, 9.36%, 13.18%, 10.38%, 5.3%, 0.2%, 2.02%, and 0.62%. Although, VMs are deployed using 1 vCPU excepting the three VMs the execution time is not same for each VM. The reason is that VMs not running at same time, it depends on the numbers of VMs running at that time interval and sharing the CPU resources to complete their tasks. When the number of VMs are greater the execution time will exceed it also depend on the VM for how long time it was running and launched inside the server. The differences in evaluation and validation results are shown insignificant which advocates reliability and validity of our model. The model shows minimum amount as compare to the experiment results because there is no computation overhead is considered whereas the experiments presents the results with the amount of computations overhead. The differences in experiments and validation results are shown < 15%, which advocates reliability and validity of our model. The evaluation shows 85% accuracy of the mathematical results when compared with the experiments and percentile difference is computed for theses values.

5.7 Conclusion

In this chapter, proposed solutions SMRS, DMRS are evaluated by comparing it with benchmarks results. The benchmarking is done by collecting the results using the OpenStack filter scheduler (default scheduler). The data is collected by sampling the parameters for thirty data traces. The best estimation point is measured by calculating the mean of 30 values for each experiment which shows the significant results by computing the 95% confidence interval. Based on the exterminates the validate of mathematical model is also proved by observing the CPU utilization and application execution time.

It is concluded that SMRS, successfully reduce the application execution time when the load nature is random in side the cloud. Besides, with static load distribution behavior it shows the similar results as gathered by the legacy scheduler. The SMRS algorithms balanced the load when the VMs are initially placed in the cloud without considering the migrations. In addition, the DMRS presents dynamic load balancing and initially placed the VMs while considering the CPU utilization and also it fairly distributes the workload based on migration technique. DMRS presents the minimum execution time when its results are compared with the standard scheduler. The DMRS shows the performance

University of Malaya

gain while satisfying the minimum numbers of migration objectives in order to balance the workload. The accuracy of the optimized DMRS validation model is validated upto 85% when compared with the DMRS experiments results.

University of Malaya

CHAPTER 6: RESULTS AND DISCUSSION

This chapter validates the system model of proposed methods SMRS and DMRS against their empirical evaluation results. The performance of proposed solutions is compared with the benchmark solution. Moreover, the Mathematical model results of DMRS method are also compared with the empirical results of DRMS to validate the correctness of the proposed model. The evaluation parameters such as CPU utilization, execution time, and numbers of migrations are considered to analyze the performance of proposed solution.

This chapter is organized into six main section. Section 6.1 analyze the performance of proposed initial VM placement algorithm (SMRS) by comparing it results with default OpenStack scheduler. This section is further classified into two subsection. Section 6.1.1.1 analyze the VM distribution behavior based on the static load whereas section 6.1.1.2 studied the VM distribution while considering the random load as a factor. Moreover, Section 6.2 compares the DMRS method results with the benchmark scheduler based on the dynamic load. Section 6.3 validated the mathematical model results by comparing it with the experimental results of DMRS. Section 6.4 presents the performance analysis based on the results of purposed DMRS, optimized DMRS, and default scheduler. Finally, section 6.5 conclusively presents the main findings of the chapter.

6.1 Performance Evaluation of Proposed SMRS Method

This section evaluates the static multi resource based scheduler performance by comparing it with nova scheduler. In order to analyze the SMRS performance, CPU utilization and application execution time parameters are considered based on the number of VM provisioning on physical hosts.

University of Malaya

6.1.1 Analysis of CPU Utilization

In this section, CPU utilization behavior is studied based on two scenarios, including static and random load factor while the initial placement of VMs. In order to generate the load, a CPU intensive application is executed inside the VM. In static analysis, each VM executes the number of applications equal to its number of cores (vCPUs) allocated to that VM. Besides, considering the random load based distribution, load generator function will decide for how many cores the application needs to execute based on the values produced by the it.

6.1.1.1 Static Load Based Distribution Considering Load as a Factor

Fig. 6.1 shows the relationship between CPU utilization and VM deployment sequence on four physical servers, including Edge1, Edge2, Edge3, and Edge4. In said Fig, number of core utilization across the VM deployment sequence is represented to highlight that how the VMs are deployed on the physical servers at time interval t considering CPU utilization factor. In order to show their deployment on the physical hosts different shapes such as triangle, circle, diamond, and square are selected to show the Edge1, Edge2, Edge3 , Edge4, respectively. The figure shows that second, third, and forth VMs are placed on the Edge2, 4, and 3. While, conducting the static analysis as mentioned in chapter 3 the cloud environment is heterogeneous based on the system memory. The VM placement shows that like the default scheduler deployment as presented in Fig. 3.3, VMs are not place in a unique and recursive order. The VM fifth and sixth are deployed on Edge1 while considering the CPU usage based distribution. The VMs are distribution is selected by the proposed static load balancing algorithm using the load filter, compute filter, and load analyzer.

University of Malaya

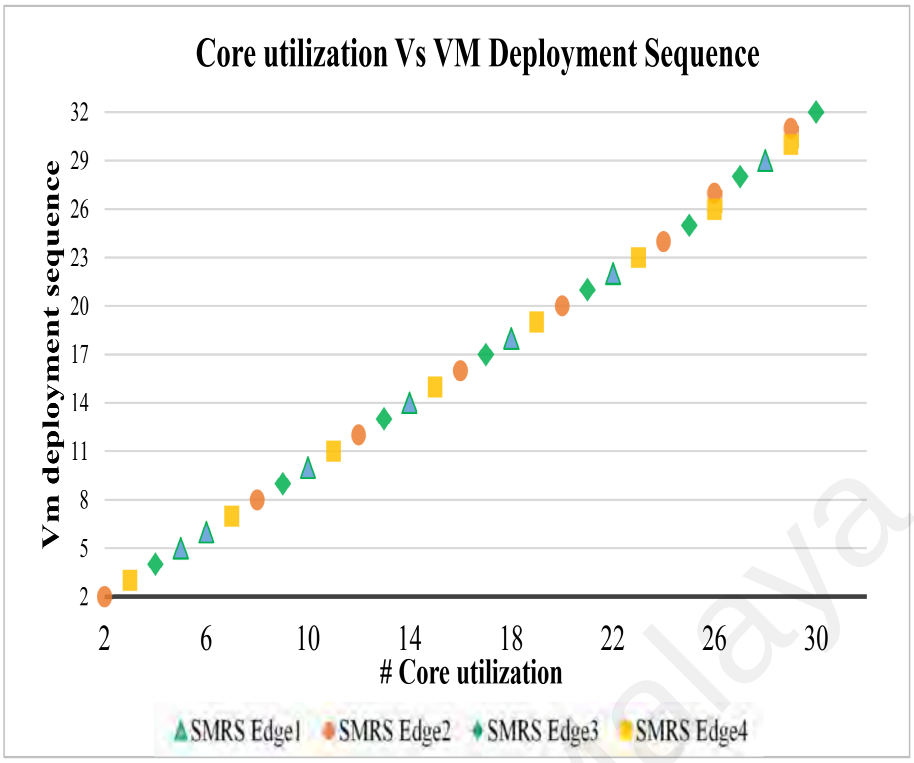


Figure 6.1: Core utilization vs VM deployment (Static Load Analysis)

University of Malaya

Fig. 6.2 depicts the CPU utilization when the number of VMs are deployed in cloud. The figure presents that at x-axis VM deployment is plotted using the deployment sequence presented in Fig. 6.1. At y-axis CPU utilization behavior is reported. When first VM is placed the load values is reported 9%, and 23% to 25%, for Edge1, Edge2, 4, and 3, respectively. At the deployment of sixth VM the CPU load is computed by the SMRS method and this VM is place on the least loaded host (Edge1). Seventh VM is also placed on the Edge1 because other three physical hosts are showing maximum CPU utilization when compared with the Edge1. Moreover, when the seventh, eight, ninth, and tenth VMs are placed the CPU usage is reported as 54% 53.5%, 52.5%, 50.9%, respectively. Each VM is deployed using 2 vCPUs and the physical CPUs, which are reported 8 for each physical host are completely allocated when the total 4 VMs are deployed on each server. Therefore, for VMs from fourteenth to onward the physical servers are showing the CPU utilization > 90%.

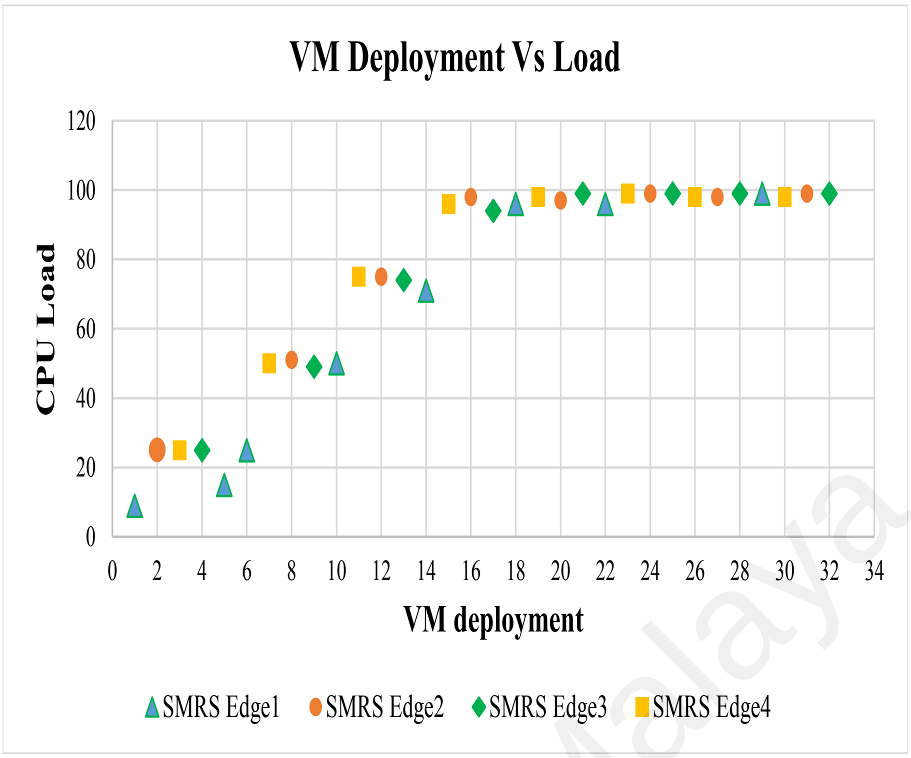


Figure 6.2: Core Utilization vs VM Deployment Sequence of SMRS

6.1.1.2 Random Load Based Distribution Considering Load as a Factor

This experiment used the same random load generator function as discussed in Fig. 3.5 to distribute a dynamic load. Using proposed SMRS method, first four VMs among a sequence of VMs are deployed on Edge 1, 4, 2, and 3 as shown in Fig. 6.3. This happened due to random load generation, which causes each VM to have dissimilar CPU utilization. This figure presents the VMs distribution behavior when the number of cores (vCPUs) requested by the VM at x-axis and y-axis. In order to balance the load, the seventh, eighth, and tenth VMs are placed on Edge2 whereas the twelfth and thirteenth VMs are placed on Edge1. The sequence shows that VMs are not deployed while considering the same placement behavior as studied in Fig. 3.5 in chapter 3. Moreover, in order to better understand the VMs deployment sequence, CPU utilization is plotted in Fig. 6.4

University of Malaya

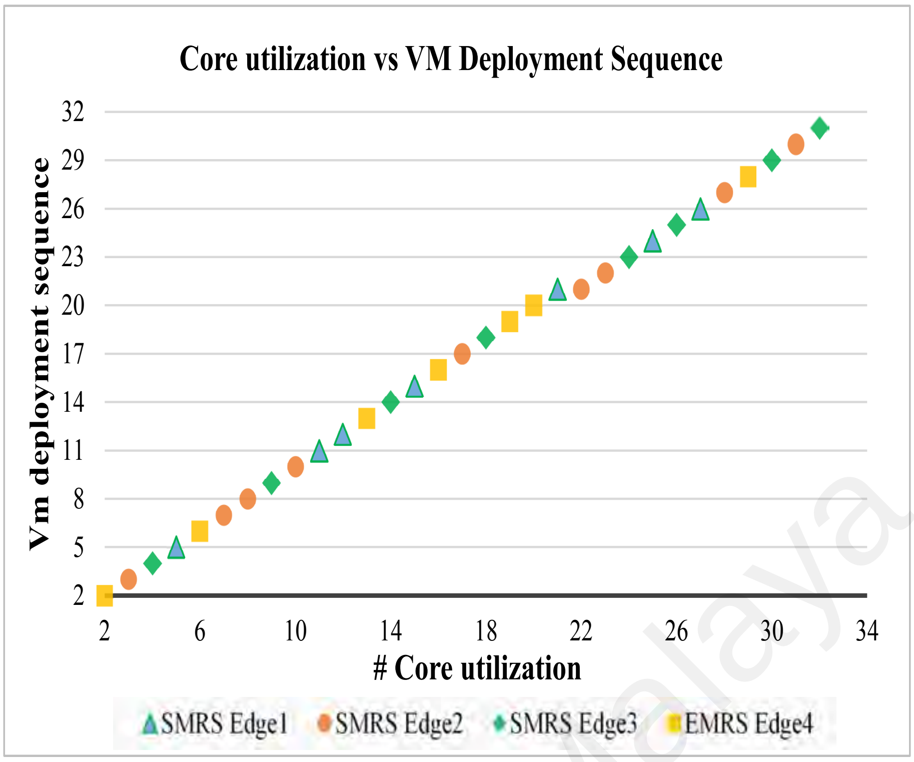


Figure 6.3: VM Distribution Behavior of SMRS (Random Load Analysis)

University of Malaya

In Fig. 6.3, VM deployment sequence is plotted across CPU usage at x-axis and y-axis. Fig. 6.3 represents that when initial VM is deployed on Edge1, the CPU utilization is 5%. After first VM placement, the other compute hosts are loaded with minimum load compared to Edge 1. However, the second, third, and forth VMs are placed on Edge 4, 2, and 3, respectively. After the deployment of first four VMs, GDE (Fig. 4.1) perceived that Edge1 has a minimum load. Therefore, it chooses Edge1 to place next incoming VM. Moreover, at the time of ninth and tenth VM, the CPU utilization is 23% and 45%, respectively. In the said figure, second VM on Edge2 presents minimum CPU utilization as no execution profile is running on it because of the lowest load generated by the random function generator.

Based on deployment sequence in Fig. 6.4, the CPU utilization is minimum for Edge2 as compared to rest of the nodes. Therefore, seventh and eighth VMs are deployed on Edge2 in order to balance the load factor. Furthermore, ninth VM that is second VM for Edge3 shows the maximum CPU utilization as it executes four different execution profiles. Therefor, after third, seventh, and eighth VM's deployment, Edge2 again leverages minimum load due to random load generator. As a result, tenth VM is also deployed on Edge2. Based on comparison VM deployment sequence of proposed algorithms is not same as in existing algorithm presented Fig. 3.5.

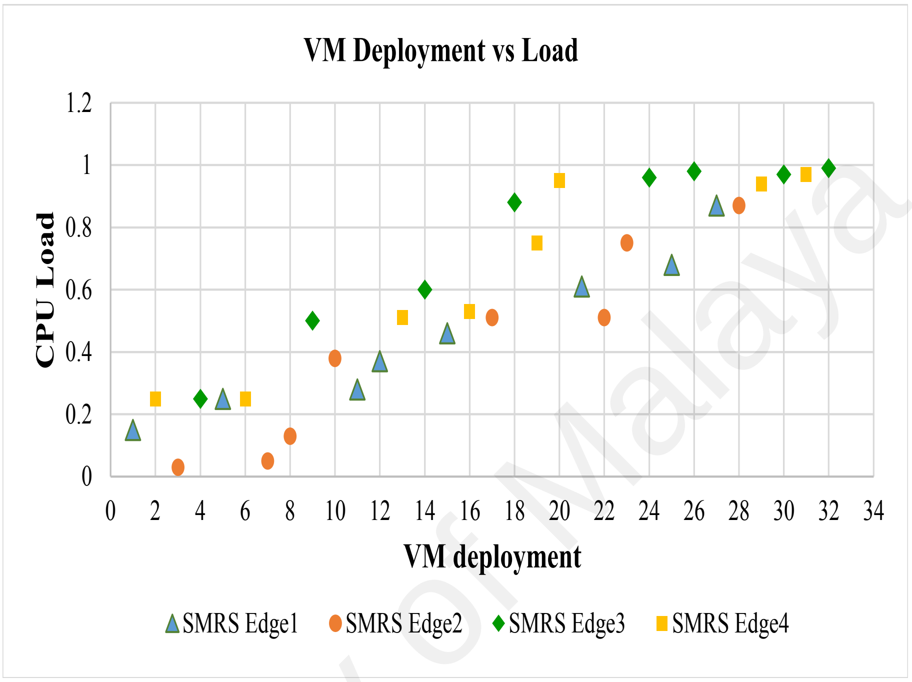


Figure 6.4: Core Utilization of SMRS (Random Load Analysis)

6.1.2 Analysis of Application Execution Time

This study analysis the application execution time while varying the CPU load as shown in Fig. 6.2 and Fig. 3.3 for static load based distribution and Fig. 6.4 and Fig. 3.5 for the random load based distribution. Based on static load distribution, as in Fig. 6.2 and Fig. 3.3 application execution time within VM is plotted in order to check the performance of proposed SMRS method and existing nova scheduler. In order to understand the results we have plotted the graph of each compute node where SMRS Edge 1, 2, 3, and 4 represents the results proposed algorithm while NovaScheduler Edge 1, 2, 3 and 4 presets the results of existing OpenStack default scheduler. Fig. 6.5 shows that existing, and proposed SMRS scheduler behavior is same when the load distribution nature is based on uniform or static load. The difference between proposed and existing algorithm

University of Malaya

is captured as a small amount of time in seconds. Each physical server is taking the 1660 to 1669 seconds in order to complete the tasks running inside the VMs.

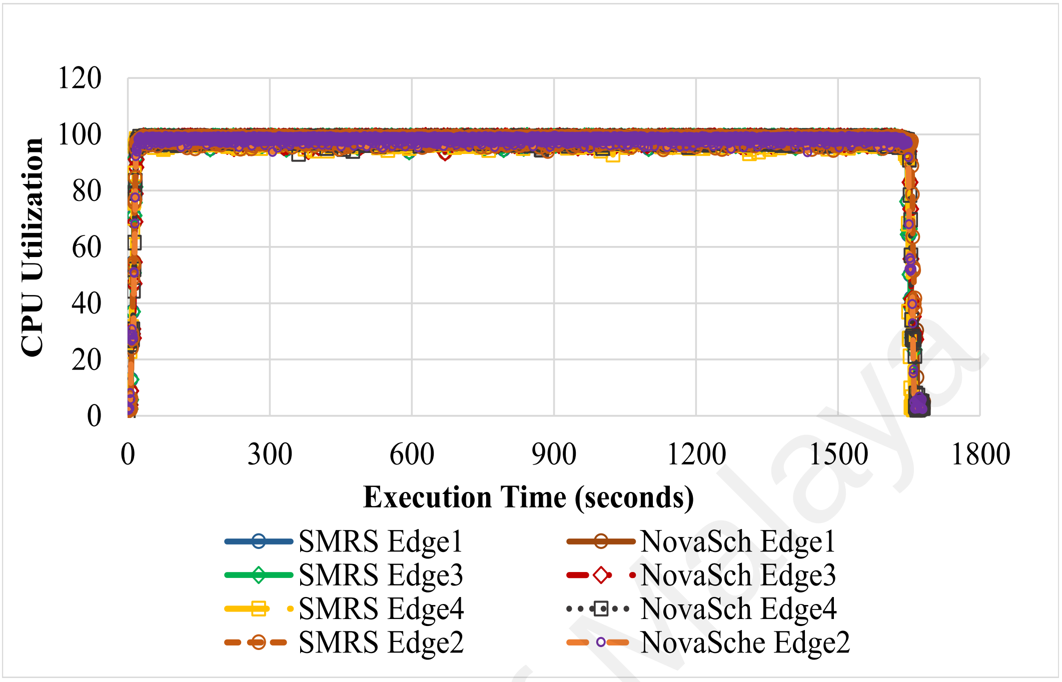


Figure 6.5: Execution Time vs CPU Utilization of SMRS based on Static Load

Moreover, Fig. 6.6 is reflecting the minimum execution time for each compute node when the VMs are deployed based on random distribution. In comparison of existing scheduler, our proposed scheduler method enhanced the performance of Edge1, Edge2, Edge3, and Edge4 up-to 33%, 50%, 33%, 44%, respectively.

University of Malaya

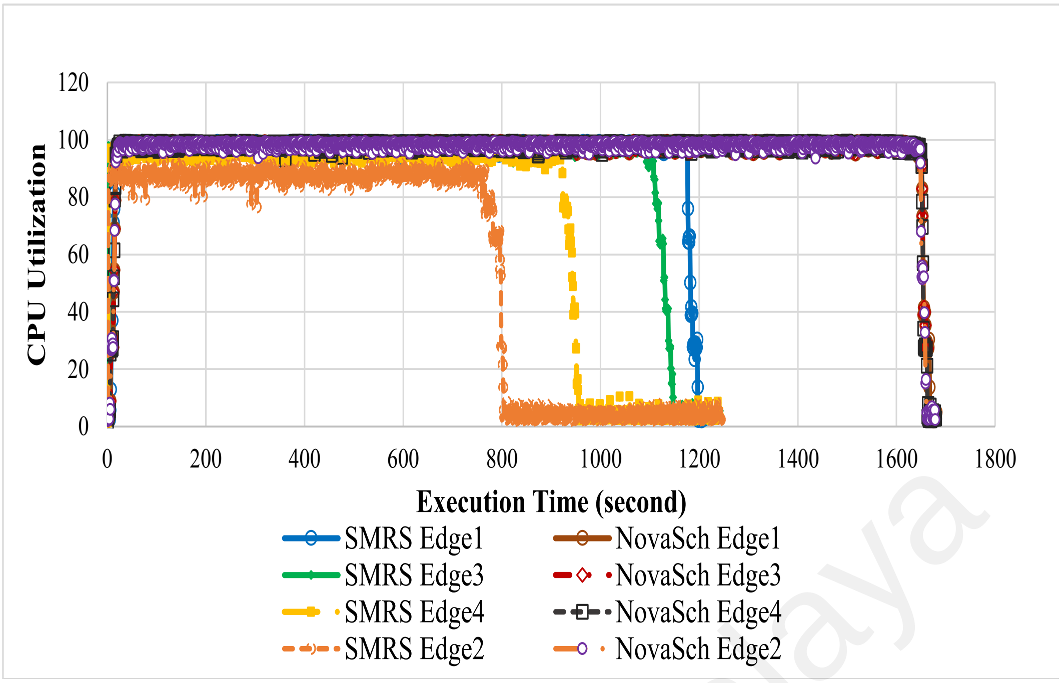


Figure 6.6: Execution Time vs CPU Utilization of SMRS based on Random Load

University of Malaya

Based on execution time analysis (Fig. 6.5 and Fig. 6.6), it is concluded that SMRS efficiently utilize the CPU resources when the load distribution is random. It is observed that when the load nature is static the proposed and existing schedulers uniformly utilize the CPU resource and the execution time taken by each server to complete the task is same. The application execution time is similar when the CPU load is kept similar for each VM. In general increase in CPU usage is directly increase the application execution time because the CPU resources are shared among the deployed VMs. In addition, when the load is distributed randomly the CPU utilization of each machine is not same as reported in Fig. 6.4. Therefore, based on distribution behavior each physical server is taking different time to execute the application running on that physical host. However, the data observed in case of random load based distribution scenario proves that the execution time is not only depended on the CPU usage but also the placement and deployment sequence criteria for the allocation of VMs to physical host based on load factor.

6.2 Performance Evaluation of Proposed DMRS Method

This section analyze the performance of DMRS method based on the dynamic load factor. This analysis presents the impact of CPU utilization on the application execution time.

6.2.1 Analysis of CPU Utilization over Execution Time

Fig. 6.7 presents the execution time as time slots at x-axis and y-axis show sows the CPU utilization. This figure shows that how the application execution time affect the CPU utilization in dynamic environments when the new VMs can be launched any time and migrated to another host when load is maximum. The time slot parameter is measured in seconds. The graph show that until slot 5 the CPU utilization is minimum because the new VMs are launched at time interval t on the physical servers. The execution time is plotted for every single physical host. The servers are represents with the DMRS Edge *n* when the VMs are deployed using the proposed method. In contrast the Nova Scheduler Edge *n* the physical hosts when VMs are placed using the default OpenStack scheduler.

University of Malaya

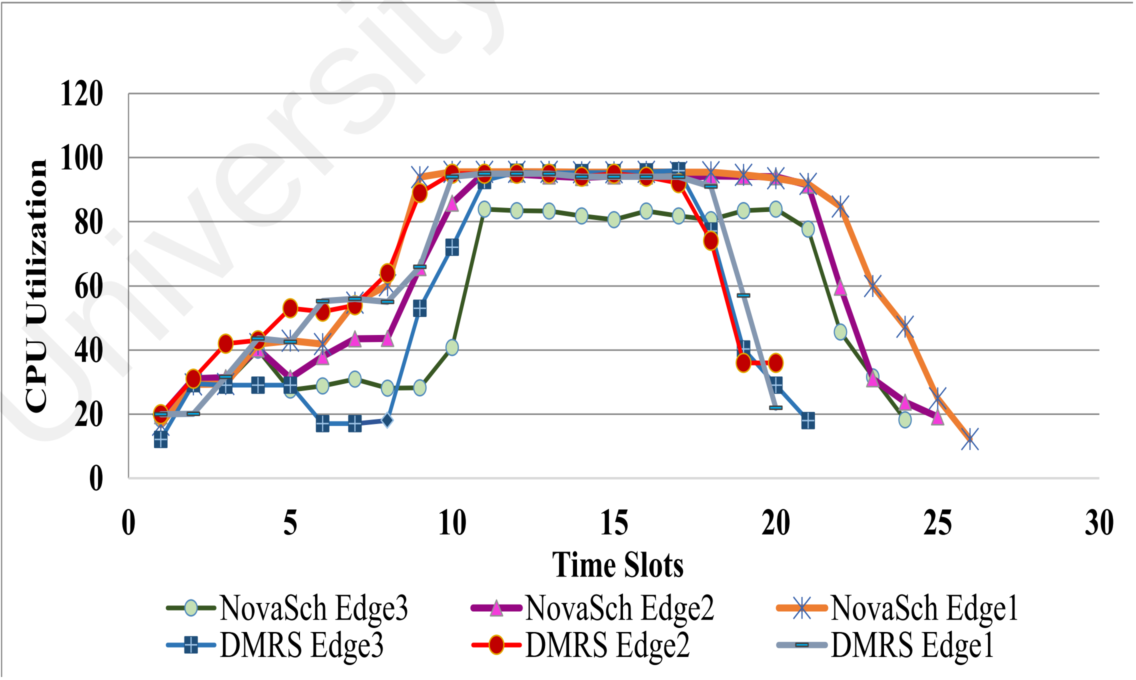


Figure 6.7: Comparison of Proposed DMRS and Default Nova Scheduler

Based on the time slots the DMRS Edge 1, 2, and 3 are showing the similar values when all VMs have completed their executions on the physical hosts. Moreover, DMRS shows that balanced load for each server when the physical CPUs allocation criteria is fulfilled with the deployment of VMs. From slot 10 to 19 the DMRS Edge1, 2, and 3 shows the equal utilization of CPU and as a results every physical hosts completed their execution at same time. In contrast, using the existing nova scheduler Edge 3 is showing the 82% of CPU utilization whereas the Edge2, and 1 are showing the maximum usage with the average load values presented as 94%. The physical server with the default scheduler impacts the CPU utilization and not evenly distributes the workload which as a result exceeds the application execution time as presented up-to 7200 seconds for Edge3, 7800 for Edge2, and 8100 reported for Edge1.

6.3 Model Validation

The accuracy of the mathematical model is validated by comparing its results with the experiments. The VM deployment behavior, execution time of individual VMs running on the physical hosts, overall execution time taken by individual physical server when multiple VMs are deployed, CPU utilization, and number of migrations are the parameters studied to evaluate the results of validation model and the results obtained from empirical studies.

6.3.1 Analysis of VM Placement

Figure. 6.8 presents the number of VMs placement on the physical hosts at x-axis and at y-axis the execution time of individual VM is plotted. The placement sequence is decided by based on the balanced load criteria in order to fairly deploy the workload on the physical machines. In order to presents the distribution of VMs on the specific PMs different colors are selected such as green presents Edge1, blue shows Edge2, and yellow presents Edge3. The first three VMs are placed on Edge2, Edge1, and Edge3,and it completed their execution in 1500, 1500, and 1200 seconds. In start VM 1, and 2

University of Malaya

have assigned the 5 slots to complete their tasks, which shows that the VMs completed their executions without any interruption when their is exactly one VM is deployed on each server. The forth, fifth, and sixth VMs are taking 1800, 1200, and 2100 in order to complete their execution. While considering the VM placement the VM 25 shows that initially it is executed on Edge2 for 1200 seconds and later on it is migrated to Edge3 and completed the remaining compactions in 1500 seconds. Form results it is observed that when numbers of VMs are maximum the CPU resources are shared among the VMs and execution time is depended on CPU utilization as studied in next section.

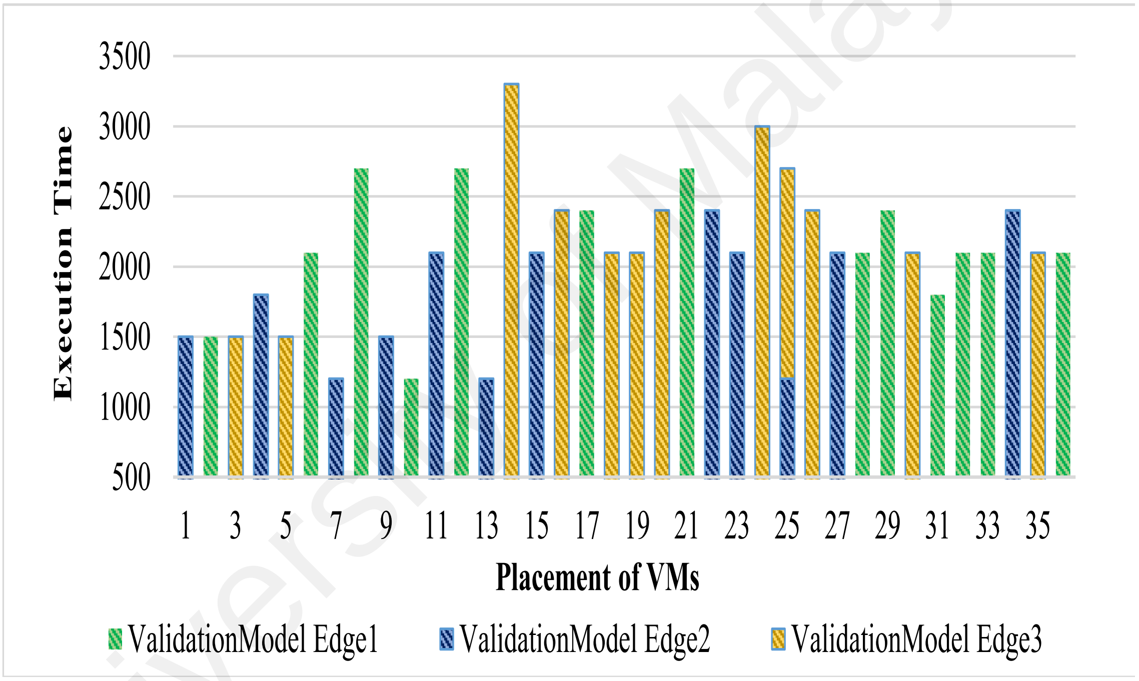


Figure 6.8: VM Distribution based on Mathematical Model

University of Malaya

6.3.2 CPU utilization

Figure. 6.9 presents the CPU utilization for physical servers Edge 1, 2, and 3 at time slot t plotted on x-axis and y-axis, respectively. The CPU utilization is presented for how long the load is running inside the VMs for each physical server. Based on migration constraint the load is balanced if the difference is reported equal to 1 vCPU. Based on experiments it is observed that when only 1 vCPUs is utilized by the VMs the load is generated by the VM reported as 12.5% to 15% of CPU usage. Until the time slot 10 the load is reported

<80% of the whole CPU utilization for three servers.

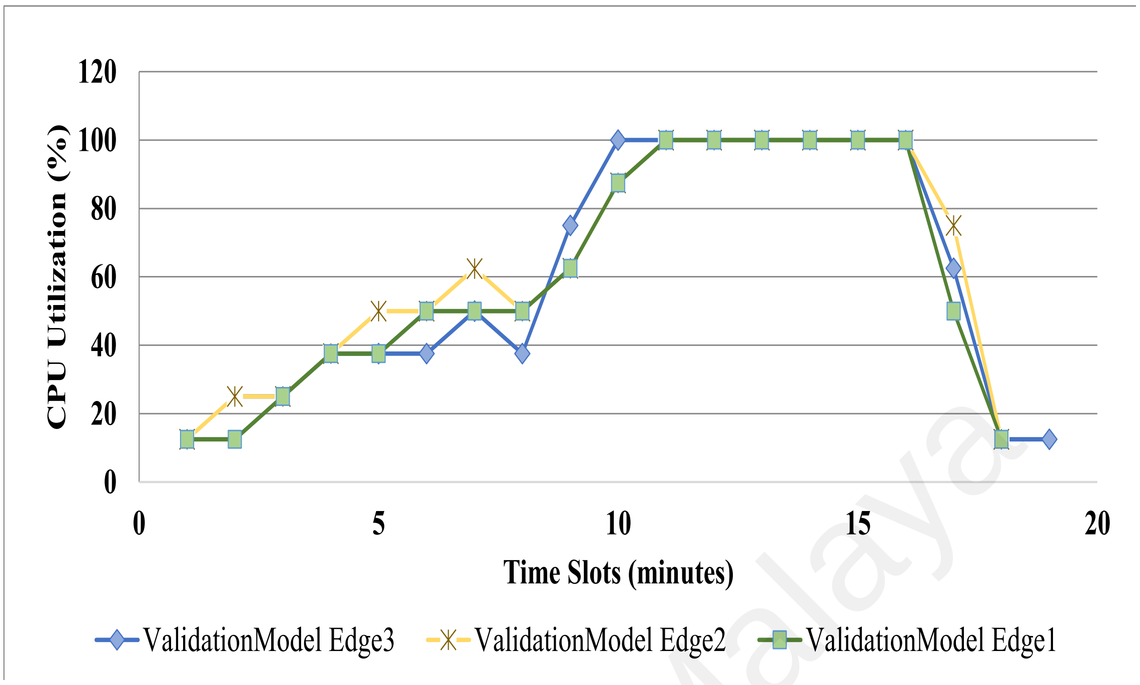


Figure 6.9: CPU Utilization of Physical Servers

The graph shows the load is increases evenly for three physical host. At slot 7 the Edge 2 is showing higher load as compare to the others. The Edge1 and Edge3 presented the CPU utilization up-to 52% whereas the Edge1 shows the 61.2%, as discussed before the difference between theses values of the servers is reported < 12% (1 vCPU), which proved that load is fairly distributed among the servers. From slot 10 to 17 the servers shows the 100% usage of CPU resources. Moreover, the time slots shows that every single host completed its execution between 18 to 19 slots.

6.3.3 Number of Migrations

Figure. 6.10 the number of migrations. The graph shows that only Vm 25 is migrated in order to balance the load when the CPU is maximum and affecting the application execution time. As presented Figure. 6.8 the VM 25 performs migration at the 13th time slot from Edge 2 to Edge 3. The migration action is justified according to the load variations of the physical hosts recorded in Figure. 6.11.

University of Malaya

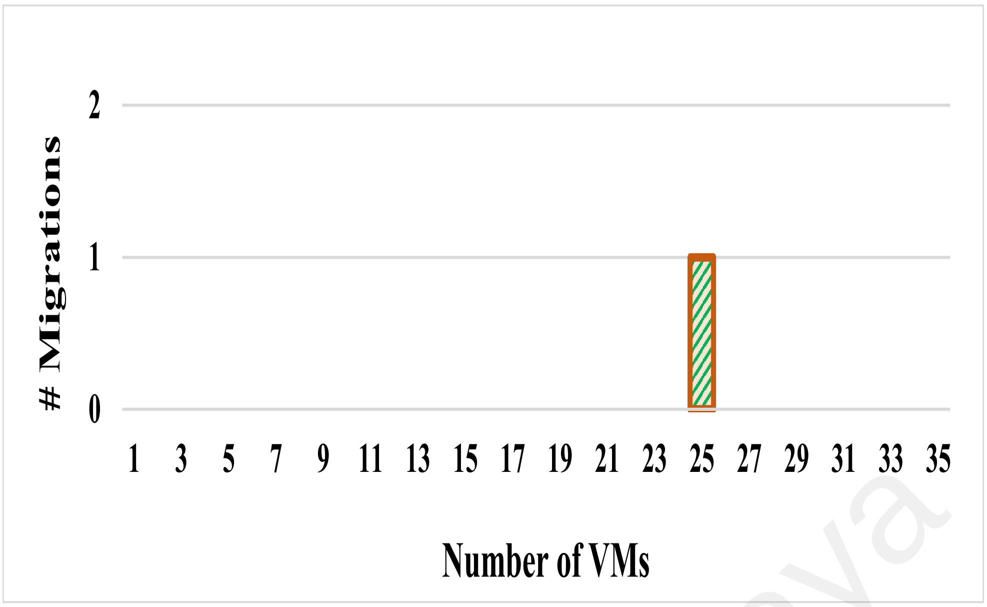


Figure 6.10: Number of VM migrations

Figure 6.11 presents the number of time slots in which VMs are allocated to the physical servers and completed their execution. The graph is plotted to shows the CPU utilization when VMs are placed requesting the different number of vCPUs. The said figure depicts that load values reaches to its maximum when 8 cores (actual physical cores) are utilized. At 11th and 12th time slots for all the three physical hosts more than 8 cores are allocated by VMs. At the 13th time slot, mathematical model decides to migrate the VM number 25 from Edge 2 to Edge 3. By the 13th time slot, the load becomes balanced between the three hosts and the system maintains its stability (remains stable) for two slots. For slots 15th the load is also balanced and only the one extra VM is running on Edge3 which has completed its execution in 16th time slot. The graph results shows that balanced load criteria is fulfilled while considering the minimum number of migrations criteria when the load is not stable after the distribution of VMs.

University of Malaya

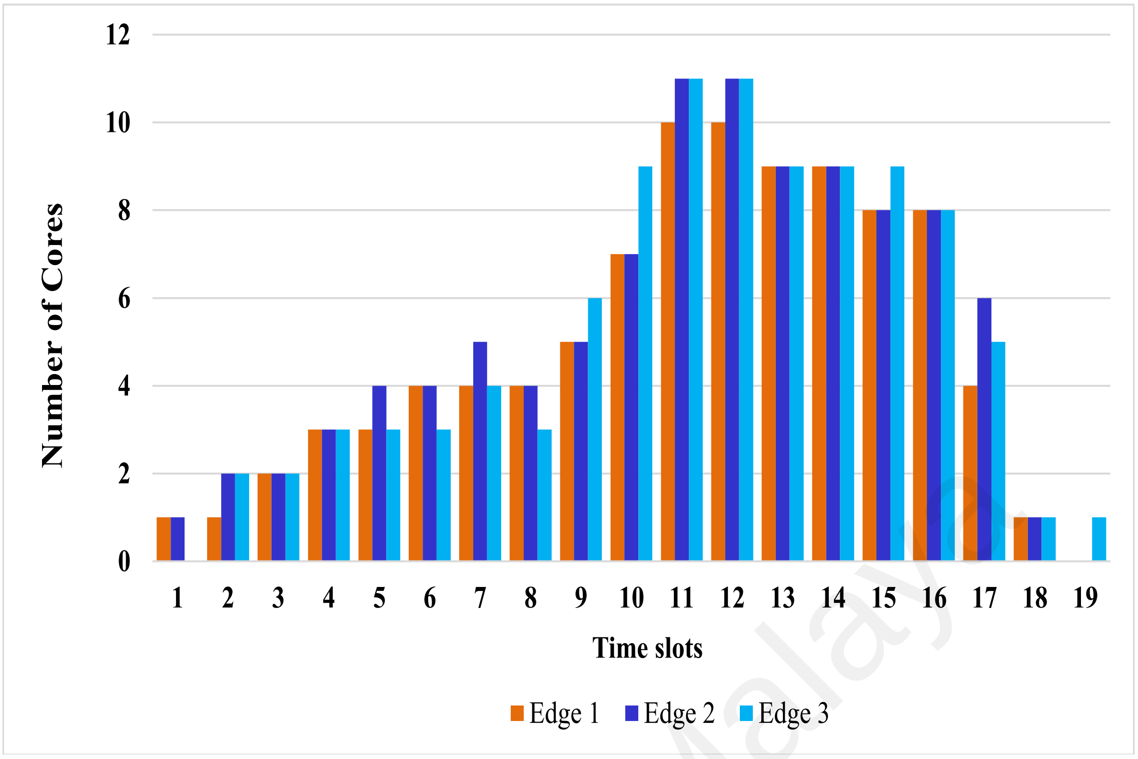


Figure 6.11: Impact of Migrations on the CPU Utilization

6.3.4 Execution Time of Individual VMs

Figure 6.12 shows the total number of VMs on each PM. The graph is plotted across the number of VMs and the execution time required for that VMs to complete the executions. The first VMs deployed on each host are taking the execution time from 1200 to 1551 seconds the execution time depends on the arrival of VMs while it is affected if there are number of VMs deployed equal to the physical capacity of the PMs. The graph compares the execution time taken from the experiments and also from the validation model. Moreover, it represents that in order to validate the correctness of model the distribution of VMs is same, which is selected by the validation model represented as optimized DMRS and proposed dynamic multi resource based scheduler based on the CPU utilization criteria. The graph results shows that VMs execution time is similar when executed based on validation model and the experiments. Based on the results the accuracy of the model is validated up-to 85% while considering the application execution time.

University of Malaya

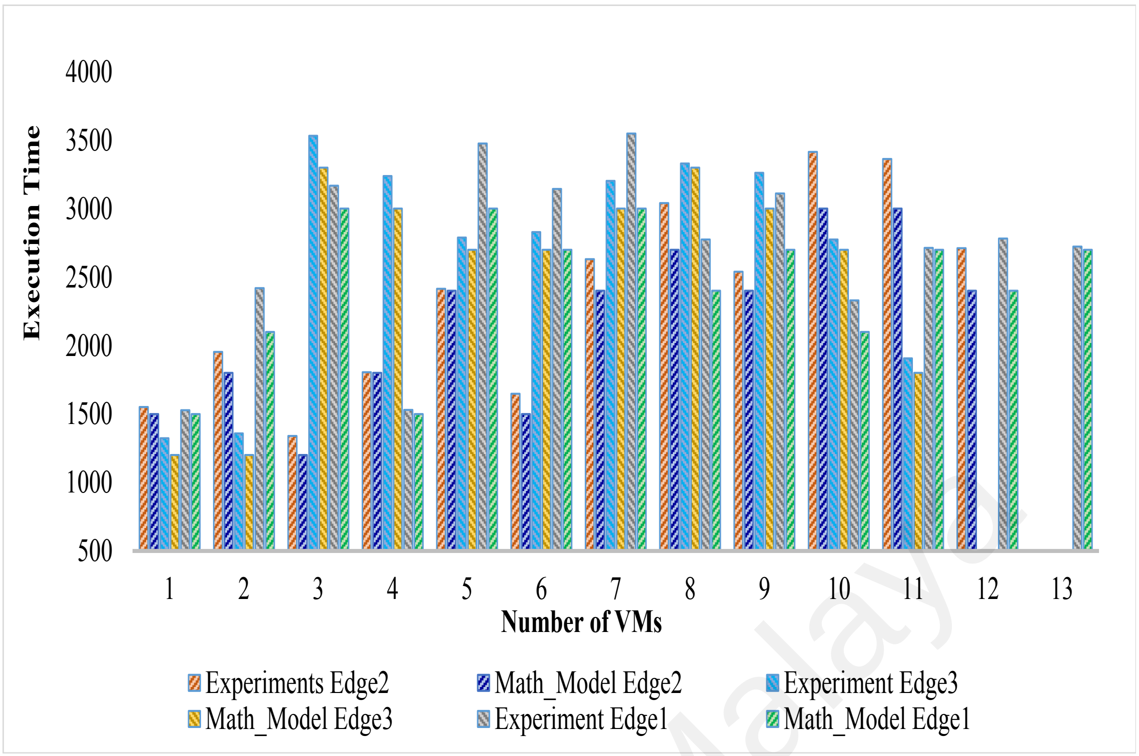


Figure 6.12: Performance Comparison of Validation Model, and DMRS Method

6.4 Comparison of DMRS, Optimized DMRS, and Nova Scheduler

In this section the performance of proposed DMRS, optimized DMRS (validation model), the default scheduler is compared based on the VM distribution and execution time parameters.

6.4.1 CPU utilization

Figure. 6.13 presents the time slots in minutes and the CPU utilization in in percentage values at x-axis and y-axis, respectively. The proposed DMRS method is compared with the optimized DMRS. The CPU utilization of optimized DMRS is obtained through the validation model results. The distribution behavior of physical servers is presents using the different marker styles. In addition, the physical server shows the results conducted using the experiments and the validation model. In the said figure the values taken by the mathematical model shows the 100% CPU usage whereas the values plotted using the experimentations shows the values >90%. In actual the load in the real environment do

University of Malaya

not reaches to the 100%; therefore' when the load is greater then 90% the server is fully utilized.

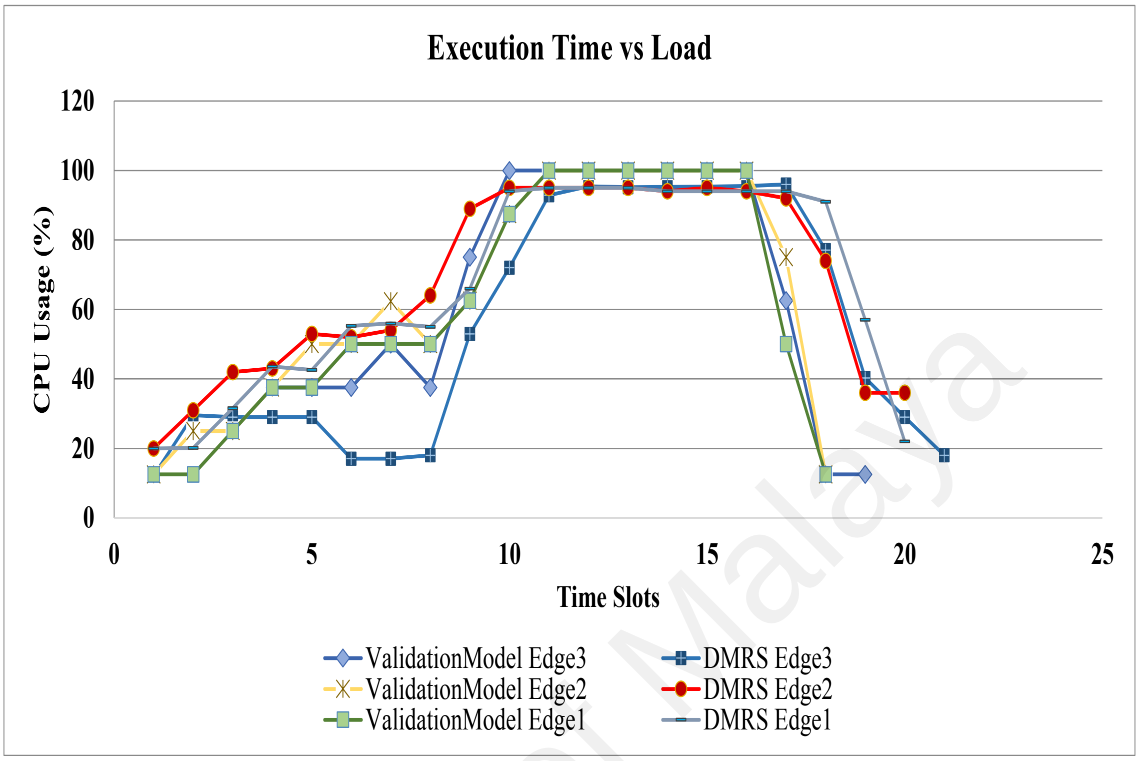


Figure 6.13: CPU Usage Comparison DMRS Experiment Vs Validation Model

The graph values from time slots 11 to 16 shows the equal utilization of the CPU. Moreover, every physical server completed their task in a same time as using the validation model results the total execution time is calculated for 12 time slots. Besides, the experiments results shows that Edge 3 is showing the balanced load values starting from 11 slot to 16. When the difference of the slots is compared based on validation results the servers Edge2 and Edge3 with the experiments completed their execution in 20 slots while the Edge has completed all executions within this time slot and running for slots 21 with the normal load up-to 19.5%.

6.4.2 Execution Time of Individual VMs

Figure. 6.14 the number of VMs deployed only at Edge1. The graph depicts the number of VMs and the execution time of each VM at x-axis and y-axis, respectively. The said figure compare the execution time of VMs obtained from the mathematical results and the

University of Malaya

experiments. This graph shows that the execution time taken by each VM in validation model of experiments is matched up-to 85%. The mathematical results shown the small difference because of no overhead is considered while conducting theses results. While the execution is affected in real time experimentation because numbers of processes are running to execute the instruction including new VMs booting statements and execution of CPU intensive applications. The term mathematical model and validation model is used interchangeably while presenting the results collected from the validation model.

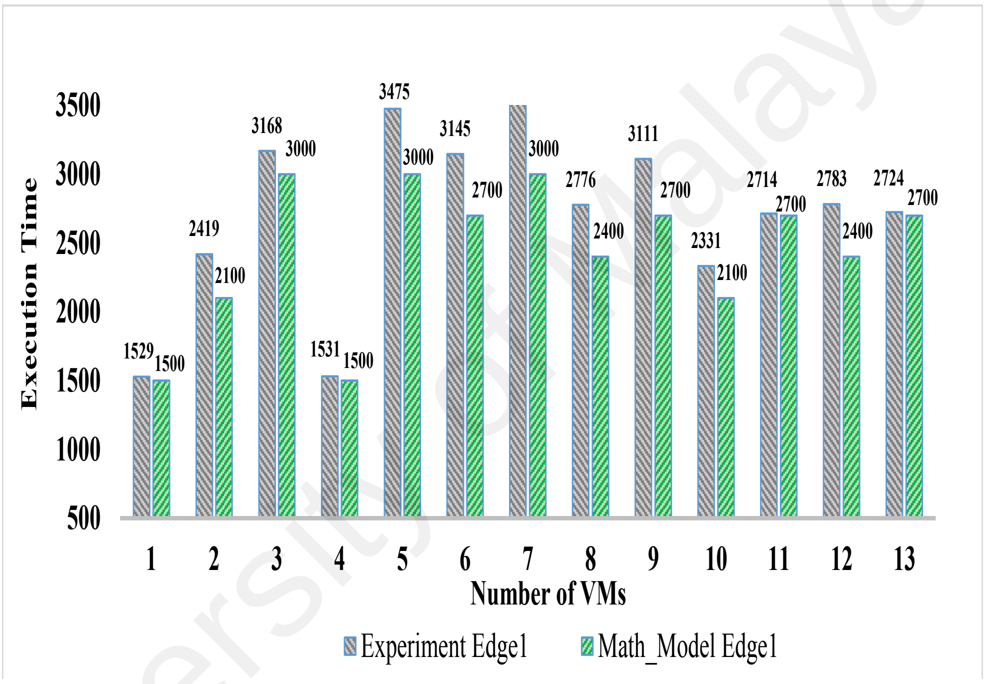


Figure 6.14: Comparison of Execution Time for Edge1

Figure. 6.15 shows the allocation of VMs on Edge2 and shows the total execution of VMs. On Edge2 the time calculated based on validation model and experiments is the small amount of difference. The time calculated using the empirical results is presented as 1551, 1954, 1339, 1805, 2415, 1647, 2631, 3040, 2540, 3416, 3364, and 2711 second for VM 1 to 12, respectively. Besides the time taken by the validation model for theses VMs is reported as 1500, 1800, 1200, 1800, 2400, 1500, 2400, 2700, 2400, 3000, 3000, and 2400. The results shows the execution time reported with the model results is similar to the experimental results, which validated the correctness of mathematical model.

University of Malaya

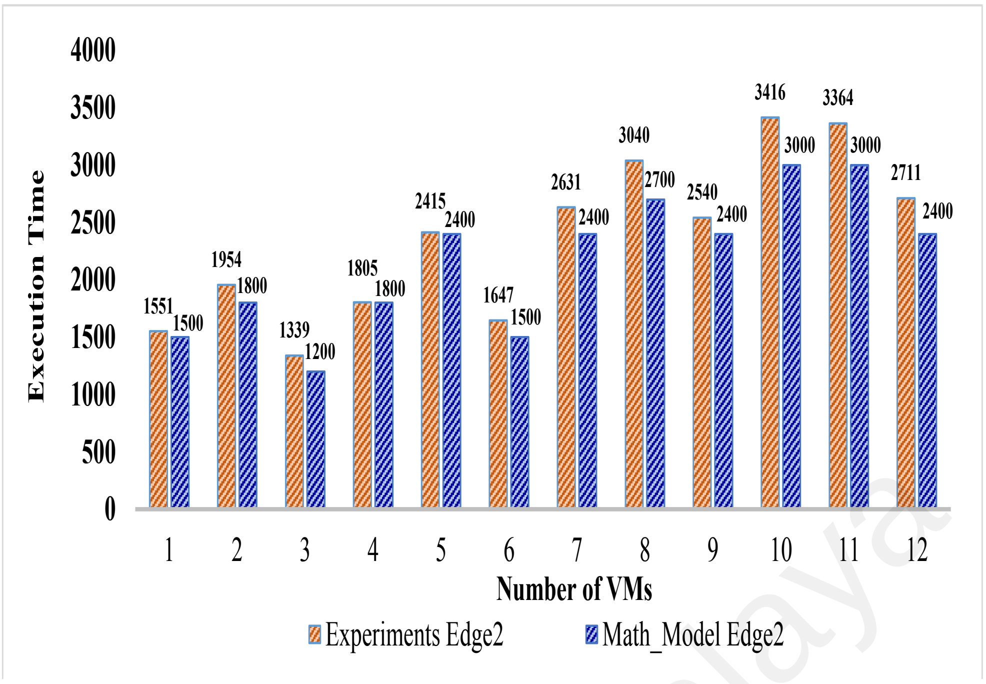


Figure 6.15: Comparison of Execution Time for Edge2

Figure. 6.16 presents the VMs execution time when deployed on the Edge machines. It shows that total 11 VMs are deployed on the Edge3 while satisfying the balanced load criteria. The validation model and proposed DMRS deployed VMs in a similar way to fulfill the even CPU utilization objective. The graph shows that the execution time taken by individual VM is approximately same with the small amount of difference. The mathematical model shows that third Vm is executed eleven time slots and each slots represents the 300 seconds with the execution time 3300 whereas the experiment shows the time value up-to 3533 for that machine with the percentile difference 6.5%. Moreover, the percentile difference calculated for each VM on that server is reported as 9.3%, 11.6%, 6.5%, 7.3%, 3.1%, 4.5%, 6.3%, 0.9%, 8.0%, 2.7%, and 5.6%, respectively.

University of Malaya

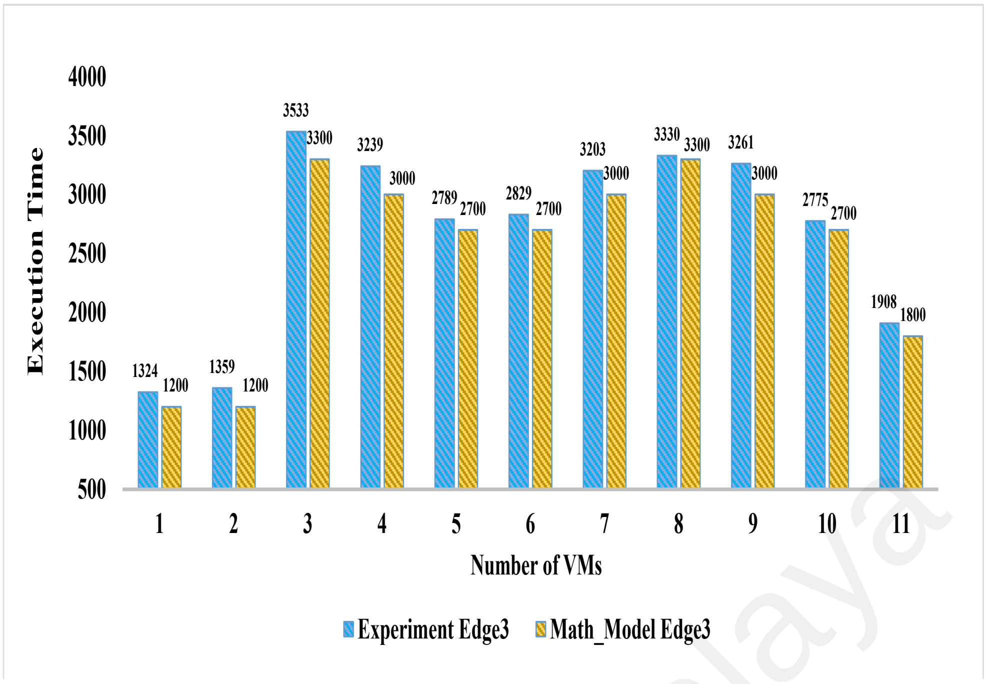


Figure 6.16: CPU Utilization Analysis using Mathematical Model, and DMRS

Figure. 6.17 shows the execution time for all the VMs that are hosted on Edge1, Edge2, and Edge3 during the whole observation period The said figure presents the percentile difference between the validation model and experiments. The bar presented as experiments shows the values computed via real time experimentation of execution time. Besides, the right bar displays the estimated values by computed through the validation model. Moreover, the percentile differences is presented to measure the correctness of the mathematical model. Based on the results the percentile difference calculated for theses results shows the difference <15% in total, which advocates the effectiveness and reliability of our mathematical model in offering an acceptable accuracy.

University of Malaya

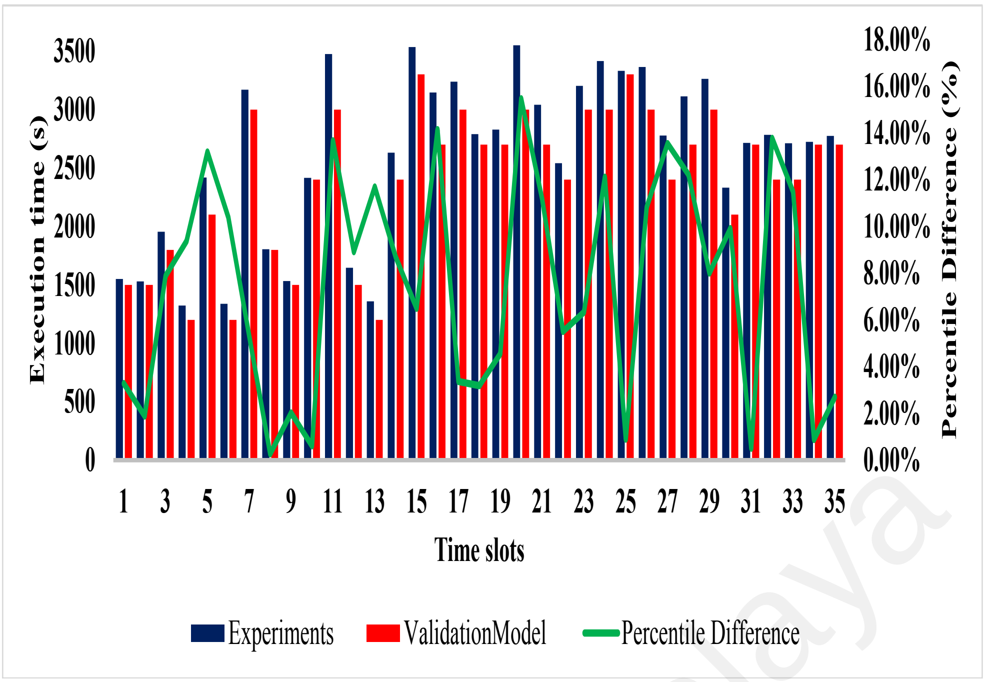


Figure 6.17: Percentile Difference of Execution Time

Figure. 6.18 presents the overall execution time when the CPU utilization is maximum and VMs are running inside the physical hosts. The said figure presents the CPU usage computed using the results obtained from validation model, proposed DMRS method, and default OpenStack scheduler. The graph shows that when the load is deployed using the default scheduler it is not showing the even CPU utilization for Edge1,2, and 3. The Edge3 server showing the minimum CPU usage with the load value up-to 81% whereas the load observed through the Edge1, and Edge2 is reported as maximum value of CPU (90%). Moreover, the difference between the servers is reported more then one VM. As the default scheduler do not balanced the load; therefore; the execution time taken by each VMs is maximum when compared with the validation model and DMRS results.

University of Malaya

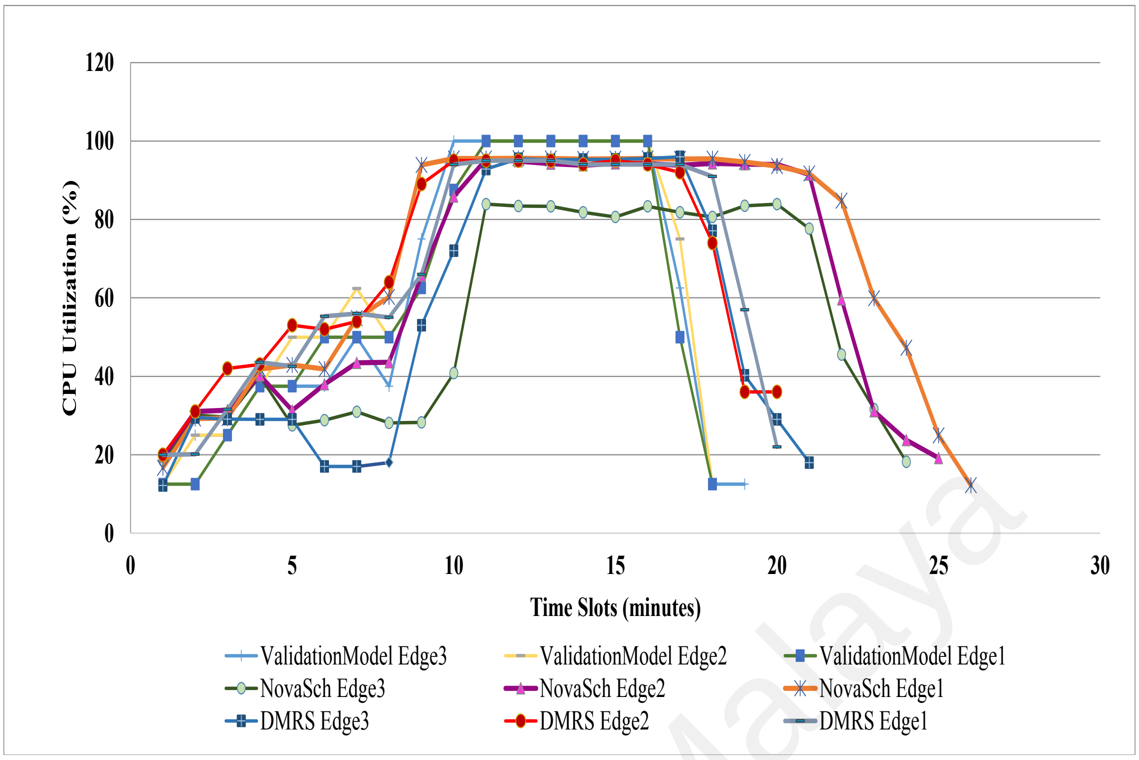


Figure 6.18: CPU Utilization Analysis (Validation Model, DMRS, and Default Scheduler)

Figure. 6.18 shows that the validation model showing the better execution time which is presented as 44% while. Furthermore, percentage difference computed for the validation model results and DMRS experiments results is reported as average value of 5.6% with the 94.4% similar results based on the CPU utilization parameter. However, the DMRS, and optimized DMRS based VM distribution maximize the performance of cloud with the minimum execution time, minimum number of migrations while satisfying the objective for fair distribution of load among the physical hosts.

6.5 Conclusion

In this chapter we have validated the performance of proposed SMRS method in the static environment while the initial placement of VMs. We have compared the SMRS with existing scheduler, which is used for the distribution of the VMs in OpenStack cloud. The CPU utilization, and execution time is considered as a performance metrics for static algorithm. The comparison results shows that fairly VM distribution based on CPU uti-

University of Malaya

lization enhance the application execution time. We have compared that when the load nature is uniform the SMRS evenly distribute the load among the servers and produce the similar results to the default scheduler. In contrast, SMRS shows the better results when load nature is different for each VM. IT is observed that existing scheduler overlooks the CPU utilization, which leads to the maximum execution time. The comparison shows that in best case the performance of proposed SMRS is increased up-to 50% when compared with the default nova scheduler. Similarly, for different load behavior it shows the performance gain up-to 44%,a nd 33% in case of static algorithm.

For dynamic load balancing the DMRS method is proposed and compared with the existing solution. The proposed DMRS shows the 44% performance improvement when compared with default method using the execution time parameter. Moreover, the proposed DMRS algorithm is validated using the set of equations designed in chapter 4 using the real experiments. The execution time results obtained through the validation model are within the 0.2% of empirical results as a best case. In addition, as a worst case difference the value is reported as 14.14%. Based on the execution time of individual VMs the validation model shows the 85% similar results equal to the empirical results. Moreover, the DMRS fulfills its balanced load objective while satisfying minimum number of migrations. The proposed DMRS results are compared with validation model results based on the CPU utilization parameter. However, the CPU utilization parameters shows the 94.4% similar results when compared with the validation results. Considering the results obtained through the performance comparison and analysis with the existing method, it is concluded that CPU utilization based initial VM placement enhance the application execution time in static load balancing. Further, for dynamic load balancing when the VM are initially distributed using the CPU utilization factor the number of migrations are reduced and execution time is increased.

University of Malaya

CHAPTER 7: CONCLUSION

This chapter presents the conclusion of overall study conducted in this thesis. The conclusive analysis is performed by reflecting the set of research objectives presented in the first chapter.This chapter highlights the research contributions and summarize the future research directions of this study.

This chapter is categorized into four sections. Section 7.1 presentees the reassessment of the proposed objectives of this thesis. Section 7.2 discusses the contribution of this study. The scope and limitation of this study are examined in section 7.3. Section 7.4 highlights the future direction of this study.

7.1 Reappraisal of Research Objectives

The problem of uneven distribution of workload while the initial placement of VMs in cloud and its adverse impact on the application execution time has been addressed in this thesis. This section presents the road-map that is followed to achieve the research objectives highlighted in section 1.4.

Objective 1: To critically review the current state-of-the-art cloud load balancing schemes while the placement of VMs to gain insight to the performance limitations.

The first objective of this study was to qualitatively analyze the existing state-of-the-art cloud load balancing schemes to highlight their limitations. In order to achieve the first research objective, this study critically review the cloud load balancing schemes and derived the thematic taxonomies to categorize the exiting literature based on the selected parameters which were common in most of the studies. The state-of-the-art literature has been studied from the online digital libraries including ACM, IEEE, Web of Science, Elsevier, and Springer. In the broader domain of cloud resource management and cloud load balancing, we have collected ad studied the 130 papers and reviewed the current

University of Malaya

literature on VM placement methods by selecting the 30 methods, which are published during last five years. Moreover, the selected methods are compared based on the proposed taxonomies to highlight the variances and commonalities among them. The aim of this exercise was to investigate the issues and challenges to propose the future research directions in this domain. We found that existing studied do not fairly distribute the cloud workload. Several studies initial allocate the VMs in the cloud while overlooking the CPU utilization. Besides, majority of the studied incorporates the dynamic load balancing based on the VM migration method which leads to the maximum migration overhead. Therefore; an efficient load balancing method is required that fairly distribute the CPU load at the time of initial deployment of workload and also minimize the number of migrations.

Objective 2: To investigate the workload distribution of cloud load balancing schedulers to reveal inefficiencies in existing schemes without considering the CPU utilization and current load while the placement of VMs.

The second objective of this study was to investigate and analyze the impact of workload distribution on the application execution time and CPU utilization. To accomplish this research objective, we have examined the VM allocation behavior in difference cases using the default OpenStack scheduler. We classified the workload nature as a static and random load based distribution while the placement of VM in cloud. The empirical analysis highlighted that workload nature affect CPU utilization and application execution time. This study showed that the existing schedulers consider the RAM availability factors only to select servers for VM deployment. The analysis shown that existing cloud load balancing schedulers were not distributed the workload based on CPU utilization criteria. Moreover, existing schemes were not able to adequately mitigate the impact of CPU utilization on the application execution time while the initial deployment.

Objective 3: To design and propose a multi resource-based scheduler to min-

University of Malaya

imize the deficiencies of current cloud schedulers based on CPU utilization, and application execution time while the placement and migration of VMs.

The third objective of this research was to design and propose the solution that efficiently distribute the workload and save the application execution time. We have proposed the multi-resource based objective schemes for static and dynamic workload distribution. A static multi resource based scheduler is proposed for the static load balancing to address the issue of CPU load at the time of initial placement. In addition, a dynamic multi resource based scheduler is proposed to address the dynamic load balancing while minimize the number of migrations after the initial placement of workload. The proposed solutions collect the CPU utilization based information using the proposed load analyzer, load filter, and compute load algorithms. The proposed solutions minimize the application execution time and allocated the workload based on balanced utilization of CPU. In dynamic algorithm application execution time is minimized while satisfying the objective of uniform workload distribution and minimum number of migrations.

Objective 4: To evaluate the performance of proposed multi resource based algorithms and compare it with the state-of-the-art current VM placement cloud scheduler, and to validate the developed mathematical model.

The final objective was to develop the mathematical model of the proposed solutions and their validation. The mathematical model is validated by comparing its results with the results obtained from the empirical study. The empirical results are conducted based on real time experiments using the OpenStack cloud. We have evaluated the performance based on CPU utilization, number of migrations, Overall execution time taken by the physical server, and the application execution time of individual VMs parameters. Moreover, the performance of static and dynamic algorithms are compared with the default nova scheduler.

The proposed static algorithm fairly distribute the workload with the performance

University of Malaya

gain up-to 91% using the CPU utilization parameter and minimize the appellation execution time 50% when compared with the existing solution. Moreover, the dynamic load balancing algorithm enhance the performance 88% based on CPU utilization metric and save the execution time up-to 44% when compared with the existing solution. The validation model and empirical results shows difference of 0.2% in the best case while in the worst case it shows the difference less than 15%. In addition, the validity of the mathematical model is proved 94.4% and 85% for CPU utilization and application execution time when compared with the empirical results of proposed solution.

7.2 Contributions

This section highlights the main contribution of this research. The scholarly articles as contribution are listed in Appendix A. This study summarizes to the body of knowledge as follows.

- **Thematic Taxonomy:** This research proposed the thematic taxonomies to classify the existing state-of-the-art cloud load balancing schemes. The main research categories are highlighted in the corresponding domain based on the proposed taxonomies. These taxonomies highlight the critical aspects related to workload distribution, resource selection, and allocation in cloud. Moreover, the comprehensive literature analysis lead to the identification of open research issues.

- **Performance Evaluation of VM deployment schemes:** A detailed analysis based on default scheduler of OpenStack cloud is performed to analyze the VM placement selection criteria. The performance evaluation based on deployment behavior and unfair CPU usage revealed insights to the issues in the existing load balancing methods.

- **Static Multi Resource Based Scheduler Method (SMRS):** An efficient method is

University of Malaya

proposed for the static load balancing. The proposed method selected the minimum loaded host to allocate the new request (VMs). We empowered default scheduler to consider the RAM capacity and number of vCPUs in addition with CPU utilization (CPU load) for initial placement of VMs. The proposed algorithm fairly distributes the workload and improve the performance by minimizing the application execution time.

- **Dynamic Multi Resource Based Scheduler Method (DMRS):** An efficient dynamic load balancing algorithms is proposed to efficiently utilize the cloud resources in terms of RAM, CPU utilization, and number of vCPUs. Proposed DMRS method uniformly distributed the workload at the time of initial deployment and manage the CPU utilization among all the servers by adapting the migration technique. The proposed algorithm controls the number of migrations with the balanced load distribution and successfully minimize the application execution time.
- **Validation and Evaluation of DMRS:** We have modeled the proposed DMRS mathematically using and set of equation that are derived to validate the model. The model is validated by comparing its results with the empirical results. The performance of the model is evaluated based on the CPU utilization, application execution time, and number of migrations.

University of Malaya

7.3 Scope and Limitations

The proposed multi resource based scheduler algorithms are effective for all interactive cloud data centers. The proposed algorithms will works for legacy applications in addition with the newly designed applications. Moreover, this algorithms are also for all types of cloud specially form OpenStack cloud.

The proposed DMRS is suitable for efficient load balancing with the minimum number of migrations based on the CPU utilization factor. VM migration is not free and consumes a significant amount of sender and receiver resources in terms of power. This study do not address the power consumption while the migration of the workload. Moreover, proposed algorithms do not address the network communication pattern when VMs are deployed and migrated to another hosts.

7.4 Future Research Directions

This research was an effort to contribute towards the cloud load balancing domain. However, a single PhD thesis is not enough to covers the all aspects to a particular domain. The following lines, we presents the insight to the some of possible research directions. The focus of this research is to only efficiently balance the workload among physical servers. The propose dynamic load balancing method balance the load while adapting the migration technique. When the VMs are migrated proposed algorithm do not consider the migration time (overhead) and downtime from the overall execution time recorded. i.e., we assumed that once a particular VM is stopped, it is directly resumed on the target one at the same instant. Hence, the future work include to extending the scope of this study by addressing the migration overhead while the VM migration. Moreover, in this study, it is assumed that each VM is running with the uniform load before its execution time, we aim to address that assumption as a future work. In addition, like CPU and memory, network is considered as a shared and critical resource for cloud application. Network traffic affects the performance of cloud. Therefore, as a future research direction we aim to address the affect of workload on the network traffic based on the internal cloud traffic.

University of Malaya

REFERENCES

Adhikari, J., & Patil, S. (2013). Double threshold energy aware load balancing in cloud computing. In *Computing, communications and networking technologies (icccnt), 2013 fourth international conference on* (pp. 1–6).

Ahmad, R. W., Gani, A., Hamid, S. H. A., Shiraz, M., Yousafzai, A., & Xia, F. (2015). A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *Journal of Network and Computer Applications*, *52*, 11–25.

Ahmed, E., Gani, A., Khan, M. K., Buyya, R., & Khan, S. U. (2015). Seamless application execution in mobile cloud computing: Motivation, taxonomy, and open challenges. *Journal of Network and Computer Applications*, *52*, 154–172.

Ai, L., Tang, M., & Fidge, C. (2011). Resource allocation and scheduling of multiple composite web services in cloud computing using cooperative coevolution genetic algorithm. In *International conference on neural information processing* (pp. 258–267).

Ammal, R. A., Kumar, K. A., Alka, B., & Renjith, B. (2015). Experiences on setting up on-premise enterprise cloud using openstack. In *International conference on cloud computing* (pp. 114–125).

Ardagna, D., Casolari, S., Colajanni, M., & Panicucci, B. (2012). Dual time-scale distributed capacity allocation and load redirect algorithms for cloud systems. *Journal of Parallel and Distributed Computing*, *72*(6), 796–808.

Armstrong, R., Hensgen, D., & Kidd, T. (1998). The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions. In *Heterogeneous computing workshop, 1998.(hcw 98) proceedings. 1998 seventh* (pp. 79–87).

Ashwin, T., Domanal, S. G., & Guddeti, R. M. R. (2014). A novel bio-inspired load balancing of virtualmachines in cloud environment. In *Cloud computing in emerging markets (ccem), 2014 ieee international conference on* (pp. 1–4).

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., ... Warfield, A. (2003). Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, *37*(5), 164–177.

Bayer, R., & Unterauer, K. (1977). Prefix b-trees. *ACM Transactions on Database Systems (TODS)*, *2*(1), 11–26.

Bellifemine, F., Bergenti, F., Caire, G., & Poggi, A. (2005). Jade a java agent development framework. In *Multi-agent programming* (pp. 125–147). Springer.

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

University of Malaya

Beloglazov, A., & Buyya, R. (2010). Energy efficient resource management in virtualized cloud data centers. In *Proceedings of the 2010 10th ieee/acm international conference on cluster, cloud and grid computing* (pp. 826–831).

Beloglazov, A., & Buyya, R. (2012). Openstack neat: A framework for dynamic consolidation of virtual machines in openstack clouds–a blueprint. *Cloud Computing and Distributed Systems (CLOUDS) Laboratory*.

Bernsmed, K., Jaatun, M. G., Meland, P. H., & Undheim, A. (2011). Security slas for federated cloud services. In *Availability, reliability and security (ares), 2011 sixth international conference on* (pp. 202–209).

Bhadani, A., & Chaudhary, S. (2010). Performance evaluation of web servers using central load balancing policy over virtual machines on cloud. In *Proceedings of the third annual acm bangalore conference* (p. 16).

Bobroff, N., Kochut, A., & Beaty, K. (2007). Dynamic placement of virtual machines for managing sla violations. In *Integrated network management, 2007. im'07. 10th ifip/ieee international symposium on* (pp. 119–128).

Bramson, M., Lu, Y., & Prabhakar, B. (2010). Randomized load balancing with general service time distributions. In *Acm sigmetrics performance evaluation review* (Vol. 38, pp. 275–286).

Buyya, R., Garg, S. K., & Calheiros, R. N. (2011). Sla-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions. In *Cloud and service computing (csc), 2011 international conference on* (pp. 1–10).

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

Calcavecchia, N. M., Biran, O., Hadad, E., & Moatti, Y. (2012). Vm placement strategies for cloud scenarios. In *Cloud computing (cloud), 2012 ieee 5th international conference on* (pp. 852–859).

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., & Buyya, R. (2011). Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, *41*(1), 23–50.

Carlini, E., Coppola, M., Dazzi, P., Ricci, L., & Righetti, G. (2011). Cloud federations in contrail. In *European conference on parallel processing* (pp. 159–168).

Chaczko, Z., Mahadevan, V., Aslanzadeh, S., & Mcdermid, C. (2011). Availability and load balancing in cloud computing. In *International conference on computer and software modeling, singapore* (Vol. 14).

Chang, V. (2014). The business intelligence as a service in the cloud. *Future Generation Computer Systems*, *37*, 512–534.

University of Malaya

Chang, V., Walters, R. J., & Wills, G. (2012). Business integration as a service. *International Journal of Cloud Applications and Computing (IJCAC)*, *2*(1), 16–40.

Chen, H., Wang, F., Helian, N., & Akanmu, G. (2013). User-priority guided min-min scheduling algorithm for load balancing in cloud computing. In *Parallel computing technologies (parcomptech), 2013 national conference on* (pp. 1–8).

Chen, M., Zhang, H., Su, Y.-Y., Wang, X., Jiang, G., & Yoshihira, K. (2011). Effective vm sizing in virtualized data centers. In *Integrated network management (im), 2011 ifip/ieee international symposium on* (pp. 594–601).

Chernicoff, D. (2009). *The shortcut guide to data center energy efficiency*. Realtimepublishers. com.

Corradi, A., Fanelli, M., & Foschini, L. (2014). Vm consolidation: A real case based on openstack cloud. *Future Generation Computer Systems*, *32*, 118–127.

Dash, S. K., Sahoo, J. P., Mohapatra, S., & Pati, S. P. (2012). Sensor-cloud: assimilation of wireless sensor network and the cloud. In *Advances in computer science and information technology. networks and communications* (pp. 455–464). Springer.

Di, S., Wang, C.-L., & Chen, L. (2013). Ex-post efficient resource allocation for self-organizing cloud. *Computers & Electrical Engineering*, *39*(7), 2342–2356.

Domanal, S. G., & Reddy, G. R. M. (2013). Load balancing in cloud computingusing modified throttled algorithm. In *Cloud computing in emerging markets (ccem), 2013 ieee international conference on* (pp. 1–5).

Domanal, S. G., & Reddy, G. R. M. (2014). Optimal load balancing in cloud computing by efficient utilization of virtual machines. In *Communication systems and networks (comsnets), 2014 sixth international conference on* (pp. 1–4).

Egi, N., Greenhalgh, A., Handley, M., Hoerdt, M., Huici, F., Mathy, L., & Papadimitriou, P. (2010). A platform for high performance and flexible virtual routers on commodity hardware. *ACM SIGCOMM Computer Communication Review*, *40*(1), 127–128.

Elzeki, O., Reshad, M., & Elsoud, M. (2012). Improved max-min algorithm in cloud computing. *International Journal of Computer Applications*, *50*(12).

Erdogmus, H. (2009). Cloud computing: does nirvana hide behind the nebula? *Software, IEEE*, *26*(2), 4–6.

Espadas, J., Molina, A., Jiménez, G., Molina, M., Ramírez, R., & Concha, D. (2013). A tenant-based resource allocation model for scaling software-as-a-service applications over cloud computing infrastructures. *Future Generation Computer Systems*, *29*(1), 273–286.

Etminani, K., & Naghibzadeh, M. (2007). A min-min max-min selective algorihtm for grid task scheduling. In *Internet, 2007. ici 2007. 3rd ieee/ifip international conference in central asia on* (pp. 1–7).

University of Malaya

Fan, W.-J., Yang, S.-L., Perros, H., & Pei, J. (2015). A multi-dimensional trust-aware cloud service selection mechanism based on evidential reasoning approach. *International Journal of Automation and Computing*, *12*(2), 208–219.

Farokhi, S., Jrad, F., Brandic, I., & Streit, A. (2014). Hierarchical sla-based service selection for multi-cloud environments. In *4th international conference on cloud computing and services science* (pp. 722–734).

Foster, I., Zhao, Y., Raicu, I., & Lu, S. (2008). Cloud computing and grid computing 360-degree compared. In *Grid computing environments workshop, 2008. gce'08* (pp. 1–10).

Gautam, P., & Bansal, R. (2014). Extended round robin load balancing in cloud computing. *International Journal of Engineering Computer Science*, *3*(8), 7926–31.

Ghribi, C., Hadji, M., & Zeghlache, D. (2013). Energy efficient vm scheduling for cloud data centers: Exact allocation and migration algorithms. In *Cluster, cloud and grid computing (ccgrid), 2013 13th ieee/acm international symposium on* (pp. 671–678).

Goiri, Í., Berral, J. L., Fitó, J. O., Julià, F., Nou, R., Guitart, J., … Torres, J. (2012). Energy-efficient and multifaceted resource management for profit-driven virtualized data centers. *Future Generation Computer Systems*, *28*(5), 718–731.

Greenpeace, I. (2010). *Make it green: Cloud computing and its contribution to climate change.*

Grossman, R. L. (2009). The case for cloud computing. *IT professional*, *11*(2), 23–27.

Guo, C., Lu, G., Wang, H. J., Yang, S., Kong, C., Sun, P., … Zhang, Y. (2010). Secondnet: a data center network virtualization architecture with bandwidth guarantees. In *Proceedings of the 6th international conference* (p. 15).

Gutierrez-Garcia, J. O., & Sim, K. M. (2013). Agent-based cloud service composition. *Applied intelligence*, *38*(3), 436–464.

Hassan, M. M., Hossain, M. S., Sarkar, A. J., & Huh, E.-N. (2014). Cooperative game-based distributed resource allocation in horizontal dynamic cloud federation platform. *Information Systems Frontiers*, *16*(4), 523–542.

Hassan, M. M., Song, B., & Huh, E.-N. (2011). Game-based distributed resource allocation in horizontal dynamic cloud federation platform. In *International conference on algorithms and architectures for parallel processing* (pp. 194–205).

Heller, B., Seetharaman, S., Mahadevan, P., Yiakoumis, Y., Sharma, P., Banerjee, S., & McKeown, N. (2010). Elastictree: Saving energy in data center networks. In *Nsdi* (Vol. 10, pp. 249–264).

Hermenier, F., Lorca, X., Menaud, J.-M., Muller, G., & Lawall, J. (2009). Entropy: a consolidation manager for clusters. In *Proceedings of the 2009 acm sigplan/sigops international conference on virtual execution environments* (pp. 41–50).

University of Malaya

HoseinyFarahabady, M., Lee, Y. C., & Zomaya, A. Y. (2014). Randomized approximation scheme for resource allocation in hybrid-cloud environment. *The Journal of Supercomputing*, *69*(2), 576–592.

Hu, J., Gu, J., Sun, G., & Zhao, T. (2010). A scheduling strategy on load balancing of virtual machine resources in cloud computing environment. In *Parallel architectures, algorithms and programming (paap), 2010 third international symposium on* (pp. 89–96).

Jaikar, A., & Noh, S.-Y. (2015). Cost and performance effective data center selection system for scientific federated cloud. *Peer-to-Peer Networking and Applications*, *8*(5), 896–902.

James, J., & Verma, B. (2012). Efficient vm load balancing algorithm for a cloud computing environment. *International Journal on Computer Science and Engineering*, *4*(9), 1658.

Jararweh, Y., Jarrah, M., Alshara, Z., Alsaleh, M. N., Al-Ayyoub, M., et al. (2014). Cloudexp: A comprehensive cloud computing experimental framework. *Simulation Modelling Practice and Theory*, *49*, 180–192.

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

Kalim, U., Gardner, M. K., Brown, E. J., & Feng, W.-c. (2013). Seamless migration of virtual machines across networks. In *Computer communications and networks (icccn), 2013 22nd international conference on* (pp. 1–7).

Kang, J., & Park, S. (2003). Algorithms for the variable sized bin packing problem. *European Journal of Operational Research*, *147*(2), 365–372.

Kaur, P., & Kaur, P. D. (2015). Efficient and enhanced load balancing algorithms in cloud computing. *International Journal of Grid and Distributed Computing*, *8*(2), 9–14.

Kllapi, H., Sitaridi, E., Tsangaris, M. M., & Ioannidis, Y. (2011). Schedule optimization for data processing flows on the cloud. In *Proceedings of the 2011 acm sigmod international conference on management of data* (pp. 289–300).

Kokilavani, T., & Amalarethinam, D. D. G. (2011). Load balanced min-min algorithm for static meta-task scheduling in grid computing. *International Journal of Computer Applications*, *20*(2), 43–49.

Kolda, T. G., Lewis, R. M., & Torczon, V. (2003). Optimization by direct search: New perspectives on some classical and modern methods. *SIAM review*, *45*(3), 385–482.

Kousiouris, G., Cucinotta, T., & Varvarigou, T. (2011). The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks. *Journal of Systems and Software*, *84*(8), 1270–1291.

Kusic, D., Kephart, J. O., Hanson, J. E., Kandasamy, N., & Jiang, G. (2009). Power

University of Malaya

and performance management of virtualized computing environments via lookahead control. *Cluster computing*, *12*(1), 1–15.

Lenk, A., Klems, M., Nimis, J., Tai, S., & Sandholm, T. (2009). What's inside the cloud? an architectural map of the cloud landscape. In *Proceedings of the 2009 icse workshop on software engineering challenges of cloud computing* (pp. 23–31).

Liaqat, M., Chang, V., Gani, A., Ab Hamid, S. H., Toseef, M., Shoaib, U., & Ali, R. L. (2017). Federated cloud resource management: Review and discussion. *Journal of Network and Computer Applications*, *77*, 87–105.

Liaqat, M., Ninoriya, S., Shuja, J., Ahmad, R. W., & Gani, A. (2016). Virtual machine migration enabled cloud resource management: A challenging task. *arXiv preprint arXiv:1601.03854*.

Litvinski, O., & Gherbi, A. (2013). Openstack scheduler evaluation using design of experiment approach. In *Object/component/service-oriented real-time distributed computing (isorc), 2013 ieee 16th international symposium on* (pp. 1–7).

Mell, P., & Grance, T. (2009). The nist definition of cloud computing. *National Institute of Standards and Technology*, *53*(6), 50.

Meshkova, E., Riihijärvi, J., Petrova, M., & Mähönen, P. (2008). A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks. *Computer networks*, *52*(11), 2097–2128.

Mills, K., Filliben, J., & Dabrowski, C. (2011). Comparing vm-placement algorithms for on-demand clouds. In *Cloud computing technology and science (cloudcom), 2011 ieee third international conference on* (pp. 91–98).

Mishra, M., & Sahoo, A. (2011). On theory of vm placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach. In *Cloud computing (cloud), 2011 ieee international conference on* (pp. 275–282).

Mondal, B., Dasgupta, K., & Dutta, P. (2012). Load balancing in cloud computing using stochastic hill climbing-a soft computing approach. *Procedia Technology*, *4*, 783–789.

Nicolae, B., & Cappello, F. (2012). A hybrid local storage transfer scheme for live migration of i/o intensive workloads. In *Proceedings of the 21st international symposium on high-performance parallel and distributed computing* (pp. 85–96).

Nielson, S. J., Crosby, S. A., & Wallach, D. S. (2005). A taxonomy of rational attacks. In *International workshop on peer-to-peer systems* (pp. 36–46).

Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., & Zagorodnov, D. (2009). The eucalyptus open-source cloud-computing system. In *Cluster computing and the grid, 2009. ccgrid'09. 9th ieee/acm international symposium on* (pp. 124–131).

Palmieri, F., Buonanno, L., Venticinque, S., Aversa, R., & Di Martino, B. (2013). A

University of Malaya

distributed scheduling framework based on selfish autonomous agents for federated cloud environments. *Future Generation Computer Systems*, *29*(6), 1461–1472.

Papagianni, C., Leivadeas, A., Papavassiliou, S., Maglaris, V., Cervello-Pastor, C., & Monje, A. (2013). On the optimal allocation of virtual resources in cloud computing networks. *IEEE Transactions on Computers*, *62*(6), 1060–1071.

Patel, G., Mehta, R., & Bhoi, U. (2015). Enhanced load balanced min-min algorithm for static meta task scheduling in cloud computing. *Procedia Computer Science*, *57*, 545–553.

Petrides, P., Nicolaides, G., & Trancoso, P. (2012). Hpc performance domains on multi-core processors with virtualization. In *International conference on architecture of computing systems* (pp. 123–134).

Phan, L. T., Zhang, Z., Zheng, Q., Loo, B. T., & Lee, I. (2011). An empirical analysis of scheduling techniques for real-time cloud-based data processing. In *Service-oriented computing and applications (soca), 2011 ieee international conference on* (pp. 1–8).

Rahmeh, O. A., Johnson, P., & Taleb-Bendiab, A. (2008). A dynamic biased random sampling scheme for scalable and reliable grid networks. *INFOCOMP journal of computer science*, *7*(4), 1–10.

Ristenpart, T., Tromer, E., Shacham, H., & Savage, S. (2009). Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th acm conference on computer and communications security* (pp. 199–212).

Rosado, T., & Bernardino, J. (2014). An overview of openstack architecture. In *Proceedings of the 18th international database engineering & applications symposium* (pp. 366–367).

Rubinstein, A. (1982). Perfect equilibrium in a bargaining model. *Econometrica: Journal of the Econometric Society*, 97–109.

Samal, P., & Mishra, P. (2013). Analysis of variants in round robin algorithms for load balancing in cloud computing. *IJCSIT*, *4*(3), 416–9.

Sefraoui, O., Aissaoui, M., & Eleuldj, M. (2012). Openstack: toward an open-source solution for cloud computing. *International Journal of Computer Applications*, *55*(3), 38–42.

Shah, M. M. D., Kariyani, M. A. A., & Agrawal, M. D. L. (2013). Allocation of virtual machines in cloud computing using load balancing algorithm. *International Journal of Computer Science and Information Technology & Security (IJCSITS)*, *3*(1), 2249–9555.

Shaw, S. B., & Singh, A. (2014). A survey on scheduling and load balancing techniques in cloud computing environment. In *Computer and communication technology (iccct), 2014 international conference on* (pp. 87–95).

University of Malaya

Shiraz, M., Gani, A., Shamim, A., Khan, S., & Ahmad, R. W. (2015). Energy efficient computational offloading framework for mobile cloud computing. *Journal of Grid Computing*, *13*(1), 1–18.

Sidhu, A. K., & Kinger, S. (2013). Analysis of load balancing techniques in cloud computing. *International Journal of Computers & Technology*, *4*(2), 737–41.

Sim, K. M. (2006). Grid commerce, market-driven g-negotiation, and grid resource management. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *36*(6), 1381–1394.

Sim, K. M. (2008). Evolving fuzzy rules for relaxed-criteria negotiation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *38*(6), 1486–1500.

Sim, K. M. (2012). Agent-based cloud computing. *IEEE Transactions on Services Computing*, *5*(4), 564–577.

Singh, A., Juneja, D., & Malhotra, M. (2015). Autonomous agent based load balancing algorithm in cloud computing. *Procedia Computer Science*, *45*, 832–841.

Singh, A., Korupolu, M., & Mohapatra, D. (2008). Server-storage virtualization: integration and load balancing in data centers. In *Proceedings of the 2008 acm/ieee conference on supercomputing* (p. 53).

Sommers, J., Barford, P., Duffield, N., & Ron, A. (2005). Improving accuracy in end-to-end packet loss measurement. In *Acm sigcomm computer communication review* (Vol. 35, pp. 157–168).

Son, J. (2013). *Automated decision system for efficient resource selection and allocation in inter-clouds* (Unpublished doctoral dissertation). The University of Melbourne.

Sotiriadis, S., Bessis, N., Xhafa, F., & Antonopoulos, N. (2012). Cloud virtual machine scheduling: modelling the cloud virtual machine instantiation. In *Complex, intelligent and software intensive systems (cisis), 2012 sixth international conference on* (pp. 233–240).

Sotomayor, B., Montero, R. S., Llorente, I. M., & Foster, I. (2009). Virtual infrastructure management in private and hybrid clouds. *IEEE Internet computing*, *13*(5).

Stage, A., & Setzer, T. (2009). Network-aware migration control and scheduling of differentiated virtual machine workloads. In *Proceedings of the 2009 icse workshop on software engineering challenges of cloud computing* (pp. 9–14).

Stawski, S. (2015). *Inflection point: How the convergence of cloud, mobility, apps, and data will shape the future of business*. FT Press.

Sundareswaran, S., Squicciarini, A., & Lin, D. (2012). A brokerage-based approach for cloud service selection. In *Cloud computing (cloud), 2012 ieee 5th international conference on* (pp. 558–565).

Tolba, A., & Ghoneim, A. (2015). A stepwise self-adaptive model for improving cloud

University of Malaya

efficiency based on multi-agent features. *JSW*, *10*(8), 1037–1044.

Truong, D. (2010). How cloud computing enhances competitive advantages: A research model for small businesses. *The Business Review, Cambridge*, *15*(1), 59–65.

Uhlig, R., Neiger, G., Rodgers, D., Santoni, A. L., Martins, F. C., Anderson, A. V., ... Smith, L. (2005). Intel virtualization technology. *Computer*, *38*(5), 48–56.

Vilutis, G., Butkiene, R., Lagzdinyte Budnike, I., Sandonavicius, D., & Paulikas, K. (2013). The qogs method application for selection of computing resources in intercloud. *Elektronika ir Elektrotechnika*, *19*(7), 98–103.

Voorsluys, W., Broberg, J., Venugopal, S., & Buyya, R. (2009). Cost of virtual machine live migration in clouds: A performance evaluation. In *Ieee international conference on cloud computing* (pp. 254–265).

Wang, G., & Ng, T. E. (2010). The impact of virtualization on network performance of amazon ec2 data center. In *Infocom, 2010 proceedings ieee* (pp. 1–9).

Wang, Q., & Varela, C. A. (2011). Impact of cloud computing virtualization strategies on workloads' performance. In *Utility and cloud computing (ucc), 2011 fourth ieee international conference on* (pp. 130–137).

Wang, S.-C., Yan, K.-Q., Liao, W.-P., & Wang, S.-S. (2010). Towards a load balancing in a three-level cloud computing network. In *Computer science and information technology (iccsit), 2010 3rd ieee international conference on* (Vol. 1, pp. 108–113).

Wickremasinghe, B., Calheiros, R. N., & Buyya, R. (2010). Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In *Advanced information networking and applications (aina), 2010 24th ieee international conference on* (pp. 446–452).

Woo, S. S., & Mirkovic, J. (2014). Optimal application allocation on multiple public clouds. *Computer Networks*, *68*, 138–148.

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

Wood, T., Shenoy, P. J., Venkataramani, A., & Yousif, M. S. (2007). Black-box and gray-box strategies for virtual machine migration. In *Nsdi* (Vol. 7, pp. 17–17).

Wuhib, F., Stadler, R., & Lindgren, H. (2012). Dynamic resource allocation with management objectives implementation for an openstack cloud. In *Network and service management (cnsm), 2012 8th international conference and 2012 workshop on systems virtualiztion management (svm)* (pp. 309–315).

Xu, P., Zheng, W., Wu, Y., Huang, X., & Xu, C. (2010). Enabling cloud storage to support traditional applications. In *Chinagrid conference (chinagrid), 2010 fifth annual* (pp. 167–172).

University of Malaya

Yau, S. S., & Yin, Y. (2011). Qos-based service ranking and selection for service-based systems. In *Services computing (scc), 2011 ieee international conference on* (pp. 56–63).

Younge, A. J., Henschel, R., Brown, J. T., Von Laszewski, G., Qiu, J., & Fox, G. C. (2011). Analysis of virtualization technologies for high performance computing environments. In *Cloud computing (cloud), 2011 ieee international conference on* (pp. 9–16).

Zhang, I., Denniston, T., Baskakov, Y., & Garthwaite, A. (2013). Optimizing vm checkpointing for restore performance in vmware esxi. In *Usenix annual technical conference* (pp. 1–12).

Zhao, M., & Figueiredo, R. J. (2007). Experimental study of virtual machine migration in support of reservation of cluster resources. In *Proceedings of the 2nd international workshop on virtualization technology in distributed computing* (p. 5).

Zuo, X., Zhang, G., & Tan, W. (2014). Self-adaptive learning pso-based deadline constrained task scheduling for hybrid iaas cloud. *IEEE Transactions on Automation Science and Engineering*, *11*(2), 564–573.

University of Malaya

LIST OF PUBLICATIONS

1. **Liaqat, M**., Chang, V., Gani, A., Ab Hamid, S. H., Toseef, M., Shoaib, U., Ali, R. L. (2017). Federated cloud resource management: Review and discussion. Journal of Network and Computer Applications, 77, 87−105.

2. **Liaqat, M**., Gani, A., Anisi, M. H., Ab Hamid, S. H., Akhunzada, A., Khan, M. K., Ali, R. L. (2016). Distance−Based and Low Energy Adaptive Clustering Protocol for Wireless Sensor Networks. PloS one, 11(9), e0161340.

3. Hamedani, S. R., **Liaqat, M.**, Shamshirband, S., Al−Razgan, O. S., Al−Shammari, E. T., Petkovic, D. (2015). Comparative study of soft computing methodologies for energy input–output analysis to predict potato production. American journal of potato research, 92(3), 426−434.

4. **Liaqat, M.**, Laurian−Ioan, P., Othman, W. A. M., Ali, A., Gani, A., Ozel, C. (2016). Estimation of inequalities for warped product semi-slant submanifolds of Kenmotsu space forms. Journal of Inequalities and Applications, 2016(1), 239.

5. **Liaqat, M**., Ninoriya, S., Shuja, J., Ahmad, R. W., Gani, A. (2016). Vir tual Machine Migration Enabled Cloud Resource Management: A Challenging Task. arXiv preprint arXiv:1601.03854.

6. **Liaqat, M**., Javaid, N., Akbar, M., Khan, Z. A., Ali, L., Hafizah, S., Gani, A. (2014, May). HEX clustering protocol for routing in wireless sensor network. In Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on (pp. 549−554). IEEE.

7. Butt, F. S., **Liaqat, M.**, Khan, M. R., Nisar, W., Munir, E. U. (2013). Common Factors in the Successful Software Projects in Pakistan‘s Software Industry. World Applied Sciences Journal, 23(9), 1176−1185.

University of Malaya