Thesis no: MSEE-2016:17

Smart Placement of Virtual Machines
Optimizing Energy Consumption

Raywon Teja Kari

Department of Communication Systems
School of Computing of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering with emphasis on Telecommunication Systems. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author(s):
Raywon Teja Kari
E-mail:
kari.raywonteja@gmail.com
rakb15@student.bth.se

University advisor:
Dr. Dragos Ilie

University examiner:
Dr. Kurt Tutschku

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

ABSTRACT

Context: Recent trends show that there is a tremendous shift from IT companies following traditional methods by hosting their applications/systems in self-managed on premise data centers to using the so-called cloud data centers. Cloud computing has received immense popularity due to its architecture and the ease of usage. Due to this increase in demand and shift in practices, there has been a tremendous increase in number of data centers over a period, resulting in increase of energy consumption. In this thesis work, a research is carried out on optimizing the energy consumption of a typical cloud data center. OpenStack cloud computing software is chosen as the platform in this research. We have used live migration as a key aspect in this research.

Objectives: In this research, our objectives are as follows:

- Design an OpenStack testbed to implement the migration of virtual machines.
- To estimate the energy consumption of the data center.
- To design a heuristic algorithm to evaluate the performance metrics and to optimize the overall energy consumption.

Methods: We have used PowerAPI, a software tool to estimate the energy consumption of hosts as well as virtual machines. A heuristic algorithm is designed and implemented in an instrumental OpenStack testbed to optimize the energy consumption. Server consolidation and load balancing of virtual machines methodologies are used in the heuristic algorithm design. Our research is carried out against the functionality of Nova scheduler of OpenStack.

Results: Results section describes the values of performance metrics yielded by carrying out the experiment. The obtained results showed that energy can be optimized significantly by modifying the way OpenStack nova scheduler can work. The experiment is carried out on vanilla OpenStack and OpenStack with the heuristic algorithm in place, In the second case, the nova scheduler algorithms are not used but the heuristic algorithm is used instead. The CPU utilization and CPU load were noticed to be higher than the metrics observed in case of OpenStack with nova scheduler. Energy consumption is observed to be lesser than the consumption in OpenStack design with nova scheduler.

Conclusions. The research tells that energy consumption can be optimized significantly using desired algorithms without compromising the service quality it offers. However, the design impacts on CPU slightly as the metrics are observed to be higher when compared to that in case of OpenStack with nova scheduler. Although it won't have noticeable impact on the system

Keywords: Energy Consumption, OpenStack, Optimization, Virtual Machines.

ACKNOWLEDGEMENT

The completion of this research work could not have been possible without the continuous support and guidance of my supervisor Dr. Dragos Ilie. It gives me a lot of happiness and pleasure to express my deepest and sincere thanks to him for his comments, suggestions and feedback throughout the year. I would also like to thank him for giving me an amazing opportunity to learn and develop skills voluntarily apart from regular courses and assignments. He was very helpful and is a great teacher.

I would like to thank Dr. Kurt Tutschku, our thesis examiner for conducting and managing this course. I would also thank him for his lectures and motivational discussions throughout the thesis course.

I would like to thank Dr. Patrik Arlos for helping us out with the hardware setups and troubleshooting issues when the hardware crashed.

I would like to thank all the Communication Systems department for this opportunity.

Last but not the least, I would like to thank my family and friends for their continuous love and support which helped me a lot to finish this thesis work and my Masters.

CONTENTS

ABSTRACT ..... **I**

ACKNOWLEDGEMENT ..... **II**

CONTENTS ..... **III**

LIST OF FIGURES ..... **V**

LIST OF TABLES ..... **VI**

1 INTRODUCTION **7**

1.1 CLOUD COMPUTING 7
1.1.1 Software-as-a-Service (SaaS) 7
1.1.2 Platform-as-a-Service (PaaS) 9
1.1.3 Infrastructure-as-a-Service (IaaS) 10
1.2 VIRTUALIZATION 12
1.2.1 Importance of Using Virtualization Technology in Cloud Computing 13
1.3 ENERGY EFFICIENCY IN CLOUD COMPUTING 13
1.4 MOTIVATION 14
1.4.1 Aims and Objectives of the Thesis 15
1.4.2 Research Questions 15
1.5 OUTLINE OF THE DOCUMENT 15

2 A SKETCH OF THE TECHNOLOGY **17**

2.1 OPENSTACK 17
2.1.1 Swift (Object Storage) 19
2.1.2 Keystone (Identity) 19
2.1.3 Nova (Compute) 20
2.1.4 Neutron (Networking) 20
2.1.5 Cinder (Block Storage) 20
2.1.6 Glance (Image Service) 20
2.1.7 Horizon (Dashboard) 21
2.2 MIGRATION OF VIRTUAL MACHINES 21
2.2.1 Live Migration 23
2.2.2 Live Migration in OpenStack 25
2.3 POWERAPI 26
2.4 OPTIMIZATION ALGORITHMS 27
2.4.1 Heuristic Method 28

3 STATE OF THE ART **29**

3.1 LIVE MIGRATION OF VIRTUAL MACHINES 29
3.2 RESOURCE MANAGEMENT IN DATA CENTERS 30
3.3 OPTIMIZING ENERGY CONSUMPTION IN DATA CENTERS 30

4 METHODOLOGY **32**

4.1 EXPERIMENT MODEL 32
4.2 HEURISTIC ALGORITHM DESIGN 32
4.2.1 Algorithm 33
4.3 SIMULATION 38
4.4 WORKLOAD GENERATION 39
4.5 COMPARISON METHOD 39
4.6 IMPLEMENTATION PHASES 40
4.6.1 System 41
4.6.2 Test Bed 41
4.7 EVALUATION PROCEDURE 42

4.8 TEST BED SPECIFICATIONS 43
4.8.1 Case – I 43
4.8.2 Case – II 44
4.9 PERFORMANCE METRICS 44
5 RESULTS & ANALYSIS 45
5.1 SCENARIO – I 45
5.1.1 Results 45
5.1.2 Analysis 46
5.2 SCENARIO – II 49
5.2.1 Results 49
5.2.2 Analysis 50
6 CONCLUSIONS 53
6.1 ANSWER TO RESEARCH QUESTIONS 53
6.2 FUTURE WORK 54
REFERENCES 55
APPENDIX 58

LIST OF FIGURES

Figure 1: Cloud Computing Services [2] 7
Figure 2: Software-as-a-Service Model 8
Figure 3: Platform-as-a-Service (PaaS) Model 9
Figure 4: Infrastructure-as-a-Service Model 11
Figure 5: Virtualization Architecture 12
Figure 6: OpenStack Software Diagram [15] 18
Figure 7: OpenStack Conceptual Architecture [16] 19
Figure 8: Overview of VM Migration 22
Figure 9: Example of Live Migration 24
Figure 10: Example of BitWatts Functionality [25] 27
Figure 11: Architecture of Setup - 1 42
Figure 12: Comparison of Energy Consumption 45
Figure 13: CPU Utilization 46
Figure 14: Energy Saving for 2 hours 47
Figure 15: Estimate of Energy Saved in One Day 47
Figure 16: Estimate of Money Savings in One Day 48
Figure 17: Estimate of Money Saved in One Month 48
Figure 18: Comparison of Energy Consumption 49
Figure 19: CPU Utilization 50
Figure 20: Energy Saving in 2 Hours 51
Figure 21: Estmate of Energy Saved in One Day 51
Figure 22: Estimate of Cost Saving in One Day 52
Figure 23: Estimate of Cost Saving in One Month 52

LIST OF TABLES

Table 1: Specification of Testbed – 1 43
Table 2: Specification of Testbed - II 44
Table 3: Comparison of Values 46
Table 5: Comparison of Energy Consumption values 50

1 INTRODUCTION

1.1 Cloud Computing

In the Information Age, Cloud Computing has emerged as a fascinating paradigm for managing and delivering services over the Internet to common public. This emergence has changed the view of information technology. In general, Clouds are categorized in the way their deployments are done such as private, community, public and hybrid clouds. Clouds are also different in the way their services are offered [1]. They often form a stack like structure with services they offer i.e., software, platform and infrastructure as a service (SaaS, PaaS, IaaS respectively) as shown in **Figure 1** [2].

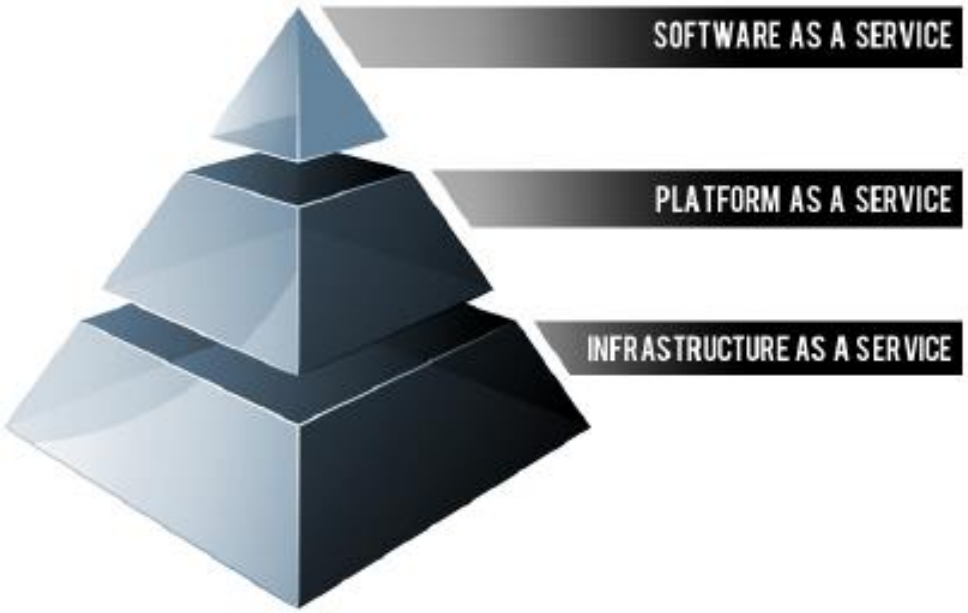


Figure 1: Cloud Computing Services [2]

1.1.1 Software-as-a-Service (SaaS)

SaaS is a software which is deployed over the internet. These Softwares are licensed to customers as a service on demand through a subscription. It is referred to as "pay-as-you-go" model or as a free service through which revenues are generated in the form of advertisements [2]. **Figure 2** shows the basic model of a Software-as-a-Service system. Salesforce, LinkedIn, Concur Technologies are the few noted companies which use SaaS.

These services are possessed with the following characteristics:

- Software services developed in this case are provided with web access.
- Software is managed from a central location.

- Software can be used by many users simultaneously.
- Users need not update/upgrade its patches, as this is taken care by the company providing these services.
- Application Programming Interfaces APIs are designed and implemented to link between several software services[2].

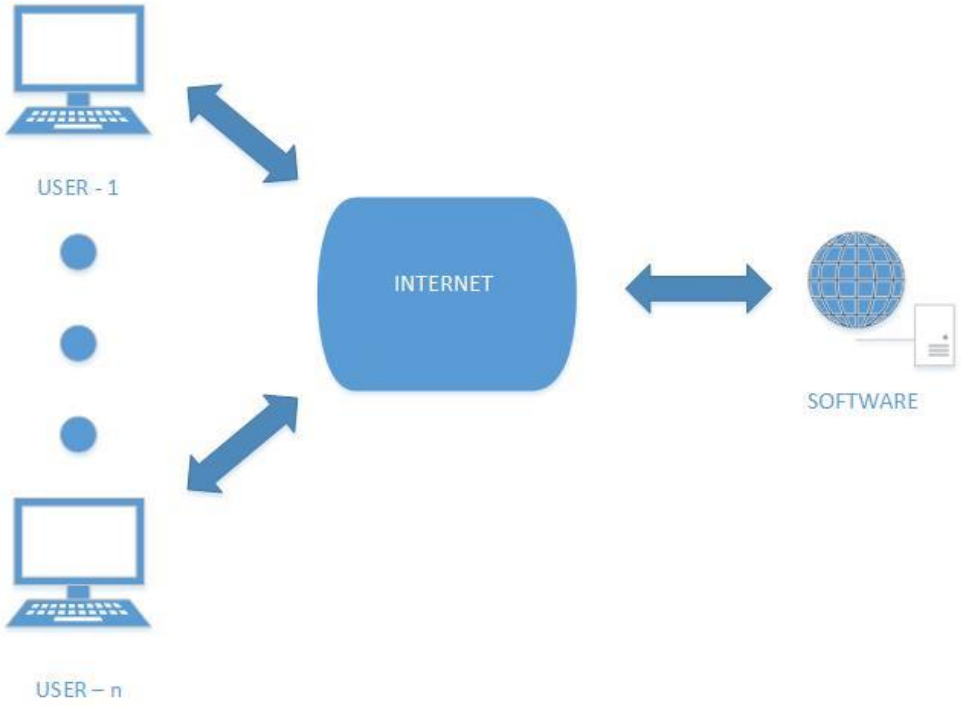


Figure 2: Software-as-a-Service Model

1.1.1.1 Advantages of SaaS

SaaS is the most used platform in Cloud Computing [2]. Few of the noted advantages of this service. They are as follows:

- User Advantages:
  - Low costs
  - Low overhead
  - Customizable
  - Decentralized
- Provider Advantages
  - Scalable applications
  - Highly customizable
  - Stable and easy maintenance
  - Maximum efficiency [3]

1.1.1.2 Disadvantages of SaaS

Although SaaS is considered the most used platform in Cloud Computing, it has got its own disadvantages. Few of them are as follows:

- Security concerns: as the service is open to public, it is quite difficult to ensure security.

- Performance: As the service is accessed using internet, the performance depends on the how good the internet connection is and hence the performance is not dependent on the provider but the user and therefore the performance varies and cannot be the best for all the users [4].

1.1.2 Platform-as-a-Service (PaaS)

PaaS allows the users to create and manage web applications without buying software services and infrastructure underneath it. PaaS is similar to SaaS but in PaaS the user need not use the web to access software but use the platform for the deploymentof software which is thereby delivered over the web. **Figure 3** shows the basic model of a Platform-as-a-Service system. Amazon Web Services, Windows Azure, IBM Smart Cloud, Google are the few noted products which use PaaS.

These services are possessed with the following characteristics:

- Software services developed on the platform are built with user interfaces which are accessed over the web and also through the platform to create, manage, host and maintain the applications in the same integrated development environment.
- Multiple tenants can access the same application running on the platform via web interfaces and platform.
- PaaS has the feature to integrate with web services and databases [2].

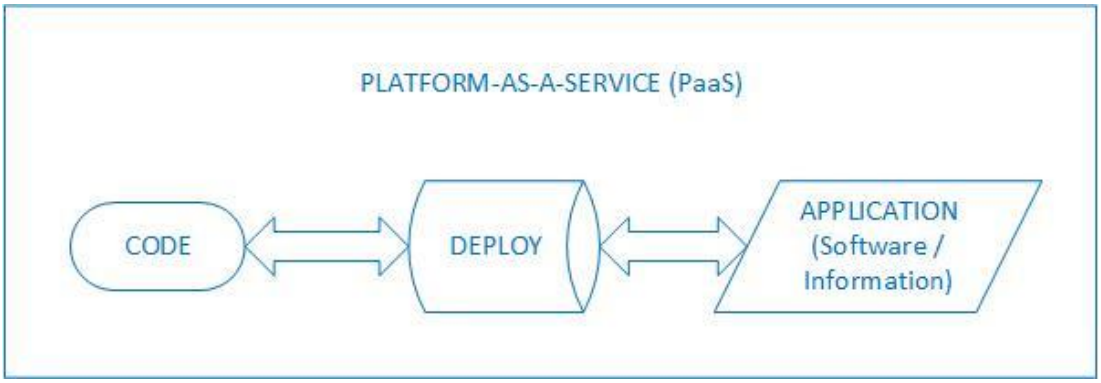


Figure 3: Platform-as-a-Service (PaaS) Model

1.1.2.1 Advantages of PaaS

- PaaS allows the users to code high level programs with less complexity and workload because buying and maintaining the software and infrastructure underneath it is managed by the vendor, and hence the overall application development can be more effective as it is equipped with inbuilt infrastructure and thereby enabling the option of easy maintenance and enhancement.
- Multiple users from different geographical locations can work simultaneously through the internet is possible [5].

1.1.2.2 Disadvantages of PaaS

- All the users might not use the tools equipped in the platform to fullest due to security i.e., the platform will not be open, but configured with firewalls which blocks all users by default [5].
- If the application built on the platform brings up a situation where the underlying hardware needs to be configured in a different way, then the PaaS model may not be helpful as the underlying hardware cannot be changed [2].

1.1.3 Infrastructure-as-a-Service (IaaS)

IaaS delivers the infrastructure such as servers, storage, network, and operating systems as a service to the users. This measure ensures that the users need not buy the physical equipment rather use the equipment as a fully outsourced service on demand. IaaS clouds are often categorized as Public Cloud and Private Cloud [2]. Amazon web services, IBM, Microsoft, VMWare, CSC, OpenStack, Rackspace are few products and companies which make use of the IaaS.

Public Cloud: public cloud consists of shared resources which are deployed on self-service basis over the internet. All the users can use these kind of resources.

Private Cloud: private cloud means that the resources are deployed on a private network which cannot be used by other users other than those with access to the private network.

There is another kind of cloud developed which is named as Hybrid Cloud. Hybrid cloud is the combination of both public and private clouds. In this kind of cloud infrastructure, both public clouds and private clouds are present and one is able to access the other cloud whenever demand increases [2]. **Figure 4** shows the basic IaaS system.

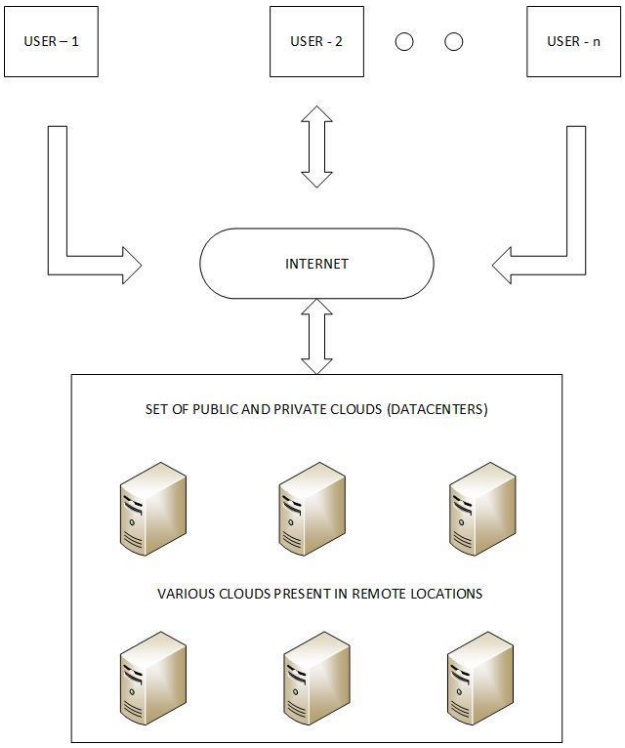


Figure 4: Infrastructure-as-a-Service Model

The characteristics of IaaS are as follows:

- Available hardware resources are used as a service.
- Dynamic scaling of the resources is possible when demand increases
- Has the possibility of multiple users using the same hardware simultaneously without interrupting each other's work on shared basis [2].

1.1.3.1 Advantages of IaaS

- The amount of resources used by the users are metered by the cloud which are later used to bill the users for the amount they have used. This enables the users to use the available resources with responsibility and thereby managing the resources while still having control over the system.
- Ensures collaboration between individuals and companies to make use of the available resources and expanding when necessary [6].

1.1.3.2 Disadvantages of IaaS

- An IaaS may host several company's workloads and can lead to a potential security threat by exposing each companies bulk data to the other.
- Allocation of resources to its users can be tricky as it is difficult to look after each user out of all the users [6].
- Infrastructure lock-in by vendors.

1.2 Virtualization

Virtualization is a method of creating a virtual version of an object such as a computer hardware platform, an operating system, a storage device, a computer network etc. Virtualization is often implemented as hardware virtualization. In this method, virtual machines are created which acts like a real computer with an operating system [7]. **Figure 5** shows us the basic architecture of a physical machine with virtual machines on top of it.

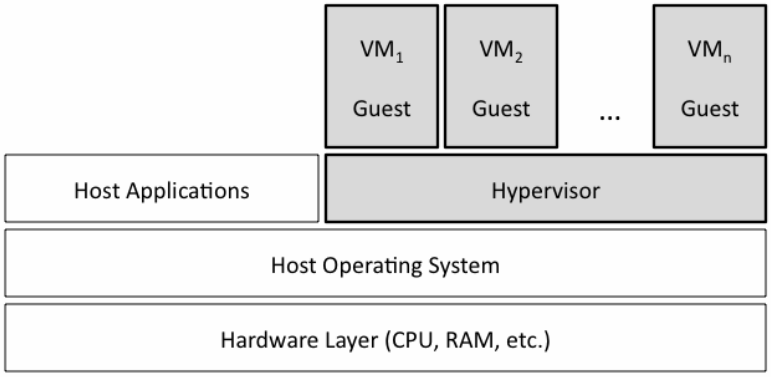


Figure 5: Virtualization Architecture

There are several types of virtualization introduced. Few of them are as follows [7]:

- Hardware virtualization: In this type of virtualization, virtual machines are created and an operating system is installed in them. Any software which is executed in the virtual machine is separated from the underlying hardware equipment, which means a windows host can have Linux virtual machines in it and vice versa. The software which is used to control the virtual machines on the host is referred as a hypervisor or virtual machine manager. Types of hardware virtualization are as follows:
  - Full virtualization
  - Partial virtualization
  - Para virtualization

- Desktop virtualization: In this type of virtualization, several desktop operating systems are stores as virtual machines in a server and these machines are accessed from client devices such as PCs.

- Application virtualization: In this type of virtualization, several applications are wrapped in a run-time layer that separates them from the underlying operating system and the users can access these applications from client devices such as PCs.

- Storage virtualization: In this type of virtualization, data is stored in a cloud environment, which is a collection of virtualized resources rather than storing in the physical machine directly. This ensures safety as the virtual environment can be migrated to other devices ensuring connectivity and safety. This method is incorporated often

in data centers which host storage as its service such as Google Drive, Dropbox etc.

- Memory virtualization: In this type of virtualization, multiple hosts are connected to each other and all the hosts contribute to a common memory pool. The operating system ensures that applications in these hosts can access this pool of memory in a transparent way and therefore enabling applications to use more memory than a single host can have.

- Network virtualization: In this type of virtualization, network elements are grouped together in a particular configuration and these network elements are bridged together which enables all the elements to function together there by providing more network resources to the machines or applications deployed to them. This ensures that separate physical connections need not be done in between machines to enable connectivity between virtual machines as the existing connections are made use of.

- Data virtualization: This type of virtualization allows an application to access, retrieve and change the data without knowing how the data is formatted and where it is physically present.

1.2.1 Importance of Using Virtualization Technology in Cloud Computing

Virtualization is a crucial element to cloud computing, as it enables the technology allowing to create an abstraction layer thereby hiding the complexity of the underlying software or hardware. By virtualizing the system, one can abstract the computer resources and also ensures scalability of the available resources. Cloud computing is possible without virtualization but would be very costly and ineffective because management of the resources would be very difficult in that case. Virtual machines are more safe and flexible when compared to the physical machines with same operations [8].

Internet is a large collection of several networks present at different geographical locations throughout the world connected to each other. Prior the emergence of cloud computing, communication between these networks are done through traditional server-client model. After the emergence of cloud computing, the view of the Internet has changed tremendously and the key element in the large-scale deployment of cloud computing is virtualization. Maintenance of the IT resources can be easier if virtualization is used in cloud computing and the availability of the hardware in virtual servers can be kept at a high level. Configurations can be changed easily while the servers are in running state [9].

1.3 Energy Efficiency in Cloud Computing

In recent times there is a tremendous increase in the usage of Information and Communication Technologies. Cloud computing has received great attention as it delivers the ICT services thereby increasing the utilization of a data center resources. As the users increase, the utilization also increases which means the equipment which a data center has, is utilized at the fullest and brings up a demand for expansion at a point of time. Hence, the maintenance costs such as energy usage, network usage, memory usage etc. increase [10].

Cloud computing offers several services to its users and the user satisfaction is one of the most important goals of a data center. Although the maintenance costs cannot be neglected in satisfying the users demand. It is equally important to run a data center efficiently without compromising the service quality which it offers. Running a data center with greater efficiency and without compromising the service quality is the ultimate case which one provider can think of. Currently ICT industry is consuming energy which is increasing day by day but it is also required for increasing the productivity [10].

At present, ICT is facing energy, power and other associated costs concerns due to the increase in the utilization of its services. Energy consumption is a key parameter for a data center along with other parameters such as the high-availability of its services with maximum utilization of network, memory and applications etc. There is an increase in carbon dioxide emissions and other greenhouse gases and it is alarming to know that the temperature is beyond the limit which earth is to be in [11]. Hence it is very important for a data center to run in an energy efficient way without compromising on the quality of service they offer to its users [12].

1.4 Motivation

Large data centers often consist of several hundreds or even thousands of physical nodes. These nodes work collectively to offer respective services which the data center is entitled to. This large number of nodes host an even greater number of virtual machines, which can be various services, applications etc. Proper management of these resources is mandatory because any error or negligence in maintaining these resources can lead to break down of the data center permanently, as there will be several users who depend on these services. Therefore, proper management is a must. Other concerns are the maintenance costs of these resources and the service quality they offer. There should not be service discrepancies at any point of time. Hence, proper management is needed without compromising the service quality they provide.

Parameters such as CPU utilization, memory usage, network usage etc are important factors for determining the data centers capacity and the service quality they offer. Another important metric that is receiving much attention is the energy consumption. As many as 2.5 billion people are online throughout the world which is 566% more when compared to the number of people online in the year 2000 [13].

Live Migration is moving a virtual machine from one physical machine to other, with an unnoticeable downtime. This measure is introduced to relocate virtual machines to increase the efficiency of the data center. The working of a virtual machine depends on the physical machine which provides the resources to run a virtual machine. By implementing this technique, better performance can be achieved as the virtual machines can run with minimum downtime or service discrepancy because the virtual machines can now be migrated to other hosts whenever necessary or in certain conditions such as host being over loaded etc. Live migration can be used to conserve the resources as physical machines can be turned into sleep mode and other physical machines which are running below their optimal level can be used to host these virtual machines.

1.4.1 Aims and Objectives of the Thesis

The aim of this thesis is to use heuristic model to investigate the optimal location of VMs in order to minimize the energy consumption. We use PowerAPI to estimate the energy consumption of physical machines as well as virtual machines in an OpenStack environment.

The objectives of the thesis are as follows:

- To estimate the energy consumption of various heterogeneous hosts using PowerAPI
- To design an OpenStack testbed to implement VM migration
- To calculate the energy consumption of hosts before migrating, while migration and after migrating the virtual machines.
- To design a heuristic algorithm to evaluate the performance metrics such as energy consumption and optimizing it.

1.4.2 Research Questions

The following research questions will be explored in this thesis:

1. How should the relationship between workload and energy consumption be defined to enable comparisons in a heterogeneous hardware environment?

2. How should a heuristic model be defined to provide an efficient and simple method to optimize energy consumption in a data center?

3. How much cost can be saved using the heuristic model defined in question 2 over a period of one day, one month?

1.5 Outline of The Document

- Chapter 1 i.e., this chapter describes the basic overview and background of the research stream considered i.e., cloud computing and virtualization. It also tells why this stream is considered along with the research questions, aims and objectives.

- Chapter 2 illustrates the technologies considered in this work. It describes about Openstack, Live Migration, PowerAPI, Optimization Models in detail.
- Chapter 3 brings up the state of the art i.e., a literature review of the related works done in the same stream considered.
- Chapter 4 describes the methodology chosen to implement and execute the thesis work.
- Chapter 5 shows all the results obtained while experimenting.
- Chapter 6 provides the overview of the analysis done after obtaining the results of the thesis work.
- Chapter 7 ends the thesis work by providing conclusions and by opening up the possible areas of future research work that can be done to further develop and motivate this thesis work.

2 A SKETCH OF THE TECHNOLOGY

This section gives a brief idea about the technological concepts used in this thesis work. The areas covered are as follows:

- OpenStack
- Live Migration
- PowerAPI
- Optimization Algorithms
    - Heuristic Algorithm

2.1 OpenStack

OpenStack is a free and Python-based open source software platform used in cloud computing. OpenStack is a tool used as a platform in IaaS. This software platform consists of components which are interrelated to each other controlling the hardware equipment underneath used for processing, storage and networking resources in a data center. These components can be removed, added or modified at any point of time without effecting the functioning of OpenStack. The users have the feature to operate OpenStack using web based dashboard, command line utilities (CLI) and through a RESTful API. OpenStack is simple to implement and it has great scalability. OpenStack was jointly started by Rackspace Hosting and NASA in 2010. OpenStack Foundation manages this software platform, which is a non-profit corporate entity established in September 2012 to promote this platform and its community. More than 500 companies joined this project [14].

OpenStack has several services embedded in the project, where each service is managed as a separate project and has its own team to manage and develop the respective service to achieve great scalability and flexibility in operation. OpenStack is powered by three major projects namely Compute, Object Storage and Compute+ Object Storage. OpenStack can be used with one service mentioned above or all the services at a time [15]. See **Figure 6** for a basic understanding of OpenStack and its functionality.

Openstack can be deployed on physical hardware as well as using virtual machines itself. The technical specifications required to run an Openstack environment is pretty high as normal domestic laptops and desktops cannot be used to run the traditional Openstack environment, although several tools and software's such as devstack have been developed. Devstack can be installed on any laptop with at least 2GB RAM and can be used to host 1 or 2 virtual machines and can experience how it would be to work with openstack as the dashboard is pretty same in both the environments. Mirantis is a software company which has gained immense popularity in recent times because of its tool Fuel which is a very simple and user friendly tool used to deploy an Openstack environment in simple steps.

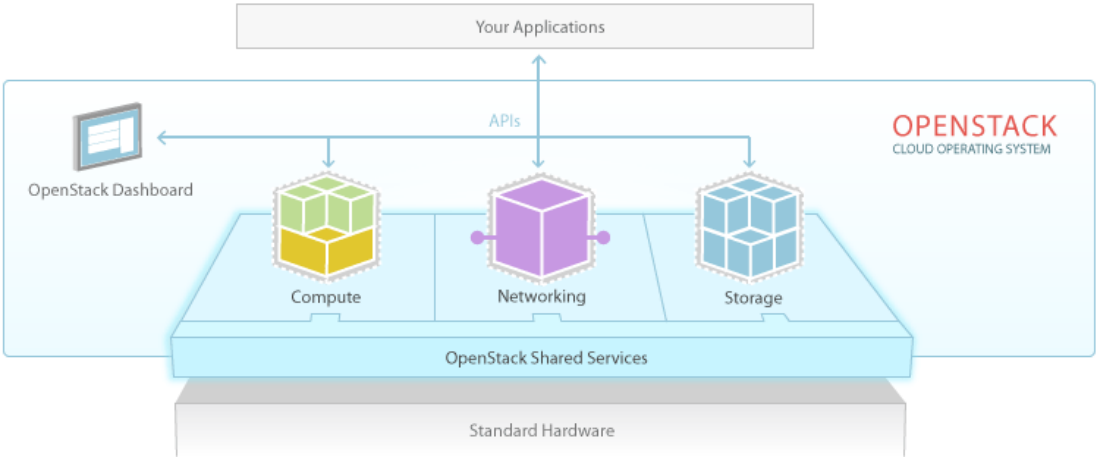


Figure 6: OpenStack Software Diagram [15]

OpenStack has additional core services different from the services mentioned above and are inbuilt in them. See **Figure 7** for OpenStack's conceptual architecture [16]. The core services which are offered by OpenStack with their project names are as follows:

- Swift (Object Storage)
- Keystone (Identity)
- Nova (Compute)
- Neutron (Networking)
- Cinder (Block Storage)
- Glance (Image Service)

OpenStack has few other optional services which can be used to enhance the cloud operating systems performance and experience. They are as follows:

- Horizon (Dashboard)
- Ceilometer (Telemetry)
- Heat (Orchestration)
- Trove (Database)
- Sahara (Elastic Map Reduce)
- Ironic (Bare-Metal Provisioning)
- Zaqar (Messaging Service)
- Manila (Shared Filesystems)
- Designate (DNS Service)
- Barbican (Key Management)
- Magnum (Containers)
- Murano (Application Catalog)
- Congress (Governance)

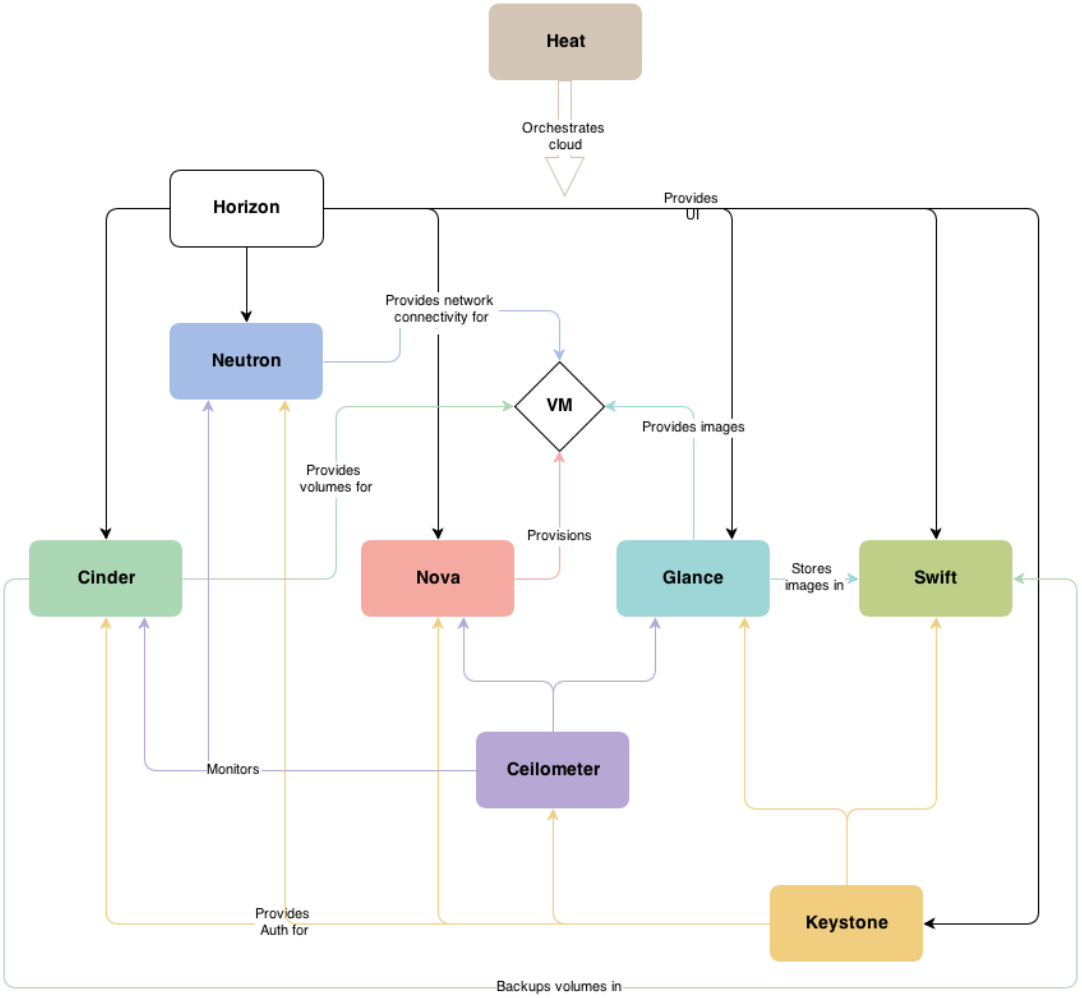


Figure 7: OpenStack Conceptual Architecture [16]

Brief discussion about the major projects involved in OpenStack is as follows [16]:

2.1.1 Swift (Object Storage)

Swift also known as OpenStack Object Storage is a highly available service and is used as a distributed storage. It allows its users to store and retrieve files. This service can be used by the Cinder component to back up the VM volumes [17]. Swift stores and retrieves arbitrary data objects via a RESTful API and HTTP based API. It is fault tolerant with its data replication and scale out architecture [16].

2.1.2 Keystone (Identity)

Keystone is responsible for authentication and identifying its users. It uses token system to differentiate between its users. Authentication is done by using the correct tokens issued by this service. Tokens are used in the form of URLs and a URL with the right token gets the access as the identification process is completed by using the right token for accessing a particular service. It provides a catalog of endpoints for all the Openstack services [16].

2.1.3 Nova (Compute)

Nova is the most important service among all the other services as it is responsible for the core operation of Openstack i.e., spawning, scheduling and decommissioning of virtual machines on demand. It also managed the lifecycle of compute instances in an Openstack environment [16]. It provides virtual servers on demand which interact with hypervisors such as KVM, Xen, VMWare or Hyper-V etc. Openstack supports all the above mentioned hypervisors in it. It is responsible for providing services for cloud resource management through its APIs and is cable of orchestrating running instances, networks and access controls [17].

2.1.4 Neutron (Networking)

Another important service is neutron which is responsible for the overall network connectivity of the cloud operating system. All the virtual machines hosted in the Openstack environment need to be connected to the virtual network which is established by Neutron. Neutron provides a cloud computing service to Openstack i.e., Network-Connectivity-as-a-Service for all the other services of Ppenstack. Neutron is very flexible and user friendly as it provides the feature to assign networks (internal and floating) and routers (internal and external) by the user. All the popular networking vendors and technologies are supported by having this service in Openstack [16].

Neutron provides capabilities for managing DHCP, static IPs and even has the feature to enable VLANs along with other advanced policies and topologies. Openstack has the feature to install plugins as well. Firewall-as-a-Service, VPN-as-a-Service etc. can be installed in an Openstack environment and Neutron plays crucial role in setting up these plugins. These plugins are to be installed manually by the users on demand and when required. Plugins are installed from source code just like the way a software package is installed on a device from source code [17].

2.1.5 Cinder (Block Storage)

This service provides block storage to the instances running in the Openstack environment. Creating and managing of the block storage devices are easy and simple to manage by the user. This service can be used to back up the volumes attached to virtual machines in the Openstack environment. Cinder will interact with Nova to provide the block storage i.e., volumes to the virtual machines running. Volume types can be assigned manually and manipulation of the volumes can be done freely as the volumes can be backed up in the form of snapshots which can later be booted up if needed [16] [17].

2.1.6 Glance (Image Service)

This service is responsible for hosting necessary cloud images which are uploaded by the user in the Openstack environment either by using the dashboard or by using command line utilities. These cloud images are later used by nova

compute to initiate virtual machine launch and thereby provisioning this image to the virtual machine [16].

2.1.7 Horizon (Dashboard)

Horizon is the service responsible for the Openstack dashboard. The dashboard is very user friendly as all the operations which Openstack is entitled to, can be operated using the dashboard. Horizon is generally accessed by an IP address which is allotted from the IP range while setting up the Openstack environment. Using the dashboard, user can do the basic operations such as launching an instance, assigning internal as well as floating IP addresses, configuring access controls, attaching volumes, taking snapshots, and many more operations [16].

Horizon can be used to create and manage several projects in an Openstack environment and has the feature to assign users to various projects and an administrator has this privilege to control how its users can work with the Openstack environment. The administrator is responsible for creating tenant IDs and provisioning them with the project the user can work in.

2.2 Migration of Virtual Machines

A virtual machine monitor also known as hypervisor is responsible for managing the interaction between the virtual machine and the underlying hardware resources on the platform. A hypervisor runs directly on the hardware just like an operating system. A hypervisor is generally operating system independent. Hypervisor can support the operation of different guest operating system images known as virtual machine images. The number of virtual machine images hosted on the hypervisor depends upon the capacity of the underlying hardware resources. In cloud data centers, where there are several applications running which provide various services to its users, these applications are stored in virtual machines which are hosted by servers present. A specific application is hosted in a virtual machine whereas the database management system can be present in another virtual machine. These virtual machines are connected to each other to send and retrieve necessary data from each other [18].

Migration of a virtual machine is transferring a virtual machine from one hypervisor to other hypervisor i.e., from one physical machine to other physical machine. The physical machines can be referred to as source machine and destination machine. See **Figure 8**, the source and destination are two physical machines which can host virtual machines in it. By using live migration, one or more virtual machines can be live migrated to other physical machine i.e., source to destination.

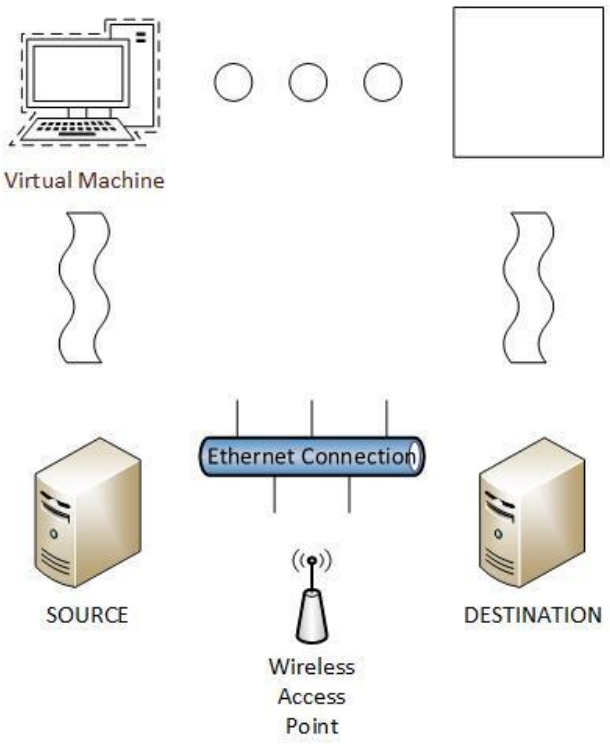


Figure 8: Overview of VM Migration

Virtual machine migration is achieved by using various migration techniques and types. Virtual machine manager supports network migration: an entire network can be migrated from one location to the other which has several machines interconnected, storage area network (SAN) migration: the data is stored in a network of storage devices, quick storage migration etc [19].

Virtual machine migration can be categorized as follows:

- Cold Migration: In this type of migration, the virtual machine which is subjected to migration, is shut off on the physical machine and it is migrated through the medium where the source and destination physical machines are connected, and after successful reaching on the destination machine, the virtual machine is switched on.

- Warm Migration: In this type of migration, the virtual machine which is subjected to migration, is suspended in the state which it is in, and it is migrated through the connection medium where the source and destination physical machines are connected, and after successful reaching on the destination machine, the virtual machine is brought to its earlier state by switching it on. In this case, a copy of the RAM and CPU are taken and sent to the destination which is used to restore its state back.

- Live Migration: In this type of migration, the virtual machine is not switched on or suspended for noticeable period but it is halted or paused for a very short time depending on the round trip time RTT between the hosts, but keeps running while the migration is done in the background from source machine to the physical machine without effecting the performance of the virtual machine and thereby avoiding

service downtime and discrepancy. This type of migration is achieved by copying across RAM and sending it to the destination and dirty RAM pages are marked and re-copied later.

2.2.1 Live Migration

Migration of virtual machines across different physical machines allows a clean separation between hardware and software, and it facilitates fault management, load balancing, low-level system maintenance and dynamic resource management. By carrying out the majority of migration while the operating system of the virtual machine is continuously running leads to impressive performance with a very minimum service downtime and discrepancies. Hence live migration is a key concept in day to day data centers to effectively monitor the resources efficiently in terms of service quality [20]. See **Figure 9**, it has two physical machines with virtual instances in it, and the figure says the migration if these virtual instances from one machine to the other.

CPU state, memory state, network state and disk state are transferred from source machine to a destination machine in live migration process. Out of the mentioned states, memory state transfer is of greater importance as the sensitive part of a device i.e., memory which has data stored in it, is to be transferred with utmost care and concern.

Memory state transfer is categorized in three phases [21]:

- Push Phase: In this phase, memory pages are transferred or pushed to the destination machine in an iteration fashion, while the VM is running on the source machine. Memory pages modified during this process are referred as dirty pages. These dirty pages are re-sent in the next iteration to ensure all of the memory is transferred.

- Stop-and-Copy-Phase: In this phase, the VM is stopped at the source machine, as soon as it is done, remaining memory pages are transferred to the destination machine and into the VM thereafter VM is started.

- Pull Phase: In this phase, if a memory page is not transferred from the source, a page fault is generated and this page is pulled from the source machine through the network connection.

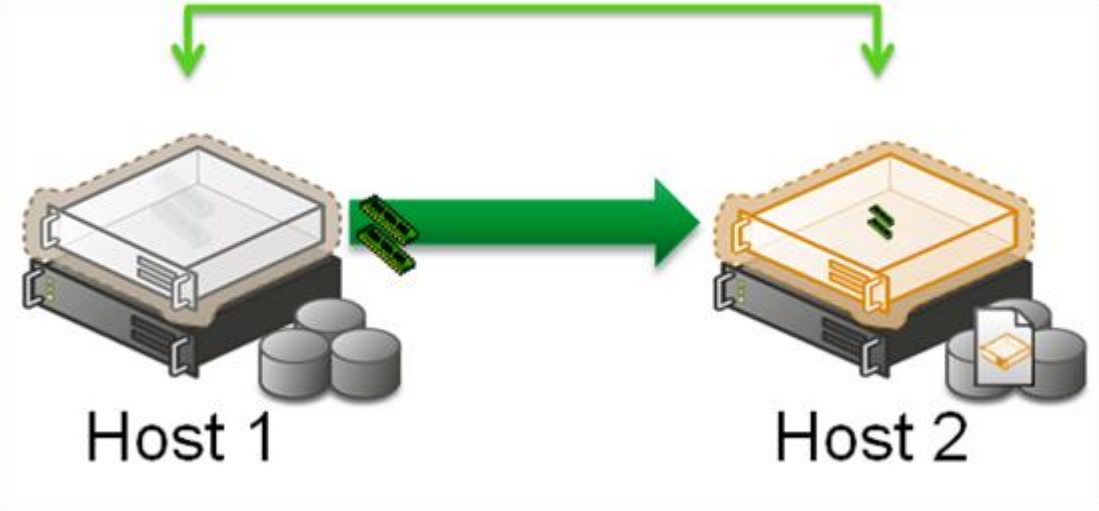


Figure 9: Example of Live Migration

2.2.1.1 Techniques of Live Migration

There are two main migration techniques used, which are a combination of the phases mentioned earlier. They are as follows [21]:

- Pre-Copy
  - This technique is the combination of push phase and stop-and-copy phase. Firstly, a dirty page tracker is enabled at the source machine. This tracker keeps note of all the dirty pages occurred in the iterations. In the first iteration, all memory pages are transferred to the destination machine.
  - In the meantime, dirty pages are identified and transferred in the next iteration to make sure the VM does not disrupt its services. After all the memory pages are transferred, the VM is suspended and the VM state is transferred to the destination and resumed.
  - Pros:
    - Does not require fasts interconnection between physical hosts.
    - Memory is transferred in succession of iterations until the remaining dirty memory can be transferred, this does not prolong the VM downtime.
  - Cons:
    - If the host dies, the VMs data cannot be recovered.

- Post-Copy
  - This technique is the combination of stop-and-copy phase and pull phase for live migrating a VM. Firstly, the VM is suspended and the minimal state of VM is transferred to the destination machine thereby resuming the VM.
  - Now the VM starts fetching the remaining memory pages from the source machine and hence referred to as pull phase.
  - Pros:

        - This method reduces the downtime and total migration time.
        - Data is transferred in the background or on demand making it flexible method of live migration
    - Cons:
        - This method incurs service degradation due to page faults which must be resolved over the network by the source host.

2.2.1.2 Properties of Live Migration [22]

- Service continuity: the services hosted in the virtual machines must be running all the time, which means its users will be accessing the data all the time and hence the service must be continuous in nature and live migration plays a crucial role in having this feature enabled all the time, which means using live migration the service continuity can be achieved as physical machines needs to be taken down during maintenance period.
- Low resource usage: live migration uses memory on both the source and destination machines and this memory usage is very minimal when compared to the virtual machines memory or the physical machines memory.
- Robustness: the source machine, destination machine or the virtual machine should not crash is the ideal functionality of a system. With live migration in place, we can achieve this by choosing suitable hosts for placing the virtual machines.
- Predictability: the time required to live migrate a virtual machine from source to destination can be predictable depending on the size of the virtual machine and the source, destination machines.
- Transparency: live migration of virtual machines need not be aware of the operating systems of both the source and destination and also the virtual machine. Live migration is possible without this data.

2.2.2 Live Migration in OpenStack

Openstack is an IaaS platform which can be used to implement multiple features of cloud computing such as hosting virtual instances, store data, network services etc. Live Migration is one such feature, which is used to migrate instances between compute nodes. In traditional networks, migration of virtual machines is done across physical machines whereas in Openstack, compute nodes host the instances. Openstack being user friendly, live migration can be done in various ways i.e., by using command line utilities, API and web dashboard. Non-live migration can also be done in openstack.

2.2.2.1 Non-Live Migration

In non-live migration, a particular instance is shut off and later migrated to another compute node. Non-live migration can be achieved using dashboard and command line utilities. In the dashboard, i.e., horizon, under the instances tab,

list of all instances are present, and at the end of a particular instance column, a drop down box is present wherein there is an option named migrate instance, which initiates the migration process across compute nodes. This feature migrates to a compute node which the nova scheduler finds on its own depending on the underlying policies and restrictions. Using the command line utilities, *nova migrate* command is used to migrate the instance.

nova list command lists all the virtual machines running in the openstack environment. A particular virtual machine is considered, its ID is known by executing the command *nova show VM_ID*. The ID of the instance which is to be migrated is noted down and used in the next command which migrates the instance. *nova migrate VM_ID* command when executed, migrates the particular virtual machine automatically [23].

2.2.2.2 Live Migration

Live migration is also referred to as seamless live migration where the instances offer no downtime of the services they host and thereby avoiding service discrepancies. Openstack has inbuilt live migration utilities in it. There are several types of live migration incorporated with openstack. They are as follows [21]:

- Shared Storage-Based Live Migration:
  - In this type, both the hypervisors have access to a shared storage configured to operate to support this scenario.
  - Several databases can be used as these shared storages. Few of them are GlusterFS, MySQL etc.
- Block Live Migration:
  - In this type, no shared storage is required, as the disk state is also migrated along with the virtual machine.

- Volume-Backed Live Migration:
  - In this type as well, no shared storage is required, as the virtual machines are backed by volumes which are migrated along with the virtual machine.

2.3 PowerAPI

PowerAPI is an open-source project developed by the Spirals research group (University of Lille and Inria) and the project is fully managed using *sbt*, an interactive build tool. PowerAPI is a middleware toolkit used to build software-defined power meters. These power meters are configurable software libraries which are used to estimate the power consumption of a software in real time scenarios. PowerAPI also supports the addition of raw metrics from a wide range of sensors such as physical meters, processor interfaces, hardware counters, OS counters etc. The power consumption estimates can be obtained in various forms such as file system, networks, web, graphical etc. It offers the capability of assembling power meters to accommodate user requirements [24].

Energy efficient computing is becoming increasingly important. Due to the increase in consumption of resources of the data center, energy efficiency is becoming a prime technological challenge. For a data center, it is very important to know which applications are consuming more energy thereby bringing up the demand for energy measuring devices, which increase the maintenance costs and investments. Hence, software tools which has the same capability to meter the energy consumption are in great demand. In virtual environments, one does not have direct access to the physical CPUs and one can observe only the behavior of a virtual machine. Hence BitWatts, a software tool is built which is an extension to the PowerAPI project mentioned above. BitWatts provides process level power estimation in virtual machine bases systems [25]. See **Figure 10** for basic understanding of the BitWatts functionality. Here, each virtual instance and hosts have BitWatts installed in it, which estimates respective power consumption, these values are used to convert into energy consumption.

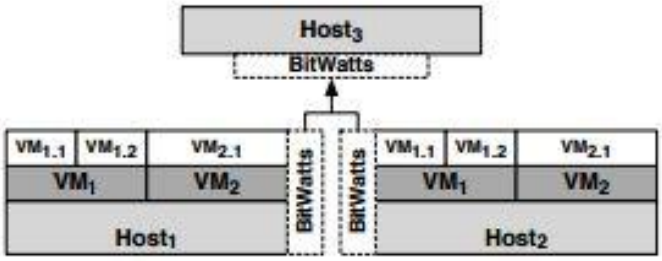


Figure 10: Example of BitWatts Functionality [25]

2.4 Optimization Algorithms

In mathematics and computer science, an algorithm is a self-contained step by step set of operations which are to be performed. Algorithms can perform calculation, data processing and automated reasoning. In general, an algorithm is a method to express in a finite amount of space and time and in a well-defined formal language used to calculate a particular function, for example, a software program is usually written in some random programming language which when executed does the necessary operations present in the code, here before the software code is developed a basic high level representation is prepared to understand the work flow and also to define what function is to be written and what that function should determine, this is referred to as an algorithm. Starting from an initial state and an initial input, the set of instructions describe a computation that when executed, proceeds through a finite number of well-defined successive states, eventually producing an output and terminating at a final ending state. Algorithms are essential to the way computers process data. Many computer programs contain algorithms that detail the specific instructions a computer should perform in a specific order to carry out a specified task. Hence algorithm can be considered to be any sequence of operations that can be simulated by a Turing-complete system [26].

For optimization problems there is a more specific classification of algorithms as follows [26]:

- Linear Programming Model: In this type of model, optimal solutions are found out which are bound to linear equality and inequality constraints. These constraints can be used directly to produce optimal solutions. Simplex algorithm, maximum flow problem, integer programming etc are few algorithms which fall under linear programming.

- Dynamic Programming: In this type of model, optimal solution of a system can be found out by evaluating the optimal solutions of each of the sub problems and overlapping sub problems. Floyd-Warshall algorithm is an example of dynamic programming.

- The Greedy method: This model is similar to that of dynamic programming, where examining the substructures is the method to find the optimal solution. Huffman Tree, Kruskal, Prim, Sollin are few greedy methods.

- The Heuristic method: This model is used to find the solution which is close to the optimal solution in cases where it is difficult to find the optimal solution or is impractical.

2.4.1 Heuristic Method

Heuristic method or technique, called as a heuristic, is an approach to solve problems that employs a practical method, that does not guarantee to be ideal or perfect, but it is sufficient enough to solve the problems. In general, when finding out an ideal solution is near to impossible or impractical, using a heuristic method can be used to speed up the process for finding a satisfactory solution. Few notable examples of heuristic are rule of thumb, educated guess, stereotyping, profiling or common sense [28].

In computer science, artificial intelligence and mathematical optimization, a heuristic is a technique designed for solving a problem quickly, when classic methods fail to find an exact solution. Heuristic methods can be referred to as shortcuts to find out solutions to problems which are close to the ideal solution. A heuristic has certain branches to follow at each step, this gives an approximate to the exact solution. Few notable examples are the Simpler problem, Traveling Salesman problem, Virus scanning etc [29]. In general, a heuristic algorithm consists of few set of input values, which are given to the algorithm. Algorithm consists of several steps, and is executed one step at a time, by making use of the given input values and based on the operations in the algorithm, outputs are generated.

3 STATE OF THE ART

This section explains the related works done in the following areas which are used in this thesis work.

- Live migration of virtual machines.
- Resource management in a data center.
- Optimizing energy consumption in a data center.

3.1 Live Migration of Virtual Machines

Live migration of virtual machines ensures significant benefits for virtual machines mobility with minimal service disruption. However, the cost required to migrate a virtual machine is different for different workloads. Authors of [30] investigate this aspect and experimented to predict the performance and energy consumed while migrating a virtual machine from one server to the other.

The authors have designed application models to predict the cost by making use of the knowledge about the workloads at the hypervisor level. Their results showed a 90% prediction accuracy in comparison with the actual cost measurements [30].

After the virtualization has evolved into a significant and a must have technology in a data center, high availability(HA) of the services hosted using this technology is demanded by the users. A high availability of a service means that, it must be up all the times. These services or machines should have a near zero down time. Authors of [31] have conducted a survey of mechanisms which ensure a near zero down time. Replication, a method to host multiple copies of particular data has been introduced to ensure a near zero down time and several such replication methods have been developed.

In this paper, authors have compared the performance of all these mechanisms and concluded that, migration mechanisms or replication mechanisms based on memory, are efficient and faster when compared to the other mechanisms such as, process migration, suspend/resume migrations [31].

Automating a process without manual interruption is a measure to ensure speed processing in a particular system. Authors of [32] have developed such system where the live migration of virtual machines is possible without manually triggering the process or any such method where manual intervention is needed The authors have developed algorithms to achieve the goal, which are push strategy, an algorithm where in the virtual machines are pushed from the source machine and pull strategy, where in the virtual machines are pulled from the destination machine. The performance of these algorithms were framed using simulations.

Their results proved to be complementary in both the cases depending on the workload considered. The pull strategy is found to be fast in the case where the workload is in the range between low and medium and the push strategy is found to be fast in the case where the workload is in the range between medium and high [32].

3.2 Resource Management in Data Centers

Resource management is one of the most important aspects in establishing and running a data center. Proper usage and provisioning of the available resources can lead to better system management which in turn avoids any breakdown of the system. Authors of [33] focused on different virtual machine migration techniques and correlated these techniques to manage the resources dynamically.

They have discussed the role of live migration in resource management dynamically in particular, cloud systems. They have concluded that, live migration enabled several management goals such as consolidation, load balancing, hot spot mitigation [33].

In recent times, there is shift in data centers running on dedicated hardware to a cloud based systems, the difference is that, in dedicated hardware systems, all the services are hosted in a set of machines in a data center, where as in cloud systems, the serices are hosted in collection of machines which are spread across multiple data centers. This change is due to the advantages of cloud computing, which provides flexibility, scalability and security aspects. The authors of [34] have derived a mechanism to the use the resources dynamically in a cloud data center without wasting the resources. The research area is focused on CPU, memory, hard disk usage of the data center. Their mechanism is properly demonstrated using an algorithm based on bin packing and best fit.

The results of their work shows that, utilizing a particular host to its maximum capacity produced better results. Their aim is to have less number of tasks migrated with maximum allocation of resources. This task was achieved by server consolidation, which is the offloading of a particular host that is under-utilized [34].

A cloud system ensures the provisioning of resources whenever demand arises and whenever needed. It ensures scalability in the infrastructure and the present resources to be used based on incoming demand. This measure enables enterprises to scale their resources up and down. Effective use of these resources can lead to smooth functioning of the system and there by eradicating certain performance degrading aspects such as slow run of systems, down time, hot spots etc. The authors of [35] focused mainly on allocation of virtual machines to users based on the characteristics of the task which the virtual machine is to perform.

The main aim of this work is to avoid execution of low priority tasks over high priority tasks and to allot the resources dynamically based on the task priority. To achieve this aim, the authors have developed an algorithm which executes the tasks based on their priority and there by allocating resources dynamically which support this measure [35].

3.3 Optimizing Energy Consumption in Data Centers

In large data centers, significant amount of total energy consumption in spent on cooling. The temperature of hardware devices increases, when there is an increase in the workload processed by a specific device. This temperature needs to be reduced in order to safe guard the device from permanent damage. In order to achieve this goal, several air conditioning appliances are used which are made for this purpose. The authors of [36] proposed a scheme, which is based on combination of limiting the temperature of the processors by using a technique called Dynamic Voltage and Frequency Scaling (DVFS) and frequency-aware load balancing, thereby reducing the energy consumption used by the air conditioners to cool down the devices physically. This measure also prevents hot spot mitigation. Their scheme is primarily based upon the ability to control processors' temperature, minimizing the timing penalty and cooling energy savings [36]. Their scheme is designed as an algorithm and is implemented on hardware dedicated for this purpose.

The results proved that, hot spots were neutralized and core temperatures of the processors were constrained. The algorithm was able to reduce the total energy consumption by 28% by paying a timing penalty of 11%. Broad scope for future work was created within this experiment [36].

In large data centers, in a user perspective, it is optimal to have the system avoid violating the performance constraints of the applications hosted. In an administrator's perspective, it is optimal to minimize the power consumption without introducing any service disruptions to the applications hosted. The authors of [37] proposed a runtime virtual machine mapping framework in the data center to save the energy/power consumption. Their main objective was to host the applications without violating its performance constraints and also to minimize the power consumption of the data center. One probabilistic and one heuristic algorithm were developed as part of their work and each algorithm has its own way of functioning, on comparing both of them heuristic algorithm came out to work better [37].

The results obtained shows that, there was a significant amount of power consumption saved when compared to the original system. They could save up to 69.2% over base with a few performance penalties. The effectiveness and performance insights are also analytically verified [37].

As of 2016, energy consumption by data centers worldwide were estimated to be 15% of the global electricity use and is expected to be higher. Hence there is a great demand to reduce the usage of resources without compromising the service quality. The author of [38] presented novel techniques, models, algorithms and software for distributed dynamic consolidation of virtual machines in cloud data centers, all of them majorly developed in the form algorithms, which are in turn implemented in OpenStack framework. The major goal is to improve the utilization of computing resources and reduce the energy consumption under workload independent quality of service constraints. This project was majorly implemented on OpenStack [38].

4 METHODOLOGY

This chapter describes the methodology followed to accomplish the goals this thesis work is entitled to, which are:

1. Literature review of the related works in the same area to gain knowledge and information on how energy consumption can be optimized in data centers.
2. Design a heuristic algorithm with all the steps involved in running a data center right from launching of instances and terminating them.
3. Simulate the algorithm to know the data flow involved in the algorithm and to validate all the steps involved in the algorithm.
4. The heuristic algorithm design is transformed into a python code.
5. Twoimplementation phases are considered as follows:
    a. 2 Compute nodes & 1 Controller-Cinder node
    b. 4 Compute nodes & 2 Controller-Cinder nodes
6. In each of the implementation phase, the same experiment is conducted twice as mentioned below,
    a. CleanOpenstack
        i. VMs are launched until the environment cannot host more VMs using the nova-scheduler.
        ii. Each VM is embedded with a script running in the background that imposes load on the CPU periodically, and repeats the process continuously.
        iii. After the full capacity of the environment is reached, termination of VMs is done one by one.
        iv. In parallel, the energy consumption of all the hosts are measured and stored.
    b. CleanOpenstack with algorithm
        i. The same set of steps mentioned above, are repeated but with algorithm placing the VMs, re-arranging VMs in between depending upon their performance.
        ii. After the process is done, the energy consumption values in both the cases are compared.

4.1 Experiment Model

There are several methods defined to study or evaluate a particular system and to know in depth functionality, relation between components and analyze its performance. The selection of method depends upon the outcomes, available resources and the desired accuracy level. In this thesis work, combination of two such methods are considered i.e., simulation and physical model of the actual system [21].

4.2 Heuristic Algorithm Design

A simple heuristic algorithm is designed in expectation to optimize the energy consumption of a data center. The algorithm majorly focusses on the hosts which have the VMs launched over it. The algorithm comes into play whenever

a VM is to be launched and when the VMs are to be re arranged. An overview of the algorithm is as follows:

- When a new VM is to be launched, the algorithm finds the best host to place this VM.
- The algorithm re-arranges VMs whenever a possibility of re-arrangement occurs.

4.2.1 Algorithm

The heuristic algorithm design is a combination of two algorithms which operate in synchronization with each other. They are as follows:

1. Spawning a new VM algorithm.
2. Re-arranging VMs algorithm.

4.2.1.1 Spawning a New VM

Whenever a new VM is to be spawned, this part of the algorithm does the operations as follows:

1. The algorithm retrieves the current cumulative energy consumption, available resources such as number of CPU cores, RAM, Disk storage of all the hosts, and virtually arranges them as a list in decreasing order.
    a. Hosts: Host 1 > Host 2 > Host 3
2. The algorithm retrieves the VMs requirements such as number of cpu cores required, RAM and disk storage, and virtually arranges them as a list in increasing order. For example:
    a. VMs: VM1 < VM2 < VM3
3. First VM in the list is chosen, the requirements of VM and current availability of the first host in the list is matched. If there is enough space to launch this VM, it is launched in that particular host. If not, next host in the list is chosen and same operation is done.

4.2.1.1.1 Motivation

If we execute this algorithm, all the hosts in the data center are filled up with VMs as far as there is enough space in it. This way we can use the hosts as much as possible and we can avoid using a host which is under-utilized. This method can enable the option to put off hosts into sleep mode or to switch it off, which saves lot of energy.

4.2.1.1.2 Input Functions

The algorithm is given with some input functions and values to retrieve data from. These input functions, values are used directly in the algorithm to trigger the launching VM operation in the data center.

Input functions given to the algorithm are as follows:

1. Current energy consumption, CPU cores, RAM, Disk storage of each host (running), assuming ‘n’ total hosts:
    - $H1 = H_{1EC}, H_{1Cores}, H_{1RAM}, H_{1Disk}$

- H2 = H_{2EC}, H_{2Cores}, H_{2RAM}, H_{2Disk}
- H3 = H_{3EC}, H_{3Cores}, H_{3RAM}, H_{3Disk}
- ….
- Hn = H_{nEC}, H_{nCores}, H_{nRAM}, H_{nDisk}
- All.Hosts() = (H1, H2, H3, …., Hn)

2. Required parameters of VMs which are to be launched, assuming 'm' VMs are to be launched (m ≠ 0):
   - V1 = V_{1Cores}, V_{1RAM}, V_{1Disk}
   - V2 = V_{2Cores}, V_{2RAM}, V_{2Disk}
   - V3 = V_{3Cores}, V_{3RAM}, V_{3Disk}
   - ….
   - Vm = V_{mCores}, V_{mRAM}, V_{mDisk}
   - All.VMs() = (V1, V2, V3, …., Vm)

4.2.1.1.3 Spawning Algorithm

Algorithm

Require: All.Hosts(), All.VMs()
Input: H1,..Hn; V1,..Vn;

```
1   for All.Hosts() do
2   .  H.Dec("H1, .., Hn") ← Sort hosts in Decreasing order as per EC
3   .  All.VMs() ← List of VMs to be launched
4   .  for All.VMs() do
5   .  .        V.Inc("V1, .., Vn") ← Sort VMs in Increasing Order
6   .  end for
7   .  foreach V.Inc() do
8   .  .        foreach H.Dec() do
9   .  .        .        if H1canhostV1 do
10  .  .        .        .        Launch V1 in H1
11  .  .        .                 else
12  .  .        .        .        .        exit loop
13  .  .        .        .        .        next Host;
14  .  .        .        end if
15  .  .        .        Update H.Dec()
16  .  .        end foreach
17  .  .        Update V.Inc()
18  .  end foreach
19  end for
```

4.2.1.1.4 Operations

Spawning algorithm is triggered whenever a request for launching a new VM occurs. The algorithm performs the following operations whenever such a request occurs:

1. The algorithm retrieves current energy consumption, available number of CPU cores, RAM and Disk storage of all the hosts and virtually arranges the host in decreasing order of their energy consumption.
2. The algorithm retrieves the VM flavor details i.e., number of CPU cores required, RAM and Disk storage required and virtually arranges the host in increasing order as per the number of CPU cores required.
3. The algorithm choses first VM from the list and first host from the list and see if the host can place the VM in it. If the parameters required by the VM are available at the host, the VM is placed successfully, if the host does not have enough area, the next host from the list is chosen and placed.
4. After each VM placement, the lists are updated according to their current usage.

4.2.1.2 Re-Arrangement of VMs

In a data center, launching and termination of virtual machines is a common process and happens very frequently. It is important to manage these operations as the hosts might turn into idle state which consume some energy to keep the system in running state. This way the resources are wasted and proper management is required to overcome this phase. Hence, re-arrangement of VMs is necessary to save the energy consumption and to avoid unnecessary wastage of resources. Hence the goal of this algorithm is to push as many hosts as possible into sleep mode and thus using less number of hosts.

The heuristic algorithm developed for this purpose does the operations as follows:

1. The algorithm retrieves the current cumulative energy consumption of all the hosts and virtually arranges the hosts in two distinct groups.
    a. Group – I: Hosts with Energy consumption above 50%
    b. Group – II: Hosts with Energy consumption below 50%
2. Group – I hosts are sorted in decreasing order as per their consumption and group – II hosts are sorted in increasing order as per their consumption.
3. First host present in both groups are considered and VMs of the host of group – II are live migrated to the first host of group – I.

4.2.1.2.1 Motivation

In data centers, several hundreds of hosts are used to host various virtual machines and applications running in it with variable work load. During a period of time, the virtual machines might be terminated or shut off depending the usage of it by its users. Hence it is important for administrators to dynamically allocate and distribute the resources in order to save energy. This can be achieved by pushing the hosts into sleep mode or shut it off, to save energy and by utilizing the available resources to the fullest.

4.2.1.2.2 Input Functions

The algorithm is given with some input functions and values to retrieve data from. These input functions, values are used directly in the algorithm to trigger the re-arrangement of VMs in the data center.

The input functions, values given to the algorithm are as follows:

1. Current energy consumption, CPU cores, RAM, disk storage of each host (running), assuming 'n' total hosts:
    a. $H1 = H_{1EC}, H_{1Cores}, H_{1RAM}, H_{1Disk}$
    b. $H2 = H_{2EC}, H_{2Cores}, H_{2RAM}, H_{2Disk}$
    c. $H3 = H_{3EC}, H_{3Cores}, H_{3RAM}, H_{3Disk}$
    d. ….
    e. $Hn = H_{nEC}, H_{nCores}, H_{nRAM}, H_{nDisk}$
    f. All.Hosts() = (H1, H2, H3, …., Hn)
2. Number of CPU cores, RAM, disk storage of each VM running in each host of group II.
    a. Host – I = {
        $V_{1Cores}, V_{1RAM}, V_{1Disk}$
        $V_{2Cores}, V_{2RAM}, V_{2Disk}$
        …..
        $V_{mCores}, V_{mRAM}, V_{mDisk}$
        }
    b. Host – II = {
        $V_{1Cores}, V_{1RAM}, V_{1Disk}$
        $V_{2Cores}, V_{2RAM}, V_{2Disk}$
        …..
        $V_{mCores}, V_{mRAM}, V_{mDisk}$
        }
    c. …..
    d. Host – n = {
        $V_{1Cores}, V_{1RAM}, V_{1Disk}$
        $V_{2Cores}, V_{2RAM}, V_{2Disk}$
        …..
        $V_{mCores}, V_{mRAM}, V_{mDisk}$
        }
    e. Hosts.AllVMs() = (Host-I, Host-II, …., Host-n)

4.2.1.2.3 Re-arrangement Algorithm

Algorithm

Require: All.Hosts(), Hosts.AllVMs
Input: H1, H2, H3, …., Hn; Host-I, Host-II, Host-III, …., Host-n;

```
1  for All.Hosts() do
2  .        H.All() ← List of all hosts and respective current usage
3  .        for H.All() do
4  .        .        H.Group1 ← List of Hosts above 50% (↓)
5  .        .        H.Group2 ← List of Hosts below 50% (↑)
6  .        end for
7  .        foreach H.Group2 do
8  .        .        VM.Host($_) ← Arrange VMs in ↑ order
9  .        end for
10 .        foreach Hosts.AllVMs do
11 .        .        foreach H.Group-II do
12 .        .        .        foreach H.Group-I do
13 .        .        .        if H(x)canhostV(y) do
14 .        .        .        .        LiveMigrate V(y)
15 .        .        .        .        else
16 .        .        .        .        Migrate to H(x+1)
17 .        .        .        .        Update H.Group-I
18 .        .        .        end if
19 .        .        .        end foreach
20 .        .        Push Host($_) into Sleep Mode
21 .        .        end for
22 .        .        Update Hosts.AllVMs
23 .        end foreach
24 end for
```

4.2.1.2.4 Operations

The re-arrangement algorithm is triggered periodically over a period of time. The algorithm virtually arranges the hosts into two groups according to their current energy consumption values and the following operations are performed:

1. Group 1: Hosts with energy consumption above 50% of their capacity, are arranged in decreasing order and Group 2: Hosts with energy consumption below 50% of their capacity, are arranged in increasing order.
2. The VMs of group 2 hosts are virtually arranged in increasing order in respective hosts. Virtually arranged is nothing but the algorithm is aware of the virtual machines parameters and values and it is arranged in the in-memory in increasing order.
3. The VMs of each host are live migrated to the hosts of group 1.

a. Required parameters of first VM of first host is matched with that of available parameters of first host of group 1.
b. If the host have enough space, the VM is live migrated. Otherwise, next host in the list is chosen and same operation is repeated.

4. This process goes until all the VMs of the host are live migrated and the host is put into sleep mode.
5. If the hosts of group-1 does not have enough space at all, the algorithm halts, and re checks after one minute and does the same operation.
6. The lists of both groups are updated every one minute and the algorithm automatically triggers every 5 minutes and does necessary operations wherever required. These respective time periods are considered because workload generation is varied every one minute hence the list of groups are updated every one minute as well and the algorithm is triggered in that fashion because there is a time gap for 5 minutes between a virtual machine being launched and terminated hence the algorithm is triggered at that point of time will yield better results.

4.3 Simulation

This thesis work is implemented using two evaluation models i.e., simulation model and implementing on actual physical model. As part of the validation of the algorithm design, simulation of the python code is done. In the simulation, we tested the functionality of the program used for the experimentation on real time scenarios and machines. Using the simulation method, the following aspects are studied:

- Performance of the python code when different types of inputs are given to python code such as:
  - Number of Virtual Machines to be spawned with different required parameters (RAM, CPU cores, Disk storage).
  - Number of running hosts to accommodate incoming virtual machine spawn requests.
  - Termination of virtual machines running.
- To check the data flow through the components of the python code such as
  - Verifying the re-arrangement of VMs among hosts
  - Validating the algorithm operations such as virtually arranging the hosts in respective order based upon their energy consumption.
- Verifying the VM live migration operation triggers based upon the logic

After the execution, the simulation program yielded expected results that is the data flow and the triggering operations were functioning as expected and proof of this is that the program displays outputs on the terminal periodically and based on the data displayed and time synchronization we concluded that the program was functioning as expected.

4.4 Workload Generation

In any data center, be it a cloud or dedicated hardware setup, there are several hundreds of hosts running which are built and configured to run many number of instances. These instances are entitled to work in a way the administrator wants it to run. The administrator sets up the instance in a way their respective users are willing to use at an acceptable quality of service and level. As the services hosted run in their respective way, it is ideal to assume that hosts and respective instances run with variable workload all the time throughout the year depending upon the number of users accessing particular applications or services.

In this thesis work, a similar scenario is considered. All the instances spawned are embedded with certain python scripts which are developed using a benchmarking tool developed to test the performance of CPU under variable load and stress called *stress-ng*, which impose variable workload on the CPU and run continuously with variable workload. The workload imposed here is restricted to only CPU workload. Key aspects of the workload generator script are as follows:

- CPU is given with number of operations depending upon the number of CPU cores it contains.
    - For example, if the instance is having 2 CPU cores, 2 instances of the workload generator script are launched.
    - Each instance of the workload generator script, imposes variable workload in synchronization with each other.
- The workload generator script, computes Fibonacci series.
- The tool used to run this operation is *stress-ng*
    - *stress-ng --cpu <number of cores> --cpu-method fibonacci*
    - By executing the above command, the CPU is given with instances equal to its number of cores to compute fibonacci series.
    - This method imposes load on CPU continuously as the system is kept at 100% CPU utilization constantly throughout the period and the load gradually increases with time. When the system reaches its maximum load level (Load average nearly equals its number of CPU cores value), the process is halted until the system reaches its 0 load level. After that, the same process is initiated again.
- Using this method, the following aspects are achieved:
    - Instance is embedded with continuous workload varying with time similar to any instance in a typical data center.
    - A real time scenario like test case is achieved.

4.5 Comparison Method

To test an algorithm and to compare the performance and efficiency of it, we need a standard method or a benchmark method. In this thesis work, the algorithm is compared with the algorithm embedded with *nova-scheduler*, which is the main component in launching instances in Openstack. In general, the default method used to launch instances is done by the *nova-scheduler*. The compute node utilizes the *nova-scheduler* to determine how to dispatch compute

requests. For example, *nova-scheduler* determines on which host an instance is to be launched [39].

nova-scheduler uses default configurations and steps to follow, unless until it is changed. In this thesis work, the default *nova-scheduler* is used and compared. Basically *nova-scheduler* can be configured by editing the *nova.conf* file located in the *etc* directory in the path */etc/nova/*. The default scheduler used by *nova-scheduler* is the *filter scheduler*, which supports filtering and weighting based on weight of the instances, this weight is calculated using a formula which applies for the complete system, for example, free RAM is considered to be one of the parameter and the value is multiplied with a standard constant and all the values are compared and are separated according to the outcome of the value. There are several filtering mechanisms based upon certain parameters like disk, cores, RAM etc, supported by *nova-scheduler* few of them are as follows:

- Aggregate Core Filter
- Aggregate Disk Filter
- All Hosts Filter

In data centers, all the hardware resources are stacked in server rooms and a typical server room consists of different number of servers depending on the size of the data center. In this thesis work, a minimized replica of each data center is considered and a similar working atmosphere is chosen. Every server has instances launched and each instance has a lifetime and there is rate at which a new instance is to be spawned and there is a specific period of time in a day when the availability of resources is minimal, i.e., 0 availability, this assumption is considered because in any sized data center, there is a particular point of time when the system is 100% or atleast near to 100% utilized and any more servers/virtual machines cannot be launched, in order to have this property as well this type of setup is considered. In this thesis work the following assumptions are made:

- Rate of new instance request is 300 seconds
- Lifetime of an instance is 3600 seconds
- Non availability of resources is 900 seconds
- Small, medium and large sized flavors are chosen in a sequence to all the instances launched. Small sized flavor is a type where the virtual machine has 1 vCPU, 20 GB disk & 2 GB RAM. Medium sized flavor is a type where the virtual machine has 2 vCPUs, 40 GB disk & 4 GB RAM and the large sized flavor is a type where the virtual machine has 4 vCPUs, 80 GB disk & 8 GB RAM [40].

4.6 Implementation Phases

Two experiment phases are considered for evaluating this thesis work because, to test an algorithm and to evaluate its performance, the test bed needs to scaled up i.e., number of compute hosts are increased. In each scenario the system is vertically scaled up and the phases are horizontally scaled. The performance of the algorithm can be best found out if it is implemented on hardware which has the flexibility to scale up the infrastructure. In real time scenarios such as data centers, the resources are scaled up depending upon the

demand and need. Hence it is crucial to evaluate the performance of an algorithm implemented on such a scalable hardware hence openstack is chosen which has the option to scale up easily. The following phases are considered:

1. Two Compute hosts – One Controller - Cinder host
    - Two hosts with Compute role
    - One host with Controller and Cinder roles
2. Four Compute hosts – Two Controller - Cinder hosts
    - Four hosts with Compute roles
    - Two hosts with Controller and Cinder roles

4.6.1 System

In an Openstack environment, a minimum of one compute host and one controller host are mandatory to have the system running. The functionality of the system (launching instances, storage, network, CPU) is shared among services hosted by these components. Compute is responsible for spawning, scheduling and termination of instances. It interacts with hypervisors and does the operations. Controller node on the other hand, is responsible for identity, dashboard, authentication and the whole functionality of the system and communications among all the services. Cinder is focused majorly on storage purpose as it enables block storage for the instances launched.

4.6.2 Test Bed

This section explains briefly how the hardware system is designed and how the algorithm is implemented in the physical model i.e., hardware. Spawning instances, migrating instances and termination of instances is done in compute nodes. The algorithm design which is transformed into a python code, is executed by the controller. Controller host retrieves required information/data from compute hosts, instances and triggers the required operations to be performed such as spawning, re-arranging instances by live migration and termination of instances whenever a request occurs. The design of the setup is explained in the following sections.

4.6.2.1 2 Compute – 1 Controller & Cinder

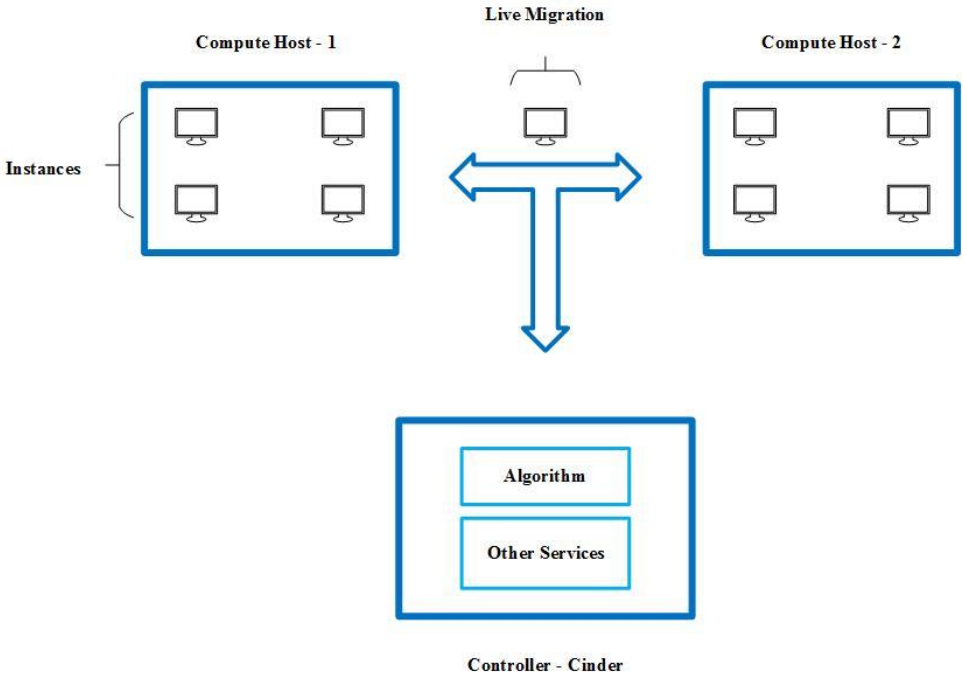


Figure 11: Architecture of Setup - 1

First case of experiment is done on an environment setup as shown in **Figure 11**. Two compute hosts communicate with each other whenever an instance is to be migrated among them. The spawning, re-arranging and termination operations are triggered from the controller-cinder host. Storage to the instances launched in the system is provided by the cinder from the controller-cinder host.

4.6.2.2 4 Compute – 2 Controller & Cinder

Second case of experiment is done on an environment setup. This scenario is considered to see how the system behaves and to study the consumption when the system is scaled. In this scenario, four compute hosts are used to host the instances. Compute hosts communicate with each other whenever an instance is to be migrated among two hosts. Spawning, re-arranging and termination operations are triggered from the controller-cinder host. One controller host is chosen to execute the algorithm. Two controller-cinder hosts are considered to overcome the redundancy factory in the system. Four compute hosts are to be handled by the controller-cinder hosts hence to overcome the redundancy factory, two controller hosts are embedded to the system i.e., to avoid system to function slow with lags.

4.7 Evaluation Procedure

To evaluate the performance and working of the algorithm, in each of the test bed cases, the following steps are done:

1. Spawn instances using *nova-scheduler* until the system cannot host any more instances.
    a. Instances are backed with volumes, which means that the virtual machines data is written on disk and this disk is a volume here and the instances use the RAM to save the in-memory.
2. Launched instances are embedded with workload generator scripts which are executed repeatedly until they are terminated.
3. The system is left running after it is filled completely with instances.
4. The instances are terminated one by one after their lifetime is reached.
5. In parallel, PowerAPI is executed on all the hosts which continuously calculate the power consumption from the beginning until all the instances are terminated. These values are later converted into energy consumption.
6. The above steps are repeated again in the test bed, now with the optimization algorithm in place. Algorithm is given with inputs which launch the instances and terminates the instances when their lifetime is reached and in parallel, re-arrangements are done.
7. In parallel, PowerAPI is executed on all the hosts which continuously calculate the power consumption from the beginning until all the instances are terminated. These values are later converted into energy consumption.

4.8 Test Bed Specifications

This section gives the specifications of the hardware used in the test bed. All the servers used are Dell PowerEdge servers.

4.8.1 Case – I

This section gives the specifications of the hardware used in evaluating the algorithm in case – I (2 Compute – 1 Controller & Cinder) as shown in **Table 1**.

Specification	Compute Host-1	Compute Host-2	Controller-Cinder
Server	PowerEdge R415	PowerEdge R415	PowerEdge R415
CPU	8 X 3.10 GHz	6 X 3.10 GHz	6 X 3.10 GHz
CPU Cores	8	6	6
RAM	32 GB	32 GB	32 GB
Disk Storage	0.9 TB	0.9 TB	0.9 TB
Interfaces	2 X 1 Gbps	2 X 1 Gbps	2 X 1 Gbps

Table 1: Specification of Testbed – 1

4.8.2 Case – II

This section gives the specifications of the hardware used in evaluating the algorithm in case – II (4 Compute – 2 Controller & Cinder) as shown in **Table 2**.

Specification	Compute Host-1,2	Compute Host-3,4	Controller-Cinder-1,2
Server	PowerEdge R415	PowerEdge R415	PowerEdge R220
CPU	8 X 3.10 GHz	6 X 3.10 GHz	4 X 3.10 GHz
CPU Cores	8	6	4
RAM	32 GB	32 GB	16 GB
Disk Storage	0.9 TB	0.9 TB	0.9 TB
Interfaces	2 X 1 Gbps	2 X 1 Gbps	2 X 1 Gbps

Table 2: Specification of Testbed - II

4.9 Performance Metrics

This section explains the metrics considered to evaluate the performance of the algorithm. Energy consumption of the system, CPU load of all the hosts and CPU utilization of all the hosts are compared. Energy consumption values are obtained by converting power consumption values given by PowerAPI, CPU load values are fetched from */proc/loadavg* file and CPU utilization values are fetched from the output of *top* command. Here, power consumption is the amount of work done by the resource, and energy consumption is the amount of work done over a period of time. CPU utilization is the percentage of the CPU used by the instance and load is exponentially damped value of number of cores used by the instance. In both the evaluation methods, background scripts are used to run and retrieve these values and are stored in respective text files.

5 RESULTS & ANALYSIS

This chapter describes all the results obtained and brief analysis is provided based on the obtained results. This chapter is organized in two sub sections respective to the implementation phases considered as follows:

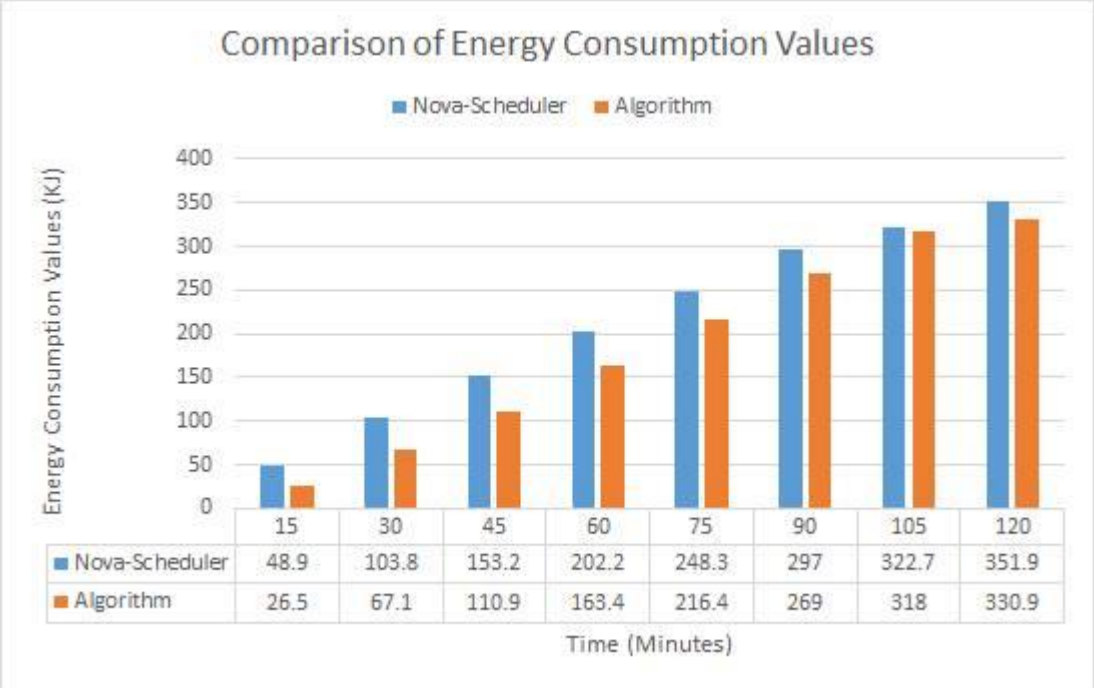
- 2 Compute hosts – 1 Controller & Cinder host
- 4 Compute hosts – 2 Controller & Cinder hosts

5.1 Scenario – I

This section focuses on the results obtained by implementing the methods of evaluation in the first test bed scenario i.e., Openstack environment with 2 compute hosts and 1 controller-cinder host. Analysis of the results obtained is described at the end of this section.

5.1.1 Results

The cumulative energy consumption of the system is shown in below figures and tables. The system with nova-scheduler and the algorithm are compared and individual trends are displayed.



	15	30	45	60	75	90	105	120
Nova-Scheduler	48.9	103.8	153.2	202.2	248.3	297	322.7	351.9
Algorithm	26.5	67.1	110.9	163.4	216.4	269	318	330.9

Figure 12: Comparison of Energy Consumption

Time (Minutes)	Energy in KJ (nova-scheduler)	Energy in KJ (algorithm)
15	48.9	26.5
30	103.8	67.1
45	153.2	110.9
60	202.2	163.4
75	248.3	216.4
90	297	269
105	322.7	318
120	351.9	330.9

Table 3: Comparison of Values

5.1.2 Analysis

Energy consumption for both the systems are compared And the results says that the energy consumption of the system running using the algorithm, is less when compared to that of the energy consumption of the system running using nova-scheduler.

CPU utilization and load levels of the system are observed to be similar in trends with the algorithm in place consuming slightly more CPU as shown in **Figure 13**. The figure displays the utilization values over the experiment duration.

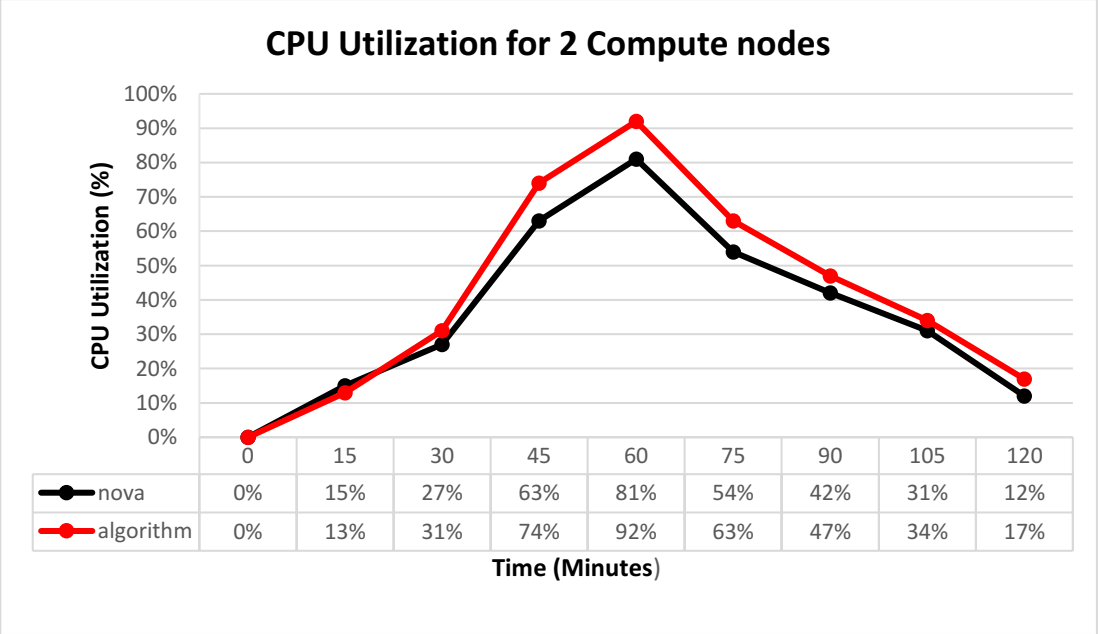


Figure 13: CPU Utilization

5.1.2.1 Energy & Cost Saving

This section provides an estimate of the total energy saved and equivalent money saved over a period of time. In **Figure 14**, the amount of energy saved is displayed and this values are obtained from subtracting the respective energy consumptions of the system with algorithm from the system with nova-scheduler. **Figure 15** gives us the estimate of amount of energy saved in one day by taking the values obtained in the experimentation as the practical values.

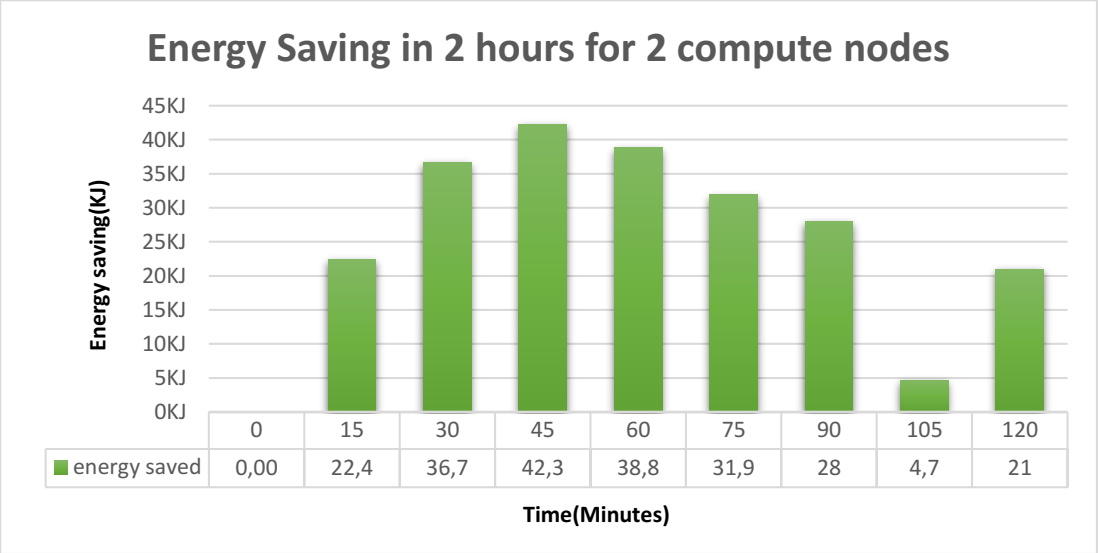


Figure 14: Energy Saving for 2 hours

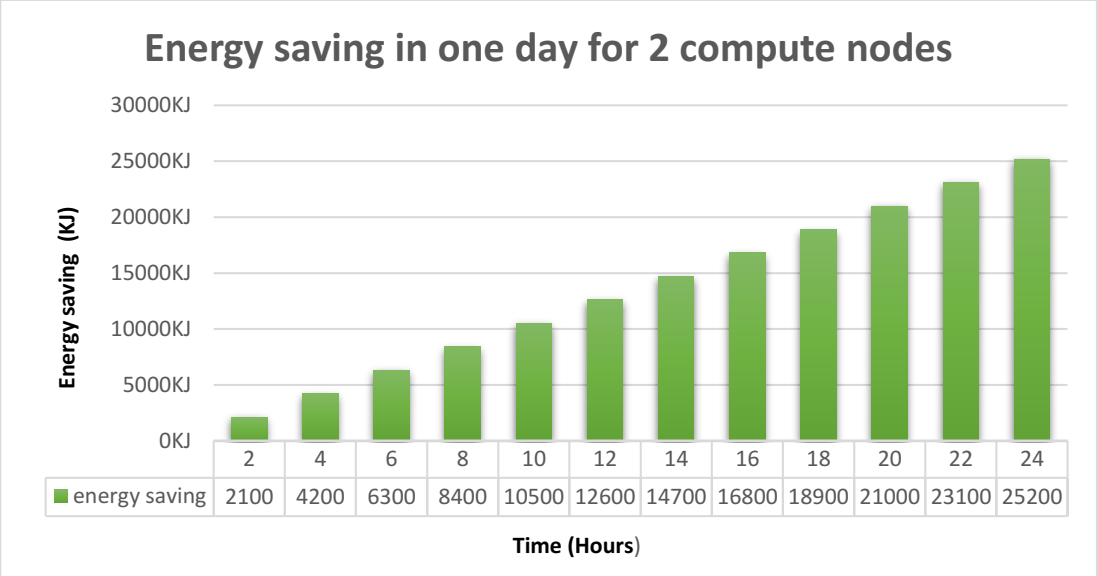


Figure 15: Estimate of Energy Saved in One Day

In 2016, the cost of energy consumption in the commercial sector as per the US government is 10.35 cents per every 1 KWh (Kilo Watt Hour), and 1 KWh Is equivalent to 3600 KJ. This data is used to estimate the amount of money saved in this research. In **Figure 16**, the amount of money saved is calculated for a

system with 10 compute nodes in one day. **Figure 17** gives us the estimate of amount of money saved in one month by a system with compute nodes, assuming that the same trends observed in this research's experimentation are followed.

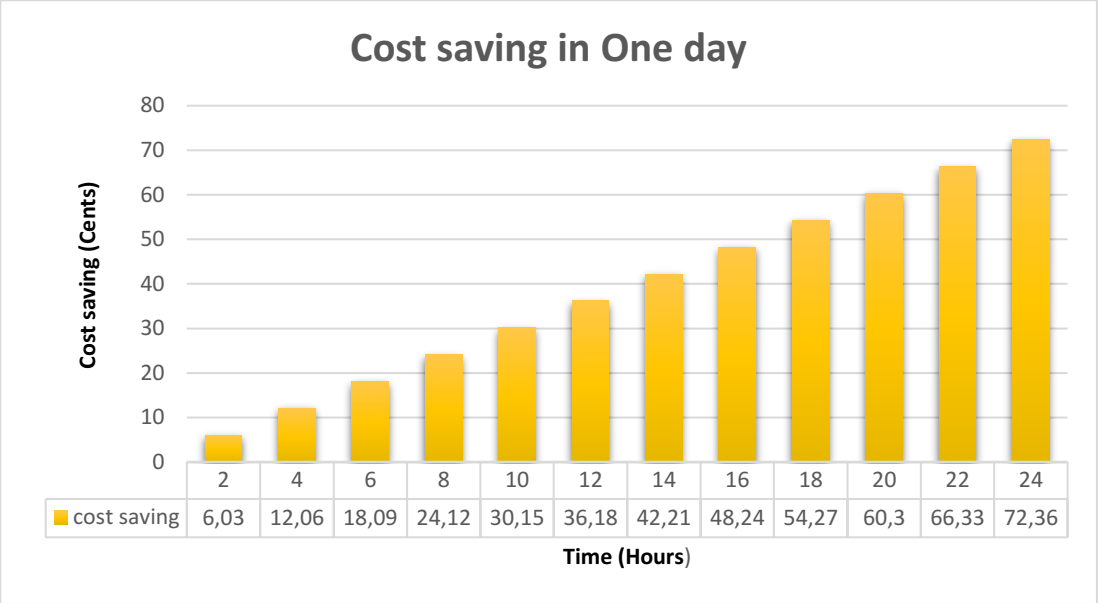


Figure 16: Estimate of Money Savings in One Day

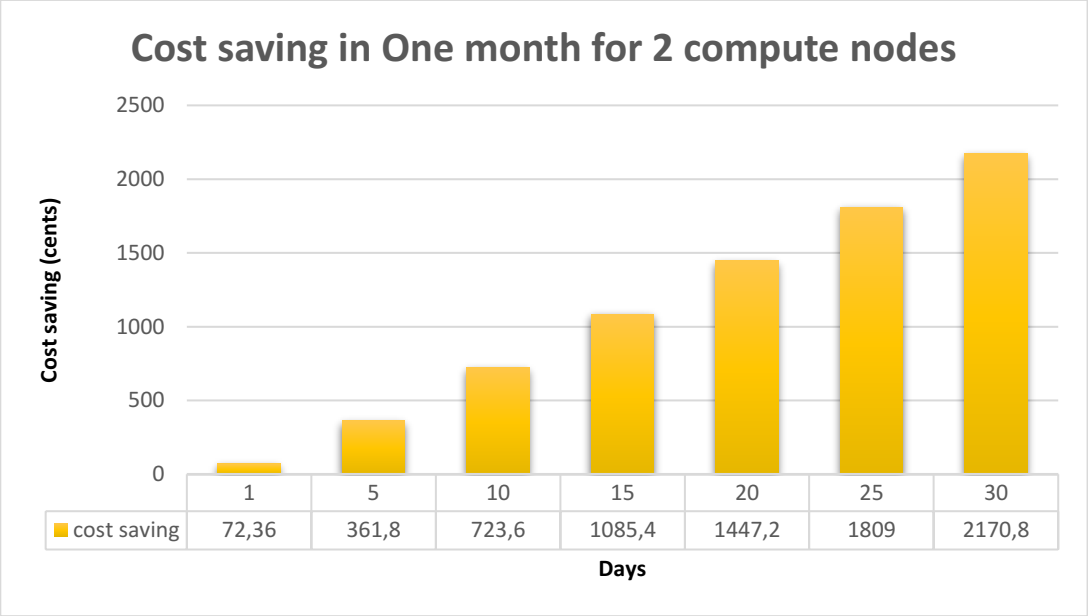


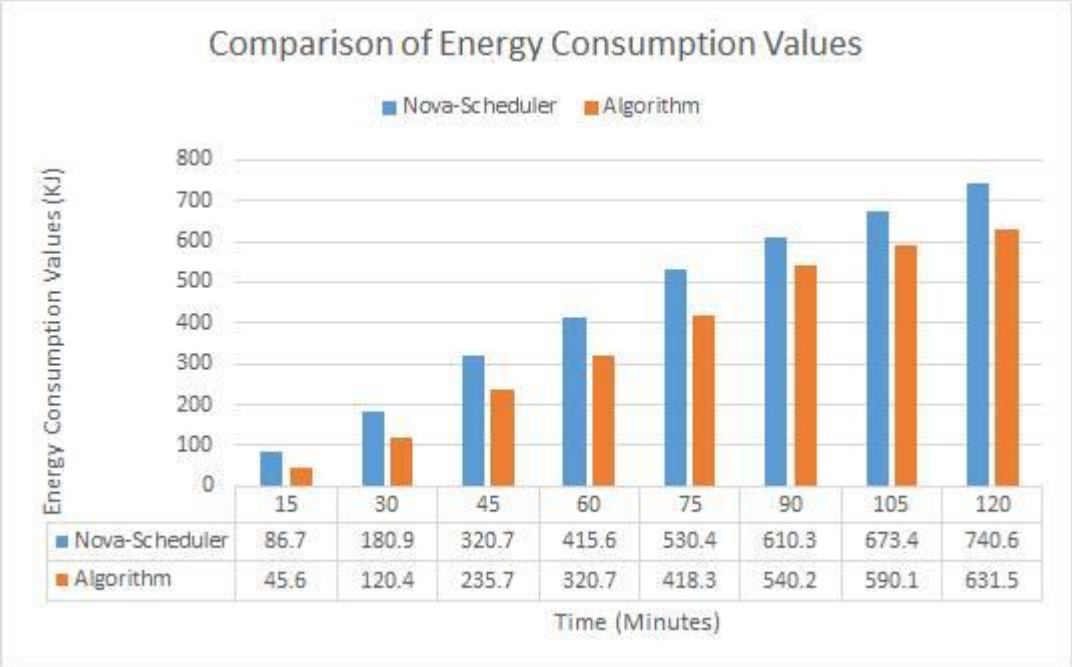
Figure 17: Estimate of Money Saved in One Month

5.2 Scenario – II

This section focuses on the results obtained by implementing the methods of evaluation in the second test bed scenario i.e., Openstack environment with 4 compute hosts and 2 controller-cinder hosts. Analysis of the results obtained is described at the end of this section.

5.2.1 Results

The energy consumption of the system is shown in below figures and tables. The system with nova-scheduler and the algorithm are compared and individual trends are displayed.



	15	30	45	60	75	90	105	120
Nova-Scheduler	86.7	180.9	320.7	415.6	530.4	610.3	673.4	740.6
Algorithm	45.6	120.4	235.7	320.7	418.3	540.2	590.1	631.5

Figure 18: Comparison of Energy Consumption

Time (Minutes)	Energy in KJ (nova-scheduler)	Energy in KJ (algorithm)
15	86.7	45.6
30	180.9	120.4
45	320.7	235.7
60	415.6	320.7
75	530.4	418.3
90	610.3	540.2
105	673.4	590.1
120	740.6	631.5

Table 4: Comparison of Energy Consumption values

5.2.2 Analysis

Energy consumption for both the systems are compared And the results says that the energy consumption of the system running using the algorithm, is less when compared to that of the energy consumption of the system running using nova-scheduler.

CPU utilization and load levels of the system are observed to similar in trends with the algorithm in place consuming slightly more CPU as shown in **Figure 19**.

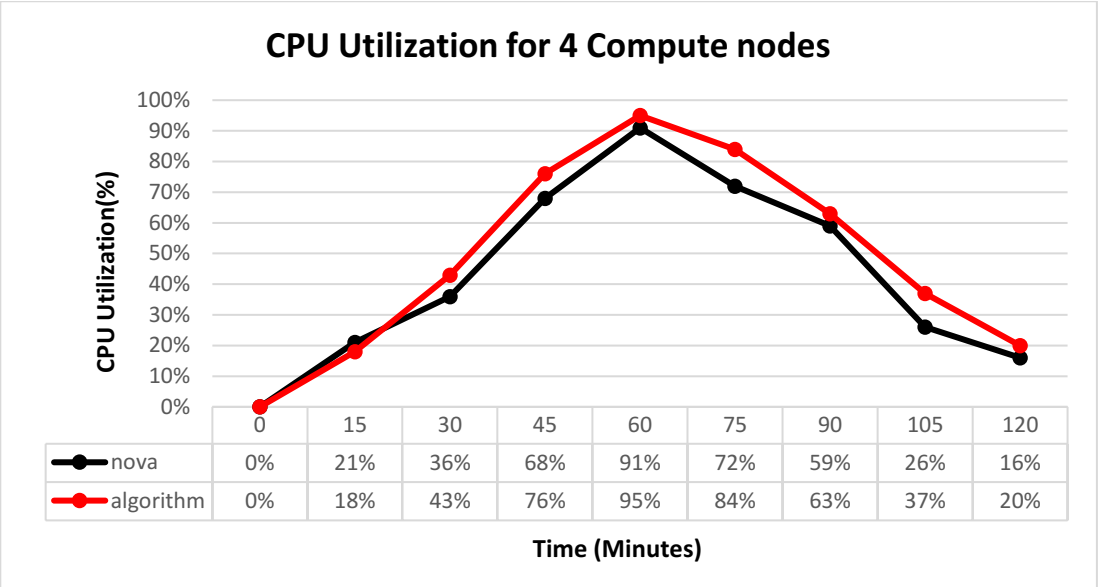


Figure 19: CPU Utilization

5.2.2.1 Energy & Cost Saving

This section provides an estimate of the total energy saved and equivalent money saved over a period of time. In **Figure 20**, the amount of energy saved is displayed and this values are obtained from subtracting the respective energy consumptions of the system with algorithm from the system with nova-scheduler. **Figure 21**gives us the estimate of amount of energy saved in one day by taking the values obtained in the experimentation as the practical values.

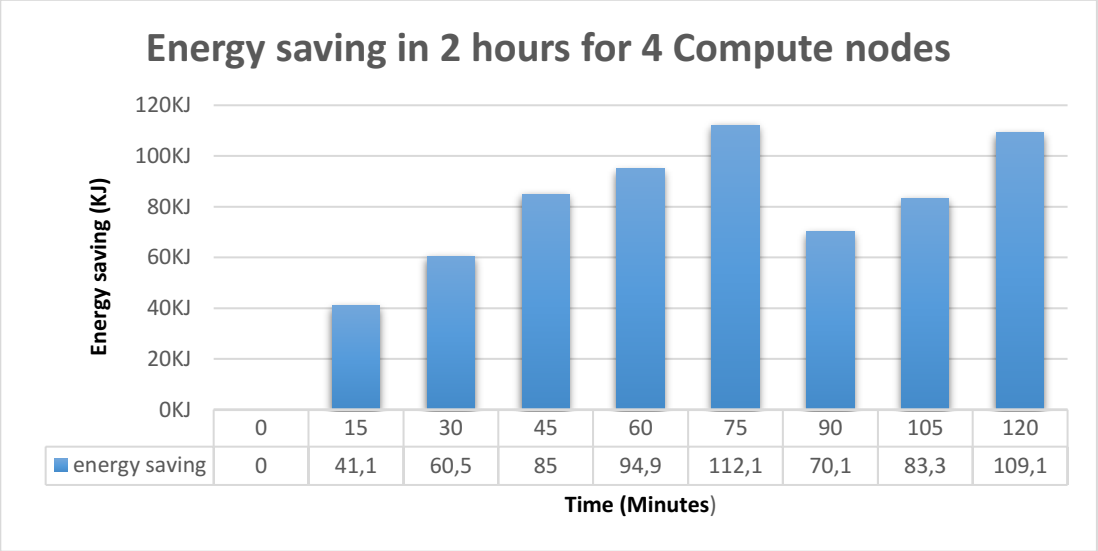


Figure 20: Energy Saving in 2 Hours

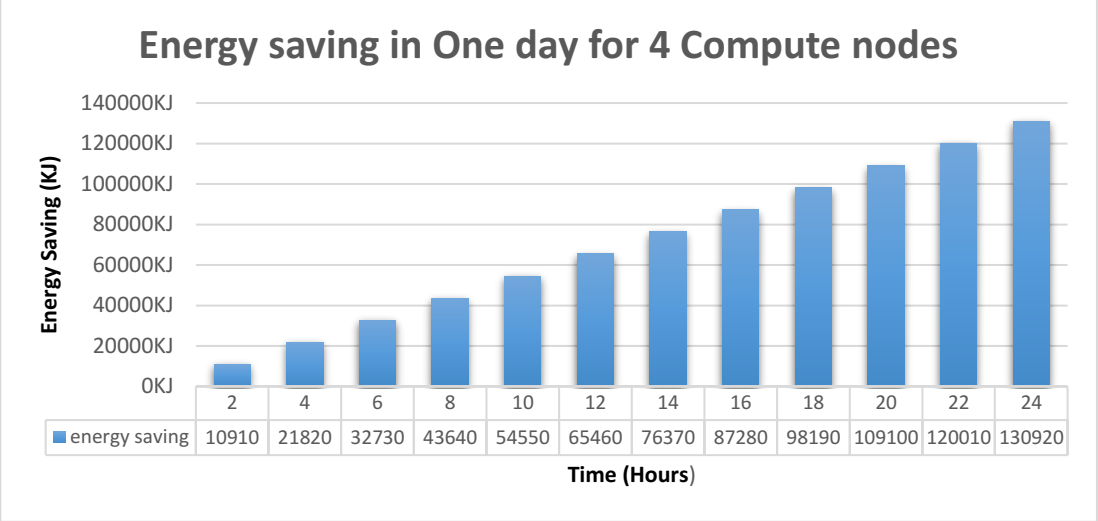


Figure 21: Estmate of Energy Saved in One Day

In **Figure 22**, the amount of money saved is calculated for a system with 10 compute nodes in one day. **Figure 23** gives us the estimate of amount of money saved in one month by a system with compute nodes, assuming that the same trends observed in this research's experimentation are followed.

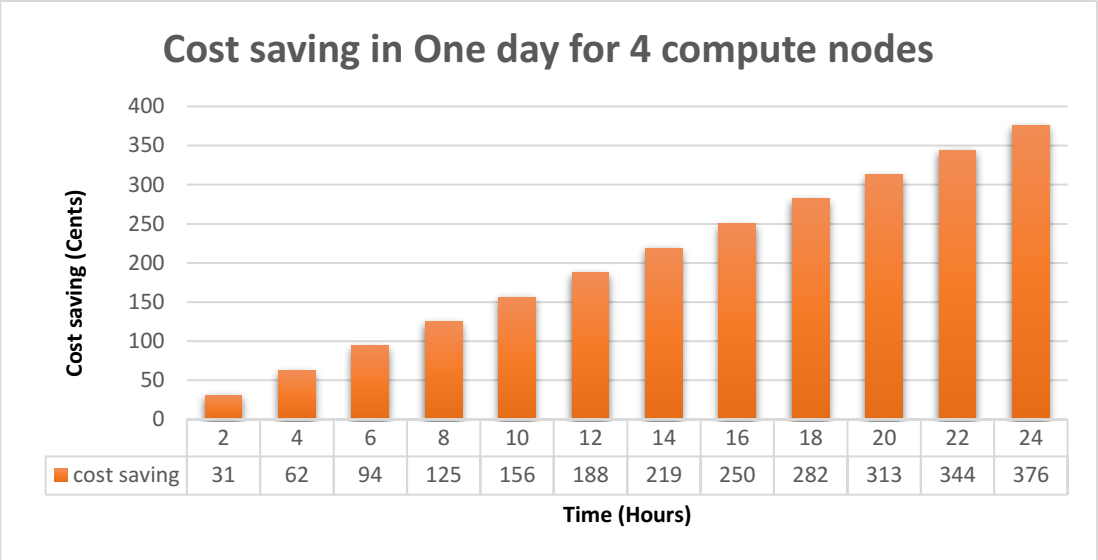


Figure 22: Estimate of Cost Saving in One Day

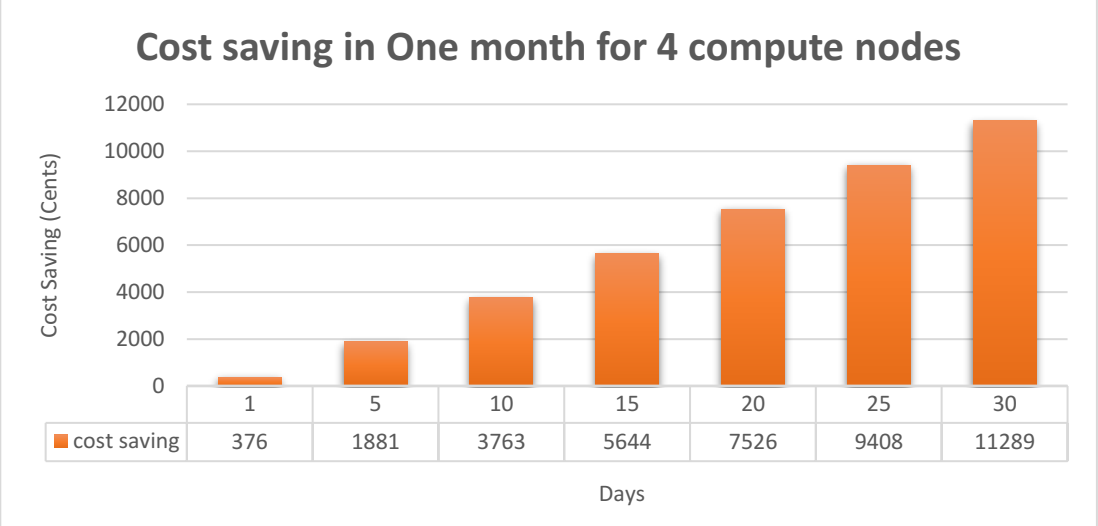


Figure 23: Estimate of Cost Saving in One Month

6 CONCLUSIONS

This research was done in order to find and evaluate if we can optimize the energy consumption in cloud data centers without effecting the performance in a noticeable fashion. The outcomes of this research were as expected. Energy can be optimized in cloud data centers with the use of live migration of virtual machines. Openstack, an open source cloud computing software is used as the cloud platform and a replica of a data center is considered as the experimental setup. In this research, we used heuristic algorithms to carry out experiments which was designed from a model.

In order to enable comparisons, the heuristic algorithm was embedded in the Openstack and the system let to run, and in the other experimental scenario, the system was let to run with nova-scheduler as the default component to launch and terminate instances.

To estimate the energy consumption of the system, we used BitWatts software tool, which is a software defined power metering tool developed by Spirals team. BitWatts is a tool developed on top of PowerAPI. This tool gives the estimate of power consumption for processes in the operating system. We use these power values to convert to energy consumption.

Major component of this research is the live migration of instances between compute nodes, and the heuristic algorithm developed as part of this research uses live migration as the main component or a triggering component to ensure the energy consumption is optimized.

The results proved that energy can be optimized in cloud data centers using live migration and also that the performance was not degraded in a significant way, this conclusion was driven from the CPU utilization values obtained.

6.1 Answer to Research Questions

RQ1: How should the relationship between workload and energy consumption be defined to enable comparisons in a heterogeneous hardware environment?

Answer: In a server or a machine, the work load performed by it is majorly depends upon the amount of work performed by its CPU. CPU being the central processing unit is the most important component of a server and the performance of the server depends upon the performance of the CPU. Here, the workload performed by CPU holds the major share or most of the percentage of the workload performed by the server/machine overall and most of the energy consumed by the server/machine overall is consumed by the CPU. In this research we have taken this parameter as one of the crucial requirements. In this section, **Heuristic Algorithm Design** we explained how we have taken the CPU and used it in this research and in this section **Results & Analysis**, we have put up our results.

RQ2: How should a heuristic model be defined to provide an efficient and simple method to optimize energy consumption in a data center?

Answer: In this research we have designed an experimental setup and experimental procedures by assuming a replica of an IT data center with a set of servers. In this section **Algorithm**, we have explained how the algorithm performs and what assumptions we have taken and the work flow of the system. We have taken the inputs and assumptions keeping in mind how a data center is operated in a daily basis i.e., number of virtual resources spawning periodically, terminating the virtual resources when its usage is completed and when the data center is operating in full capacity i.e., additional virtual resources cannot be spawned due to unavailability of hosting capacity.

RQ3: How much cost can be saved using the heuristic model defined in question 2 over a period of one day, one month?

Answer: We have designed an algorithm which uses live migration to study if we can optimize energy consumption and in turn saving money in long run. We were able to optimize the energy consumption and estimated the amount of money we can save using this algorithm in full scale system data centers. In this section **Energy & Cost Saving-1** & **Energy & Cost Saving-2**, we have estimated the amount of money we can save in one day and one month depending upon the amount of energy saved by using this algorithm.

6.2 Future Work

The following components can be studied as part of the future work:

- In this research, we used Openstack as the cloud platform. The same research can be carried out in other cloud platforms and see the performance of the algorithm. This is because the algorithm developed, uses few components of Openstack such as REST API, nova, cinder which are Openstack specific. Hence the performance can vary if we implement in different cloud platforms.
- We assumed that the research will be carried out in heterogeneous environment. However, we cut down on the number of hosts we use to 6GB & 8GB RAM hosts. We can implement on hosts with more RAM.
- The work load generator used in this research is a tool which imposes fake load on the system. However, we can carry the experiment by using real time load generating experiments, such as video streaming between server and client, file transfers, compute intensive loads etc. This enables use to make sure the system is just like a production system and hence a better solid conclusion can be driven.

REFERENCES

[1] A. Shawish and M. Salama, "Cloud Computing: Paradigms and Technologies," in *Inter-cooperative Collective Intelligence: Techniques and Applications*, F. Xhafa and N. Bessis, Eds. Springer Berlin Heidelberg, 2014, pp. 39–67.

[2] "Understanding the Cloud Computing Stack: SaaS, PaaS, IaaS." [Online]. Available: https://support.rackspace.com/white-paper/understanding-the-cloud-computing-stack-saas-paas-iaas/. [Accessed: 16-Apr-2016].

[3] K. V. K. M. Kumar, "Software As A Service For Efficient Cloud Computing." [Online]. Available: http://esatjournals.net/ijret/2014v03/i01/IJRET20140301028.pdf. [Accessed: 16-Apr-2016].

[4] "The Benefits and Drawbacks of SaaS models | Claims," *ClaimVantage*, 29-Jan-2014. .

[5] "Platform as a service," *Wikipedia, the free encyclopedia*. 14-Apr-2016.

[6] J. Gibson, R. Rondeau, D. Eveleigh, and Q. Tan, "Benefits and challenges of three cloud computing service models," in *2012 Fourth International Conference on Computational Aspects of Social Networks (CASoN)*, 2012, pp. 198–205.

[7] "Virtualization," *Wikipedia, the free encyclopedia*. 11-Feb-2016.

[8] R. Kumar and S. Charu, "An Importance of Using Virtualization Technology in Cloud Computing," *Glob. J. Comput. Technol.*, vol. 1, no. 2, pp. 56–60, Mar. 2015.

[9] K. tej Koganti, E. Patnala, S. S. Narasingu, and J.N. Chaitanya, "Virtualization Technology in Cloud Computing Environment," *Int. J. Emerg. Technol. Adv. Eng.*, vol. 3, no. 3, p. 3, Mar. 2013.

[10] A. Berl, E. Gelenbe, M. D. Girolamo, G. Giuliani, H. D. Meer, M. Q. Dang, and K. Pentikousis, "Energy-Efficient Cloud Computing," *Comput. J.*, vol. 53, no. 7, pp. 1045–1051, Sep. 2010.

[11] P. Association, "Greenhouse gas emissions rise at fastest rate for 30 years," *The Guardian*, 09-Sep-2014.

[12] keqiu L. Awada Uchechukwu, "Improving Cloud Computing Energy Efficiency," 2012.

[13] Josh Whitney and Pierre Delforge, "Data Center Efficiency Assessment." NRDC and Anthesis, Aug-2014.

[14] "OpenStack," *Wikipedia, the free encyclopedia*. 17-Apr-2016.

[15] "Software » OpenStack Open Source Cloud Computing Software." [Online]. Available: http://www.openstack.org/software/. [Accessed: 17-Apr-2016].

[16] "Chapter 1. Architecture - OpenStack Installation Guide for Ubuntu 14.04 - juno." [Online]. Available: http://docs.openstack.org/juno/install-guide/install/apt/content/ch_overview.html. [Accessed: 17-Apr-2016].

[17] T. Rosado and J. Bernardino, "An Overview of Openstack Architecture," in *Proceedings of the 18th International Database Engineering & Applications Symposium*, New York, NY, USA, 2014, pp. 366–367.

[18] A. Hayward and S. P. Mullen, "Migration of virtual machines," US8850430 B2, 30-Sep-2014.

[19] "Migrating Virtual Machines and Storage Overview." [Online]. Available: https://technet.microsoft.com/en-us/library/jj628158(v=sc.12).aspx. [Accessed: 17-Apr-2016].

[20] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, “Live Migration of Virtual Machines,” in *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, Berkeley, CA, USA, 2005, pp. 273–286.
[21] S. C. Pasumarthy, *Live Migration of Virtual Machines in the Cloud : An Investigation by Measurements*. 2015.
[22] Petter Svard, Steve Walsh abd Benoit Hudzia, and Johan Tordsson and Erik Elmroth, “The Noble Art of Live VM Migration - Principles and Performance of precopy, postcopy and hybrid migration of demanding workloads.”
[23] “OpenStack Docs: Migrate a single instance to another compute host.” [Online]. Available: http://docs.openstack.org/admin-guide/cli_nova_migrate.html. [Accessed: 18-Apr-2016].
[24] “PowerAPI.” [Online]. Available: http://www.powerapi.org/. [Accessed: 18-Apr-2016].
[25] M. Colmant, M. Kurpicz, P. Felber, L. Huertas, R. Rouvoy, and A. Sobe, “Process-level power estimation in VM-based systems,” 2015, pp. 1–14.
[26] “Algorithm,” *Wikipedia, the free encyclopedia*. 12-Apr-2016.
[27] “Linear programming,” *Wikipedia, the free encyclopedia*. 06-Apr-2016.
[28] “Heuristic,” *Wikipedia, the free encyclopedia*. 27-Jun-2016.
[29] “Heuristic (computer science),” *Wikipedia, the free encyclopedia*. 14-Jun-2016.
[30] H. Liu, C.-Z. Xu, H. Jin, J. Gong, and X. Liao, “Performance and Energy Modeling for Live Migration of Virtual Machines,” in *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, New York, NY, USA, 2011, pp. 171–182.
[31] V. Medina and J. M. García, “A Survey of Migration Mechanisms of Virtual Machines,” *ACM Comput Surv*, vol. 46, no. 3, pp. 30:1–30:33, Jan. 2014.
[32] M. Forsman, A. Glad, L. Lundberg, and D. Ilie, “Algorithms for automated live migration of virtual machines,” *J. Syst. Softw.*, vol. 101, pp. 110–126, Mar. 2015.
[33] M. Mishra, A. Das, P. Kulkarni, and A. Sahoo, “Dynamic resource management using virtual machine migrations,” *IEEE Commun. Mag.*, vol. 50, no. 9, pp. 34–40, Sep. 2012.
[34] A. Ngenzi, S. R, and S. R. Nair, “Dynamic resource management in Cloud datacenters for Server consolidation,” *ArXiv150500577 Cs*, May 2015.
[35] N. Nagaveni, R. Renuga, A. Kunthavai, M. Sangeetha, T. Ramraj, A. T. Saraswathi, Y. R. A. Kalaashri, and S. Padmavathi, “Graph Algorithms, High Performance Implementations and Its Applications (ICGHIA 2014)Dynamic Resource Allocation Scheme in Cloud Computing,” *Procedia Comput. Sci.*, vol. 47, pp. 30–36, Jan. 2015.
[36] O. Sarood, P. Miller, E. Totoni, and L. V. Kale, “‘Cool’ Load Balancing for High Performance Computing Data Centers,” *IEEE Trans Comput*, vol. 61, no. 12, pp. 1752–1764, Dec. 2012.
[37] X. Liao, H. Jin, and H. Liu, “Towards a Green Cluster Through Dynamic Remapping of Virtual Machines,” *Future Gener Comput Syst*, vol. 28, no. 2, pp. 469–477, Feb. 2012.
[38] A. Beloglazov, “Energy-efficient management of virtual machines in data centers for cloud computing,” 2013.
[39] “OpenStack Docs: Scheduling.” [Online]. Available: http://docs.openstack.org/mitaka/config-reference/compute/scheduler.html. [Accessed: 13-Jul-2016].

[40] “Openstack Administrators Guide.” [Online]. Available: http://docs.openstack.org/admin-guide/cli-manage-flavors.html. [Accessed: 29-November-2016].

APPENDIX

Scripts used in this research:

Finding RAM of the VM:

```
#!/usr/bin/python
import time
import os
import sys
import subprocess
from subprocess import call

call.("source admin-openrc.sh -p admin",shell = True)
with open ("ip.txt", "r") as myfile:
   data=myfile.read().replace('\n', '')

name = subprocess.check_output("cat /etc/hostname",shell=True)

value = name.strip()

start = time.time()
while (time.time() - start < 21600):

                                RAM = subprocess.check_output("cat /proc/meminfo |
grep MemTotal | awk '{print $2}'", shell=True)


                                f =
open('/home/node_scripts/textfiles/%s/%s_RAM.txt' %(value, data), 'w')
                                print >> f, RAM
                                f.close()
                                subprocess.call("sudo sshpass -p '****' scp -r
/home/node_scripts/textfiles IP:/home && sudo sshpass -p '*****' scp -r
root@IP:/home/textfiles root@NodeIP:/home", shell=TRUE)
                                time.sleep(60)
```

Finding EC of the VM

```
#!/usr/bin/python
import time
import os
import sys
import subprocess
from subprocess import call

call.("source admin-openrc.sh -p admin",shell = True)
with open ("ip.txt", "r") as myfile:
   data=myfile.read().replace('\n', '')
   #print data
```

```
name = subprocess.check_output("cat /etc/hostname",shell=True)
#print name
value = name.strip()

one = "nova list --host "
two = value
three = " | awk 'NR==4{print$2}'"

end = one+two+three

host = subprocess.check_output(end ,shell=True)
hostvalue = host.strip()


count = 0
while (count < 21600):

                        if hostvalue:
                                print "vms exist"
                                retval = os.getcwd()
                                os.chdir( "/home/Spirals-
Team/bitwatts/bitwatts-cli/target/universal/stage" )
                                call('./bin/bitwatts modules procfs-cpu-
simple monitor --frequency 1000 --targets all --console duration 60 >>
/home/node_scripts/output.txt' , shell=True)
                                count = count+60
                        else:
                                count = count+1
```

Finding the CPU Cores

```
#!/usr/bin/python
import time
import os
import sys
import subprocess
from subprocess import call

call.("source admin-openrc.sh -p admin",shell = True)
with open ("ip.txt", "r") as myfile:
   data=myfile.read().replace('\n', '')

name = subprocess.check_output("cat /etc/hostname",shell=True)
print name
value = name.strip()

start = time.time()
while (time.time() - start < 21600):
```

```
                                Cores = subprocess.check_output("nproc --all",
shell=True)

                                x = Cores.strip()
                                print x
                                f =
open('/home/node_scripts/textfiles/%s/%s_cpucores.txt' %(value, data), 'w')
                                print >> f, x
                                f.close()
                                subprocess.call("sudo sshpass -p '****' scp -r
/home/node_scripts/textfiles IP:/home && sudo sshpass -p '****' scp -r
root@IP:/home/textfiles root@NodeIP:/home", shell=True)
                                time.sleep(60)
```

Finding CPU Utilization

```
#!/usr/bin/python
import time
import os
import sys
import subprocess
from subprocess import call

name = subprocess.check_output("cat /etc/hostname",shell=True)
#print name
value = name.strip()


call.("source admin-openrc.sh -p admin",shell = True)
one = "nova list --host "
two = value
three = " | awk 'NR==4{print$2}'"

end = one+two+three

host = subprocess.check_output(end ,shell=True)
hostvalue = host.strip()

count = 0
while (count < 21600):

                                        if hostvalue:
                                                print "vms exist"
                                                subprocess.call("top -b -d1 -n
60 |grep Cpu | cut -c 11-25 >> top.txt" , shell = True)
                                                count = count+60
                                        else:
                                                count = count+1
```

Finding the Floating IP

```
#!/usr/bin/python
import time
import os
import sys
import subprocess
from subprocess import call

namehost = subprocess.check_output("cat /etc/hostname",shell=True)
#print name
value = namehost.strip()

name = subprocess.check_output("ifconfig br-fw-admin | awk
'NR==2{print$2}'",shell=True)



x = name.strip()

ip = x.split(':', 1)

print ip[1]

f = open('ip.txt', 'w')
print >> f, ip[1]

h = "sudo mkdir "
mkdir = h+value

os.chdir( "/home/node_scripts" )
call("sudo mkdir textfiles", shell = True)
os.chdir( "/home/node_scripts/textfiles" )
call(mkdir , shell = True)
os.chdir( "/home/node_scripts" )
```

Finding the Aggregate of values from text files using bash scripts:

```
#!/bin/bash

file=$1
if [[ -n $file && -f $file ]];then
output_file="$(echo $file | sed 's/\(\w\)\.txt/\1/g')_avg.txt"
echo -e "\e[95mScript has begun it's processing, Don't mess with it.\nAfter
Script completion $output_file.txt file is created.\e[0m"
sleep 1
a=60
j=0
output_file="$(echo $file | sed 's/\(\w\)\.txt/\1/g')_avg.txt"
values_60_set=0
values=()
if [[ $file =~ energy ]]; then
values=($(cat $file | sed 's/.*power\=\([0-9]*\.[0-9]*\)/\1/g'))
elif [[ $file =~ load ]]; then
values=($(cat $file))
```

```
elif [[ $file =~ util ]]; then
values_us=($(cat $file | sed 's/.*\:\s*\([0-9]*\.[0-9]*\)\s*us\,.*/\1/g'))
values_sy=($(cat $file | sed 's/.*\,\s*\([0-9]*\.[0-9]*\)\s*sy\,.*/\1/g'))
for i in $(seq 0 $((${#values_us[@]}-1))); do
#echo "ADD: ${values_us[$i]}+${values_sy[$i]}"
values+=($(bc <<< ${values_us[$i]}+${values_sy[$i]}))
done
fi
echo $((${#values[@]}/$a))
for i in $(seq 1 $((${#values[@]}/$a))); do
for k in $(seq $j $(($j+$(($a-1))))); do
#echo $k
values_60_set=$(bc <<< $values_60_set+${values[$k]})
done
values_avg=$(bc <<< "scale=4;$values_60_set/$a")
echo "VALUE: $values_avg"
echo $values_avg >> $output_file
#echo '-----------------------' >> $output_file
values_60_set=0
j=$(($k+1))
done
else
echo -e "\e[31mFUCK OFF!!!"
echo -e "USAGE:\n$0 <inputFile>.txt\e[0m"
fi
```