



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

A thesis
For the Degree of Doctor of Philosophy

Service-oriented Resource Orchestration – A Resource
Allocation Approach

Afaq Muhammad

Department of Computer Engineering

GRADUATE SCHOOL

JEJU NATIONAL UNIVERSITY

August, 2017

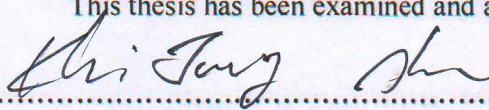

Service-oriented Resource Orchestration – A Resource Allocation Approach

Muhammad Afaq
(Supervised by Professor Wang-Cheol Song)

A thesis submitted to the Department of Computer Engineering and the Faculty of Graduate School of Jeju National University in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Engineering

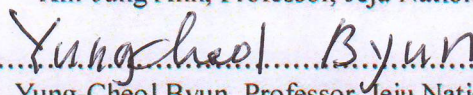

2017. 06.

This thesis has been examined and approved

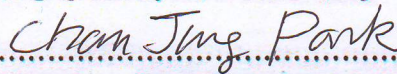

 
.....

Thesis Committee Chair

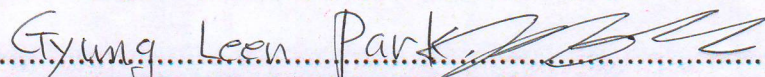
Khi-Jung Ahn, Professor, Jeju National University

 
.....

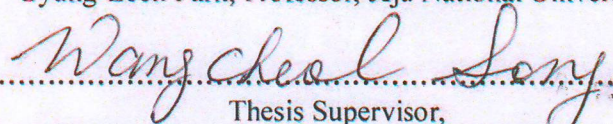
Yung-Cheol Byun, Professor, Jeju National University

 
.....

Chan-Jung Park, Professor, Jeju National University


.....

Gyung-Leen Park, Professor, Jeju National University


.....

Thesis Supervisor,

Wang-Cheol Song, Professor, Jeju National University

Department of Computer Engineering

Graduate School

Jeju National University



*Dedicated to
my dearest parents, loving brothers and sisters, beloved wife, and lovely son.*

ACKNOWLEDGMENTS

First and foremost, my humble praise and gratitude to Allah Almighty; the most gracious, the most merciful, for showering His endless blessings on me throughout my life. Many blessings and salutations on Prophet Muhammad (PBUH) who taught and emphasized the importance of learning and seeking knowledge.

I am extremely grateful to my supervisor Prof. Wang-Cheol Song for his continuous guidance, suggestions, and constant interaction throughout my PhD studies at Jeju National University. I would like to extend my sincere gratitude towards my Korean language teachers, Prof. Gwon-Jin Choi, Madam Anna Young, Madam Kyeongok Song, Jong, Madam Cho Yeon Ha, Madam Lee Hyo Suk, Madam Kwak Min Jong, Sir Sa Sang Heong, Madam Jang Kyung Mi, Madam Hee Soon Lee, Madam Yina Yoon, and Madam Bak Yoon for their tireless hard work, dedication, devotion, and inspiration during my one year stay at Inha University.

I offer my humble gratitude to Prof. Khi-Jung Ahn, Prof. Yung-Cheol Byun, Prof. Chan-Jung Park (Department of Computer Education), and Prof. Gyung-Leen Park (Dept. of Computer Science and Statistics) for their valuable suggestions and extremely important comments during the process of my thesis evaluation. I would like to thank all the professors and teachers for their contributions throughout the program.

I am grateful to National Institute for International Education (NIIED), Ministry of Education, South Korea for awarding me the Korean Government Scholarship to pursue such an exciting program in two universities. Sincere thanks to all the KGSP coordinators of Inha University and Jeju National University especially Madam Junghyo Lee, Madam Ji Hyojung, Madam Hanna Baek, Madam Ji-Yon Song, and Madam Heekyoung Kwon.

I would like to acknowledge the company, cooperation, and help of my friends and colleagues during the tenure of the study. I offer special thanks to Dr. Safdar Ali, Dr. Ghayas Ud Din Siddiqi, Dr. Ahmar Rashid, Dr. Shafqat Ur Rehman, Sham Ur Rehman, Mohammad Kawsar Manik, Ibrahim Hashlamon, Fasihullah Khan, Muhammad Tahir Abbas, Hafiz Mutee Ur Rehman, and Mr. Waqar Kiyani for their enormous help and support.

I offer my special gratitude to Mufti Asim Khan Ashrafi who has been very kind and supportive to me in all circumstances since the beginning of my stay in South Korea. He has been a constant source of guidance in many religious matters that I faced during my stay. I greatly value the time I spent with him.

I owe a debt of special thanks to Zubair Amjad for his cordial cooperation during many occasional light and dark moments of research that I shared with him. His encouragement, determination in helping me, kindness, and care allowed me to finish this journey. We joined

Network Convergence Lab on the same day and shared many great moments together along the way. I would also like to offer special thanks to Muqet Rehman Mughal who has always been my true well-wisher. This journey would have never been easy without his prayers and support.

An honorable mention goes to my dearest parents, loving brothers and sisters, and all the family members. I am forever indebted to my mother and father for their unconditional love and endless prayers. No words can actually describe their everlasting love to me. I owe a lot to them; they encouraged and helped me at every walk of my life. Their unwavering faith and confidence in my abilities always motivated me.

Last but certainly not least, I wish to thank my beloved wife Zari for her love, for sharing the ups and downs, and for her faith in me. My deepest love and respect to her for her tremendous patience, constant moral support, and endless prayers, which were invaluable in completing this journey. Special mention goes to my innocent, joyful and lovely son Hashir in whose company I forget all my worries.

Afaq Muhammad
June 2017

Service-oriented Resource Orchestration – A Resource Allocation Approach

Afaq Muhammad

Supervisor: Prof. Wang-Cheol Song

ABSTRACT

This thesis is primarily concerned with three main topics in cloud platforms using OpenStack as a case study: allocation of resources to meet the demands of a service requested by remote end-user, migration of virtual machines (VMs) instances to offload the overloaded compute nodes, and monitoring of utilized resources. The overall framework architecture consists of three subsystems: 1) An orchestrator that enables to automate resource management and provisioning in OpenStack, 2) sFlow-based subsystem to monitor resource performance counters of OpenStack compute nodes, and 3) A resource utilization-based subsystem for dynamic VM migration in OpenStack.

The proposed orchestrator manages and provisions resources by: 1) exploiting application program interfaces (APIs) provided by the cloud provider in order to control/manage/allocate storage and compute resources; 2) interacting with software-defined networking (SDN) controller to get the details of the available resources, and instructing the changes/rules to manage the network based on the cloud service requirements. For resource allocation, an algorithm is proposed, which allocates resources on the basis of unutilized resources in a pool of virtual machines. An algorithm has been taken into account for comparison with the proposed resource allocation algorithm. The experiment results show that the considered algorithm is outperformed by the proposed algorithm. Furthermore, a monitoring system has been

implemented, which collects and stores samples of the performance counters related to infrastructural resources, such as CPU performance, memory faults, and network performance.

A framework is proposed for dynamic VM migration in a cloud computing platform. The framework implements the proposed overload detection, VM selection, and VM allocation algorithms for dynamic VM migration in clouds. With the help of experiments, it is shown that the proposed algorithms outperform the algorithms that are considered for the purpose of evaluation.

CONTENTS

I. Introduction	1
1.1 Resource Orchestration	3
1.2 VM Migration.....	5
1.3 Monitoring of Utilized Resources	6
1.4 Research Problems and Objectives.....	7
1.5 Thesis Organization	9
II. Related Work	11
2.1 Cloud Computing	11
2.1.1 Cloud Computing Methodologies	13
2.1.2 The Cloud Architecture	15
2.1.3 Cloud Services	19
2.1.4 Cloud Applications.....	21
2.2 Cloud Resource Orchestration.....	23
2.3 Monitoring of Infrastructural Resources.....	25

2.3.1 Existing Monitoring Frameworks	27
2.4 Dynamic VM Migration.....	28
III. Overview of OpenStack and its Deployment	32
3.1 Introduction	32
3.2 Keystone.....	34
3.2.1 Architecture	35
3.3 Nova.....	36
3.3.1 Architecture	36
3.4 Neutron.....	38
3.4.1 Architecture	39
3.5 OpenStack Deployment	40
3.5.1 OpenStack-ONOS Integration	42
3.5.2 Visualization of OpenStack Deployment in ONOS GUI.....	43
IV. Resource Orchestration Framework and its Major Components	45
4.1 Proposed Architecture.....	45
4.2 Service Abstraction Model.....	46
4.2.1 Service Abstraction Communicator	48
4.3 The Orchestrator	49
4.3.1 Network Resource Communicator.....	51
4.3.1 Cloud Resource Communicator.....	52
4.4 sFlow-based Monitoring System.....	55
4.5 Implementation of Proposed Resource Orchestration Framework	57
4.5.1 Performance Analysis of Proposed MRMC Algorithm	58
V. A Framework for Resource Utilization-based Migration of VMs in Cloud	62
5.1 System Model.....	64
5.1.1 Stats Aggregator.....	65
5.1.2 Stats Database	66

5.1.3 Migration Manager.....	66
5.1.4 Algorithm Repository.....	66
5.2 Proposed Structure and Algorithms.....	67
5.2.1 Overload Detection Algorithm	68
5.2.1 VM Selection Algorithm	70
5.2.1.1 VM Selection Criteria.....	72
5.2.1 VM Allocation Algorithm	73
5.3 Experiment and Results	74
VI. Conclusions	81
Bibliography	84

List of Figures

Figure 1.1: The thesis organization.....	10
Figure 2.1: Basic cloud computing architecture	16
Figure 2.2: Layered architecture for a customized cloud service	17
Figure 2.3: Cloud applications.....	22
Figure 3.1: OpenStack conceptual architecture	34
Figure 3.2: Keystone architecture	35
Figure 3.3: Nova architecture	38
Figure 3.4: Neutron architecture.....	40
Figure 3.5: Cloud computing platform-based testbed.....	41
Figure 3.6: OpenStack-ONOS integration	42
Figure 3.7: Visualization of OpenStack deployment in ONOS GUI.....	44
Figure 4.1: Proposed orchestration framework	46
Figure 4.2: The service abstraction model	47
Figure 4.3: Service abstraction communication module.....	49
Figure 4.4: Network resource communicator module.....	52
Figure 4.5: Cloud resource communicator module.....	54
Figure 4.6: sFlow host structure	55
Figure 4.7: Architecture of sFlow monitoring system	56
Figure 4.8: Exchange of messages between different modules for VM creation	58

Figure 4.9: Performance analysis of MRMC for minimum occupied RAM.....	60
Figure 4.10: Performance analysis of MRMC for minimum CPU utilization	61
Figure 5.1: The proposed VM migration process	64
Figure 5.2: Proposed system model	65
Figure 5.3: Exchange of messages between components for VM instance migration	68
Figure 5.4: Performance analysis of proposed overload detection algorithm	76
Figure 5.5: Performance analysis of the proposed minRmaxC-based VM selection algorithm ...	78
Figure 5.6: Minimum migration time versus RAM assigned to VMs	79
Figure 5.7: Performance analysis of the proposed VM allocation algorithm.....	80

List of Tables

Table I: Performance counters.....	26
Table II: Monitor information from OpenNebula monitoring framework.....	27

Chapter 1

Introduction

Cloud computing signifies a transition from computing as an owned product, to computing as a service that is provided to consumers over the internet through large-scale datacenters/clouds. This shift has a major effect on the ways that software is obtained, deployed and developed. This is similar to the effect of changing from mainframes to PCs. Clouds were primarily used by technology start-ups such as Dropbox [1] and Twitter [2] as they provided an adaptable and comparatively inexpensive infrastructure layer, on top of which corporations could install their systems. Recently, more developed enterprises have shown an interest in cloud computing owing to its possible advantages, which include reliability, scalability, ease of deployment and cost-effectiveness [3].

On top of cloud platforms [4], services can be created, managed, and scaled on-demand. Novel orchestration algorithms and efficient virtualization techniques enable optimal and flexible usage of underlying resources. Along with storage resources and virtual compute, basic networking is also provided in order to connect the VMs [5]. However, with a rapid increase in the number and variety of workloads and applications moving to the cloud, cloud service providers have extended their offerings to include various services beyond basic storage, volumes, virtual servers, and network connectivity. Recent commercial cloud platforms support creation of virtual networks with access controls [6], various server types, and different storage models [7, 8].

Furthermore, cloud computing platforms such as IBM's Blue Cloud [9], Microsoft's Azure [10], and Amazon's EC2 [11] host various distributed applications. Service Level Agreement (SLA) of these services is a major issue for determining service providers' profit and user loyalty [12], but SLA requirements are not easy to be satisfied because of the high variations of Internet characteristics and workloads. Deploying new storage elements and new servers are costly choice that may result in high system management costs and low server utilization. Migration of VMs across PMs has the ability to improve service SLA compliance by reducing resource contention.

More importantly, cloud computing makes it possible to provision infrastructure, platform, or software as a utility to users. The technology that lies under the aforementioned cloud computing models is virtualization [13]. It enables sharing of infrastructural resources of servers, such as cores, processing I/O, and memory. However, the performance of an application running on cloud hosts may be significantly affected by varying service and infrastructural loads [14]. It

leads to the requirement of building a framework that allows monitoring of infrastructural resources.

1.1. Resource Orchestration

In the context of cloud computing, resource management is the process of allocating storage, compute, networking, and (indirectly) energy resources to a group of applications, in a way that offers to mutually satisfy the performance requirements of the applications, the users of the cloud resources and the infrastructure providers. The requirements of the providers revolve around effective and efficient resource utilization within the restrictions of SLAs with the cloud users. Effective resource utilization is usually attained through virtualization technologies, which enable statistical pooling of resources across applications and customers. The aims of the cloud users are influenced by the application performance, their availability, and scaling of available resources as a result of varying application demands [15].

Virtualization is one of the main technologies that enables cloud computing environments and that is utilized by cloud resource management processes. It refers to the process of creating an emulation of software or a hardware environment that appears to a user as a complete occurrence of that environment. Usually, virtualization software is utilized in cloud environments to create virtual storage disks, emulated computing servers (VMs), and sometimes to create virtual networks. In the context of resource management, virtualization enables to support the dynamic sharing of data center infrastructure between cloud hosted applications [16].

From a virtual network infrastructure perspective, the evolution of the SDN [17] paradigm provides seamless integration of application provisioning in the cloud with the network by means of automation and programmable interfaces [18]. With cloud applications requiring more command over the network, SDNs are naturally appropriate, whether in public or private

infrastructure as a service (IaaS) [19] or platform as a service (PaaS) [20] clouds. Several SDN solutions have been suggested for establishing virtual networks in multi-tenant clouds based on standard protocols such as using overlay networks and encapsulation (e.g., as discussed in [21, 22]), OpenFlow (e.g., as presented in [23]), and commercial solutions such as [24].

There are not only several advantages of the SDN-based cloud networking solutions (e.g., in terms of performance, flexibility, or scalability), but they also show support for particular models of cloud network orchestration, or specific types of network environments. For example, in case of overlay virtualization, policies and network tunnels are handled virtual switches located in hypervisors at the edge of the data center. This is a suitable solution for large-scale scenarios where only logical isolation and multi-tenancy is required, but it does not support fine-grained supervision over network routes to attain aims such as quality of service (QoS) or fast failover. OpenFlow offers this required level of fine-grained control by means of its programmable interface for packet forwarding and handling in virtual and physical switches [18].

In this thesis, an orchestration framework for resource management and provisioning is proposed, which enables to automate resource management and provisioning for users across clouds and network by: 1) exploiting APIs provided by the cloud provider in order to control/manage/allocate storage and compute resources; 2) interacting with SDN controller to get the details of the available resources, and instructing the changes/rules to manage the network based on the cloud service requirements. For resource allocation, a proposed algorithm allocates resources on the basis of unutilized resources in a pool of virtual machines. Furthermore, a service abstraction model is used, which is delivered to the orchestrator for provisioning the resources systematically and dynamically based on the requirements of the requests generated by remote end-user [106, 107].

1.2. VM Migration

Recently, the power consumption of cloud computing has increased, which is dependent upon the resource usage, especially the CPU usage. The problem of power consumption encouraged numerous researchers all over the world to study the power consumption of CPU [25], network [26], etc. Whereas other researchers focus on the consolidation of VM to reduce the number of active PMs to decrease the power consumption, but majority of these researchers did not study the adverse effects on performance to ensure a viable service to clients. The cloud computing suppliers introduce the cloud service in a couple of ways: a) assured service class (reserves the physical resource for VM demand), and b) best effort class (shares the cloud platform and guarantees the effort) [27]. The cloud providers provide assured service class more importance by preserving the physical resource for clients' instance. Even though the delivery and performance of services are assured, but still service with more security and high QoS is expensive. On the other hand, the second choice warrants the effort in distributing cloud platform among cloud clients. Therefore, such kind of service attains lesser cost than the first option. However, cloud computing in terms of SLA and QoS still experiences major challenges about the availability, performance, economic costs, and power consumption for the cloud operators. Thus, service providers need to strictly cope with the physical resource to assure a facility of high QoS without violating SLA. In IaaS, the fully loaded physical machine or aggressive consolidation of VMs on the same PM is the origin of SLA violation. Hence, a fit methodology is needed by the cloud providers to achieve the resource in a vibrant way as to warranty QoS for the clients. Thus, we consider that mostly common process which originates an overloaded machine is a destructive consolidation that can be eluded by making live migration at an appropriate time, as along with taking into account the cost of migration.

The main challenge is that which VM should be transferred and where the transferred machine should be located. Most studies focus on resource utilization to make decision for migration action [28, 29], but in this case migration action may result in performance degradation if not handled properly during the migration process. Different VMs have different workload characteristics and configurations [30] that lead to different migration costs.

In this thesis, a framework is proposed for dynamic VM migration in a cloud platform. The framework implements the proposed overload detection, VM selection, and VM allocation algorithms for dynamic VM migration. Furthermore, with the help of experiments, it is shown that the proposed algorithms outperform the algorithms that are considered for the purpose of evaluation [104, 105].

1.3. Monitoring of Utilized Resources

Various service models of the cloud can be monitored. Since platform and software cloud computing models like PaaS and software as a service (SaaS) respectively are a result of the abstractions built over the IaaS model, it becomes essential to monitor infrastructural resources in the first step in order to monitor at the platform or application level. There is no way that a software's (application's) performance can be guaranteed unless performance guarantees at the level of infrastructural resources like CPU, I/O Devices, and Memory are not given [31].

Both Cloud provider and clients (which could be Service providers in case of PaaS Clouds, or end users) are the beneficiaries of resource monitoring. Cloud providers have to monitor the current status of allocated resources in order to handle future requests from their users efficiently and to keep an eye on malicious users by identifying anomalous usage behavior [32]. Monitoring is also beneficial to the end-users since it helps them to analyze their resource requirements, and ensure that they get the requested amount of resources they are paying for. Also, it enables them

to know when to request for more resources, when to relinquish any underutilized resources, and what proportion of various physical resources are appropriate for the kind of applications they are running.

In this thesis, the monitoring of hosts in a cloud platform is conducted by implementing a new sFlow-based monitoring system that is not only tailored to the requirements of the cloud platform, but can also provide real-time visibility. It is carried out by deploying open-source Host sFlow agents with a Graphite collector which offers a complete, highly scalable monitoring solution. It periodically fetches sFlow metrics and other statistics from sFlow agents, and stores them in time-series format in Whisper RRD database, which can then be used for further analysis [101, 102, 103].

1.4. Research Problems and Objectives

This thesis tackles the research challenges in three broad areas of cloud computing. More precisely, the following research problems are investigated:

- **How to abstract service requirements.** It is required to abstract the service requirements, so that the orchestrator can easily understand the requirement of the requested services and provision/maintain the performance of networks to support the services.
- **How to allocate resources.** A module is required to allocate resources in the form of VMs in order to meet the requirements of the requested applications/services, which are requested by remote end-users.
- **How to manage cloud and network resources.** It should be made possible to automate resource management and provisioning for users across clouds and networks.

- **How to monitor hosts in a cloud platform.** A mechanism is required to monitor cloud and network infrastructure resources of a host in a cloud computing platform.
- **Where to store the monitored statistics.** An internal database server has to be deployed in order to store the monitored statistics retrieved from a host, which can then be used when allocating resources to a requested service.
- **When to initiate VM migration.** A decision has to be made on an appropriate time for migrating VMs from an overloaded host to avoid performance degradation.
- **Which VMs to select for migration.** Once an overloaded host is detected, it is necessary to select those VMs which are utilizing most of the resources, and place them on efficient hosts. The issue is to select the VMs to migrate from a pool of VMs.
- **How to choose the destination host for migration of VMs.** Selecting the most efficient host for the VMs selected for migration is another concern that impacts the performance of the VM migration process.

To overcome the challenges associated with the above research problems, the following objectives have been outlined:

- Examine, classify, and analyze the research in the area of resource allocation, monitoring, and VM migration to grasp the existing approaches and techniques.
- Deploy a physical testbed based on OpenStack to address the above issues in a cloud platform.
- Design a service abstraction communicator to deliver the abstracted service requirements to the orchestrator.
- Develop an algorithm to select the most efficient VMs for allocation.

- Design a mechanism to monitor the infrastructural resources of a host, and then store them in internal database server.
- Design and implement algorithms to detect an overload host, select VMs to migrate, and place them on the most efficient host.

1.5. Thesis Organization

The core chapters of this thesis are structured as shown in Figure 1.1. They are organized as follows:

- Chapter 2 presents an overview of the related and previous literature on resource allocation in cloud computing, monitoring of resources in clouds, and VM migration.
- Chapter 3 provides an overview of OpenStack, some of its core components, and its deployment on a physical testbed.
- Chapter 4 proposes an orchestrator that enables to automate resource management and provisioning for users across clouds and networks. The comparison of the performance analysis of the proposed algorithm for resource allocation with the random resource allocation scheme is performed. Moreover, the mechanism adopted for the monitoring of infrastructural resources of a host in the cloud computing platform is presented in this chapter.
- Chapter 5 proposes a framework for dynamic VM migration in OpenStack clouds. The framework implements the proposed overload detection, VM selection, and VM

allocation algorithms for dynamic VM migration in OpenStack. Furthermore, with the help of experiments, it is shown that the proposed algorithms outperform the algorithms that are considered for the purpose of evaluation.

- Chapter 6 concludes the thesis with a summary of the work in this research, discussion of possible future research directions, and final remarks.

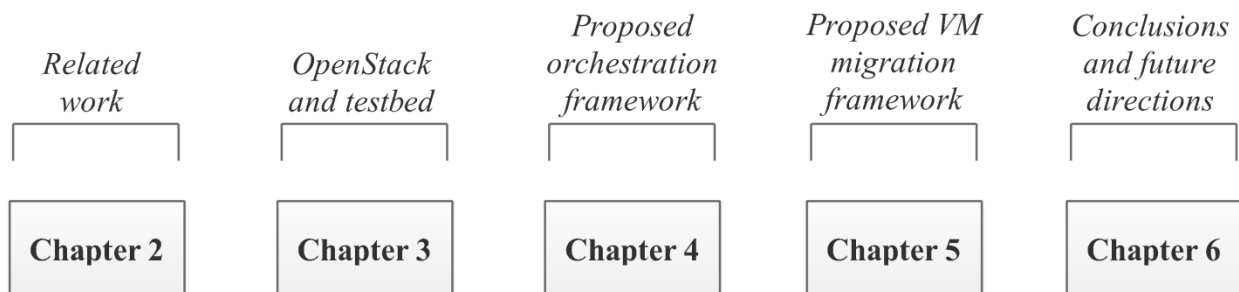


Figure 1.1: The thesis organization

Chapter 2

Related Work

Since the work in this research mostly revolves around a cloud computing platform, it is important to first discuss some major concepts in cloud computing. The discussion is then extended to summarize the related work already done in the area of orchestration of resources, their monitoring, and the migration of VMs. This chapter shows a direction on how to reduce or avoid the limitations of the existing mechanisms.

2.1. Cloud Computing

Cloud computing, with the groundbreaking potential of turning computing into a fifth utility, after gas, electricity, water, and telephony, has the potential to alter the face of Information Technology (IT), especially the features of service management and service-renderion. Though

there are numerous ways of describing the phenomenon of Cloud Computing, we chose the one proposed by NIST (National Institute of Standards and Technology). According to NIST, Cloud Computing is defined as *"A model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., net works, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"* [33]. Lightly speaking, Cloud computing signifies a novel way to organize computing technology to provide users the ability to store information, access, share and work on using the Internet [34]. The cloud itself is a net-work of data centers, each poised of several thousands of computers working together with the ability to perform the functions of software on a business or personal computer by providing users access to influential platforms, applications, and services distributed over the Internet. It is in spirit a set of network enabled services that is capable of providing inexpensive, customized and scalable computing infrastructures on demand, which could be retrieved in a pervasive and simple way by a variety of geographically isolated users. The Cloud also guarantees application based QoS assurance to its users. Thus, Cloud Computing provides the customers with large pools of resources in a clear way along with a mechanism for handling the resources so that a user can access it universally and without experiencing needless performance overhead. The best way to describe Cloud Computing would be to term it as "Everything as a Service" abbreviated as XaaS [35]. Below, the key features of Cloud Computing are summed up:

- Agility – assists in quick and low-cost re-provisioning of assets.
- Location Independence – resources can be retrieved from anywhere and everywhere.
- Multi-Tenancy – resources are distributed amongst an enormous pool of users.
- Reliability – reliable availability of resources and computation.

- Scalability – vibrant provisioning of data aids in eluding various bottleneck scenarios.
- Maintenance – users (organizations/companies) have less work in terms of resource management and upgrades, which in the new model will be controlled by service providers of Cloud Computing.

However, Cloud Computing doesn't infer that it contains only a single cloud. The term "Cloud" represents the Internet, which in itself is a network of networks. Also, not all types of remote computing are Cloud Computing. On the contrary, Cloud Computing is nothing but services presented by providers who might have their own systems in place. [36]

2.1.1. Cloud Computing Methodologies

Cloud computing is based on two main technologies – (i) Service Oriented Architecture and (ii) Virtualization.

- Service Oriented Architecture (SOA)*: Since the model of Cloud computing distinguishes all tasks accomplished as a "Service" rendered to users, it is said to follow the SOA. This architecture includes a flexible set of design principles used during the phases of system integration and growth. The deployment of a SOA-based architecture will deliver a loosely-integrated set of services that can be used within multiple business dominions. The supporting technologies in SOA allow services to be composed, discovered, and implemented. For instance, when an end-user desires to complete a particular task, a service can be employed to determine the necessary resources for the task. This will be tracked by a composition service which will propose the road-map to deliver the desired quality and functionality of service to the end-user. [37, 38]

ii. *Virtualization*: The idea of virtualization is to relieve the user from the burden of resource installation and purchases. The Cloud brings the resources to the users. Virtualization may refer to *Hardware, Memory, Storage, Software, Data* and *Network* [39]. Virtualization has become an essential ingredient for almost every Cloud; the most certain reasons being the ease of encapsulation and abstraction. Amongst the other vital reasons for which the Clouds incline to approve virtualization are:

- Server and application consolidation – as many applications can be run on the identical server resources can be applied more efficiently.
- Configurability – as the resource necessities for various applications could vary considerably, (some require higher computation capability, some need large storage) virtualization is the only clarification for customized configuration and accumulation of resources which are not realizable at the hardware level.
- Increased application availability – virtualization allows rapid recovery from unintended outages as simulated environments can be backed up and migrated with no disruption in services.
- Improved responsiveness – maintenance, monitoring and resource provisioning can be programmed, and common resources can be reused and cached. [40]

In addition, these benefits of virtualization tend to facilitate the Cloud to meet stringent SLA requirements in a business setting which otherwise cannot be easily achieved in a cost-effective manner. Without virtualization, systems have to be over provisioned to handle peak load and hence waste valuable resources during idle periods.

2.1.2. The Cloud Architecture

Geared with the knowledge of Virtualization and SOA, we now take a look at the complete Cloud architecture. From the end user's viewpoint, Figure 2.1 represents a basic Cloud Computing architecture including multiple components. Cloud architecture closely resembles the UNIX viewpoint of involving many components which work together over worldwide interfaces [34]. Recall that the Cloud computing model represents a Service oriented mechanism of dispatching and managing resources. Before we enquire the real architecture of Cloud computing it will be useful to inspect the possible characteristics that will be necessary to understand such a system. It is common understanding that the architectural necessities of the Cloud will differ depending on the use for which the Cloud is being applied. For example, social networking applications like Orkut and Facebook will have a very dissimilar set of constraints, requirements, and deliverables from the architecture in comparison to, say, a distant patient health monitoring application. However, some common architectural features can still be recognized. For example, (i) the system should be scalable with the ability to comprise thousands to perhaps tens of thousands of members. (ii) It should be able to operate between numerous service needs and effectively share resources between its users. (iii) The system should be easy to upgrade and maintain, preserving user transparency during these practices. (iv) As defined previously, handling resources like servers and storage devices almost, thereby forming a virtual organization is extremely crucial.

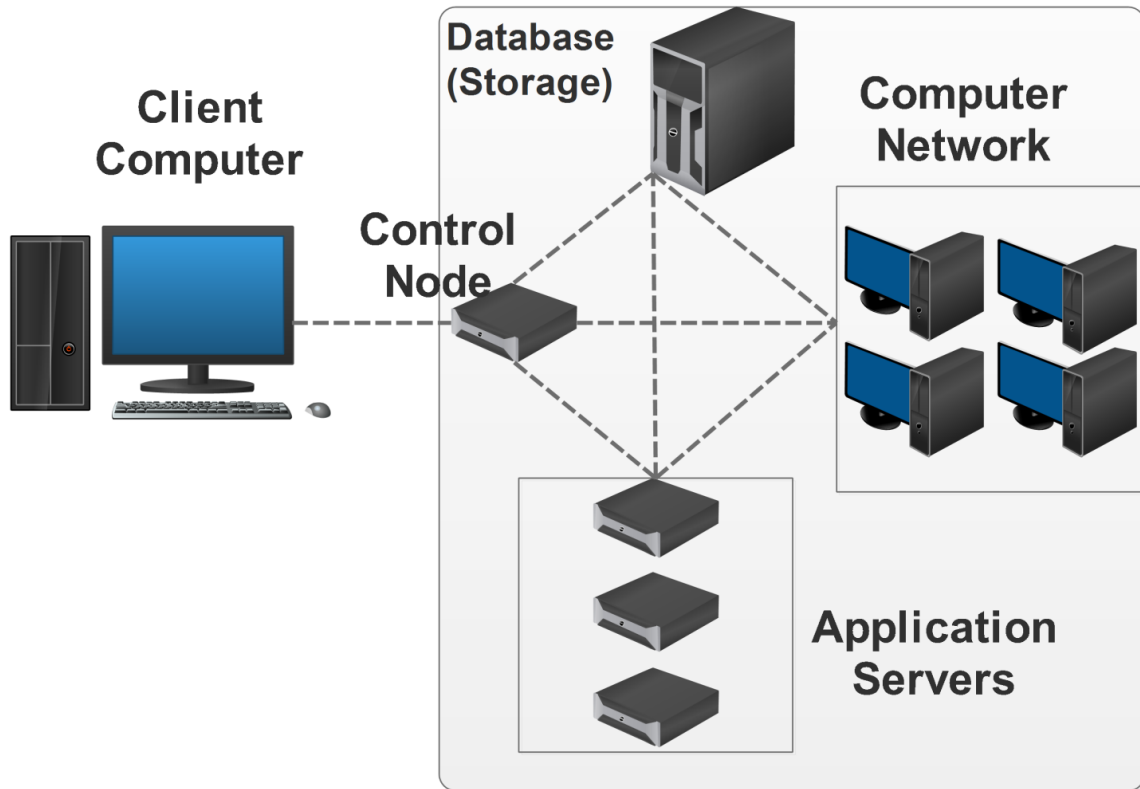


Figure 2.1: Basic cloud computing architecture

To diminish the problem of designing modified Cloud architecture for each application and also to rationalize the architecture design process of the Cloud, researchers resorted to the age old concept of a comprehensive "Layered approach" [41]. As with the standard 7-layer OSI model of data networks [42], the layered model in Cloud computing functions the same general purpose. Depending on the service requisite of an application, these layers are scuffled to produce a customized architecture. The layered architecture follows the standard of SOA which forms the primary Cloud computing model. The components of a basic layered architecture are illustrated in the Figure 2.2, below [43], namely the Client, its mandatory Services, the Storage requirement, the Applications run by the Client, the Platform to run these applications, and lastly the Infrastructure needed to back the Client's computing requirements. We dedicate a few sentences on each mentioned component below.

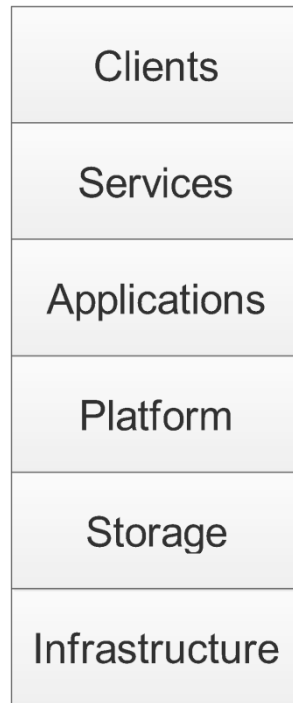


Figure 2.2: Layered architecture for a customized cloud service

Clients: the Clients of a Cloud include computer software and computer hardware that relies on the computational competence of the Cloud for application delivery. Examples include mobile devices, computers, browsers and operating systems.

Services: this refers to the different service prototypes made available by the Cloud. These services include IaaS, SaaS, and PaaS. This layer acts as a distributor between the user and the enormous amount of resources available to the user. A resource comprises of "solutions, services, and products that are supplied and consumed in real time over the Internet" [43], e.g. Search Engines and location services.

Application: the Cloud allows user activity tracking and resource management from central locations rather than at individual customer's site, allowing customers to access applications distantly through the Internet. Cloud application services provide software as a service over the

Internet, eradicating the need to install and run the application on the computer of client, thereby simplifying support and maintenance at the customer's end e.g. Peer-to-Peer computing and Web Application.

Platform: it enables distribution of applications without the complexity and cost of managing and buying the basic software and hardware layers. This layer supplies a solution stack and a computing platform as a service, often using Cloud infrastructure and supporting Cloud applications e.g. Web Application Frameworks such as Web Housing and Ruby on Rails.

Storage: the storage layer contains computer software and computer hardware products that are explicitly designed for the storage of Cloud services. Computer hardware includes enormous data centers that are used for resource sharing e.g. Nirvanix SDN (Storage Delivery Network) and Amazon SimpleDB.

Infrastructure: this layer supplies computer infrastructure, normally a platform virtualization environment as a service. It contains management of virtual resources as well. Rather than buying software, servers, data center space, customers instead buy those resources as a fully subcontracted service. e.g. Database services and Network Attached Storage.

The main benefit of such a layered architecture is the simplicity with which they can be altered to suit a specific service. The way in which these components interrelate leads to various architectural styles. The two basic architectural styles on which most of the services are based are:

- **Outside-In:** This architectural style is intrinsically a top-down design highlighting the functionality of the components. Applying this style leads to an improved architectural

layering with several functionalities. It infuses more viability allowing better interoperability and integration of components.

- **Inside-Out:** This architectural style is intrinsically a bottom-up design which takes an infrastructural perspective of the components. This style is more inclined towards application than services. [44]

It is to be highlighted that integrating new functionalities in a pre-existing architectural system is done in an incremental fashion. The ease of converting an existing architecture into another depends on the integration, difficulty of the architecture and the functionalities of the components. The vast landscape of the services and their increasing difficulty has led to application of groundbreaking architectural styles and several mix architectures. [45, 44]

2.1.3. Cloud Services

The Cloud can deliver us with numerous service models and services. They comprise PaaS, SaaS, DaaS ([Development, Database, Desktop] as a Service), IaaS, HaaS (Hardware as a Service), FaaS (Framework as a Service), BaaS (Business as a Service), and OaaS (Organization as a Service) amongst others [35]. Nevertheless, Cloud Computing products can be generally categorized into three main Services (IaaS, PaaS, and SaaS) which are concisely described below:

Infrastructure-as-a-Service (IaaS): This service supplies hardware related services such as virtual servers and storage on a pay-as-you-go basis. The main benefit of IaaS is the usage of modern technology at all times with respect to computer infrastructure which permits users to attain quicker service. Organizations can utilize IaaS to rapidly form novel versions of applications without experiencing needless purchase and configuration delay. On-demand scaling through use-based billing and resource virtualization makes IaaS capable enough for any

sort of businesses. The main companies already supplying IaaS are Amazon [46, 47], Rackspace, GoGrid, AT&T and IBM. [48]

Platform-as-a-Service (PaaS): PaaS offerings may contain services for application development, application design, deployment, testing, and hosting as well as application services like web service integration and marshalling, team collaboration, scalability, security, persistence, storage, state management, application instrumentation, application versioning and developer community facilitation. These services may be provisioned as a combined solution over the web, providing an ongoing managed higher-level software infrastructure for developing specific classes of applications and services. The platform comprises the use of core computing resources, usually billed similar to IaaS products, while the infrastructure is vague away below the platform. Foremost companies providing PaaS are Google's AppEngine [49], Microsoft Azure, and Force.com etc. [50, 43, 44]

Software-as-a-Service (SaaS): Delivers explicit already developed applications as partially or fully remote services. Occasionally it is in the form of web-based applications and other times it contains of typical non-remote applications with Internet-based storage. It allows an operator to use the provider's application using a thin client interface. Operators can acquire a software application hosted by the Cloud seller on pay- per-use basis [51]. It is a multi-tenant platform. The creator in this field has been Salesforce.com presenting online Customer Relationship Management (CRM) space. Other examples are Microsoft's hotmail, Google docs, online email suppliers such as Google's Gmail and Microsoft's online form of office called BPOS (Business Productivity Online Standard Suite). [52,50,43,44]

2.1.4. Cloud Applications

Cloud Applications are not only developed with a business viewpoint but also take into account events concerned with sharing and socializing information. This data may be as simple as testing news headlines or more delicate in nature, like medical information or searches for health. Thus Cloud Computing is often a superior choice than local servers managing such applications.

Virtualization is the core foundation of Cloud Computing, thus it is completely enabled with virtual appliances. A virtual appliance is an application that has all its components hustled and rationalized with the operating system [34]. The main benefit of a Cloud Computing application is that the provider can run numerous occurrences of an application with least labor and expenditure. A Cloud service provider needs to expect a few problems before initiating its application in the Cloud computing environment. Keeping in mind the problems an application must be designed to tolerate failure, scale easily and include management tools [53]. These issues are conferred as follows:

- *Scale:* Application in a Cloud environment needs to have maximum scalabilities and to guarantee this; one should begin developing the application in the easiest ways avoiding difficult design patterns and improvements. The next step would be to divide the functions of an application and join them lightly. The most vital step in confirming on demand scalability of an application is sharding, which can be designated as splitting up the system into numerous minor groups instead of scaling the solo system up so as to serve all users.
- *Failures:* Any application due to one or the other reason is guaranteed to face failure. To bear failures, an application must function in an asynchronous fashion and one should spread the load across many clusters so that the impact of failure gets scattered. The best

method to bear failures would be to test the application for all types of failure circumstances, and also operators should be conscious about the real cost sustained if an application faces any kind of failure.

- *Management Tools:* Having a suitable management tool aids in programming the application configuration and updates, thereby decreasing management overhead. The management system helps in not just reducing economic expenses but also leading to improved usage of resources. The most problematic and expensive problem that can be held with a proper management tool is inconsistency [54]. This helps in bringing an application that can boast of reliable performance.

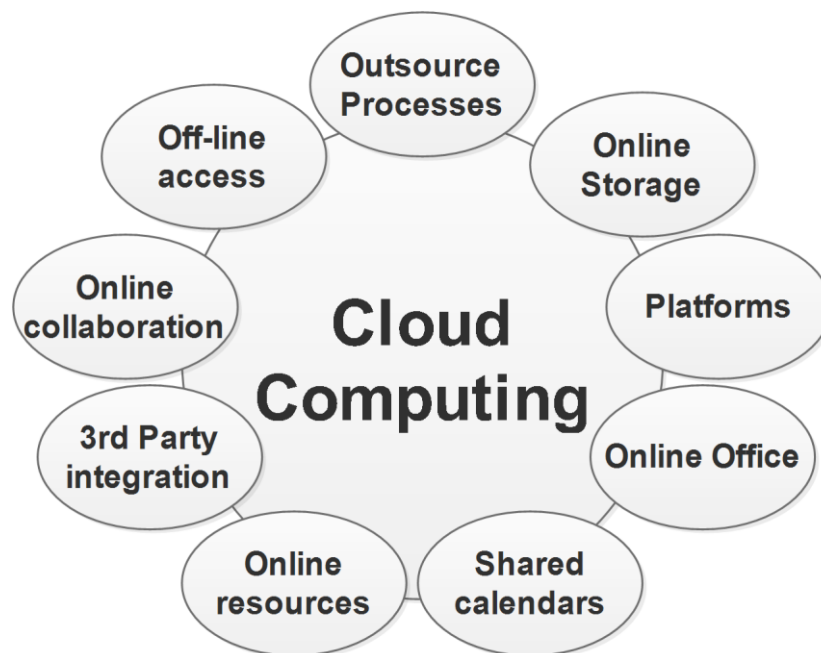


Figure 2.3: Cloud applications

Applications in Cloud Computing have been planned keeping in mind the numerous types of facilities offered by it. Cloud Computing has impacted people's awareness of applications over the Internet. The applications of Cloud computing are rationalized with each and every field of

sciences. In Figure 2.3 we have emphasized a few areas in which Cloud Computing has shown great prospect for developing various applications that have nurtured the growth of enterprises and businesses.

These applications can be classified into broad areas such as security, health industry, military applications, platforms and software usages, datacenters and storage, applications with high performance computing resources, and the growing need for virtual environments. [45, 55]

2.2. Cloud Resource Orchestration

Cloud resource orchestration [56] involves the design, management, manipulation and withdrawing of cloud resources, i.e., storage, compute, and network, to apprehend customer needs, while conforming to operational aims of the cloud service suppliers at the same time. We argue that cloud instrumentation is highly complex. First, as many new proposals [57, 58, 59, 60, 61] have uttered, cloud management is intrinsically complex due to the heterogeneity, scale, concurrent user services and infrastructure, that share a common set of physical resources. Second, arrangements of various resource types interact with each other e.g. the locations of VMs have an effect on storage placement, which in turn affect bandwidth consumption within and outside to a data center. Third, cloud resources have to be organized in a fashion that not only comprehends provider operational objectives, but also assures that the customer SLAs can be regularly met as runtime conditions change.

Resource management in cloud system is dissimilar from the customary network systems and the characteristics of the operators, cloud's services, and architecture yields some necessities in resource management which should be sensibly addressed [62, 63, 64].

- Resource Provisioning: Cloud providers should use robust resource management methods in cloud environment which competently deal with the problem of VM placement and

VM provisioning. It is vital to have clear explanation for how to match tasks to available VMs.

- **Scalability:** Cloud providers should use flexible mechanisms to vigorously scale up and down resources on request in terms of VMs. A cloud customer needs to access the resources on-demand. The cloud system may scale up to extra available resources when the system is undergoing high user demand and it may later scale down when the user demand reduces. A scalable cloud system also checks the futile VMs and turns them off if they are futile for more than a definite time. Increasing the workload on existing resources is called scaling in and increasing the workload by adding resources on demand is called scaling out. A flexible cloud system adopts these scaling methods to increase the system's utilization.
- **Load balancing:** Cloud providers should use vigorous load balancing algorithms that can intensely control the sudden changes in workload and provide improved results in diverse and dynamic environments. The applied load balancing algorithms should also inhibit unsuitable utilization of the resources in the cloud system; under-utilization results in waste of energy and resources and over-utilization leads to slower response times of the applications. To tackle this problem, migration of VMs is one solution that is called as autonomic management of resources in cloud.
- **Availability:** The system becomes inaccessible due to causes such as load fluctuations, software or hardware failures, long waiting time of jobs, etc. The customary method to deliver high availability is providing additional idle resources to be used in case of failures which can result in wasting resources. Dynamic approaches are required in cloud

systems to repeatedly sense any failure and shift the tasks to the accessible resources in a short time.

- **Overheads:** Workloads to be managed on virtualized cloud platforms may be network I/O intensive or CPU intensive. The in-depth understanding of these network bandwidth and overheads as a limitation is essential for effective resource management in the cloud. All in all, the orchestration process is complex and potentially error prone if performed manually, and motivates the need for better management tools that enable us to automate part or all of the decision process.

Recently, the cloud computing resource allocation algorithm and model have been widely studied. Among several resource allocation strategies, [4] proposed a virtual resource management model that allocates resources by means of the division and resource reserve strategy, while guaranteeing the effectiveness for the users to consume the virtual resources. In [3], authors proposed a virtual resource allocation mechanism based on utility, but the authors only considered one dimension as the CPU. In this thesis, the problem of virtual cloud resource allocation has been addressed by expanding it two dimensions, i.e., CPU and memory.

2.3. Monitoring of Infrastructural Resources

APIs are offered by modern operating systems to report the utilization about the computing resources from the kernel. By calling these APIs, operating systems also provide a various computing resource utilization utilities to help administrators understand the current states of the system [65]. The service systems are measured by Quality Attributes [66] like performance, availability, interoperability, security, and modifiability that are derived business goals. There are several different ways to express the performance quality attribute, including throughput,

response time, or constraints on resource utilization [67]. The most concerned performance counters that are collected periodically by resource utilization utilities from VMs are shown in Table I.

TABLE I. PERFORMANCE COUNTERS.

Category	Counter	Explanation
Host	Server ID	Server identification
	Time	The performance of that period of time.
Application	Transaction Count	Average transaction count in that period.
	Response Time	Average response time in second for a transaction.
CPU	% User	CPU average utilization in percentage by Application codes.
	% Wait	CPU average utilization in percentage waiting for outstanding I/O.
	% Total	CPU average utilization in percentage for everything.
	% Max	CPU peak utilization in percentage for everything.
Memory	Page Faults	Number of page faults.
	% Used	Percentage of used space in physical memory and paging space.
Disk	Read/Write-KB/s	Average data transfer rate of read and written data in kilobytes per second in the interval.

2.3.1. Existing Monitoring Frameworks

There are various open-source cloud computing tools and resources available, some of them with an inbuilt monitoring capability. For example, Eucalyptus [68] implements IaaS private cloud, which can be accessed via an API compatible with Amazon S3 and Amazon EC2 [69]. The monitoring service offered by Eucalyptus enables the guest virtual machines to be integrated with other cloud computing tools like Ganglia and Nagios.

OpenNebula [70] consists of a monitoring subsystem which is able to monitor the exclusive data transmitted/received, available memory, and CPU utilization of the created virtual machine with the help of configured hypervisor drivers. Table II shows the samples from OpenNebula for a particular VM. Net TX and RX shows total number of bytes transferred and received respectively.

TABLE II. MONITOR INFORMATION FROM OPENNEBULA MONITORING FRAMEWORK.

Metrics	Value
CPU	15%
Memory	1048576 Bytes
Net TX	133181200 Bytes
Net RX	185401946 Bytes

Despite the capabilities of Eucalyptus, and OpenNebula, collectd forms the basis of a lightweight monitoring agent which supports high resolution probing and a wide variety of metrics. collectd is also able to probe at different layers, e.g. at the OS, middleware, and the application layers, and thus can support both low-level and high-level monitoring. It can also be

customized with the help of plug-ins, which allows the addition of more application-specific probes [71].

Although Eucalyptus is an open source product for building AWS compatible private clouds, it does not support multicloud infrastructure. OpenNebula does have a support for multi-cloud infrastructure, but does not provide auto-scaling [72]. With this increasing cloud complexity, efforts needed for management and monitoring of cloud infrastructures need to be multiplied. The size and scalability of clouds when compared to traditional infrastructure involves more advanced monitoring systems that have to be more scalable, effective and fast. Technically, this would mean that there is a demand for real-time reporting of performance measurements while monitoring cloud resources and applications. Therefore, cloud monitoring systems need to be advanced and customized to the diversity, scalability, and high dynamic cloud environments. One such attempt is presented in this thesis.

2.4. Dynamic VM Migration

One of the early works [73], which focuses on dynamic VM migration, was implemented to offload an overloaded physical machine. Although, the model designed for the optimization of the dynamic VM allocation has considered the cost of VM migration, the authors did not implement any algorithm for deciding when it was vital to perform the VM allocation optimization. The designed model was invoked periodically to adjust the VM allocation, which needs a supplementary performance any necessary demand for optimization operation.

OpenStack, one of the well-known cloud platforms for both public and private clouds, was announced in 2010 [74]. Among several existing sub-projects, OpenSack Nova [75] is the core project of OpenStack, which provides IaaS on demand. An instance/VM can be launched by means of OpenStack Nova on the efficient compute node, which meets the customer's

requirements. Although, OpenStack supports live migration technique, the administrator has to manually intervene within the appropriate time to migrate a VM instance from one compute node to another. This feature of OpenStack is useful whenever a compute node, where many VM instances are running, needs to redistribute or maintain load. However, the necessary requirement for conducting a dynamic live migration, which guarantees the QoS and SLA, is the VM migration decision taken at a suitable time. This lack of dynamic VM migration leads to search for an appropriate method to monitor and measure the load in order to migrate VMs to efficient compute nodes. This method should be adaptable with the resource usage requirement of on demand up/down scaling. In addition, recent virtualization techniques do not provide enough performance isolation among the VMs [76]. The contention for physical resources among VMs leads to different performance impact level among the VMs.

A dynamic resource placement for OpenStack has been implemented in [77], which is based on a protocol to interact between peer servers. The operation was periodically performed, and the peer servers were chosen randomly without any requirement to exchange their state. Furthermore, performance degradation of migration operation was not taken into account.

In [78], authors proposed a technique for placement of VMs when the SLA of hosted applications was violated. With the same technique, authors in [79] examined the performance of VM placement decision by automating the response time which was described in the SLAs. This technique needed more time for VM placement and the duration of SLA violation was increased. In [80], authors have used the threshold based on the utilization of resources such as network bandwidth, RAM, and CPU to decide when to migrate VM(s) from an overloaded PM to an efficient one. In [81], authors handled resource placement based on the response duration of QoS conditions at the server and the cluster level by implementing control loops. In [82], a remaining

utilization-aware (RUA) algorithm has been proposed for VM allocation. In this work, the overloaded PM is first detected by means of the proposed algorithm, and then the migration for some VMs is conducted.

Authors in [83] proposed a system for managing resources of virtualized data centers by means of local and global policies. The system on the local level was responsible for implementing the power handling strategies for a guest host, whereas the global policies were responsible for managing the consolidation of VMs. However, the QoS requirements were not taken into account by the global policies.

In [84], authors proposed an algorithm for dynamic VM migration based on time-series analysis, and predicting of the resource demand. In [85], the authors devised the issue of VM placement as stochastic optimization issue with a constraint on the host overload probability, considering multiple resource constraints. A disadvantage of all the aforementioned approaches is that they are centralized, i.e., a single algorithm running on the master host (controller node in OpenStack) limits the global view of an overall system to enhance the VM instance placement. Furthermore, by the use of such a centralized approach, the scalability of the system is limited with an increase in the number of physical machines (compute nodes in OpenStack).

In order to determine the time to invoke the migration of VMs from a host, a heuristic for setting a utilization threshold was first proposed in [16]. The main idea of their heuristic was to set a utilization threshold based on the CPU utilization. In this thesis, the same approach is followed, but the host is detected to be overloaded if the mean of the last n CPU utilization quantifications is greater than the pre-defined threshold value.

After determining an overloaded host, the next step is to select the VM(s) to migrate from one host to another. The three VM selection policies used in this thesis are given as follows:

- Minimum Migration Time (MMT): According to this policy, the VM that is selected to migrate requires minimum amount of time to complete migration, compared to other VMs allocated to the host.
- Random Choice (RC): This policy randomly selects VMs to migrate.
- Maximum Utilization (MU): Based on this policy, [3] selected VMs which are required to be migrated in such a manner that VMs with highest CPU utilization, compared to other VMs, are considered first.

The VM placement problem is handled by mapping VMs to the most efficient hosts. In the last couple of years, there has been tremendous interest in this area and several algorithms have been proposed for VM placement. Most works focus on the CPU utilization as the most important resource and characterize hosts in terms of their CPU capacity and VMs in terms of their CPU load [1-5, 7, 8, 10, 20, 25]. In [X], authors use the First Fit (FF) algorithm which checks all the hosts and finds the suitable host where the CPU utilization is the minimum. On the other hand, some works make the problem multi-dimensional by also taking into account some other resource types such as I/O and memory [6, 9, 17, 18, 28]. In this thesis, the latter approach has been followed by considering RAM as well as CPU when determining efficient compute nodes for VM placement.

More precisely, in this work, an approach has been proposed and implemented in a real environment for the well-known cloud computing software called OpenStack. In this approach, every compute node sends RAM and CPU utilization statistics of each VM instance that is deployed on this compute node. Based on these statistics, the algorithms running on the control node detect an overloaded VM instance, select it, and place it on an efficient compute node.

Chapter 3

Overview of OpenStack and its Deployment

3.1. Introduction

OpenStack is an open source cloud operating system that is contained with numerous open source sub-projects that control large pools of networking resources, storage and compute throughout a datacenter, all managed through a dashboard that gives administrators control while enabling their users to provide resources through a web interface. These services are provided in order to develop an Infrastructure Service cloud [75].

All OpenStack services expose a RESTful API for communication among them and use HTTP protocol for each data exchange. Moreover, by using this type of API, fault-tolerance and

scalability is provided to the system. Nevertheless, the real communication is provided by separate processes devoted for certain items execution, perhaps running on other machines, connected through AMQP and using a message bus which is RabbitMQ by default.

Regarding the service data storage, each OpenStack service has its own SQL based database for keeping state information, but in order to provide flexibility and throughput it is likely to implement a virtual multi-master databases. In order to deliver all the resources required to create an Infrastructure service cloud, the following services of OpenStack have been used in this thesis:

- **Keystone:** Identity service
- **Nova:** Compute service
- **Neutron:** Networking service
- **Glance:** Image service
- **Horizon:** Dashboard
- **Ceilometer:** Telemetry service

Figure 3.1 depicts the conceptual architecture of a typical OpenStack environment.

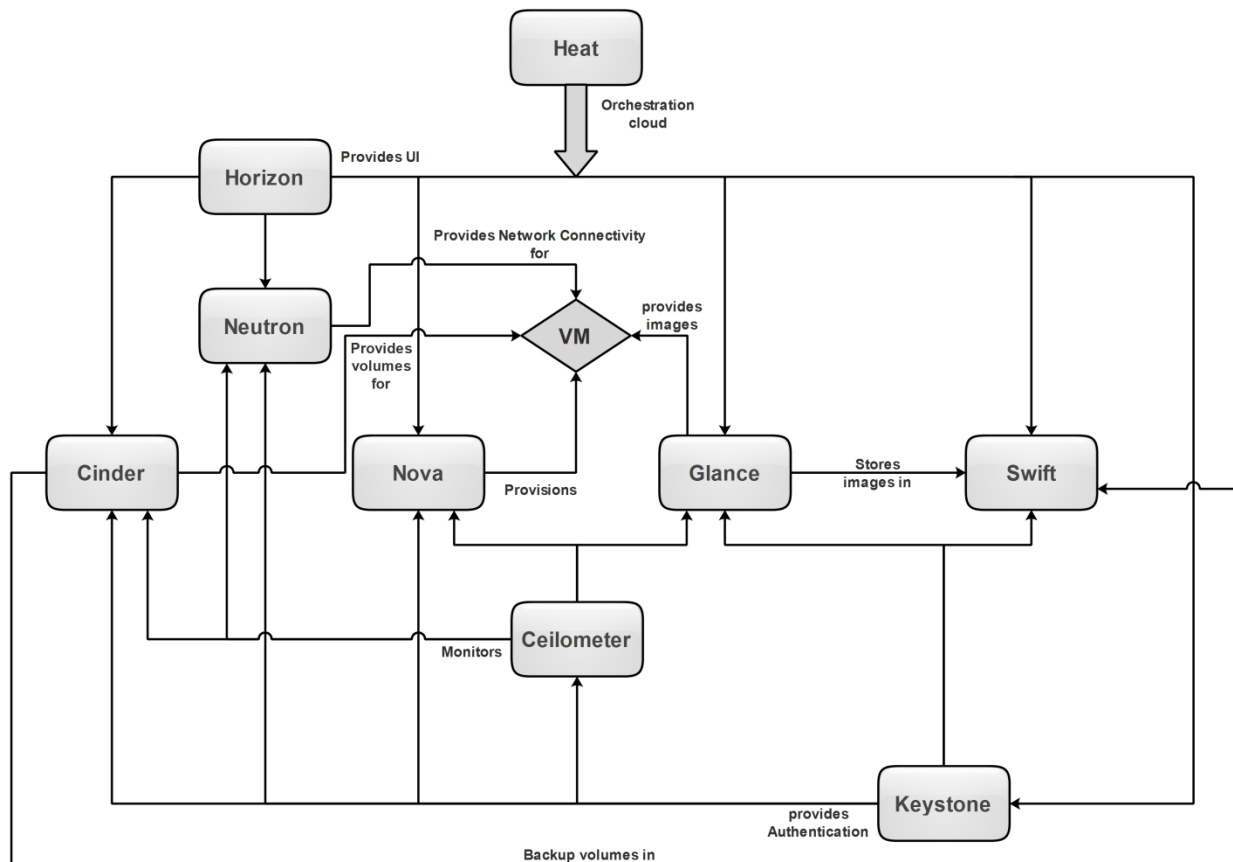


Figure 3.1: OpenStack conceptual architecture

3.2. Keystone

Keystone service [86] provides a common verification and permission store for OpenStack services. Keystone is responsible for operators, their parts, and to which project (tenants) they belong to. Furthermore, it provides a list of all other OpenStack services which confirm on Keystone a user's request. Essentially, Keystone has primary functions to control the validation and authorization of a user:

- User management: Keystone keep track of users and of what they are endorsed to do. This task is made by examining and administering the existing association among users, tenant and roles.

- Service catalog: Keystone delivers a list of accessible services and where their API endpoints are located.

3.2.1. Architecture

Keystone's architecture is organized in a group of internal services exposed on one or many endpoints which can be seen in Figure 3.2:

- Identity: The identity service delivers auth qualification authentication and data about Operators, Roles and Tenants as well as any related metadata.
- Token: The token service authenticates and manages Tokens used for validating requests once a user/tenant's credentials have already confirmed.
- Catalog: The Catalog service provides an endpoint registry used for endpoint detection.
- Policy: The policy service provides a rule-based authorization engine.

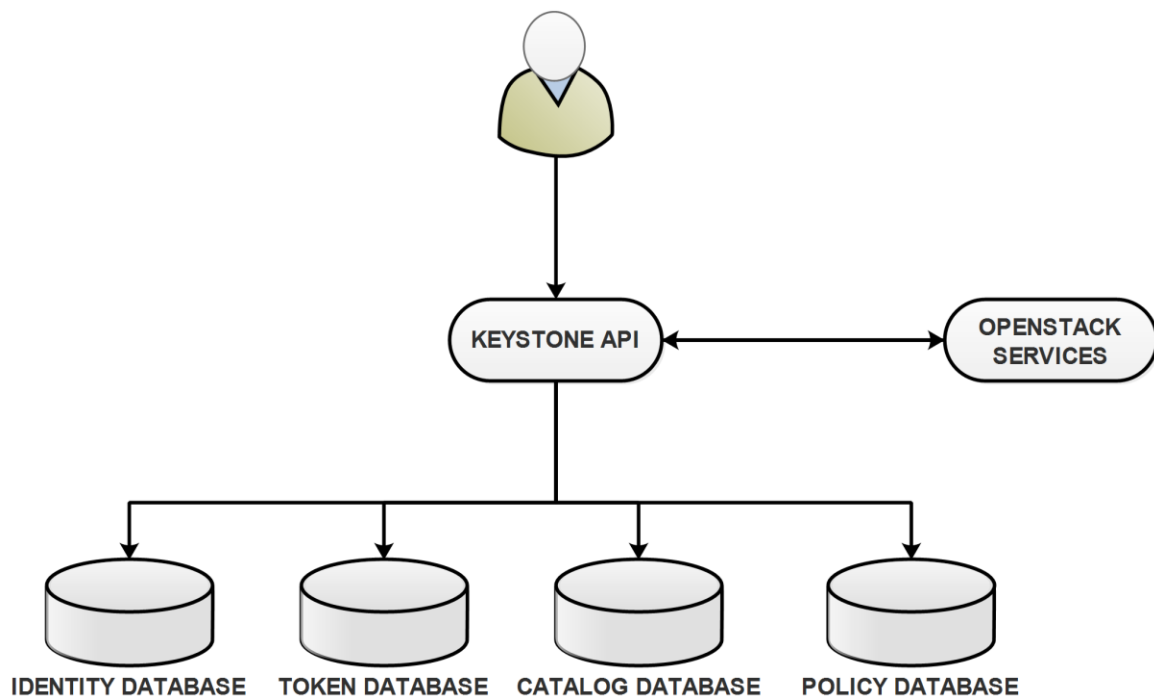


Figure 3.2: Keystone architecture

3.3. Nova

Nova [87] is liable for managing and hosting cloud computing systems, which is the main part of an IaaS system and most complex and distributed service of OpenStack. Nova interacts with Keystone for verification and Glance for disk and server images whose access is restricted by users and tenants. Its main units are applied in Python and configure a large number of processes which cooperate to turn API requests into running virtual machines.

3.3.1. Architecture

Nova uses a SQL-based central database that is mutual among all components in the system whose data fits into an SQL database appropriately. For small placements it is an optimum result but for larger ones, multiple data stores with a suitable accumulation system would be used. Nova architecture is defined by numerous components as shown in Figure 3.3.

- **API:** It is responsible for accepting HTTP requests, transforming commands and interacting with other components. It further includes:
 - nova-api service: Responds and accepts to end user compute API calls.
 - nova-api-metadata service: Receives metadata requests from instances.
- **Compute:** It is responsible for managing communication with virtual and hypervisor machines. It has the following components:
 - nova-compute service: It produces and terminates simulated machine occurrences through available hypervisor APIs, like libvirt for KVM, XenAPI for XenServer/XCP, VMWareAPI or QEMU for VMware.
 - nova-scheduler service: It takes a virtual machine occurrence request from the queue and governs on which compute server host it runs.

- nova-conductor module: It facilitates interactions between the database and nova-compute service and removes direct accesses to the cloud database made by the nova-compute service.
- **Networking:** It is responsible for managing IP addressing, VLANs and bridges. It further includes:
 - nova-network worker daemon: It receives networking tasks from the queue and controls the network. It performs tasks like changing IPtables rules or setting up bridging interfaces..
- **Console:** It permits end users to access their instance's console through a proxy. It comprises of the following components:
 - nova-consoleauth daemon: It approves tokens for users that console proxies provide.
 - nova-novncproxy daemon: It delivers a proxy for retrieving running requests through a VNC connection.
 - nova-cert daemon: It provides x509 certificates.
- **Image:** This area manages the interaction with Glance for image use. The components for this communication are:
 - nova-objectstore daemon: It is an S3 interface for registering images with the OpenStack Image Service.
 - euca2ools client: It is a set of command-line translator commands for managing cloud resources.

- **Database:** It is responsible for storing most run-time and build-time states for a cloud infrastructure like existing instance types, available networks, instances in use, and projects.

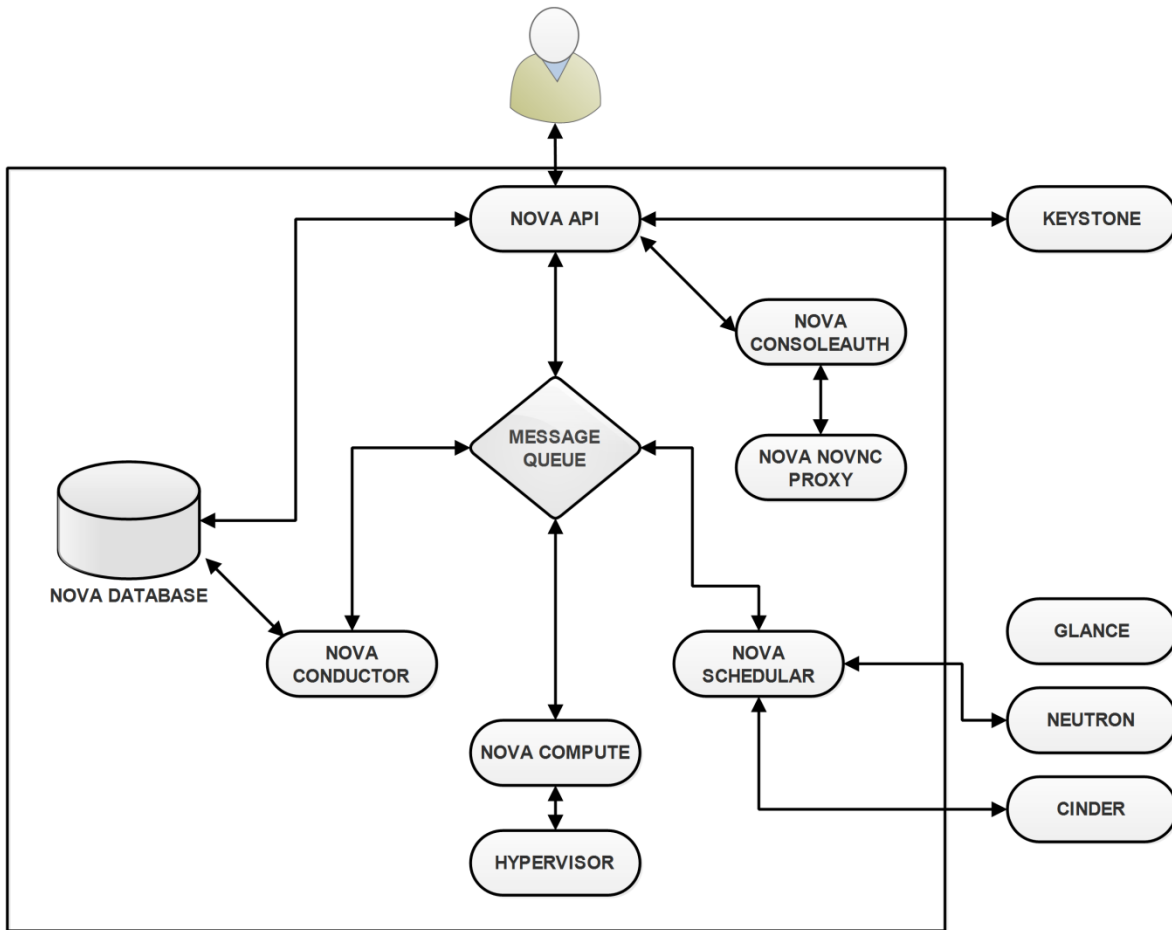


Figure 3.3: Nova architecture

3.4. Neutron

Neutron [88] delivers “networking as a service” amongst interface devices managed by other OpenStack services (e.g., nova). The central Neutron API comprises of support for IP address management (IPAM) and Layer 2 networking, as well as an addition for a Layer 3 router concept that allows routing between gateways and Layer 2 networks to external networks. Neutron

contains a list of plugins that allow interoperability with several open source technologies and commercial, including virtual switches, switches, SDN and routers.

3.4.1. Architecture

Neutron is a separate component in the OpenStack segmental architecture. It is placed alongside OpenStack services like Keystone, Nova, or Glance. Like those services, the Neutron deployment frequently involves positioning numerous services to a variety of hosts. Neutron server uses the neutron-server inspiration to depict the Networking API and allow administration of the configured Networking plug-in which needs access to a database for determined storage.

Neutron is constituted by the following components:

- **Neutron-server:** Accepts and routes API requests to the suitable Neutron plug-in for action.
- **Neutron plug-ins and agents:** Plugs and unplugs ports, provides IP addressing and creates subnets or networks. These agents and plug-ins vary depending on the technologies and vendor used in the specific cloud. Neutron ships with agents and plug-ins for Cisco physical and virtual switches, OpenvSwitch, NEC OpenFlow products, Ryu Network Operating System, Linux bridging, and the VMware NSX product. The common agents for a typical environment are plug-in agent and L3, DHCP.
- **Messaging queue:** Route information amongst the various agents and neutron-server, as well as a database to store networking state for specific plug-ins.

The Figure 3.14 shows neutron components in its architecture diagram:

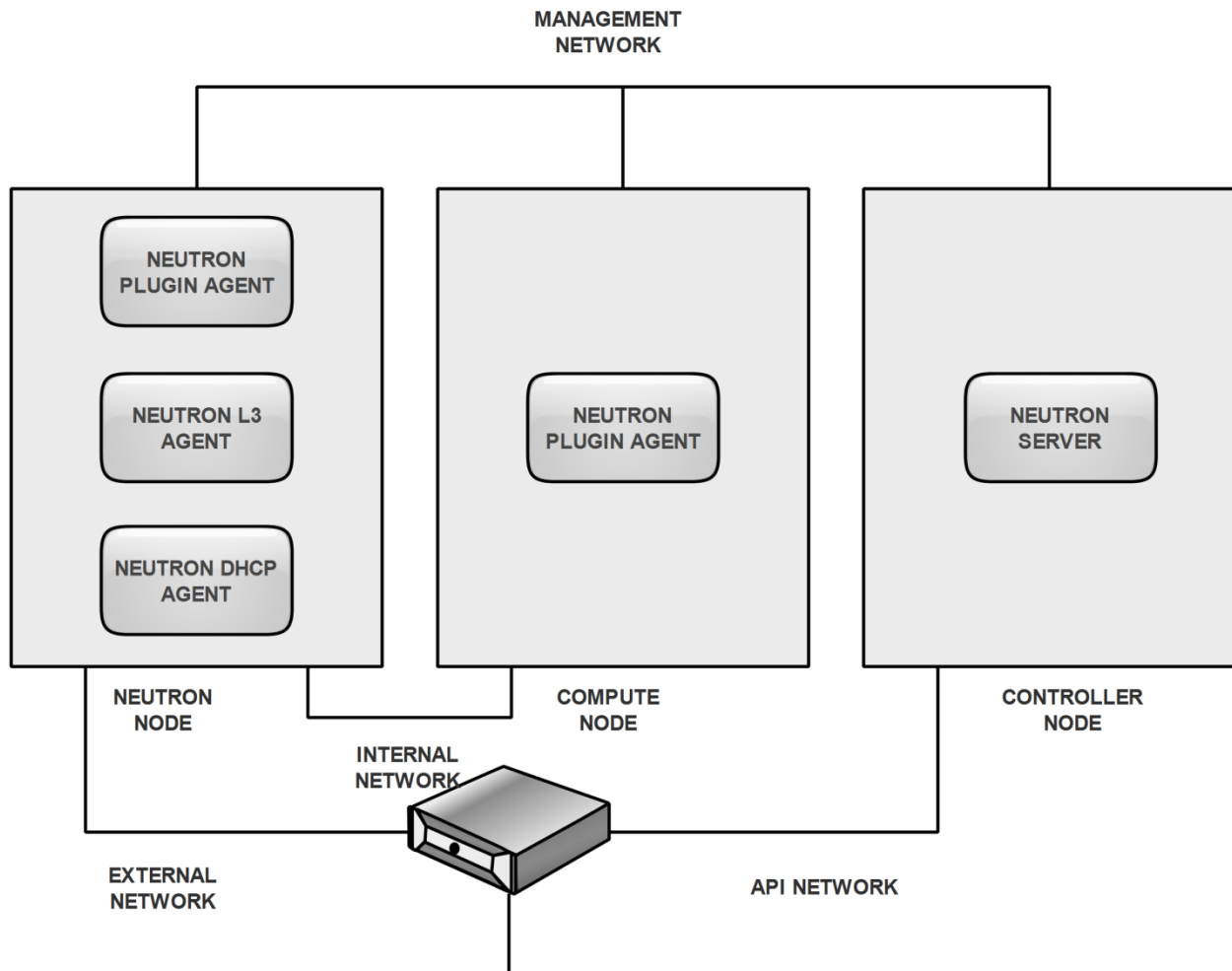


Figure 3.4: Neutron architecture

3.5. OpenStack Deployment

As illustrated in Figure 3.5, the OpenStack Mitaka [89] release has been deployed on the testbed. The deployment includes one control node and four compute nodes, all running Ubuntu 14.04 as an operating system. The OpenStack control node runs the Identity service (Keystone), Image service (Glance), management portions of Compute, management portions of Networking, various networking agents, and the dashboard (GUI). It also includes supporting services such as

a SQL database, message queue, and network time protocol (NTP). Each compute node runs the hypervisor portion of Compute that operates instances.

In our scenario, Compute uses QEMU hypervisor [90]. The compute node also runs a Networking service agent that connects instances to virtual networks and provides firewalling services to instances via security groups. The testbed also consists of a separate machine that has ONOS installed in it. Each node of OpenStack (control node and compute nodes) is connected to ONOS SDN controller via management network. Each compute node is connected to one another via a Data Tunnel Network, whereas to the Internet via External Network.

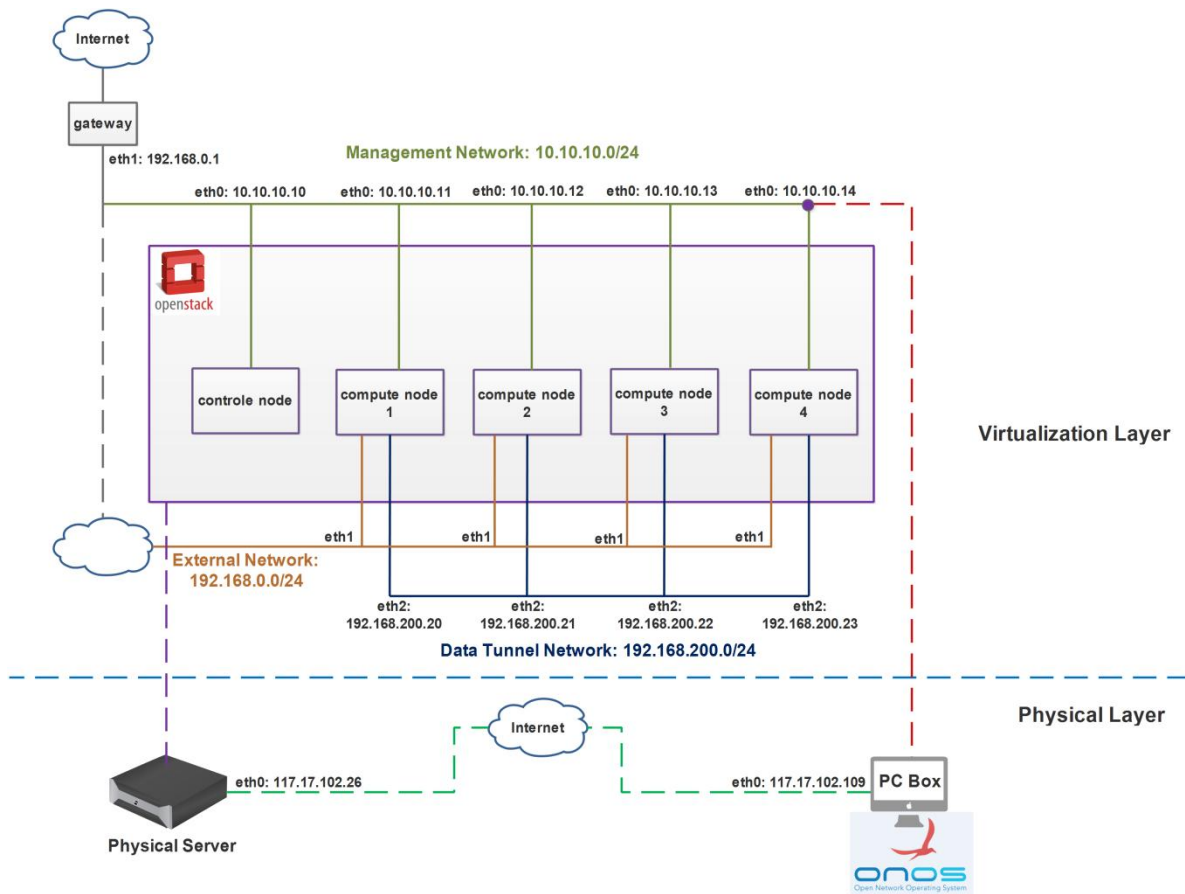


Figure 3.5: Cloud computing platform-based testbed

3.5.1. OpenStack-ONOS Integration

In OpenStack, the Module Layer2 (ML2) [91] plug-in is a framework allowing OpenStack networking to simultaneously utilize the variety of layer 2 networking technologies found in complex real-world datacenters. However, when integrating ONOS [92] SDN controller with OpenStack, the ONOS mechanism driver becomes responsible for layer 2 networking instead of the default OVS mechanism driver, as well as the default OpenStack's router plugin is replaced with ONOS L3 plug-in for layer 3 routing purpose.

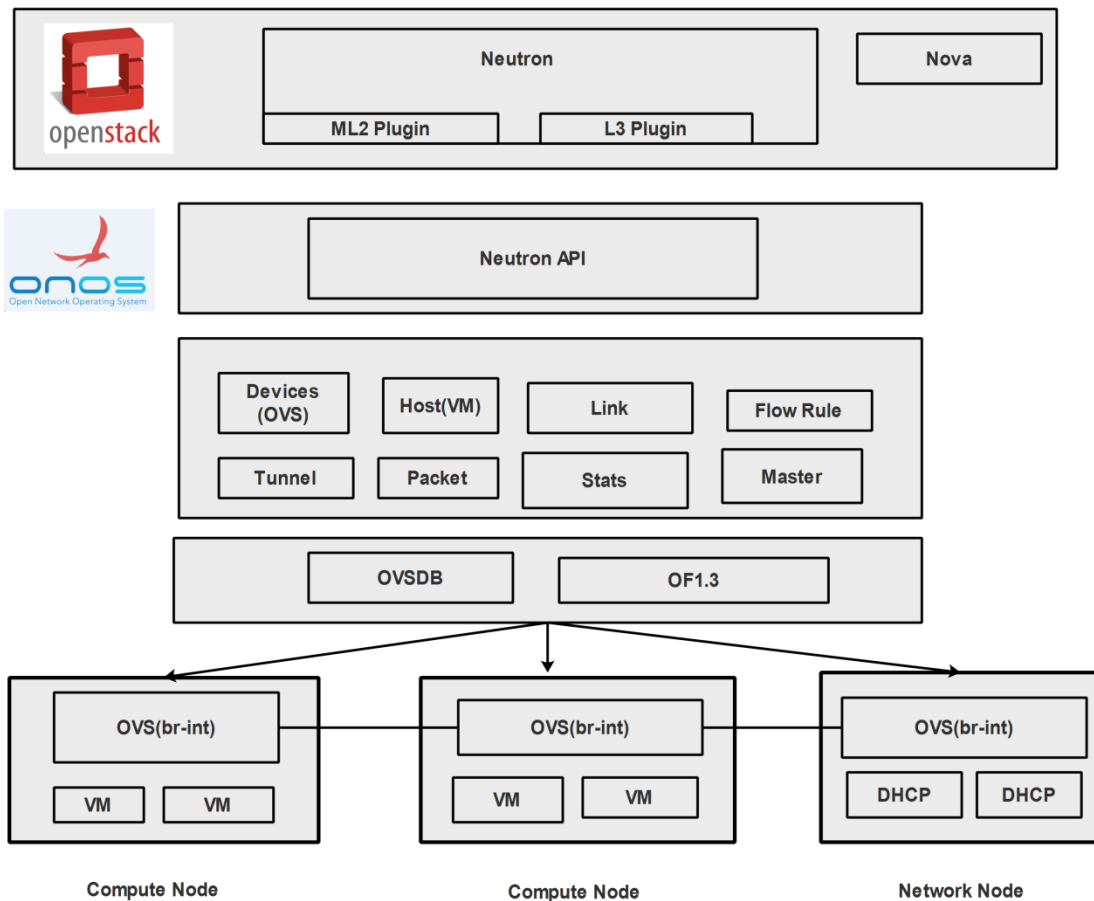


Figure 3.6: OpenStack-ONOS integration

OpenStack-ONOS integration, as shown in Figure 3.6, leads to the fact that all traffic between VM instances can be monitored, managed, and controlled by ONOS SDN controller. The northbound API of ONOS can be utilized to execute customized control applications for flow management and control. Furthermore, if visualized in ONOS GUI, the VM instances connected with OVSes of their respective compute nodes can give a better view of an overall network topology.

3.5.2. Visualization of OpenStack Deployment in ONOS GUI

As shown in Figure 3.7, there are four compute nodes in the OpenStack deployment. Since all the compute services are provided by Nova on compute node, whenever a VM instance is booted, it has to be assigned one of the existing hosts (compute nodes). In order to validate OpenStack-ONOS integration, we launched three VM instances on compute node 1, two on compute node 2, one on compute node 3, and two on compute node 4. If the integration has been done properly, the VM instances and compute nodes should appear in ONOS GUI the same as they have been launched in OpenStack. As depicted in Figure 3.7, each VM instance has been assigned an IP address, and connected with an OVS. The OVS is in fact representing a compute node on which the VM instance has been launched. The connection of one compute node with another shows the existence of a proper data tunnel network between compute nodes.

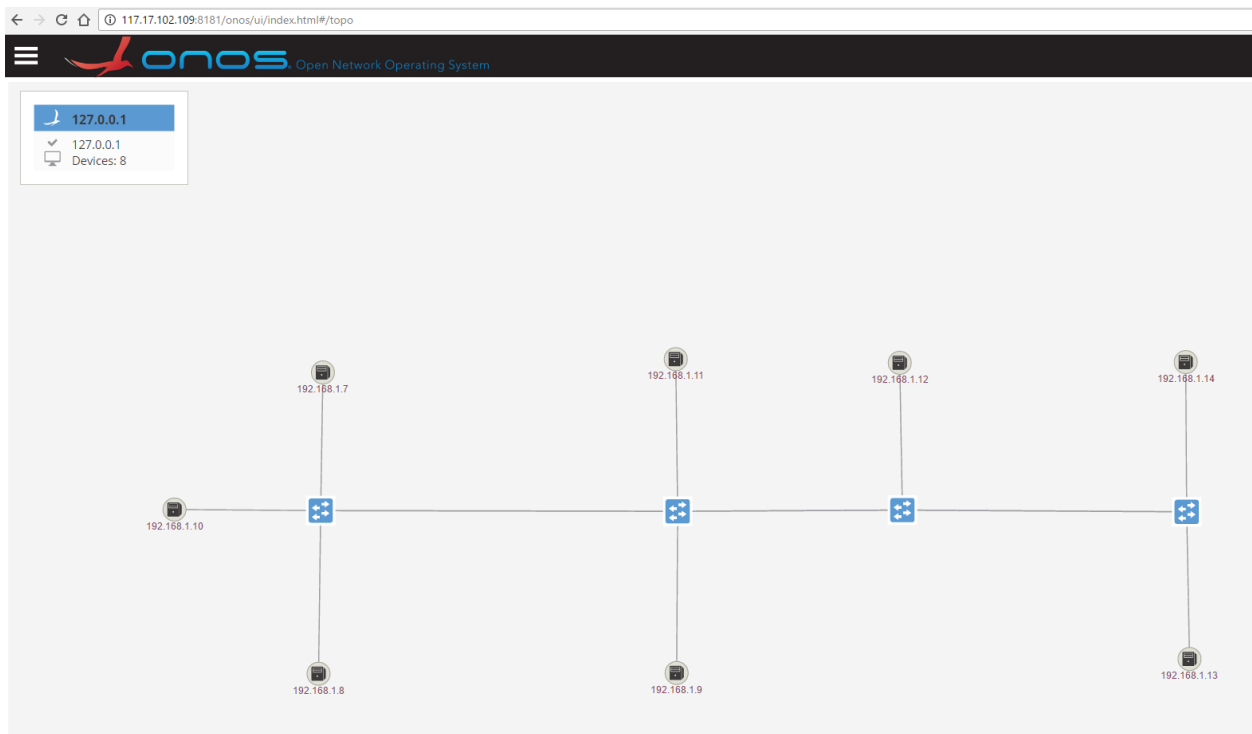


Figure 3.7: Visualization of OpenStack deployment in ONOS GUI

Chapter 4

Resource Orchestration Framework and its Major Components

This chapter presents a novel service-oriented resource orchestration framework. The aim of the orchestration framework is to provide main components in order to manage network and cloud resources required by cloud services to satisfy requests generated by remote end-users.

4.1. Proposed Architecture

The overall architecture of proposed orchestration framework is illustrated in Figure 4.1. It includes two fundamental building blocks: (i) Service Abstraction Model, and (ii) The Orchestrator. We used Service Abstraction Model for the orchestrator, with which the

orchestrator understands the requirements of the requested services and provides the resources for the services. The orchestrator is responsible for resource allocation based on the requests generated by remote end-users. The Network Resource Communicator (NRC) and Cloud Resource Communicator (CRC) modules of orchestrator are meant to manage network and cloud resources respectively. Infra Monitor module collects cloud and network infrastructure-related information stored in an internal database server for use by the Resource Allocator.

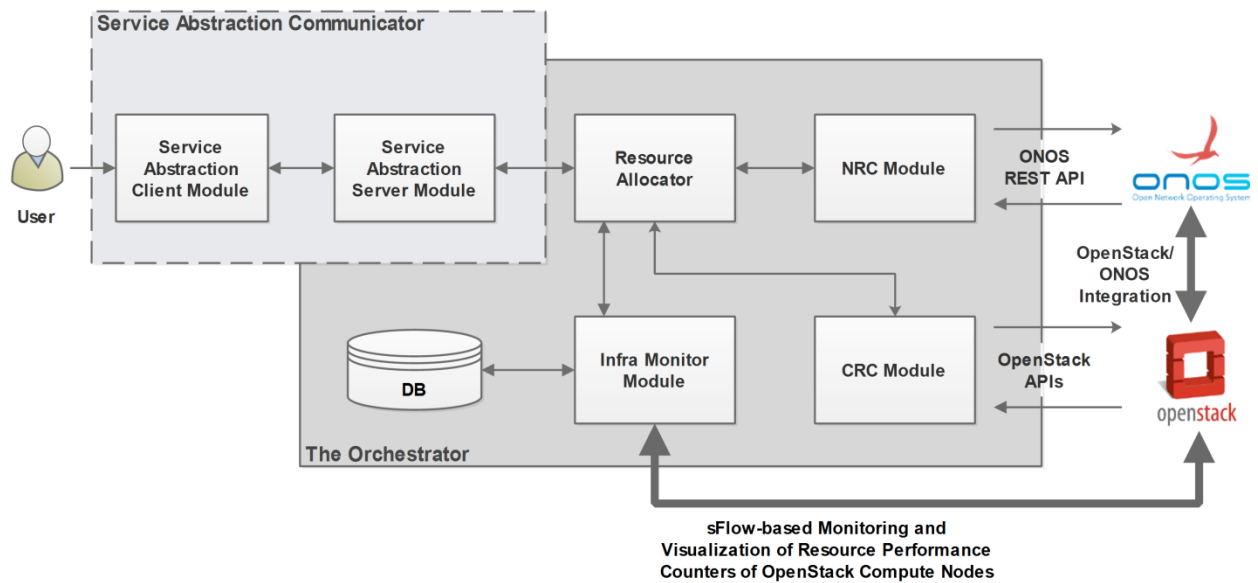


Figure 4.1: Proposed orchestration framework

4.2. Service Abstraction Model

With our existing service abstraction model [93], the orchestrator gets the abstracted requirements of the requested services, and maintains/provisions networks performance to meet the requested service requirements. The orchestrator deals with the dynamic nature of the services by easily changing the networks accordingly. For instance, the abstraction model describes any change occurred in the number of viewers of the video service, whereas the orchestrator manages the required bandwidth for this change. Our service abstraction model

abstracts the requested service parameters to support future and legacy network services. The overall concept of the service abstraction model is depicted in Figure 2.

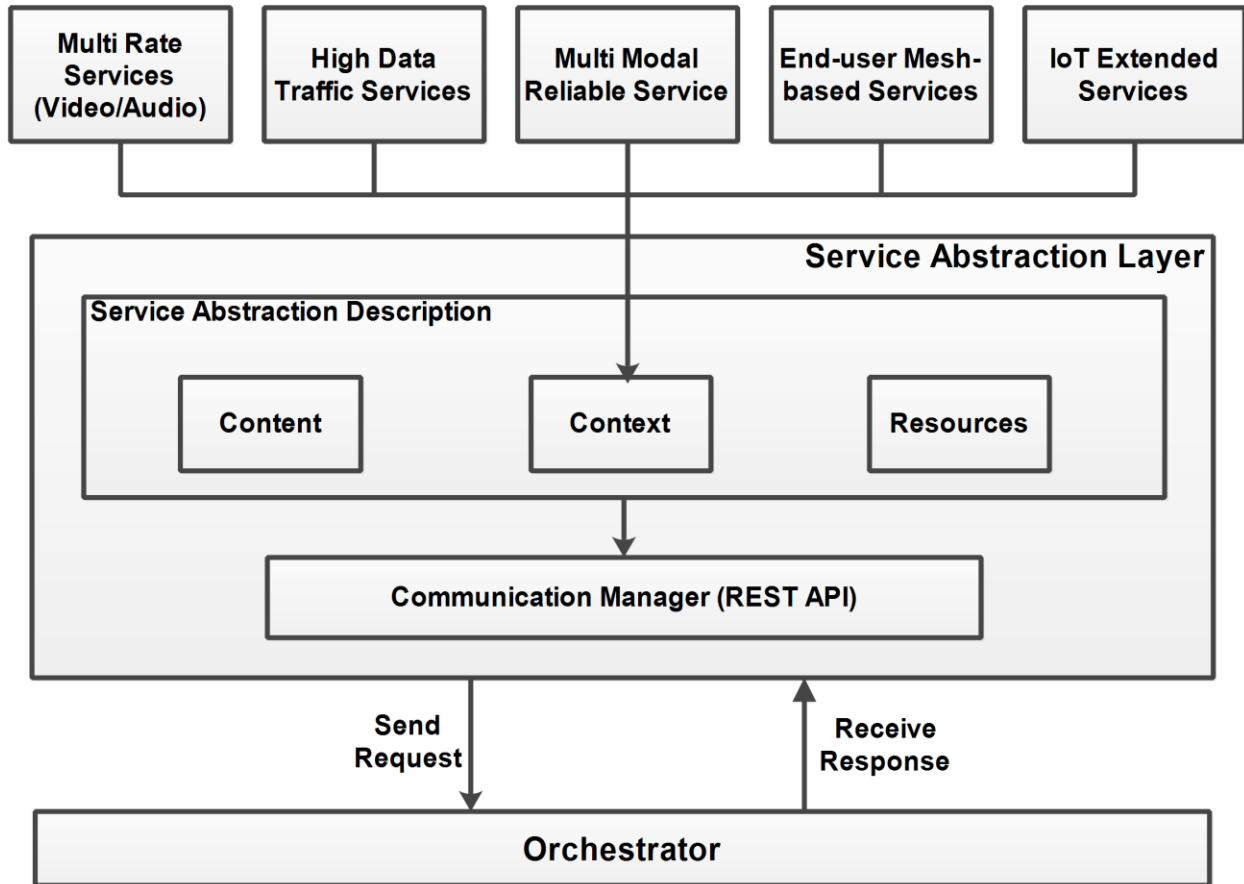


Figure 4.2: The service abstraction model

The service requirements are described in the service abstraction description of service abstraction layer. The requirements are classified into three categories, i.e., Resources, Context, and Content. The service-related parameters are provided by Content. They may be standard and resolution of a video, QoS, or audio bit-rate. The user-related parameters are covered by Context. They may be schedule and location of the service, or interest of the user. The requirements of the infrastructural resources are provided by Resource. All these three sets of parameters are converted into XML format, and are parsed at service abstraction description module. After that,

the communication manager module transfers these requirements of a service to the orchestrator. Finally, they are processed by the orchestrator, and an appropriate network is generated for the user to access the requested service.

The service parameters are received by the service abstraction layer either from the applications or inputted by the users as shown in Figure 4.2. Based on these parameters, the service is categorized by the service abstraction layer, which then generates a suitable service abstraction model in XML format and delivers it to the orchestrator.

4.2.1. Service Abstraction Communicator

A JSON-RPC server-client module is implemented to deliver abstracted service requirements to the orchestrator. The overall structure of this module consists of a function/method that is defined in the request object, and the abstracted service parameters. As depicted in Figure 4.3, the service abstraction client and server modules communicate with each other by means of a TCP channel. The abstracted service parameters are sent as a request from the service abstraction client module to the service abstraction server module where they are parsed and delivered to the orchestrator. The server also sends a message in response to the client's request message if required.

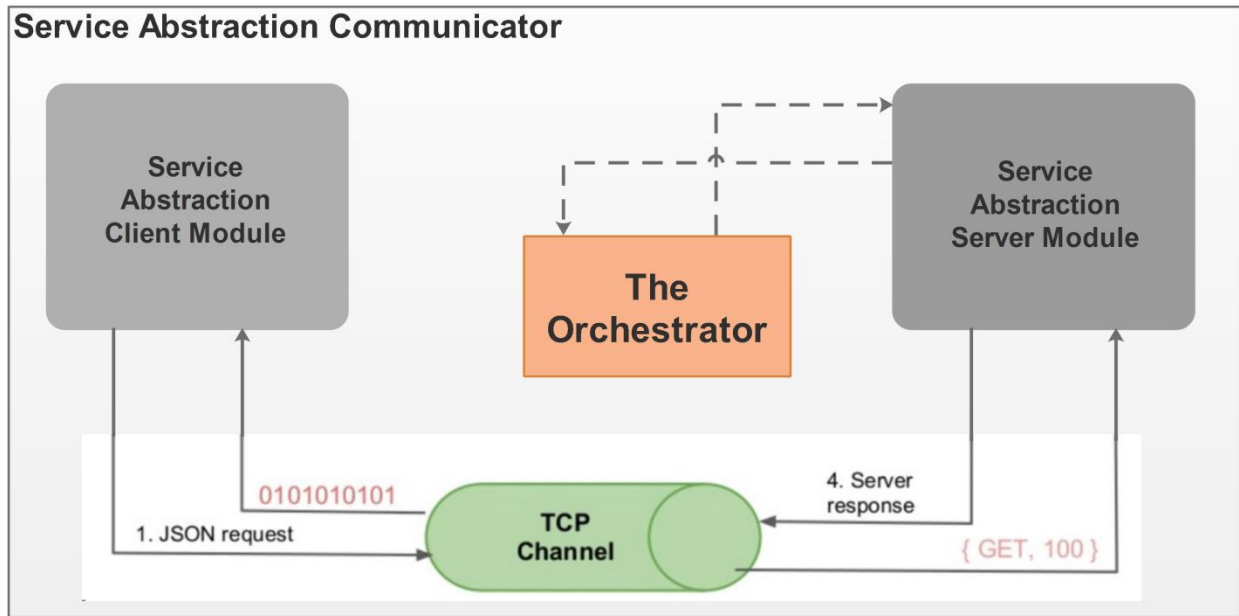


Figure 4.3: Service Abstraction Communicator Module

4.3. The Orchestrator

It is the main building block of our proposed orchestration framework. Its core module, Resource Allocator allocates resources in the form of VMs in order to meet the requirements of the requested applications/services, which are requested by remote end-users. Once a request is received, it is necessary to determine what VMs are the most appropriate to be allocated. This issue is solved by algorithms for VM selection. One such example can be selecting a VM randomly from a pool of VMs assigned to the host. Alternatively, based on our proposed algorithm, Algorithm 1, called maximum RAM minimum CPU utilization (MRMC) algorithm, VMs with the maximum amount of RAM are first selected, and then out of these selected VMs, the VM with the minimum CPU utilization averaged over the last n quantifications is selected. The algorithm takes n number of previous CPU utilizations, RAM and CPU utilization statistics of VMs, and a pre-defined CPU utilization threshold. It starts with first sorting the VMs in terms

of their RAM utilization in an ascending order. Then, from the sorted list of VMs, the mean value of the last n number of CPU utilizations of each VM is determined. If the calculated mean value of CPU utilization of any particular VM in the sorted VM list is less than or equal to the pre-defined CPU utilization threshold value, it is selected and returned as an output of the algorithm for resource allocation. The Big-O time complexity for Algorithm 1 is given as:

$$T(n) = O(m^2 + mn)$$

where m is `vm_list`.

Algorithm 1: Maximum RAM Minimum CPU utilization (MRMC) Algorithm

```

1 Input: n, vms_utilization_stats, cpu_threshold
2 Output: a VM to select
3 selected_vm ← None
4 minTempVal ← 0
5 sum ← 0
6 mean ← 0
7 for i = 0 to length(vm_list) do
8   minTempVal ← i
9   for j = i+1 to length(vm_list) do
10    |   if (vm_ram_map[j] < vm_ram_map[minTempVal]) then
11    |   |   minTempVal ← j
12    |   endif
13    done (endfor)
14    swap vm[i] with vm[minTempVal]
15 done (endfor)
16 for i = 0 to length(vm_list) do
17   sum, mean ← 0
18   for j = 0 to n do
19   |   sum ← sum + vm_cpu_map[j]
20   done (endfor)
21   mean ← sum/n
22   if mean ≤ cpu_threshold then
23   |   selected_vm ← vm_list[i]
24   |   break
25   endif
26 done (endfor)
27 return selected_vm

```

4.3.1. Network Resource Communicator

The SDN controller provides an open interface on the controller to allow for automated control of the network. The terms *northbound* and *southbound* are often used to differentiate whether the interface is to the applications or to the devices. The southbound API is the OpenFlow interface that the controller uses to program the network devices. The controller offers a northbound API, allowing software applications to be plugged into the controller and thereby allowing that software to provide the algorithms and protocols that can run the network efficiently. The northbound API of the controller is intended to provide an abstraction of the network devices and topology. That is, the northbound API provides a generalized interface that allows the software above it to operate without knowledge of the individual characteristics of the network devices themselves.

One of the results of this level of abstraction is that it provides the ability to virtualize the network, decoupling the network service from the underlying physical network. Those services are still presented to host devices in such a way that those hosts are unaware that the network resources they are using are virtual and not the physical ones for which they were originally designed.

The NRC module is responsible for managing the network infrastructure resources. As shown in Figure 4.4, the NRC module interacts with the SDN controller in order to get the details of available resource, as well as to POST the changes/rules to manage the network. We use ONOS as the SDN controller. The NRC module uses the REST API of ONOS controller in order to get the details of flows, devices etc. It can GET the FLOWS, INTENTS, DEVICES and LINKS. It can also POST intent as per the application requirements. The south bound API of ONOS provides an abstraction of the actual physical network infrastructure.

There is a method/function defined in the NRC module, which retrieves the information related to the network devices and links from the ONOS SDN controller and aggregates it in a variable. This information is shared with the orchestrator for monitoring and management purpose. The NRC communicator module also allows for installing host-to-host intents on ONOS SDN controller by means of HTTP POST method.

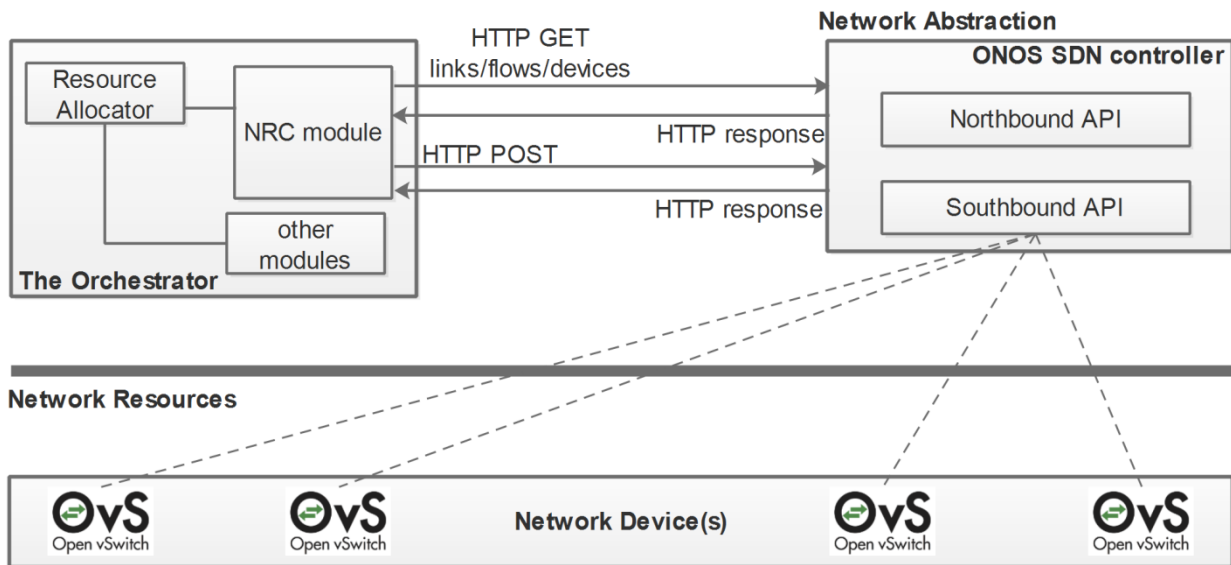


Figure 4.4: Network resource communicator module

4.3.2. Cloud Resource Communicator

The CRC module is meant for interacting with the OpenStack and managing the compute/network resources inside OpenStack according to the user requirements. As shown in the Figure 4.5, OpenStack provides different APIs that can be used to control/manage/allocate resources on the physical infrastructure. The CRC module uses the identity API to authenticate with the OpenStack cloud in the very first step. Then it uses the neutron API to create/delete/modify tenant networks inside OpenStack, and compute API to manage compute

resources. By using the compute API, CRC module can allocate new resources, modify current resources or delete any resource as per requirements of the requested application.

There is a method/function defined in the CRC module, which retrieves the information related to the already launched VM instances in OpenStack and aggregates it in a variable. Another function for creating networks uses HTTP POST request to create tenant networks in OpenStack. It first uses the identity information and creates a session with OpenStack and then creates a network with the name as specified in the parameters. There are also several other functions defined in CRC module that allow the orchestrator to delete the VM instances and networks in OpenStack clouds.

Infra Monitor module collects and stores cloud and network infrastructure-related information, and shares it with the Resource Allocator. A monitoring component is also deployed on each compute host of OpenStack and is meant for collecting data related to the resource utilization by hypervisors and VM instances, and then sending the data to the Infra Monitor module. Based on this information, the Resource Allocator module assigns resources by means of our proposed algorithm. The details of which are given in the next section.

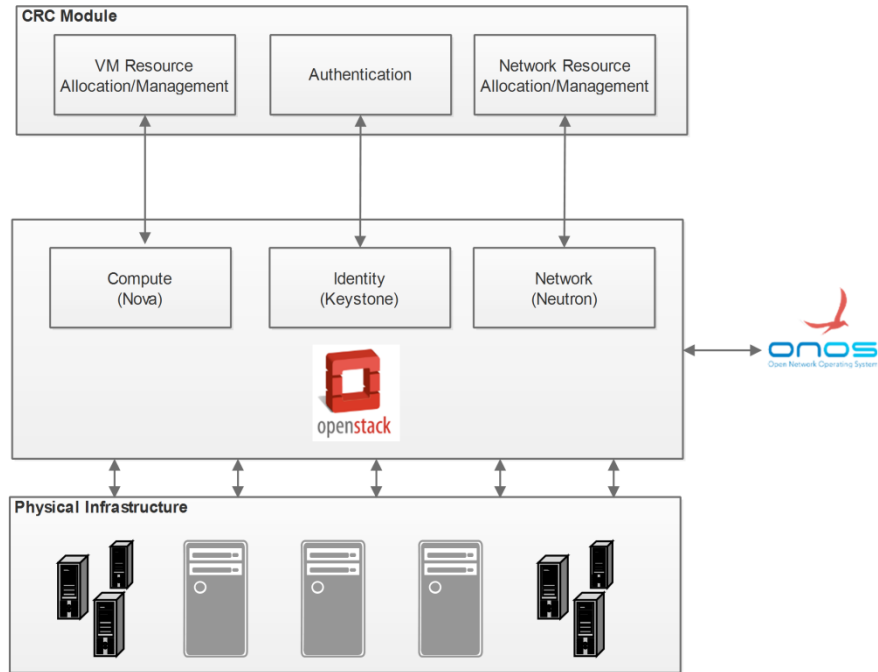


Figure 4.5: Cloud resource communicator module

The overall working of the orchestrator involves several steps. It starts from a user requesting for a service/application, which is forwarded to the web server. The web server forwards this request to service abstraction client module, which abstracts the requirements such as service type and quality, and forwards it to the service abstraction server module. It parses the request message and forwards it to the orchestrator, which then forwards set of links and bandwidth for embedding virtual topology to NRC module, and Node Information (Location etc.), Service Type, Server Capacity to CRC module. The communication of CRC and NRC modules with OpenStack and ONOS SDN controller respectively enables the orchestrator to handle the network requirements by means of SDN Controller and service requirements via Cloud Manager.

4.4. sFlow-based Monitoring System

In order to maintain a record of the resources utilized by compute nodes, a monitoring system based on sFlow technology [94] has been used in this thesis. In this monitoring system, sFlow sample from the OVSEs of compute nodes are collected and stored in a database. The Host sFlow agent, deployed on each compute node, is responsible for providing samples of performance counters related to infrastructural resources. It exports physical and virtual server performance metrics using the sFlow protocol. The agent provides scalable, multi-vendor, multi-OS performance monitoring with minimal impact on the systems being monitored. Figure 4.6 shows the sFlow host structure that allows pairing host metrics with corresponding network metrics with the help of mapping of MAC addresses [95].

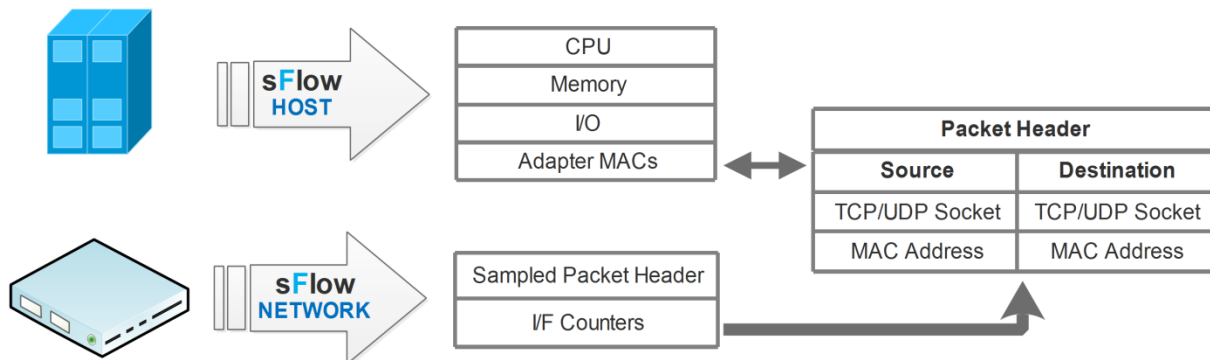


Figure 4.6: sFlow host structure

The essential components of sFlow monitoring system are depicted in Figure 4.7. The Graphite [96] real-time charting software provides a flexible way to store and plot time series data. Graphite is an open-source project that is growing in popularity among network engineers. However, while Graphite is very good at storing and plotting metrics, it offers no default monitoring capabilities. Instead, it relies on agents provided by user to make

measurements. The open source Host sFlow agent [97] is able to work in parallel with Graphite, providing a portable, lightweight agent that exports standard metrics from a wide range of systems. Host sFlow agents together with a Graphite collector offer a complete, highly scalable monitoring solution. The Host sFlow agents continuously send metrics to the Graphite collector in the form of binary sFlow data, which is converted into Graphite's text-based data and submitted to the Graphite server.

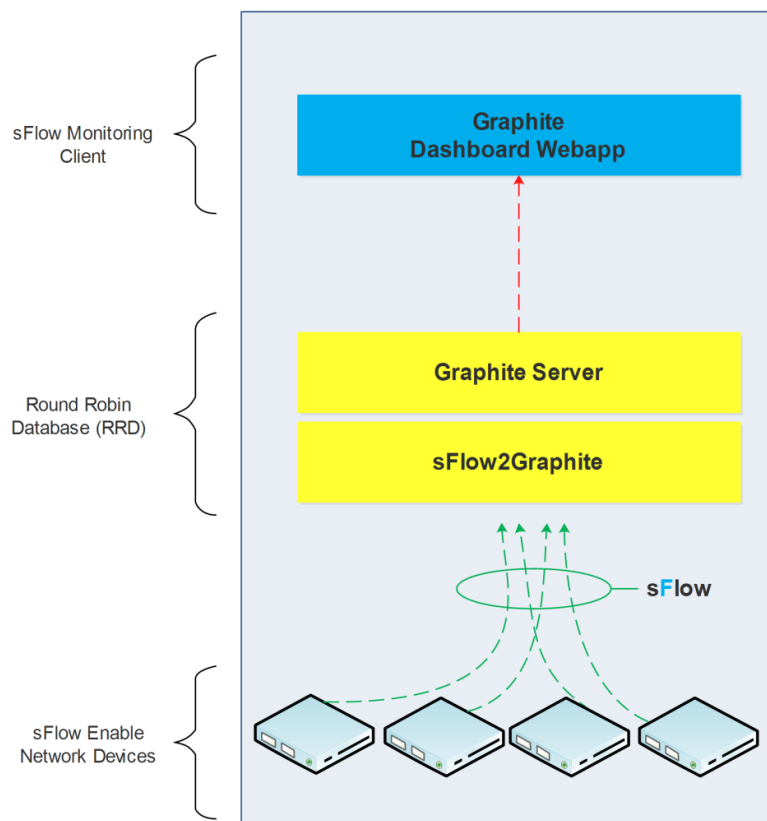


Figure 4.7: Architecture of sFlow monitoring system

4.5. Implementation of Proposed Resource Orchestration Framework

The proposed architecture has been validated in the cloud computing platform-based testbed discussed in Section 3.5. The use case that has been considered is a creation of a VM instances in OpenStack with the proposed resource orchestration framework.

As mentioned above, the CRC module of the Orchestrator manages compute/network resources in OpenStack according to the user requirements. A function/method has been defined in this module which receives several options as input variables and uses HTTP POST to create a VM/instance in OpenStack. Figure 4.8 shows the conversation between the service abstraction server module, the orchestrator, OpenStack, ONOS SDN controller, and the OpenFlow switch of the corresponding compute node on which the VM/instance is created. The orchestrator receives a parsed message from the service abstraction server module, and requests the available resources from OpenStack through HTTP GET message. After getting a reply from OpenStack, the orchestrator instructs OpenStack to create a VM/instance by means of HTTP POST message. To give network connectivity to this VM, OpenStack communicates with ONOS SDN controller, which instructs the OVS of the corresponding compute node that hosts the newly created VM.

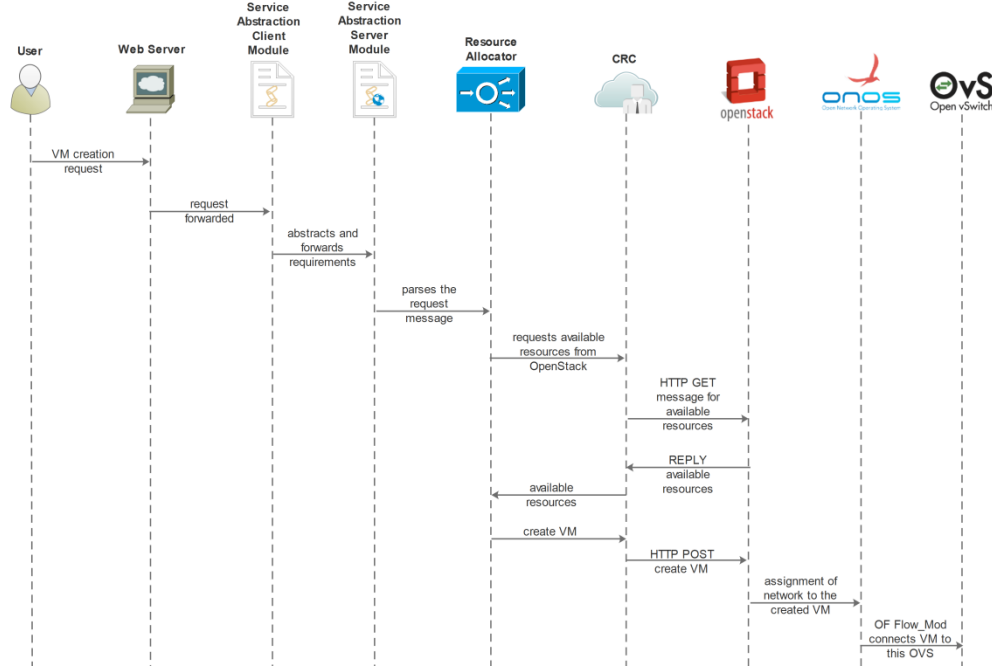


Figure 4.8: Exchange of messages between different modules for VM creation

4.5.1. Performance Analysis of Proposed MRMC Algorithm

When a request is received, it is necessary to allocate the appropriate VMs in terms of available resources. These VMs can be either selected randomly or by means of any efficient scheme. However, random VM selection may lead to performance degradation because the selected VMs may have minimum RAM and maximum CPU utilization. We address this issue by proposing MRMC algorithm, which selects VMs on the basis of maximum RAM and minimum CPU utilization. It accepts historical data on the resource usage by VMs running on the compute nodes and returns a set of VMs to be allocated.

The performance of our proposed MRMC algorithm is compared with two algorithms, i.e., the random VM selection scheme and a resource allocation algorithm given in [108]. The comparison is first performed for VM selection on the basis of the amount of utilized RAM from the pool of VMs. All the algorithms are run ten times with the time difference of five seconds

between each run. It can be clearly seen in Figure 4.10 that MRMC algorithm outperforms the random VM selection scheme. For each run, the MRMC algorithm selects VM(s) with minimum RAM utilization, whereas the random VM selection scheme selects VM(s) without taking into consideration the amount of RAM available on each VM. The proposed MRMC also outperforms the MCDA algorithm because the MCDA algorithm makes decision on the basis of available resources. It simply assigns a compute node which has the available resources to meet the requirements of the requested service. On the other hand, our proposed MRMC algorithm has two-dimensional selection criteria. It first selects those VMs from a pool of VMs which have the maximum amount un-utilized RAM for allocation. It is obvious from Fig. 4.10 that for the first run, the proposed MRMC algorithm selects a VM which has a utilized RAM of around 500 MB, whereas the MCDA algorithm chooses a VM which has the RAM utilization of around 1.1 GB.

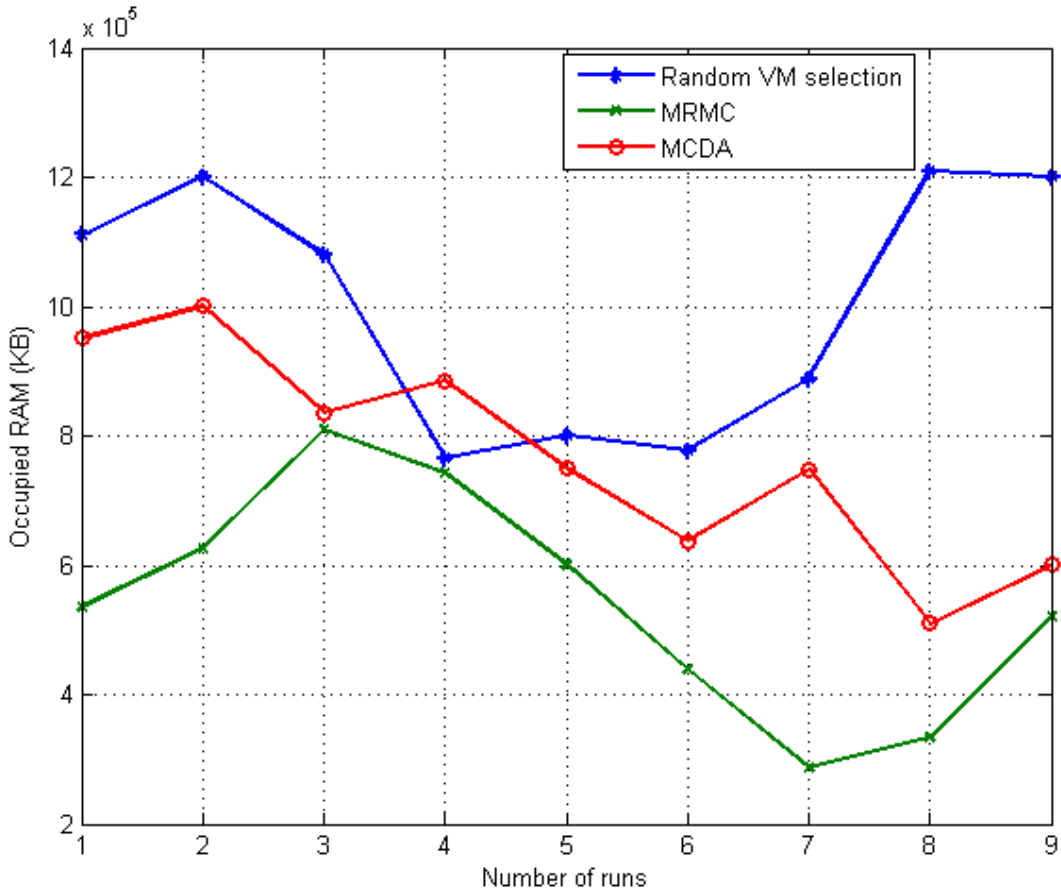


Figure 4.9: Performance analysis of MRMC for minimum occupied RAM

In the second step, the proposed MRMC algorithm allocates the resource (VM) from a set of previously selected VM(s) having the maximum amount of un-utilized RAM by determining the VM with the minimum CPU utilization averaged over the last n quantifications. On the other hand, the random VM selection and MCDA algorithms allocate resources regardless of the CPU utilization for each run. The performance of the proposed MRMC algorithm is compared with random VM selection and MCDA algorithms on the basis of the minimum CPU utilization. All the algorithms are run ten times with the time difference of five seconds between each run. It is obvious from Figure 4.11 that MRMC algorithm outperforms the random VM selection scheme and MCDA algorithms. The resource (VM) with the minimum CPU utilization is allocated by

means of MRMC algorithm, whereas the random VM selection and MCDA algorithm do not consider CPU utilization when allocating resources. For instance, MRMC algorithm selects a VM with a CPU utilization of around 28%, whereas the MCDA algorithm chooses a VM which has the CPU utilization of more than 50% for the same run.

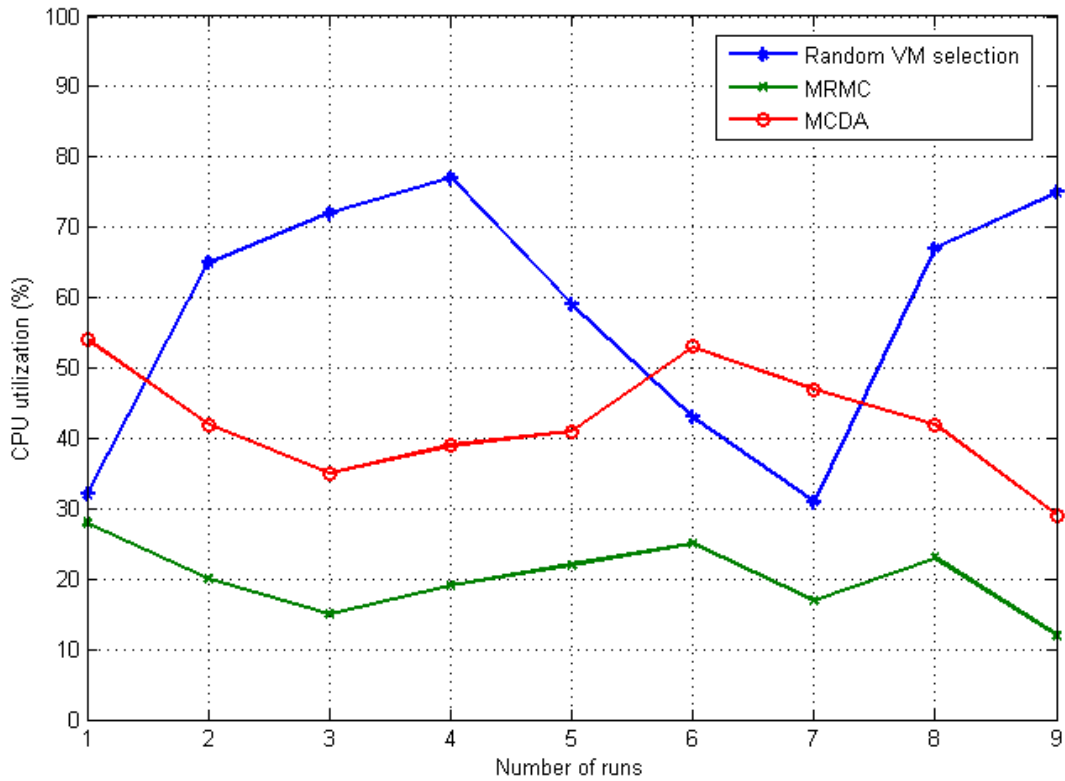


Figure 4.10: Performance Analysis of MRMC for minimum CPU utilization

Chapter 5

A Framework for Resource Utilization-based Migration of VMs in OpenStack Clouds

In this chapter, we introduce an architecture and implementation of a framework for dynamic VM migration in cloud data centers based on the OpenStack platform. The deployment of the framework includes a controller node and multiple instances of compute nodes. The purpose of the framework is to provide dynamic live migration to adjust the VM instances on compute nodes to offload a number of VMs from an overloaded compute node. We address this issue by focusing on the following key points:

- Detecting an overloaded compute node, so that some VM instances may be migrated to other efficient compute nodes.
- Based on CPU and RAM utilization, selecting the overloaded VM instance(s) from a compute node.
- Locating the selected VM instance(s) for migration on other efficient compute nodes.

The overall process of VM migration is illustrated in Fig. 5.1. At the beginning of each iteration, the migration manager module reads from the stats DB, the historical data on the resource usage by the VMs. Then, the migration manager module invokes the overload detection algorithm to determine whether the compute node is overloaded. If the compute node is not overloaded the migration manager reads the resource utilization data from the stats DB at the next iteration.

If the compute node is overloaded, the migration manager invokes the VM selection algorithm to select VMs to offload from a compute node. Once the VMs to migrate from the compute node are selected, the migration manager invokes the VM placement algorithm to determine the most efficient compute node to place the migrated VM. Finally, upon the selection of the most efficient compute node, the migration manager sends a request to the OpenStack Nova API to migrate the selected VMs.

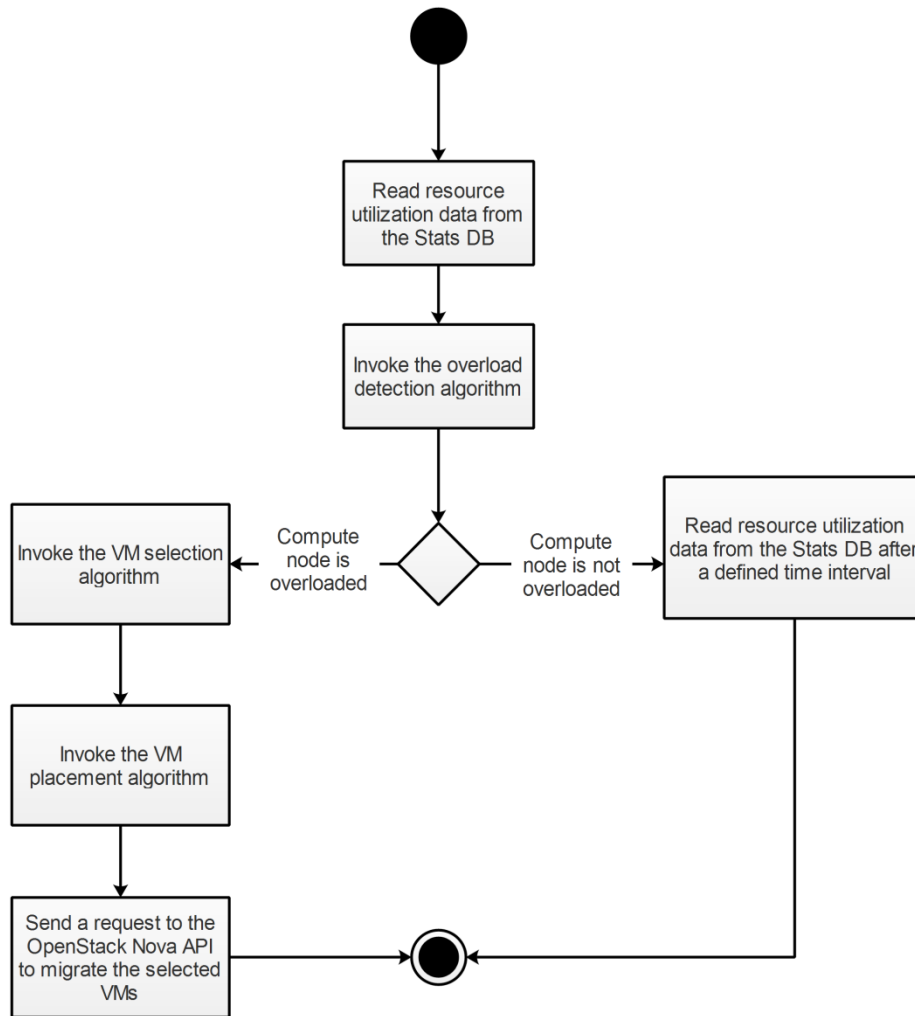


Figure 5.1: The proposed VM migration process

5.1. System Model

The framework implements essential components for monitoring hypervisors and VMs, gathering resource utilization statistics, sending messages and instructions between the system elements, and conducting VM live migrations. It enables the implementation of the three proposed algorithms for dynamic VM migration: detecting overloaded compute node, selecting a VM based resource utilization, and allocating a VM to an efficient compute node.

Fig. 5.2 depicts the overall architecture of the proposed framework that is deployed in OpenStack to avoid an overloaded compute node by conducting dynamic VM migration at

appropriate time. It includes four fundamental building blocks: (i) Stats Aggregator, (ii) Stats Database, (iii) Migration Manager, and (iv) Algorithms Repository. In the following, we will discuss each of the framework’s components in detail.

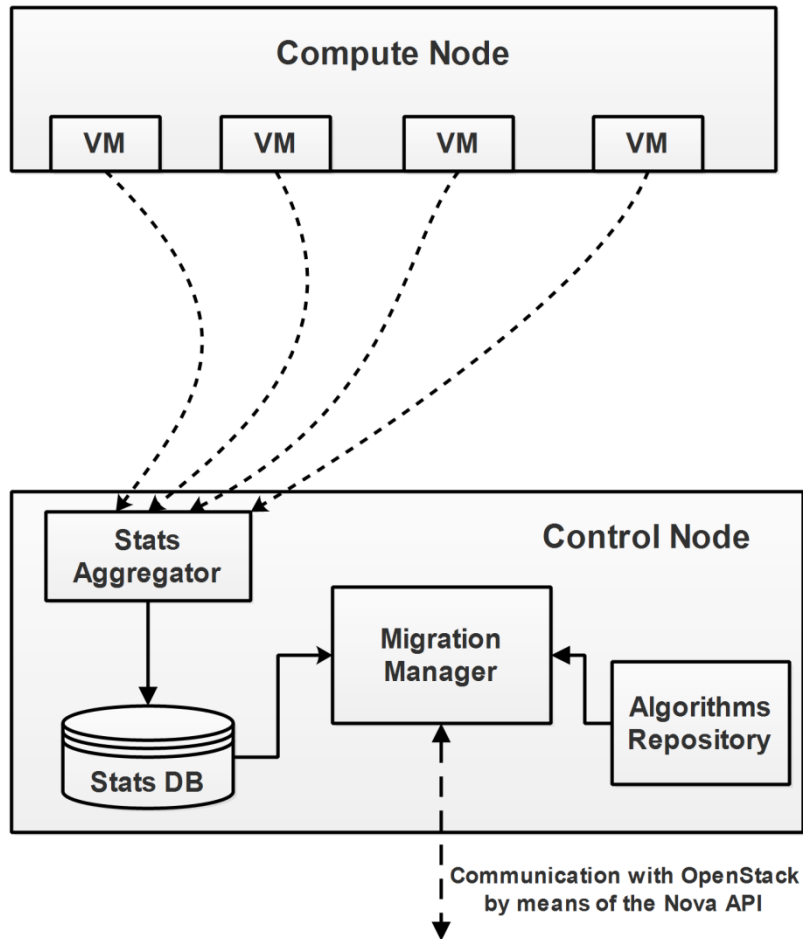


Figure 5.2: Proposed system model

5.1.1. Stats Aggregator

A component that is deployed on the control node and is responsible for collecting statistics on the resource utilization by VM instances and hypervisors and then forwarding it to the statistics database, which can also be shared with other components. The statistics are collected by means of libvirt’s API [98] in the form of the RAM consumed by VM instances and hosts. For the

collection of statistics related to CPU utilization of VM instances, we have installed and configured the Ceilometer [99] project of OpenStack on control node. The RAM and CPU statistics are both collected periodically and submitted to the Stats Database module of the framework.

5.1.2. Stats Database

The stats database is used for storing historical statistics on the resource utilization by VM instances and hypervisors. The database is populated by the stats aggregator deployed on the same control node. The RAM and CPU utilization statistics of VM instances are periodically submitted to this module, which are then used by the migration manager to determine the VM instances that are consuming most of their respective compute nodes' resources.

5.1.3. Migration Manager

The migration manager is responsible for conducting VM migrations and making VM allocation decisions, which results in offloading VMs from an overloaded compute node. It runs the overload detection algorithm when resource utilization statistics are received from the Stats DB module. If an overload condition is detected, it runs the VM selection algorithm to select the VM instances, which are utilizing maximum RAM and CPU resources. Then it determines the efficient compute nodes in order to place selected VMs on them, and invokes OpenStack API for live migration of the selected VM instances.

5.1.4. Algorithm Repository

This repository is deployed to store custom decision-making algorithms for dynamic VM migration, i.e., compute node overload detection, VM selection, and VM allocation algorithms. Based on these algorithms, the migration manager module determines the overloaded compute

node, selects the VM instances that are to be migrated, as well as initiates VM migrations and makes VM placement decisions.

5.2. Proposed Structure and Algorithms

The migration process should be initiated offload the compute nodes based on a predefined threshold. A compute node is offloaded by VM migrations, which can make the load below the predefined threshold. Figure 5.2 depicts the exchange of messages for handling a compute node overload situation. First, the migration manager detects an overload of the compute node using the overload detection algorithm. Then, by means of proposed VM selection algorithm, the migration manager selects VM instances based on their CPU utilizations. Next, the migration manager initiates the VM allocation algorithm with the list of selected VMs along with their utilized resources and states of the compute nodes obtained from the stats database as arguments. Finally, based on the VM allocation generated by the algorithm, the migration manager requests the OpenStack Nova API for the appropriate VM live migration.

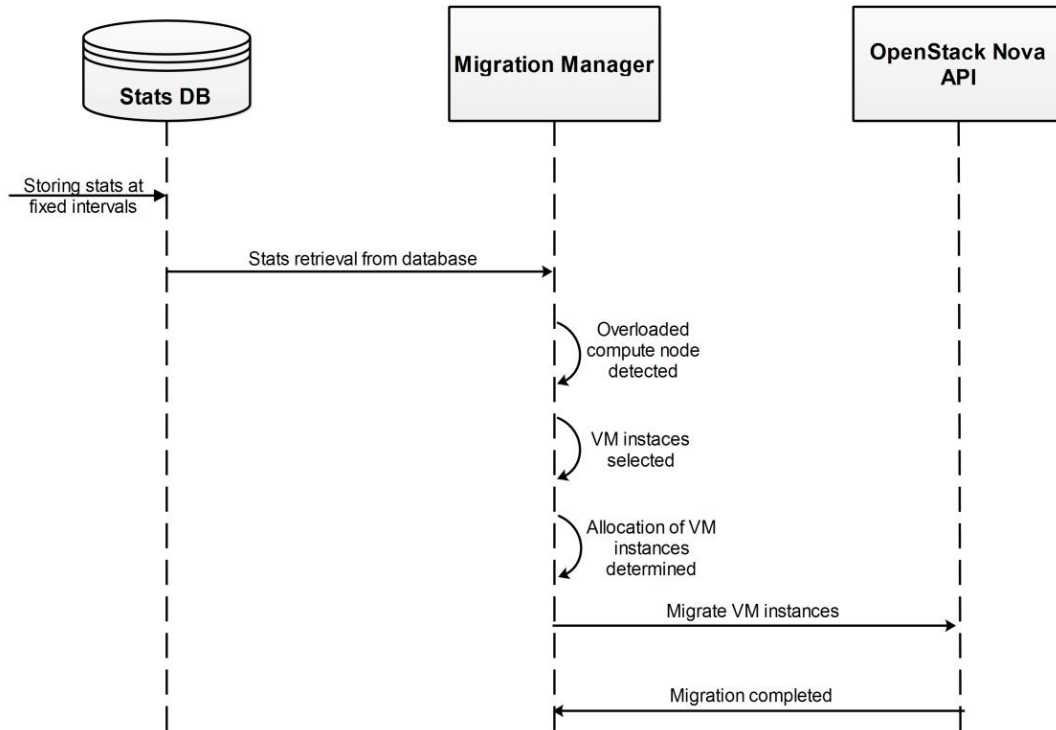


Figure 5.3: Exchange of messages between components for VM instances migration

5.2.1. Overload Detection Algorithm

The overload detection algorithm shown in Algorithm 2 is a simple algorithm which detects an overloaded compute node if the average of the last n CPU utilization measurements is greater than the pre-defined threshold value. The upper threshold value has been defined by following a traditional approach, i.e., the static threshold algorithm (STA). The STA defines an upper threshold for the hosts beforehand and the provisioning of schemes have to keep the total utilization of the CPU under the threshold limit.

The algorithm takes n number of previous CPU utilizations, RAM and CPU utilization statistics of a compute node, and pre-defined CPU utilization and RAM utilization threshold values. It starts with initializing the defined variables. Then, from the list of compute nodes, the mean values of the last n number of CPU utilizations and the last n number of RAM utilizations

for each compute node is determined. If the calculated mean value of CPU utilization or RAM utilization of any particular compute node is greater than or equal to the pre-defined CPU utilization threshold value, it is selected and returned by the algorithm as an overloaded compute node.

Algorithm 2: Compute Node Overload Detection Algorithm

```

1 Input: n, cn_utilization_stats, cpu_threshold, ram_threshold
2 Output: overloaded_cn_list
3 sum_ram, sum_cpu, mean_ram, mean_cpu ← 0
4 for i = 0 to length(cn_list) do
5     sum_ram, sum_cpu, mean_ram, mean_cpu ← 0
6     for j = 0 to n do
7         sum_cpu ← sum + cn_cpu_map[j]
8         ram_cpu ← sum + cn_ram_map[j]
9     done (endfor)
10    mean_cpu ← sum_cpu/n
11    mean_ram ← sum_ram/n
12    if (mean_cpu ≥ cpu_threshold) || (mean_ram ≥ ram_threshold) then
13        selected_cn_list ← cn_list[i]
14        break
15    endif
16 done (endfor)
17 return overloaded_cn

```

Since there is one nested for loops used in Algorithm 2, the general mathematical expression for its time complexity can be written as:

$$T(n) = \alpha + \left[\sum_{i=0}^{m-1} \sum_{j=i+1}^n (j - i) \right]$$

The Big-O time complexity for Algorithm 2 can be calculated by considering the line-wise execution of the algorithm as following:

$$T(n) = 7 + 8m + 3mn + n$$

By excluding coefficient and lower order terms, we get the following simplified equation:

$$T(n) = 3mn$$

The Big-O time complexity for Algorithm 2 is finally given as:

$$T(n) = O(mn)$$

where m is `cn_list`.

5.2.2. VM Selection Algorithm

Once an overloaded compute node is detected, it is necessary to determine what VMs are to be migrated. This issue is solved by algorithms for VM selection. One such example can be selecting a VM randomly from a pool of VMs assigned to the host. Alternatively, based on our proposed algorithm called minimum RAM maximum CPU utilization (minRmaxC) algorithm as shown in Algorithm 3, VMs with the maximum amount of RAM usage are first selected, and then out of these selected VMs, the VMs with the maximum CPU utilization averaged over the last n quantifications are selected. The algorithm takes n number of previous CPU utilizations, RAM and CPU utilization statistics of VMs, and a pre-defined CPU utilization threshold. It starts with first sorting the VMs in terms of their RAM utilization in a descending order. Then, from the sorted list of VMs, the mean value of the last n number of CPU utilizations of each VM is determined. If the calculated mean value of CPU utilization of any particular VM in the sorted VM list is greater than or equal to the pre-defined CPU utilization threshold value, it is selected and returned as an output of the algorithm for VM migration.

A VM selected for migration should have maximum CPU and RAM utilizations among all other VMs located on any particular compute node. More precisely, a VM with maximum utilization is the most efficient case. To achieve this goal, we determine the utilization of a VM by (1). Where $c(i)$ is the VM CPU utilization and $m(i)$ is its RAM utilization. Therefore, in case there are multiple VMs for migration, the one which has the highest utilization is selected.

$$U(i) = c(i) * m(i) \tag{1}$$

Since the cost of VM live migration is mostly determined by its memory footprint. Therefore, it can be said, migration time of a VM is directly proportional to the memory size of that VM. As a result, memory size of VMs is a good measure for the cost of migration. Thus, in case there are options for migration, the VM which has the lowest memory footprint is suitable for migration.

The migration cost of a VM can be calculated by the (2)

$$\text{Migration cost}(i) = \alpha * U(i) + \beta * \text{memory size}(i) \quad (2)$$

Algorithm 3: Minimum RAM Maximum CPU utilization (minRmaxC) Algorithm

```

1 Input: n, vms_utilization_stats, cpu_threshold
2 Output: a VM to migrate
3 selected_vm ← None
4 maxTempVal ← 0
5 sum ← 0
6 mean ← 0
7 for i = 0 to length(vm_list) do
8   | maxTempVal ← i
9   | for j = i+1 to length(vm_list) do
10  |   | if (vm_ram_map[j] > vm_ram_map[maxTempVal]) then
11  |   |   | maxTempVal ← j
12  |   | endif
13  |   | done (endfor)
14  |   | swap vm[i] with vm[maxTempVal]
15 done (endfor)
16 for i = 0 to length(vm_list) do
17  | sum, mean ← 0
18  | for j = 0 to n do
19  |   | sum ← sum + vm_cpu_map[j]
20  | done (endfor)
21  | mean ← sum/n
22  | if mean ≥ cpu_threshold then
23  |   | selected_vm ← vm_list[i]
24  |   | break
25  | endif
26 done (endfor)
27 return selected_vm

```

Since there are two nested for loops used in Algorithm 3, the general mathematical expression for its time complexity can be written as:

$$T(n) = \alpha + \left[\sum_{i=0}^{m-1} \sum_{j=i+1}^m (j-i) \right] + \left[\sum_{i=0}^{m-1} \sum_{j=i+1}^n (j-i) \right]$$

The Big-O time complexity for Algorithm 3 can be calculated by considering the line-wise execution of the algorithm as following:

$$T(n) = 9 + 8m + 2m^2 + 2mn + n$$

By excluding coefficient and lower order terms, we get the following simplified equation:

$$T(n) = 2m^2 + 2mn$$

The Big-O time complexity for Algorithm 3 is finally given as:

$$T(n) = O(m^2 + mn)$$

where m is `vm_list`.

5.2.2.1. VM Selection Criteria

Based on the VM selection criteria form VM migration, the VMs with the maximum CPU utilization averaged over the last n quantifications are first selected.

$$CUM_i = \frac{1}{n} \sum_{k=0}^n CU_i(t-k)$$

$M(i, j)$ is a 2D array that contains VMs that contains VMs based on the following condition:

$$VM_i = \begin{cases} 1, & CUM_i \geq x \\ 0, & CUM_i < x \end{cases}$$

Where x is a threshold for CPU utilization. VM_i is an indicator that shows if i th VM is in $M(i, j)$ matrix or not. The VM for migration is selected based on minimum RAM, which is expressed by following equation:

$$VM_b = \min (M(RAM_{util}))$$

or it can be expressed as:

$$VM_s = \min[M(:, RAM_{util})]$$

5.2.3. VM Allocation Algorithm

In order to get the efficient compute nodes for hosting VM instances, the VM allocation algorithm engages an OpenStack Nova-scheduler which conducts an overall allocation process. More precisely, it returns the efficient compute nodes on which the VM instances would be placed. Therefore, the algorithm first selects a list of the light load compute nodes for the heavy load VM instances. Then, this list is delivered to the OpenStack-Nova API in order to initiate the VM migration process between source and destination compute nodes. The pseudo-code for algorithm is shown in Algorithm 4, which explains the method of selecting the light load compute node for the heavy load VMs. The algorithm takes n number of previous CPU utilizations, RAM and CPU utilization statistics of a compute node, as well as RAM and CPU utilization statistics of VMs. It starts with initializing the defined variables. Then, from the list of compute nodes, the mean values of the last n number of CPU utilizations and the last n number of RAM utilizations for each compute node is determined. If the calculated mean value of compute node's CPU utilization is greater than the CPU utilization of any particular VM and the calculated mean value of compute node's RAM utilization is greater than the RAM utilization of that particular VM, it leads to the fact that compute node has enough resources to accommodate the migrated VM. Hence, the algorithm selects and returns this compute node for VM instance placement.

Algorithm 4: VM Instance Allocation Algorithm

```
1 Input: n, cn_stats, vm_stats
2 Output: Available_cn_list
3 sum_ram, sum_cpu, mean_ram, mean_cpu ← 0
4 for i = 0 to length(cn_list) do
5     sum_ram, sum_cpu, mean_ram, mean_cpu ← 0
6     for j = 0 to n do
7         sum_cpu ← sum + cn_cpu_map[j]
8         ram_cpu ← sum + cn_ram_map[j]
9     done (endfor)
10    mean_cpu ← sum_cpu/n
11    mean_ram ← sum_ram/n
12    if (mean_cpu > vm_cpu) && (mean_ram > vm_ram) then
13        selected_cn_list ← cn_list[i]
14        break
15    endif
16 done (endfor)
17 return available_cn_list
```

The Big-O time complexity for Algorithm 4 is finally given as:

$$T(n) = O(mn)$$

where m is cn_list .

5.3. Experiment and Results

The proposed architecture has been validated in the cloud computing platform-based testbed discussed in Section 3.5. First it is necessary to generate the work load in an appropriate way in order to reproduce a realistic data. For this purpose, software called Lookbusy [100] is used, which is a simple application for load generation on a Linux system. It enables to generate predictable, fixed loads on CPUs, and also maintain selected amounts of memory active.

Several experiments are performed to evaluate the proposed algorithms. For this purpose, the proposed overload detection algorithm has been analyzed for two cases, i.e., the impact on an overall system without and with the execution of this algorithm. The proposed VM selection and

VM allocation algorithms have been compared with random VM selection and random VM allocation schemes. A VM instance type with 32 MB amount of RAM allocated to it. Eight VM instances are launched on compute node 1, 7 VM instances on compute node 2, 5 VM instances on compute node 3, and 3 VM instances on compute node 4. Load has been generated on random VM instances of compute node 1, which results in an increased CPU utilization of the VM instances.

During the experiment, the overload detection algorithm has been executed by the migration manager module, which detects an overloaded compute node based on pre-defined threshold value. The performance of our proposed overload detection algorithm has been compared for two scenarios: 1) overload situation without any algorithm used, and 2) direct overload detection [109]. The proposed algorithm detects an overloaded compute node if the average of the last n CPU utilization measurements is greater than the pre-defined threshold value, whereas the direct overload detection detects an overloaded compute node as soon as the CPU utilization surpasses the pre-define threshold value. The drawback in direct overload detection is that most of the times a compute node goes above the threshold for a very short period of time because there are always CPU utilization spikes in real time while performing tasks. In this case, a compute node would be detected as an overload node as soon as its CPU utilization is above the threshold for a very short period of time. The granularity can be selected according to the requirement; however, in most of the cases the system is not too sensitive to the CPU overloads. In the case when no overload detection algorithm is executed, the compute nodes stay overloaded, which may result in the degradation of an overall system/service. Fig. 5.4 depicts the detection of compute node for each run of the proposed overload detection and direct overload detection algorithms. It is obvious from the figure that the compute node is detected as

an overloaded compute node by direct overload detection algorithm as soon as it surpasses the threshold value of 80% CPU utilization. On the other hand, the proposed overload detection algorithm detects an overloaded compute node if the average of the last n CPU utilization measurements is greater than the pre-defined threshold value. In the case when the algorithm is not executed, there is no overload detection even if the CPU utilization surpasses the threshold value of 80%.

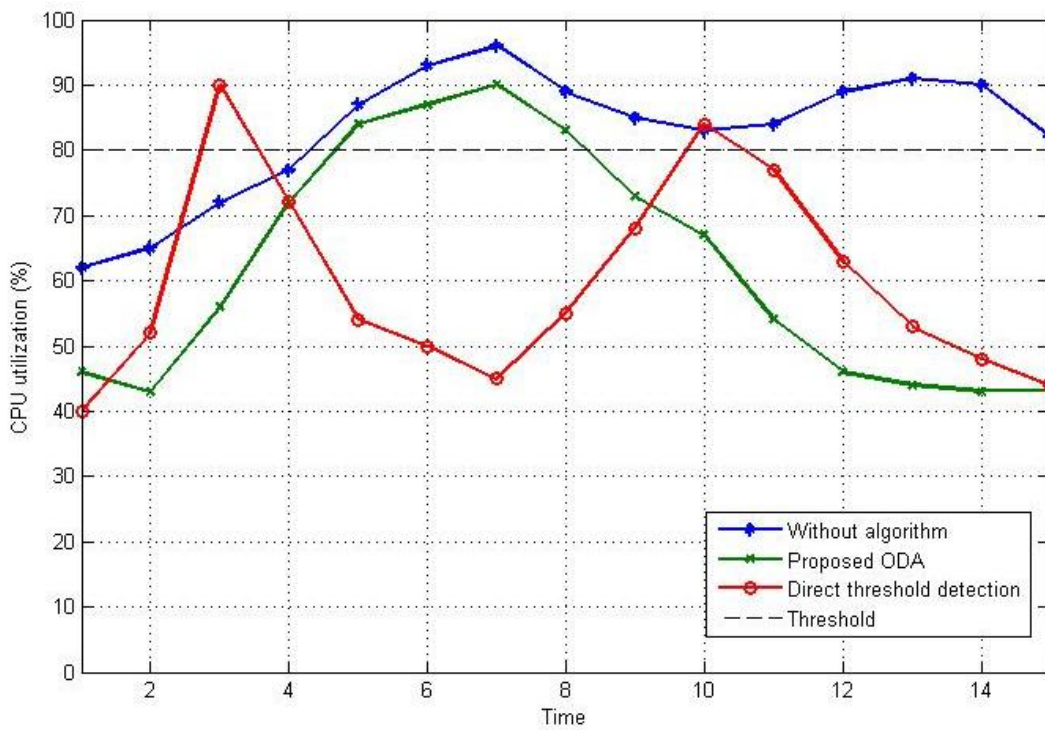


Figure 5.4: Performance analysis of the proposed overload detection algorithm

Once an overloaded compute node has been detected, the next step is to select the VM instances which are consuming most of the resource of that compute node. These VM instances can be selected randomly, but this may lead to performance degradation because the selected VM instances may be the minimum CPU utilization at the time of selection. Another issue with VM selection is that if the selected VM has the higher RAM assignment, the migration time required

for its migration would be high, which may cause a delay in the operation of the applications running on that VM. We address the aforementioned issues by proposing minRmaxC algorithm, which selects VM instances on the basis of minimum RAM and maximum CPU utilization. It accepts historical data on the resource usage by VM instances running on the compute nodes and returns a set of VM instances to be selected. The performance of the proposed minRmaxC algorithm is compared with random VM selection and maximum utilization [110] algorithms on the basis of the maximum CPU utilization. It is obvious from Fig. 5.5 that minRmaxC algorithm outperforms the random VM selection scheme. The VM instances with the maximum CPU utilization are selected by means of minRmaxC algorithm, whereas the random VM selection scheme selects VM instances regardless of the CPU utilization for each run. Although, the maximum utilization algorithm selects VMs which have higher CPU utilizations than the VMs selected by proposed minRmaxC algorithm, the former does not consider the RAM factor when selecting VMs, which may have effect on the minimum migration time required for migrating the selected VMs [110].

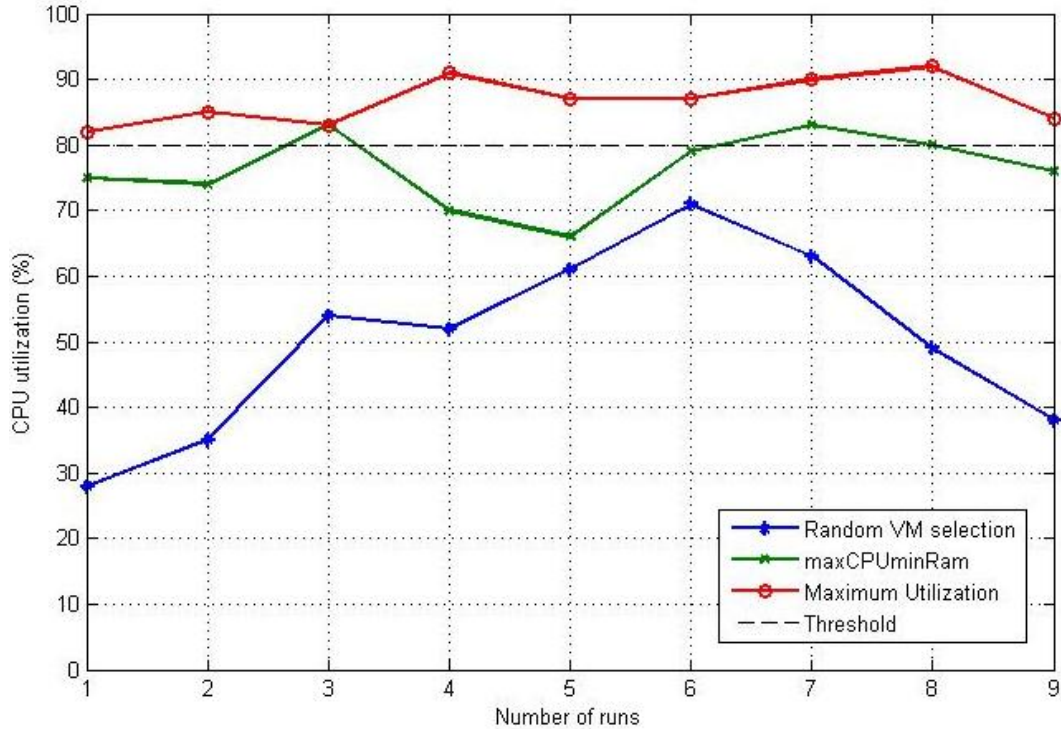


Figure 5.5: Performance analysis of the proposed minRmaxC-based VM selection algorithm

The proposed minRmaxC also addresses the issue of migration time when selecting the VMs for migration. It can be clearly seen in Fig. 5.6 that there is a significant difference in the minimum time required for migrating VMs with higher and lower RAMs. The VMs with higher RAM required higher time, whereas VMs with lower RAM require less time to complete the migration process. The performance of the proposed minRmaxC algorithm has been also compared with the maximum utilization algorithm. Since the maximum utilization considers only CPU utilization and does not consider RAM size when selecting VMs for migration, it would require more time for the migration of selected VMs. It is obvious from Fig. 5.6 that the VMs selected with maximum utilization algorithm require more time than that of selected with the proposed minRmaxC algorithm.

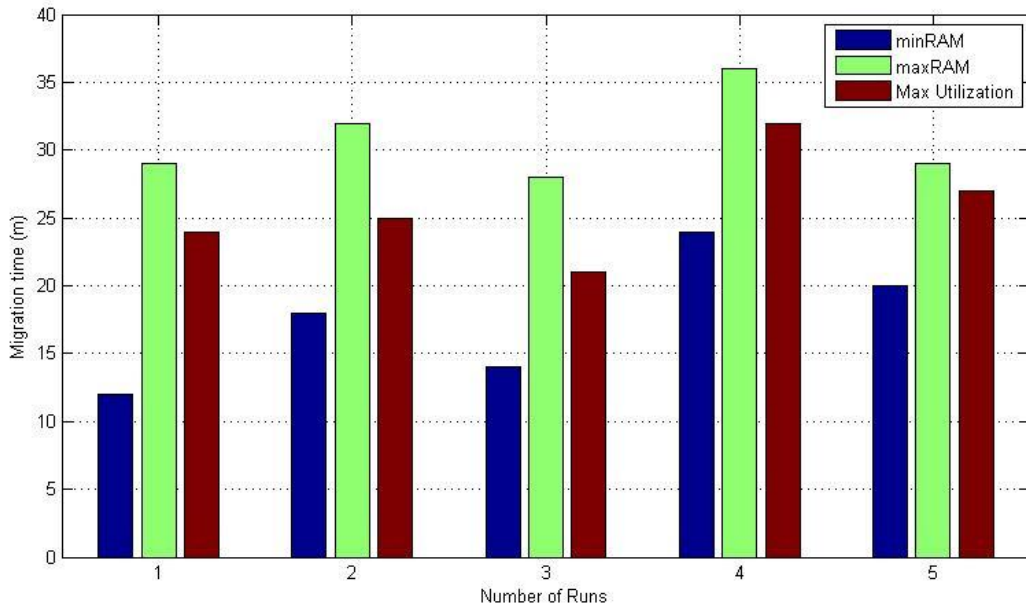


Figure 5.6: Minimum migration time versus RAM assigned to VMs

After selecting the overloaded VM instances, the migration manager module runs the VM allocation algorithm to place the selected VM instances on an efficient compute node. The performance of the proposed algorithm is compared with random VM allocation scheme and first fit VM allocation algorithm [111]. The first fit algorithm starts with the first compute node and determines the availability of the required CPU resources. If it finds enough resources, it places the migrated VM on that compute node, otherwise goes to the next compute node. On the other hand, our proposed algorithm finds the most efficient compute node among all the nodes. For this purpose, it calculates the mean CPU utilization and mean RAM utilization to determine the most efficient compute node from a list of compute nodes. If the calculated mean values satisfy the requirements of the migrated VM, it selects that compute node for VM placement. In case of random VM allocation scheme, the VMs are placed on random compute nodes. Fig. 5.7 depicts the comparison of our proposed VM allocation algorithm with the random VM allocation

algorithm and first fit VM allocation algorithm. It can be clearly seen that the proposed VM allocation algorithm outperform the first fit VM allocation algorithm by selecting light load compute nodes in terms of CPU utilization. It also clearly outperforms the random VM allocation scheme because it may place VMs on already overloaded compute nodes, which may lead to the performance degradation of an overall system.

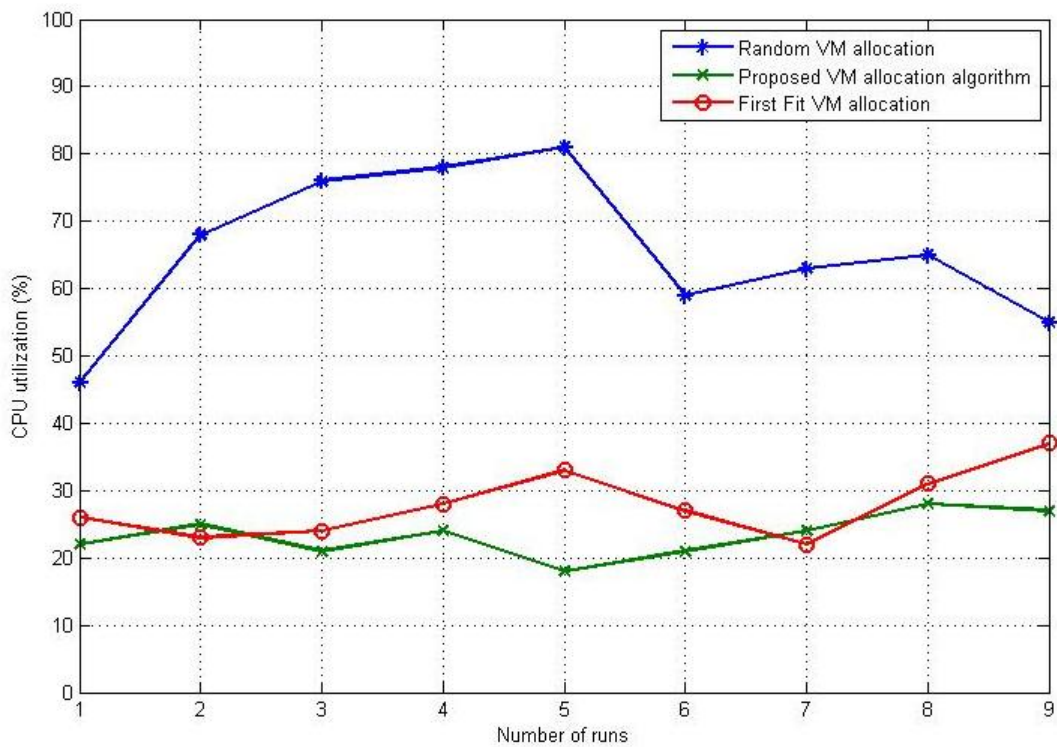


Figure 5.7: Performance analysis of the proposed VM allocation algorithm

Chapter 6

Conclusions

Cloud Computing plays a significant role in varied areas like e-business, search engines, data mining, virtual machines, batch oriented scientific computing, online TV amongst many others. Cloud computing has the potential to become an integral part of our lives. The resources residing in Cloud infrastructure such as compute, storage and network become the worthwhile infrastructure for computation, data storage and hosting network based applications. However, the dynamic nature of requests for cloud services or applications generated by remote end-users makes resource provisioning a major concern.

In this work, a resource orchestration framework for resource management and provisioning has been presented. The resource allocation is performed by implementing an algorithm, which allocates resources on the basis of unutilized resources. Two additional modules have been developed for communication with OpenStack and ONOS SDN controller to manage cloud and network resources. The framework has been validated by creating a VM instance in OpenStack with the framework. The random VM selection scheme is compared with the proposed MRMC algorithm for the VM(s) selected on the basis of the amount of utilized RAM as well as minimum CPU utilization. The experiment results show that the random VM selection scheme is outperformed in both the cases by the proposed MRMC algorithm.

An overview of OpenStack, the core services required for its deployment, and the integration with ONOS SDN controller is also presented. The integration has been validated by visualizing the OpenStack deployment in ONOS GUI. Furthermore, a monitoring system based on sFlow technology has been implemented in order to monitor the utilized resources of a compute node. A Host sFlow agent on each compute node periodically collects performance metrics related to its utilized infrastructural and network resources and stores them in a database. These metrics are then used when allocating resources or initiating VM migrations.

A design and implementation of a framework for dynamic VM migration in OpenStack clouds has been proposed in this thesis. The framework addresses the issue of the dynamic VM migration by implementing the proposed overload detection, VM selection, and VM allocation algorithms. It can be easily deployed to the default OpenStack installation by communicating with it by means of the public APIs, and without the requirement of any alterations of OpenStack's configurations. The experiment results show that the proposed framework prevents compute nodes from getting overloaded, selects the VM instances which are consuming most of

the compute node's resources, and migrates them to other efficient compute nodes. The proposed algorithms outperform the algorithms that are considered for the sake of comparison.

Despite substantial contributions of the current thesis in cloud resource orchestration, monitoring of infrastructural resources, and VM migrations, there are a number of open challenges that need to be addressed in order to further advance these areas. The proposed overload detection algorithm detects the overloaded situation on the basis of CPU utilization only. However, there are factors like bandwidth and RAM which may affect the compute node. These additional factors need to be considered to make the algorithm more efficient.

Furthermore, the proposed framework can be implemented on large-scale OpenStack deployments to improve the utilization of resources. More precisely, we intend to present a test environment for assessing the effectiveness of VM placement algorithms. This test environment builds on the CloudSim simulator, and extends it with several further components that are needed for reproducible experiments, like converters for publicly available workload traces and a workload generator. Also, there is a need to focus on the northbound API of ONOS SDN controller by executing customized control applications for flow management and control between the VM instances of OpenStack.

Bibliography

- [1] Drago, Idilio, et al. "Inside dropbox: understanding personal cloud storage services." *Proceedings of the 2012 ACM conference on Internet measurement conference*. ACM, 2012.
- [2] Quick, Darren, and Kim-Kwang Raymond Choo. "Google drive: forensic analysis of data remnants." *Journal of Network and Computer Applications* 40 (2014): 179-193.
- [3] H. Erdogmus, "Cloud Computing: Does Nirvana Hide behind the Nebula?," *IEEE Software*, vol. 26, no. 2, pp. 4–6, 2009.
- [4] Peng, Junjie, et al. "Comparison of several cloud computing platforms." *Information Science and Engineering (ISISE), 2009 Second International Symposium on*. IEEE, 2009.
- [5] Goldberg, Robert P. "Survey of virtual machine research." *Computer* 7.6 (1974): 34-45.

- [6] Chen, Yang, et al. "Resilient virtual network service provision in network virtualization environments." *Parallel and Distributed Systems (ICPADS), 2010 IEEE 16th International Conference on. IEEE*, 2010.
- [7] B Sonkoly, J Czentye, R Szabo, D Jocha, J Elek, S Sahhaf, W Tavernier, F Risso. Multi-domain service orchestration over networks and clouds: a unified approach. *ACM SIGCOMM Computer Communication Review*. 2015 Sep 22;45(4):377-8.
- [8] Wei, Yi, and M. Brian Blake. "Service-oriented computing and cloud computing: Challenges and opportunities." *IEEE Internet Computing* 14.6 (2010): 72-75.
- [9] IBM Inc. Blue Cloud project [URL]. IBM, June 2008. <http://www-03.ibm.com/press/us/en/pressrelease/22613.wss/>.
- [10] Wilder, Bill. "Cloud architecture patterns: using microsoft azure". *O'Reilly Media, Inc.*, 2012.
- [11] Ostermann, Simon, et al. "A performance analysis of EC2 cloud computing services for scientific computing." *International Conference on Cloud Computing. Springer Berlin Heidelberg*, 2009.
- [12] S. Bose, A. Pasala, D. Ramanujam A, S. Murthy, and G. Malaiyandisamy. "Sla management in cloud computing: A service provider's perspective". *Cloud Computing*, pages 413–436, 2011.
- [13] Xing, Yuping, and Yongzhao Zhan. "Virtualization and cloud computing." *Future Wireless Networks and Information Systems. Springer Berlin Heidelberg*, 2012. 305-312.

- [14] Dhingra, Mohit, J. Lakshmi, and S. K. Nandy. "Resource usage monitoring in clouds." *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing. IEEE Computer Society, 2012.*
- [15] B Jennings, R Stadler. "Resource management in clouds: Survey and research challenges." *Journal of Network and Systems Management*. 2015 Jul 1;23(3):567-619.
- [16] Swathi, T., K. Srikanth, and S. Raghunath Reddy. "Virtualization in cloud computing." *International Journal of Computer Science and Mobile Computing*, ISSN (2014): 540-546.
- [17] Paul Goransson, Chuck Black, "Software Defined Networks: A Comprehensive Approach", *1st Edition*, May 23, 2014.
- [18] Mohammad Banikazemi, D Olshefski, A Shaikh, J Tracey, G Wang. "Meridian: an SDN platform for cloud network services." *IEEE Communications Magazine*. 2013 Feb;51(2):120-7.
- [19] Bhardwaj, Sushil, Leena Jain, and Sandeep Jain. "Cloud computing: A study of infrastructure as a service (IAAS)." *International Journal of engineering and information Technology* 2.1 (2010): 60-63.
- [20] Boniface, Michael, et al. "Platform-as-a-service architecture for real-time quality of service management in clouds." *Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on. IEEE, 2010.*

- [21] IETF, "Network Virtualization Overlays," <http://datatracker.ietf.org/wg/nvo3/>, Sept. 2012, work in progress.
- [22] K Barabash et al., "A Case for Overlays in DCN Virtualization," *Proc. Wksp. Data Center-Converged and Virtual Ethernet Switching*, Sept. 2011.
- [23] T. Benson et al., "CloudNaaS: A Cloud Networking Platform for Enterprise Applications," *Proc. ACM Symp. Cloud Computing*, Oct. 2011.
- [24] Nicira, Inc., "Networking in the Era of Virtualization," white paper, 2012, <http://www.nicira.com>.
- [25] Menezes, Evandro, et al. "CPU utilization measurement techniques for use in power management." *U.S. Patent* No. 6,845,456. 18 Jan. 2005.
- [26] Gade, Anuradha, Bruce McMurdo, and Jeremy Stieglitz. "System and method for improving network resource utilization." *U.S. Patent* Application No. 11/154,204.
- [27] Karawash, Ahmad, Hamid Mcheick, and Mohamed Dbouk. "Quality-of-service data warehouse for the selection of cloud services: a recent trend." *Cloud Computing. Springer International Publishing*, 2014. 257-276.
- [28] M. Bichler, T. Setzer, and B. Speitkamp. "Capacity planning for virtualized servers". *In Workshop on Information Technologies and Systems (WITS), Milwaukee, Wisconsin, USA*, volume 1, 2006.

- [29] G. Khanna, K. Beaty, G. Kar, and A. Kochut. “Application performance management in virtualized server environments”. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pages 373–381. IEEE, 2006.
- [30] G. Jung, K. Joshi, M. Hiltunen, R. Schlichting, and C. Pu. “A cost-sensitive adaptation engine for server consolidation of multitier applications“. *Middleware 2009*, pages 163–183, 2009.
- [31] Vincent C. Emeakaroha and Marco A.S. Netto and Rodrigo N. Calheiros and Ivona Brandic and Rajkumar Buyya and Csar A.F. De Rose, “Towards autonomic detection of SLA violations in Cloud infrastructures,” *Future Generation Computer Systems*, no. 0, pp. –, 2011. [Online]. Available:
<http://www.sciencedirect.com/science/article/pii/S0167739X11002184>
- [32] Han, Fang-fang, et al. "Virtual resource monitoring in cloud computing." *Journal of Shanghai University (English Edition)* 15.5 (2011): 381-385.
- [33] Janna Anderson, Elon University, and Lee Rainie. “Technical seminar report on cloud computing”. *Intel Executive Summary*, 2005.
- [34] Wikipedia. Cloud computing. *Second International Symposium on Information Science and Engineering*, 2010.
- [35] Everything as a Service. Gathering clouds of xaas. *Second International Symposium on Information Science and Engineering*, 2010.
- [36] What is cloud computing? *Journal of Parallel and Distributed Computing*, 2008.

- [37] P. Goyal, R. Mikkilineni, and M. Ganti. “The fcaps in the business services fabric model”. In *proceedings of WETICE 2009:18th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2009.
- [38] Wikipedia. Wikipedia, free encyclopedia. 2009.
- [39] Ian Lumb, Eunmi Choi, and Bhaskar Prasad Rimal. “Virtualization for dummies”. *Second International Symposium on Information Science and Engineering*, 2010.
- [40] H. Q. Yu and S. Reiff-Marganiec. “A method for automated web service selection”. In *Services-Part I, 2008. IEEE Congress on*, pages 513-520. IEEE, 2008.
- [41] Jason Carolan and Steve Gaede. “Introduction to cloud computing architecture”. *Sun Microsystems Inc white paper*, 2009.
- [42] Wikipedia. Wikipedia, free encyclopedia. *3rd International Conference on Grid and Pervasive Computing-gpc-workshops*, 2008.
- [43] Usman Sait. “The future of cloud computing”. *Intel Executive Summary*, 2005.
- [44] Usman Sait. “Welcome to www.cloudtutorial.com”. *Intel Executive Summary*, 2005.
- [45] D. Baran. “Cloud computing basics”. *ICWS*, 2007.
- [46] Amazon. “Amazon elastic compute cloud (amazon ec2)”. *Journal of Parallel and Distributed Computing*, 2008.
- [47] Amazon. “Amazon simple storage service (amazon s3)”. *Journal of Parallel and Distributed Computing*, 2008.

- [48] IBM. IBM corp, “cloud computing:.. *Journal of Object Technology*, 2009.
- [49] Google App. Multitenancy, 2010.
- [50] Bhaskar Prasad Rimal, Eunmi Choi, and Ian Lumb. “A taxonomy and survey of cloud computing systems”. *Fifth International Joint Conference on INC, IMS and IDC*, 2009.
- [51] M. Turner, D. Budgen, and P. Brereton. “Turning software into a service”. *IEEE Computer*, Vol. 36, 2008.
- [52] “Software as a service: strategic backgrounder”. *Software and Information Industry Association*, 2001.
- [53] Jeffery F. Rayport and Andrew Heyward. “Envisioning the cloud: The next computing paradigm”. 2009. *Technical Report*, available at <http://www.hp.com/hpinfo/analystrelations/marketspace-090320-Envisioning-the-cloud.pdf>
- [54] Ya-Qin Zhang. “The future of computing in the cloud – client”. *The Economic Observer*, 2008.
- [55] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica. “Above the clouds: a Berkeley view of cloud computing”. *EECS Department, University of California, Berkley, Tech Rep*, 2009.
- [56] Liu, C., Mao, Y., Van der Merwe, J., and Fernandez, M. “Cloud Resource Orchestration: A Data-Centric Approach”. *In CIDR (2011)*.

- [57] Van der Merwe, J., Ramakrishnan, K., Fairchild, M., Flavel, A., Houle, J., Lagar-Cavilla, H. A., and Mulligan, J. “Towards a ubiquitous cloud computing infrastructure”. In *LANMAN* (2010).
- [58] Wood, T., Shenoy, P., Venkataramani, A., and Yousif, M. “Black-box and gray-box strategies for virtual machine migration”. In *NSDI* (2007).
- [59] Agarwal, S., Dunagan, J., Jain, N., Saroiu, S., Wolman, A., and Bhogan, H. “Volley: automated data placement for geo-distributed cloud services”. In *NSDI* (2010).
- [60] Wood, T., Gerber, A., Ramakrishnan, K., Shenoy, P., and der Merwe, J. V. “The case for enterprise-ready virtual private clouds”. In *HotCloud* (2009).
- [61] Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., and Vahdat, A. “Hedera: dynamic flow scheduling for data center networks”. In *NSDI* (2010).
- [62] S. T. Selvi, C. Valliyammai, and V. N. Dhatchayani. “Resource Allocation Issues and Challenges in Cloud Computing”. In *2014 International Conference on Recent Trends in Information Technology*, pages 1–6, Chennai, India, April 2014.
- [63] P. Salot. “A Survey of Various Scheduling Algorithm in Cloud Computing Environment”. *International Journal of Research in Engineering and Technology*, 2(2):131– 135, February 2013.
- [64] G. Rastogi and R. Sushil. “Cloud Computing Implementation: Key Issues and Solutions”. In *2nd International Conference on Computing for Sustainable Global Development*, pages 320–324, New Delhi, India, March 2015.

- [65] L. Uhsadel, A. Georges and I. Verbauwhede, “Exploiting Hardware Performance Counters,” *Proceedings of the 2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*, 10 August 2008, pp. 59-67.
- [66] L. O’Brien, P. Merson and L. Bass, “Quality Attributes for Service- Oriented Architectures,” *International Workshop on Systems Development in SOA Environments*, 20-26 May 2007, p. 3.
- [67] L. G. Williams and C. U. Smith, “PASA SM: A Method for the Performance Assessment of Software Architectures,” *Proceedings of the 3rd International Workshop on Software and Performance*, Rome, July 24-26, 2002, pp. 179- 189.
- [68] D. Nurmi, R. Wolski, C. Grzegorzczak, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, “The eucalyptus opensource cloud-computing system,” in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, ser. CCGRID '09. Washington, DC, USA: IEEE Computer Society*, 2009, pp. 124–131.
- [69] “Amazon Elastic Compute Cloud,” 2012. [Online]. Available: <http://aws.amazon.com/ec2/>
- [70] Yadav, Sonali. ”Comparative study on open source software for cloud computing platform: Eucalyptus, openstack and opennebula.” *International Journal of Engineering And Science* 3.10 (2013): 51-54.
- [71] octo Forster, Florian. ”Collectd, a daemon for collecting system performance statistics.” (2015).

- [72] Wang, Lizhe, et al., “eds. Cloud computing: methodology, systems, and applications”.
CRC Press, 2011.
- [73] A. Verma, P. Ahuja, and A. Neogi, “pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems,” in *Proc. of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pp. 243–264, 2008.
- [74] O. Sefraoui, M. Aissaoui, M. Eleuldj. “OpenStack: toward an open-source solution for cloud computing “. *International Journal of Computer Applications*. 2012 Jan 1;55(3).
- [75] OpenStack Nova: <http://nova.openstack.org/>, 2011.
- [76] JungYul Choi, “Virtual Machine Placement Algorithm for Saving Energy and Avoiding Heat Islands in High-Density Cloud Computing Environment,” *J. KICS*, vol. 41, no. 10, pp. 1233-1235, Oct. 2016.
- [77] F. Wuhib, R. Stadler, and H. Lindgren, “Dynamic resource allocation with management objectives—Implementation for an OpenStack cloud,” in *Proc. of Network and service management (cnsm), international conference and workshop on systems virtualization management (svm)*, pp. 309-315, 2012.
- [78] S. Kumar, V. Talwar, V. Kumar, P. Ranganathan, and K. Schwan, “vManage: Loosely Coupled Platform and Virtualization Management in Data Centers,” in *Proc. of the 6th International Conference on Autonomic Computing (ICAC)*, pp. 127–136., 2009.

- [79] W. Zheng, R. Bianchini, G. J. Janakiraman, J. R. Santos, and Y. Turner, “JustRunIt: Experiment-Based Management of Virtualized Data Centers,” in *Proc. of the 2009 USENIX Annual Technical Conference*, pp. 18–33., 2009.
- [80] X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, and D. Gmach, “1000 Islands: Integrated capacity and workload management for the next generation data center,” in *Proc. of the 5th International Conference on Auto-nomic Computing (ICAC)*, pp. 172–181, 2008.
- [81] X. Wang, and Y. Wang, “Coordinating Power Control and Performance Management for Virtualized Server Clusters,” *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 22, no. 2, pp. 245–259, 2011.
- [82] G. Han, W. Que, G. Jia, and L. Shu, “An Efficient Virtual Machine Consolidation Scheme for Multimedia Cloud Computing,” *Sensors*, vol. 16, no. 2, pp. 246, 2016.
- [83] R. Nathuji, and K. Schwan, “VirtualPower: coordinated power management in virtualized enterprise systems,” *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 265-278, 2007.
- [84] N. Bobroff, A. Kochut, K. “Dynamic placement of virtual machines for managing SLA violations“. *Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management (IM)*, Munich, Germany, 2007; 119–128.
- [85] B. Nandi, A. Banerjee, S. Ghosh, N. Banerjee. “Stochastic VM multiplexing for datacenter consolidation“. *Proceedings of the 9th IEEE International Conference on Services Computing (SCC)*, Honolulu, HI, USA, 2012; 114–121.

- [86] Keystone service documentation. <http://es.slideshare.net/openstackindia/openstack-keystone-identity-service?related=1>.
- [87] Nova service documentation.
http://docs.openstack.org/developer/nova/project_scope.html.
- [88] Neutron service documentation.
http://docs.openstack.org/admin-guide-cloud/networking_arch.html.
- [89] OpenStack: Openstack mitaka (2016). <https://www.openstack.org/software/mitaka>
- [90] QEMU - open source processor emulator, 2009. <http://www.qemu.org>
- [91] P. Singh, and S. Manickam. "Design and deployment of OpenStack-SDN based test-bed for EDoS." *Reliability, Infocom Technologies and Optimization (ICRITO)(Trends and Future Directions), 2015 4th International Conference on. IEEE, 2015.*
- [92] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, G. Parulkar. "ONOS: towards an open, distributed SDN OS". *In Proceedings of the third workshop on Hot topics in software defined networking 2014 Aug 22 (pp. 1-6). ACM.*
- [93] DN Gde, Q Nguyen-Van, TV Duc, N Nguyen-Sinh, PJ Alvin, K Kim, D Choi. "Design of service abstraction model for enhancing network provision in future network". *In Network Operations and Management Symposium (APNOMS), 2016 18th Asia-Pacific 2016 Oct 5 (pp. 1-4). IEEE.*

- [94] Phaal, Peter, Sonia Panchen, and Neil McKee, "InMon corporation's sFlow: A method for monitoring traffic in switched and routed networks", *RFC 3176*, 2001.
- [95] Some practical considerations for monitoring in OpenStack cloud, <https://www.mirantis.com/blog/openstack-monitoring/>
- [96] Graphite – Scalable Realtime Graphing, <http://www.graphite.wikidot.com/>
- [97] Host sflow, <http://www.sflow.net/>
- [98] "Libvirt, the Virtualization API," <http://libvirt.org/>, Accessed in May. 2017.
- [99] OpenStack Community, "Ceilometer," 2013, available at: <https://wiki.openstack.org/wiki/Ceilometer/>. Accessed in: May 2017.
- [100] "lookbusy -- a synthetic load generator,"
- [101] Afaq Muhammad, Song Wang-Cheol; Seok Seung-Joon; Kang M.G., "sFlow Monitoring System in a Disaster-Resilient Global SDN Testbed based on KOREN/APII/TEIN Network", *International Journal of Computer Aided Engineering and Technology*.
- [102] Muhammad Afaq, Wang-Cheol Song, "sFlow-Based Resource Utilization Monitoring in Clouds", *Network Operations and Management Symposium (APNOMS)*, 2016 18th Asia-Pacific (Kanazawa, Japan).
- [103] Afaq Muhammad, Wang-Cheol Song, "Real-time Classification, Visualization, and QoS Control of Elephant Flows in SDN", *The Journal of Korea Information and Communications Society (J-KICS)*, Vol.42 No.03: 612-622 (March 2017).

- [104] Afaq Muhammad; Zubair Amjad, Wang-Cheol Song, "A Framework for Orchestration based on Live Migration of Virtual Machines", *KSII Conference 2016* (Seoul, Korea), 17(2), 121-122,
- [105] (in review) Afaq Muhammad, Wang-Cheol Song, "A Framework for Resource Utilization-based Dynamic Migration of VMs in OpenStack Clouds", *The Journal of Korea Information and Communications Society (J-KICS)*.
- [106] (in review) Afaq Muhammad, Wang-Cheol Song, "Service Orchestration over Clouds and Networks", *The International Journal of Communication Networks and Distributed Systems (IJCND)*.
- [107] Afaq Muhammad, Wang-Cheol Song, "Deployment of OpenStack and its Integration with ONOS SDN Controller", *2017 Korea Information and Communications Society (KICS) Conference*, Jeju, South Korea.
- [108] Yazir, Yagiz Onat, et al. "Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis." *Cloud Computing (CLOUD)*, 2010 *IEEE 3rd International Conference on. IEEE*, 2010.
- [109] Abdelsamea, Amany, et al. "Virtual machine consolidation challenges: a review." *International Journal of Innovation and Applied Studies* 8.4 (2014): 1504.
- [110] Chowdhury, Mohammed Rashid, Mohammad Raihan Mahmud, and Rashedur M. Rahman. "Implementation and performance analysis of various VM placement strategies in CloudSim." *Journal of Cloud Computing* 4.1 (2015): 20.

- [111] Chowdhury, Mohammed Rashid, Mohammad Raihan Mahmud, and Rashedur M. Rahman. "Study and performance analysis of various VM placement strategies." *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2015 16th IEEE/ACIS International Conference on. IEEE, 2015.*