Aalto University
School of Science
Master's Programme in ICT Innovation

Pouya Samadi Khah

Performance Modeling of the OpenStack Controller

Master's Thesis

Espoo, August 30, 2016

Supervisor: Jukka K. Nurminen, Aalto University, Finland
Mihhail Matskin, KTH University, Sweden

Instructor: Fetahi Wuhib, Ericsson Research, Kista, Sweden

Abstract

Aalto University School of Science Degree Programme in Computer Science and Engineering Master's Programme in ICT Innovation	ABSTRACT OF THE MASTER'S THESIS
Author: Pouya Samadi Khah	

Title: Performance Modeling of the OpenStack Controller		
Number of pages: 51	Date: 2016-08-30	Language: English
Major: Distributed Systems and Services		
Supervisor: Jukka K. Nurminen, : Mihhail Matskin		
Advisor: Fetahi Wuhib		

OpenStack is currently the most popular open source platform for Infrastructure as a Service (IaaS) clouds. OpenStack lets users deploy virtual machines and other instances, which handle different tasks for managing a cloud environment on the fly. A lot of cloud platform offerings, including the Ericsson Cloud System, are based on OpenStack. Despite the popularity of OpenStack, there is currently a limited understanding of how much resource is consumed/needed by components of OpenStack under different operating conditions such as number of compute nodes, number of running VMs, the number of users and the rate of requests to the various services.

The master thesis attempts to model the resource demand of the various components of OpenStack in function of different operating condition, identify correlations and evaluate how accurate the predictions are. For this purpose, a physical OpenStack is setup with one strong controller node and eight compute nodes. All the experiments and measurements were on virtual OpenStack components on top of the main physical one.

In conclusion, a simple model is generated for idle behavior of OpenStack, starting and stopping a Virtual Machine (VM) API calls which predicts the total CPU utilization based on the number of Compute Nodes and VMs.

Keywords: **compute node, CPU Utilization, Idle Experiment, OpenStack Controller, and Per API Call.**

Declaration

This thesis is an account of research undertaken between January 2015 and August 2015 at Ericsson Research AB, Kistagången 21, Stockholm, Sweden. I hereby certify that I have written this thesis independently and have only used the specified sources as indicated in the bibliography.

Pouya Samadi Khah

Stockholm, August 2016

Acknowledgement

I would like to thank and express my deepest appreciation to my supervisor Fetahi Wuhib and my manager Azimeh Sefidcon, who gave me this opportunity to work in Ericsson and guided me throughout this work. I, not only, learned from them technically, but their knowledge and experience has helped me progress professionally. I would also like to thank my supervisor Mihhail Matskin and examiner Anne Håkansson from the KTH Royal Institute of Technology and my supervisor from Aalto University Jukka K. Nurminen. Their reviews, comments and feedback have helped me write this thesis report in the most professional way possible.

Most importantly, none of this would have been possible without the love and patience of my family. My family, to whom this thesis is dedicated to, has been a constant source of love, concern, support and strength all these years. I would also like to explicitly thank to my brother, Kaveh for being one my most significant sources of support. I couldn't have accomplished anything without him.

Tack så mycket and Kiitos!

Pouya Samadi Khah

Stockholm, August 2016

Abbreviations

AWS	*Amazon Web Service*
DHCP	*Dynamic Host Configuration Protocol*
GPL	*General Public License*
IaaS	*Infrastructure as a Service*
PaaS	*Platform as a Service*
SaaS	*Software as a Service*
SSE	*Sum squared error*
VM	*Virtual Machine*

Table of Contents

Figures

Introduction

The following section(s) provides an in depth view on the background of the approach, the existing problems, their proposed solution and the limitations faced during the course of this Master Thesis project.

1.1 Problem Statement

OpenStack continues to gain momentum in the market and is now a serious contender as more companies move from evaluation to deployment. One of the key signs of an emerging technology that is on the cusp of widespread technology is the participation of incumbent vendors. Ericsson Cloud Research is also using OpenStack infrastructure for their cloud system. However, there is a limited understanding of how much CPU is consumed or needed by components of OpenStack under different operating conditions such as number of Compute Nodes, number of running VMs, the number of users and the rate of requests to the various services.

1.2 Goals

The goal of this thesis is to model the CPU utilization of the OpenStack Controller Node in three states.

1.3 Objectives

The objective of this thesis is to study and measure OpenStack CPU consumption in three states. The first state is when the OpenStack cloud system is in the idle phase. Starting and stopping virtual machines API calls are the other phases, which have the most impact on CPU utilization of the OpenStack. This model calculates the CPU utilization based on the number of VMs and number of Compute Nodes. Within this model it is possible to predict the CPU resource consumption of the OpenStack setting based on specific number of VMS and Compute Nodes.

1.4 Project Plan

This thesis consists of 3 main work items.

1.4.1 Study and Investigation

The first task in this work item involves study of the various approaches that can be employed for performance modeling of software systems and selecting the appropriate one. The second item involves study of the OpenStack cloud management platform; its architecture, its components, etc.

1.4.2 Measurement Collection and Analysis

The OpenStack environment is provided by Ericsson Research, which gave me the possibility to run OpenStack under different operating conditions and collect measurement data. The physical server contained eight compute nodes and one controller node. It is worth mentioning that all the experiments were on virtual OpenStack environments, which were deployed on physical OpenStack cloud.

1.4.3 Developing the Model

This activity involves developing the model. It takes as input the test-bed measurement results and produces an analytical model that can predict the resource demand of the various OpenStack components under different operating conditions. Using this model, it should also be possible to propose possible configurations for an OpenStack controller given an envelop operating condition for a specific cloud deployment.

1.5 Methodology

The thesis used quantitative data for the research findings, which is part of experimental research method [20]. As for quantitative research, I ran OpenStack based on different conditions based on OpenStack state. In each state several experiments are done to get how OpenStack reacts on those scenarios. The most important variable that is been studied was CPU consumption. After collecting data, I execute statistical analysis method experiment results. Finally, I implemented my model based on the data with curve fitting method.

1.6 Risks, Consequences and Ethics

This new approach of modeling requires immaculate procedure, which is involved in design, development and implementation due to the big data process and high amount of intermediate variables, hence it requires of fine optimization, based on the available resources. The project

had limited access to physical computer nodes; therefore, the virtual OpenStack infrastructure is being used for the whole data analyses. The amount of work for development and improvement of the project overpasses the limited time of this thesis job; therefore, it has been decided to adjust additional degree of simplification to the model, which has been lead to results with somewhat compromises.

Besides that, one of the most important ethics in working at Ericsson company is that they have confidential data and it is important to keep in mind all the time which parts of the data are legal to publish and which part need to remain confidential.

1.7 Thesis Outline

Chapter 2 gives background information about cloud computing and in-depth information about OpenStack. Chapter 3 refers to measurement collection and analyses. An environment in which OpenStack can be deployed is provided. Then the OpenStack undertakes in different operating conditions in order to collect measurement data. Chapter 4 is discussions of current limitations of the scheduler in different methods to overcome those conclude the thesis. There is a deep discussion in how the model can predict the amount of resources needed for optimized CPU utilization or the other way around. The final chapter is related to future work and recommendations.

1.8 Author's Contribution

Pouya Samadi Khah has done all work during the duration of this Master Thesis project. In case of occurrence of any problem, it was solved with consultation of my advisors in Ericsson and also from support forums online.

2 Cloud Computing and OpenStack

2.1 Cloud computing

Cloud computing is a term used for delivering on-demand computing services over the Internet [1]. It also refers to application as a service as well as hardware and software in data centers which provide these services.

2.1.1 Deployment Models

Cloud computing deployment models based on cloud locations can be divided into four groups: Public, private, community and hybrid cloud [2]. If the cloud, which is known as hardware and software data centers, used as a pay-for-use basis, it is called public cloud. Private cloud refers to the internal data centers, which are not public for general use. A community cloud is multi-tenant infrastructure, which is shared among several organizations that is governed, managed and secured commonly by all the participating organizations or a third party managed service provider. A hybrid cloud contains both private and public cloud services. Usually in hybrid approach, a business might run on private cloud, but rely on a public cloud to accommodate spikes in usage.

2.1.2 Service Models

Cloud computing service models are classified into three groups: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [3]. In IaaS there is a basic cloud service structure consisting of virtual machines, block storage, firewalls, load balancers and networking services are provided. The PaaS provides a platform for developers to implement applications over the Internet. SaaS is an on-demand software service, which is a way of delivering applications over the Internet as a service. In SaaS, there is no need of installing applications. The access is remotely via the Internet. It is shown in the figure 1, how different services are placed in based on their visibility to the end users.

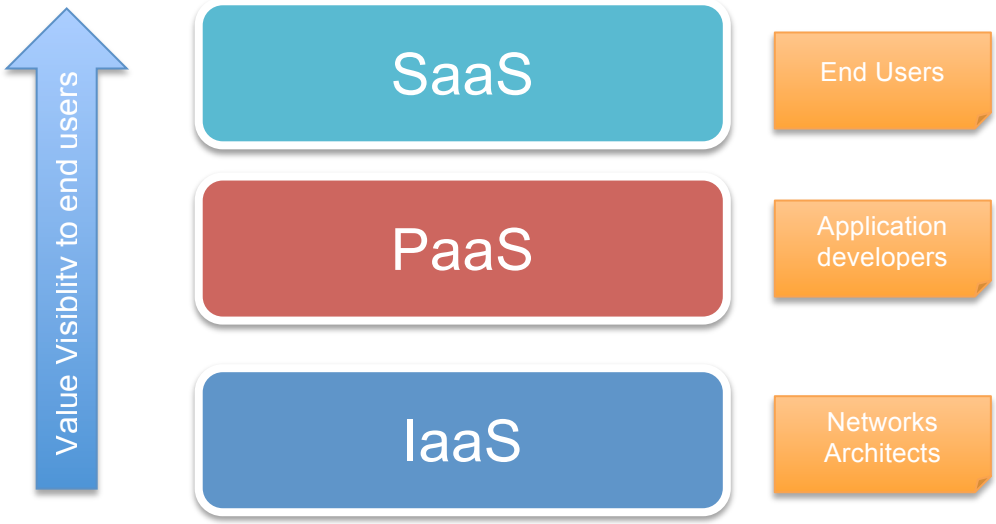

Figure 1, Cloud Service Models [32]

2.2 Different Cloud Solutions

There are a variety of different open-source cloud platforms, including Amazon Web Service (AWS) EC2, Microsoft Azure, Eucalyptus, OpenNebula, and OpenStack [5]. AWS EC2 [25] is a web service that provides a resizable compute capacity in the cloud in order to create a scalable cloud computing but the main difference compare to OpenStack is that it has an on-demand pricing model in contrast with, OpenStack which is open source and free. Eucalyptus project is released under a different license than the others, which has the GNU General Public License version 3 (GPLv3) [8], whereas OpenStack, OpenNebula and Cloudstack have Apache License version 2 [9, 10] and they are open-source. It is worth mentioning that OpenStack has the largest population and it is the most active open-source project compare to the others [15].

2.3 OpenStack

2.3.1 History

OpenStack is an open-source Infrastructure as a Service (IaaS), which is aimed for creating and managing large groups of virtual machines and data centers [4]. Originally in 2010, OpenStack

was a joint effort of Rackspace and NASA. The first release had just key components of Nova and Swift. Nova is the main component of IaaS, manages and controls compute nodes of the OpenStack environment and Swift is an unstructured redundant storage system

Additional components have been added in consecutive releases in table1 all the release dates and version [19] are shown. Today, the community consists of a significant number of individuals and companies, including Ericsson, IBM, Dell, Canonical, Red Hat, and Cisco.

Table 1, OpenStack releases [21]

Series	Status	Releases	Date
Liberty	Under development	Due	Oct 15, 2015
Kilo	Current stable release, security-supported	2015.1.0	Apr 30, 2015
Juno	Security-supported	2014.2.3	Apr 13, 2015
		2014.2.2	Feb 5, 2015
		2014.2.1	Dec 5, 2014
		2014.2	Oct 16, 2014
Icehouse	Security-supported	2014.1.5	Jun 19, 2015
		2014.1.4	Mar 12, 2015
		2014.1.3	Oct 2, 2014
		2014.1.2	Aug 8, 2014
		2014.1.1	Jun 9, 2014
		2014.1	Apr 17, 2014
Havana	EOL	2013.2.4	Sep 22, 2014
		2013.2.3	Apr 03, 2014
		2013.2.2	Feb 13, 2014
		2013.2.1	Dec 16, 2013
		2013.2	Oct 17, 2013
Grizzly	EOL	2013.1.5	Mar 20, 2014
		2013.1.4	Oct 17, 2013
		2013.1.3	Aug 8, 2013
		2013.1.2	Jun 6, 2013
		2013.1.1	May 9, 2013
		2013.1	Apr 4, 2013
Folsom	EOL	2012.2.4	Apr 11, 2013
		2012.2.3	Jan 31, 2013
		2012.2.2	Dec 13, 2012
		2012.2.1	Nov 29, 2012
		2012.2	Sep 27, 2012
Essex	EOL	2012.1.3	Oct 12, 2012

		2012.1.2	Aug 10, 2012
		2012.1.1	Jun 22, 2012
		2012.1	Apr 5, 2012
Diablo	EOL	2011.3.1	Jan 19, 2012
		2011.3	Sep 22, 2011
Cactus	Deprecated	2011.2	Apr 15, 2011
Bexar	Deprecated	2011.1	Feb 3, 2011
Austin	Deprecated	2010.1	Oct 21, 2010

OpenStack has three main advantages, which should be taken into consideration. First of all, it is highly scalable, which gives the user to store and Compute petabytes of data in distributed systems fashion. It is also compatible and flexible with different virtualization solutions such as KVM, LXC, QEMU, UML, Xen and XenServer [6].

2.4 OpenStack Versions

OpenStack has several versions. The first version, which is called Austin, is launched in October 2010. In my thesis, I used the OpenStack Juno version, which is released October 2014 and finalized in February 2015. OpenStack Juno is the stable and secure-supported version of OpenStack. The OpenStack versions are announced every six months [14].

2.5 OpenStack Architecture

In the figure below, it is shown that how different OpenStack services are communicating with each other [7].

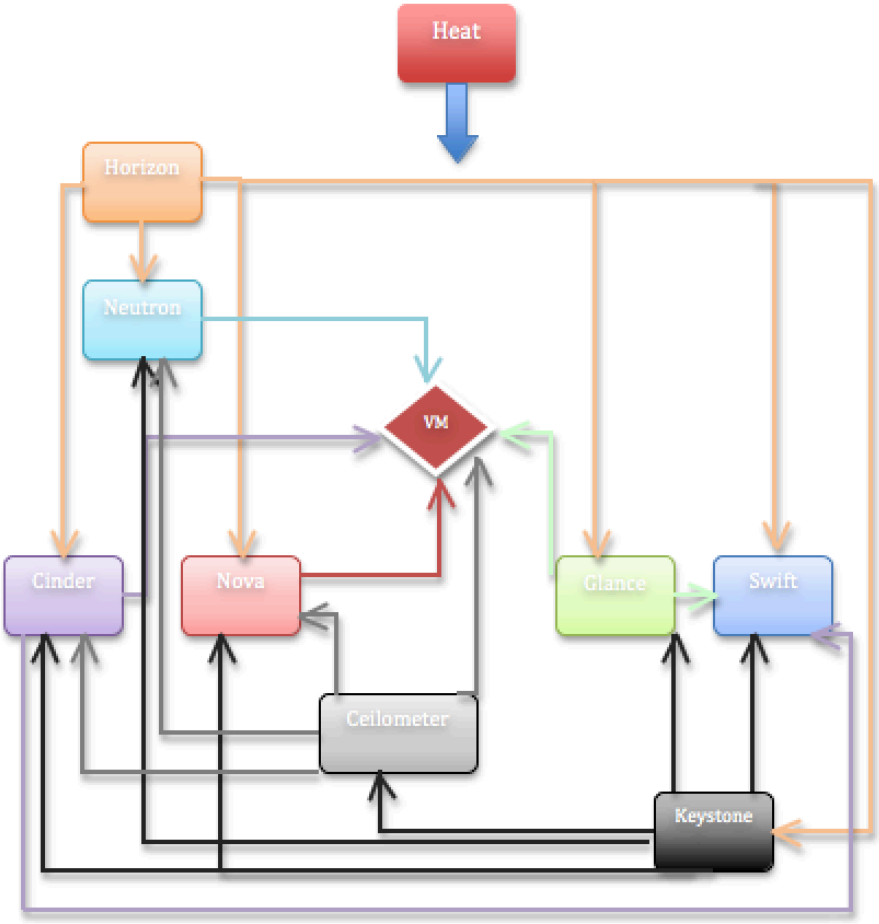

Figure 2, Simplified OpenStack Architecture [33]

2.5.1 OpenStack Services

Compute (Nova)

OpenStack Nova, which is the main component of IaaS, manages and controls compute nodes of the OpenStack environment.

Networking (Neutron)

Neutron is a system for managing and controlling network systems and IP addresses. In addition, it provides an API interface to configure network settings as a service.

Dashboard (Horizon)

Dashboard (Horizon) provides a graphical user interface in order to interact with OpenStack underlying services. The main purpose is to access, providing, and automate cloud-based resources such as launching instances, configuring IP addresses.

Object Storage (Swift)

Swift is an unstructured redundant storage system. Objects are written in multiple files in different disks in order to insure the integrity over the clusters.

Block Storage (Cinder)

Cinder provides persistent block-storage devices.

Identity Service (Keystone)

Keystone provides mapping system between users and OpenStack services. It is used for authentication and authorization.

Image Service (Glance)

Glance stores and retrieves data from disks and server images.

Telemetry (Ceilometer)

Ceilometer monitors and measures the OpenStack for statistics.

Orchestration (Heat)

Heat is used for orchestrating multiple composite cloud applications using templates

Database Service (Trov)

Trov is a Database as a service system for both relational and no relational databases [7].

2.6 OpenStack Deployments

OpenStack is based on distributed design. In other words, different OpenStack services can run on a highly distributed setting with a little restriction. Thus, the deployment architecture can

range from all-in-one to multi-node deployments. In general, there are two kind of nodes for OpenStack setting, compute nodes which host and run Virtual Machines (VMs) and controller nodes which run all the other services to support the VMs. Different OpenStack deployment techniques are demonstrated in figure 3.

2.6.1 All-in-one deployment

In this architecture both controller and Compute services are running in the same server. As an example, DevStack [21] use this architecture for the cloud deployment.

2.6.2 3+N deployment

The most common deployment architecture for multi-scale OpenStack services which also OpenStack recommends to use [24]. In these deployments, there is one node dedicated for networks and another node for storage and the third node for the remaining OpenStack services. Besides that, one or more compute nodes are used to do the computation part of the cloud infrastructure.

2.6.3 N+M deployment

It is the extended version of 3+N deployment where you could have N controller nodes and M compute nodes. This structure is used when there is need for clustering and high availability among OpenStack services.

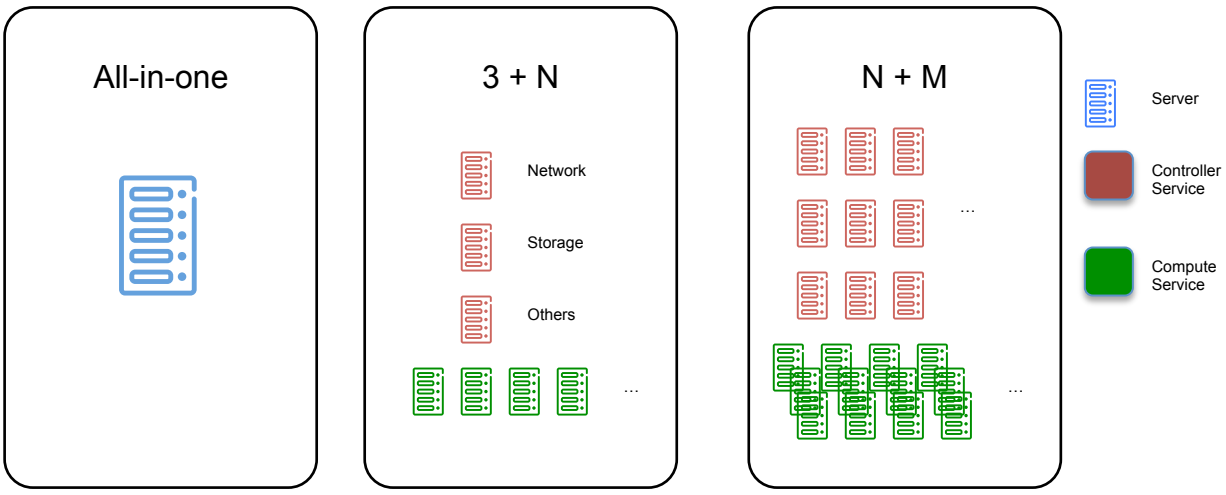

Figure 3, OpenStack Deployment [4]

2.7 Dimensioning OpenStack

There are several OpenStack deployment architectures which each of them has a unique way of scalability and dimensioning. In general, there are two ways to dimension the OpenStack, based on the compute nodes, and Controller nodes.

2.7.1 Dimensioning Computing Nodes

To do the dimensioning of compute nodes, you need to have a good knowledge of the load that you expect on the cloud. In each OpenStack system you can choose between one or more flavors [18] to use for your compute nodes. Flavors are basically virtual hardware templates defining sizes for RAM, disk, and the number of cores. In the table2, it is shown that based on the different flavors, how many VMs is expected to run on one compute node.

The first thing to consider in dimensioning compute nodes is trying to estimate the total amount of resource that is going to be consumed by the VMs. To calculate it, you need to multiply the number of VMs by the amount of resource for each flavor and sum it up for the all the flavors.

Table 2, Dimensioning OpenStack compute nodes

Flavor	RAM (MB)	Disk (GB)	Cores	#VMs
m1.tiny	512	1	1	30
m1.small	2048	20	1	30
m1.medium	4096	40	2	10
m1.large	8192	80	4	10
m1.xlarge	16384	60	8	5
Total	281600	2630	160	
OC Ratio	1.5	1	2	
HW Spec	16384	256	20	
Number of the compute nodes	11.45	10.27	4	
Max number of the Compute Nodes: 12				

Another important fact to take into calculation is the overcommitting ratio, which demonstrates how many resources a compute node reports to the scheduler. For example, if you have a server with 20 cores and an over-commit ratio of 2, the compute node will report as if it has 40 CPU cores. This is very helpful to have a high utilization OpenStack system. Thus, once you have the

over-commit ratio and the total amount of resources then you can choose one server type. As an example, which is shown in Table 2, the server with 16384MB of RAM and 256 GB of hard disk is chosen, thus, to calculate the expected numbers of compute nodes the following formula is used:

$$Compute_{Nodes} = Max\left(\frac{Total_{Resources}}{(OC_{ratio} * HW_{RAM})}, \frac{Total_{Resources}}{(OC_{ratio} * HW_{Disk})}, \frac{Total_{Resources}}{(OC_{ratio} * HW_{Cores})}\right)$$

So the result is $Compute_{Nodes} = Max\left(\frac{281600}{1.5*16384}, \frac{2630}{1*256}, \frac{160}{2*20}\right) = 11.4583333 \cong 12$

Selecting the suitable VM type also depends on scheduler algorithms. By default, the OpenStack scheduler [18] tries to balance the memory load on the compute nodes. For instance, if the cloud is utilized up to 80%, all the servers are utilized up to 80% and if the OpenStack wants to start a large instance that requires more than 20% of the capacity, cannot do with the current settings due to the fact that all the servers will be busy. The solution is either adding more compute nodes to the cloud platform or changing the OpenStack scheduler behavior. The OpenStack scheduling should fill the nodes before moving to the next node instead of trying to balance the load.

2.7.2 Dimensioning Controller Nodes

The load of the OpenStack controller depends on many facts except the load of the VMs. It depends on deployment architecture, the plugins that are used, the number of compute nodes, the configuration of services, number of objects (VMs, routers, networks...), and also how frequent users are interacting with the cloud.

2.7.2.1 Dimensioning Controller Nodes approaches

It is a challenge to come up with a generic guideline to the dimension OpenStack controller. One of the approaches to overcome this challenge is to overprovision [28] the resources that are in the controller node that you may not need to use them. Besides that, in large VM deployments, the overprovisioning strategy leads to significant under-utilization of the resources.

Another way to address this problem is oversubscription your resources at a cost which is increasing the likelihood of overload or alternatively one or more VMs do not have the resources to complete a task without encountering performance degradation.

The Ericsson Research Team in OpenStack Summit May 2015, based on my contribution, proposed two approaches to dimension OpenStack cloud [26]. The first approach is to initially

deploy the OpenStack cloud first and then measure the resource consumption based on the different scenarios. It is preferable to deploy in a test environment in order to test different parameters before to deploy in actual production environment.
The second approach, which is the main contribution of this thesis, is that to develop a model to estimate how much the resource will be consumed and then use this model as an input for the HW dimensioning accordingly.

There are three environments to study and analyze OpenStack deployment. If there is a direct access to the physical servers and nodes, the deployment can be executed in the physical environment. The advantage of this environment is that the results of the experiments will be highly accurate. However, it is usually difficult to access to the physical server production line and there is always a limitation of resources to use. The alternative environment is the emulated one, which is a virtualized environment. The problem is that the results will not be as accurate as the physical environment; however, it is highly scalable and flexible in terms of experiments. The other approach is to use the simulated environment. Although they are highly flexible, there are no good simulators for OpenStack today.

3 Experimental Setup

3.1 OpenStack Fuel

To install OpenStack on the physical layer, I used OpenStack Fuel [13], which is an open-source management tool with a GUI to deploy OpenStack platforms. The main advantage of OpenStack Fuel is bringing simplicity, error prone process of deploying and configuring different OpenStack Services. In the figure below you can see the architecture of Fuel. The key components are UI, Nailgun, Cobbler and Mcollective agents. The UI part is basically a JavaScript page which uses bootstrap and backbone underneath. The Nailgun is the main part of the Fuel, which creates Rest API and data management. Cobbler is used for providing services. Mcollective agent gives the opportunity to execute specific tasks like hardware cleaning.

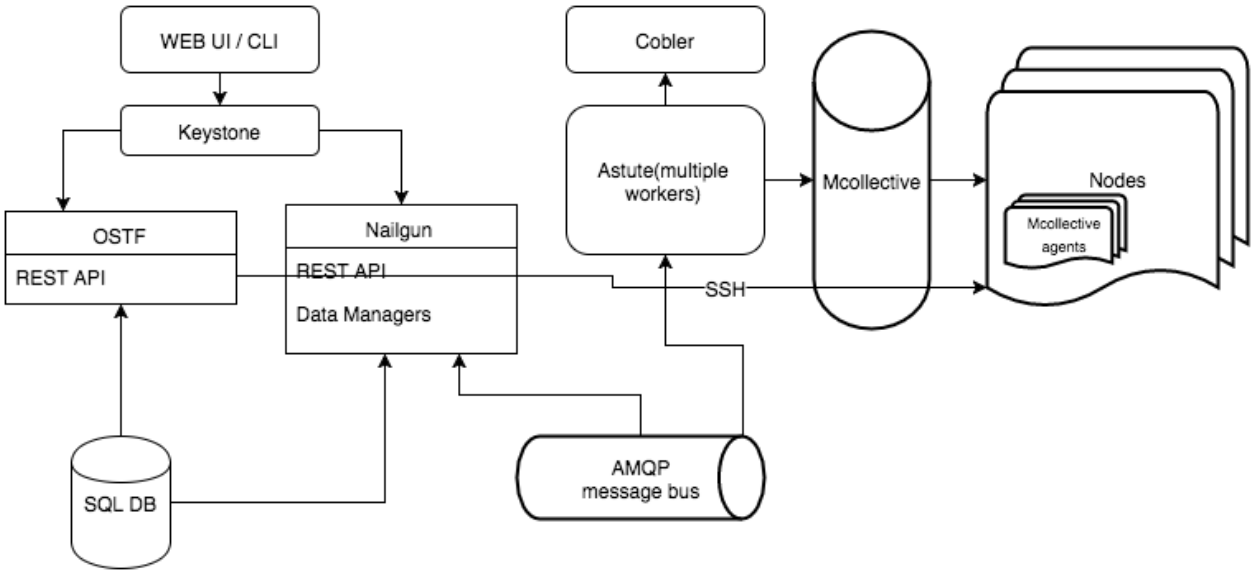

Figure 4, OpenStack Fuel Architecture [13]

3.2 OpenStack on OpenStack

Installing OpenStack is not an easy task to do even with OpenStack Fuel. First of all, it needs a physical hardware to install the under-cloud and then install and configure the suitable distribution approach. There are lots of steps to put into consideration in installing one

OpenStack. Taking into mind that experimenting different OpenStack configurations consumes huge amount of hardware devices, which makes it near impossible to work on fast and parallel.

To overcome this obstacle, making an emulated environment is a possible solution. To do that, the OpenStack platform should be deployed on the main OpenStack platform, which is installed on the physical layers [11].
The key advantages of OpenStack on OpenStack are:
→ Scalability
→ Integration
→ Decreasing drive installation and maintenance costs
→ Encapsulate the installation and upgrade process
→ Common API and infrastructure for above and below cloud [12]

3.3 OpenStack Juno configurations

This section explains about the main components of the OpenStack infrastructure, which is going to be used in my measurements and experimentations. The latest version of the OpenStack by the time of my research was OpenStack Juno.
The main tasks of the core components are described below:

- **Controller Node:**
  It is the main part of the OpenStack, which controls and configures the Service nodes and compute nodes. The controller is connected to two separate networks, which one of them is for networking and the other one is for managing the service node. All the OpenStack services are becoming executable by connecting to the controller node. As it is shown in the figure above, it can be accessed via public IP address [16].
- **Service Node:**
  It is also kind of a controller node, which provides networking services (L3 routing, dhcp) and also provides the disks for the block storage service.
- **Compute Node:**
  The computation part is based on this unit. The Compute Node is not needed to connect to a public IP address, as it is not need to be accessible directly to monitor. However, within the Controller Node, Compute Node is accessible [16].

In the figure 3, there is a snapshot of a virtual OpenStack Juno setup. All the three main nodes connected to management layer and network layer. However, the controller node and service node are connected to the public network in order to access directly to them from outside.

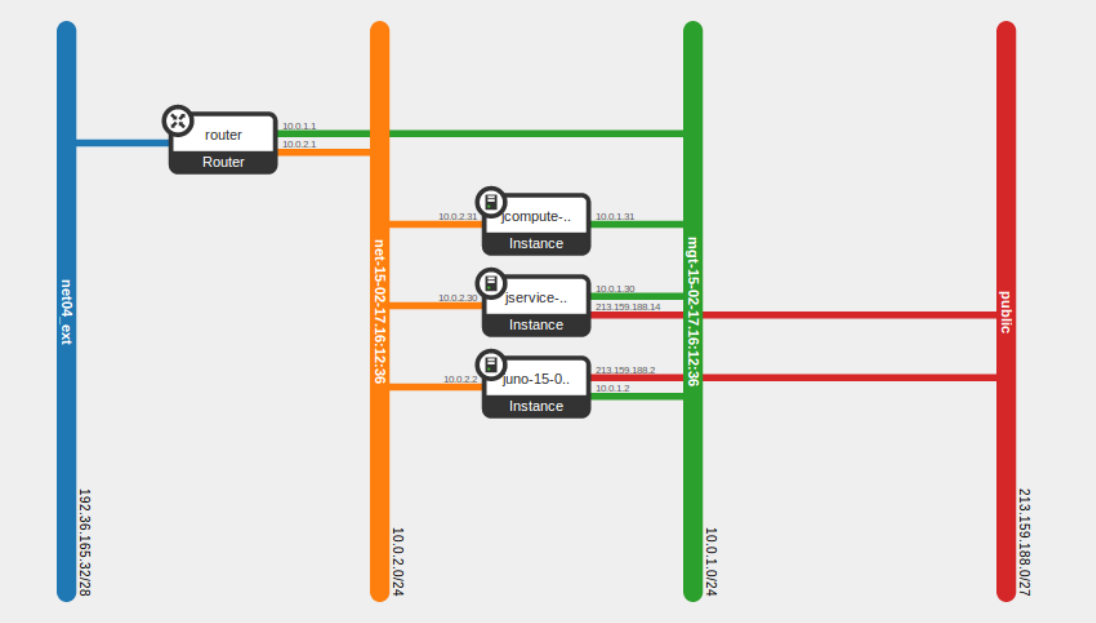

Figure 5, OpenStack Juno setup

3.4 Monitoring

The Monitoring program was written as a part of the scalability of OpenStack projects during the summer of 2014 at Ericsson [17]. The program is to be used to receive performance metrics on running processes and system utilization (CPU, memory, disk and network) in different setups. In this specific project, the program is to be used as a way to get a better understanding of how different OpenStack setups affect system performance.

The program is written in python and uses the psutil [27] library, a cross-platform library for retrieving information on running processes and system utilization. The program can be used either as a module or as a running application on a server. The program monitors the system for a period of time set by the user and writes the collected information to a database. There are two main files: *monitor.py* and *root.py*.

The Monitoring section does the sampling and uses the psutil library to collect the necessary information. It also contains a function that parses the collected data and writes it to a database.

To start the monitoring, it can be called as a function or can be used as an application. The program has successfully been used with Pecan, a Python web framework. This way, the program can be started and configured remotely when monitoring a system. By visiting the URL of the served application the monitoring can be started and stopped:

http://DATABASE_IP_ADDRESS:8080/start_monitor?duration=20&scenario_key='Test-1'
http://DATABASE_IP_ADDRESS:8080/stop_monitoring

To sum up, the *monitor.py* program can be used for monitoring system in different setups and is a proper tool for collecting system utilization metrics. Some testing can be done to improve the performance of the program and make the usage easier. For my experiments, the monitoring program was on run automatically on service node.

3.5 Developing the Model

If there are number of unknowns, for example, the number of VMs or the frequency of user interaction with the cloud, it is difficult to try and evaluate all the possible combinations of the different settings. To overcome this problem, developing a model for the resource consumption is a suitable solution. This model can be used as a tool to predict the amount of resource consumption at different parameters.

The model in general should be a function that can map the state of the cloud deployment to the consumption of various resources. The state of the cloud can be the amount of physical resources (compute nodes, storage nodes, HW, routers, etc.), objects (VMs, volumes, routers, networks, tenants, users) and user interface patterns (APIs, rate of calls, sequence of calls). The output should be the consumption indicator of the resources such as CPU, RAM, disk IO. The consumption is calculated based on two main phases. The first phase is about analyzing the resource consumption of OpenStack components is in the idle phase. This will let to study the processes that are consuming the resources frequently. The second phase is about the impact of

API calls on resource consumption. The sum of resource consumption of the idle phase and all the API calls is the output. Thus, the simplified version, which is used in this thesis, is:

$$R = R_{idle}(C,V) + \sum_{a \epsilon A} \lambda_a R_a (C,V)$$

- Resources(R): CPU, Network I/O, Disk I/O, etc.
- State: # VMs (V), # compute nodes (C), set of APIs (A), rate of calls to APIs (λ)
- Idle: There is no API call is going to the system
- Estimate R_{idle}: run deployment with varying values for C and V, measure the resource consumption on the controller
- Estimate R_a: measure resource consumption during periods where call to API a is made subtract the idle resource consumption

To use this model, the first step is to choose what are the resources you want to take it to account. CPU can be a good example for this model. The next step is to choose the state variables for this model such as number of VMs, number of compute nodes, set of OpenStack API calls and the rate of calling them. In order to complete the model, you need to have an estimation of the idle resource utilization and resource utilization per API call.

To ease the OpenStack installation process, I create a set of scripts, which are automating the whole process. The scripts are installing all the needed packages for the installation and also configure the networks between the core components of OpenStack, which are the Controller node, Service node, and compute node and then execute the experiments based on user or OpenStack's input.
We divided our experiments in two parts: Idle part and per API calls part.

3.6 Idle Experiments

In Idle experiments, it is assumed that no APIs of OpenStack services are called from the outside of the OpenStack setup. The aim is to look how the resources in the controller are affected based on number of Compute Nodes and VMs in a certain period of time. Therefore, four different scenarios are done with different settings based on Compute Nodes and VMs. Each scenario is experimented five times in order to get less error-prone results. The following resources are evaluated in these experiments:

- **CPU total usage percentage (per second):** It shows CPU utilization in percentage format

- **Average Physical Memory usage:** It shows average usage of physical memory in a certain period of time
- **Average disk usage per second:** It shows average usage of disk in a certain period of time
- **Disk I/O read kb/second:** It shows the ratio of reads in disk I/O
- **Disk I/O write kb/second:** It shows the ratio of writes in disk I/O
- **Network I/O send kb/second:** It shows the ratio of sending bytes in network I/O
- **Network I/O receive kb/second:** It shows the ratio of receiving bytes in network I/O

It is also worth mentioning that all the plots indicate the **95%** confidence interval. The evaluation of idle experiments is divided based on four different scenarios.

3.6.1 Idle Experiment 1

Table 3 shows the OpenStack settings for the Idle Experiment with zero VM and single compute node. The result for this scenario shows that CPU usage is average 16.73 % in the idle mode. There is no reading from Disk in the idle experiments. Besides that, Network I/O packets are relatively small. However, there is a considerable physical memory usage.

Table 3. OpenStack settings in Idle Experiment 1

Features	
Controller Node: 4 cores	**Service Node:** 4 cores
Compute Node: 12 cores	**Number of Compute Nodes:** 1
Number of Virtual Machines: 0	**Measurement length:** 4000 seconds

Table 4. Idle Experiment 1 Results

CPU total usage percentage (per second)	**16.73%**
Average Physical Memory usage	3318.798 MB
Average disk usage per second	134.827 B
Disk I/O read kb/second	0
Disk I/O write kb/second	1.206 MB
Network I/O send kb/second	57.897 KB
Network I/O receive kb/second	59.218 KB

3.6.2 Idle Experiment 2

In the idle experiment 2, 100 VMs is started to see how will they impact the resource consumption in the idle mode. Table 5 shows the detailed settings for the Idle Experiment with 100 VMs and single compute node.

Table 5. OpenStack settings in Idle Experiment 2

Features	
Controller Node: 4 cores	**Service Node:** 4 cores
Compute Node: 12 cores	**Number of Compute Nodes:** 1
Number of Virtual Machines: 100	**Measurement length:** 4000 seconds

Table 6 shows that the CPU consumption is increased by 2 percent compare to the experiment without VMs. Besides that, it was expected to get similar results for physical memory usage, disk usage, and Disk I/O read, since VMs are not affecting them. The small difference is because of the noise. However, in theory the difference should be zero. In addition, network I/O ratio is also increased because of the connections that VMs created.

Table 6. Idle Experiment 2 Results

CPU total usage percentage (per second)	**19.77%**
Average Physical Memory usage	3294.201 MB
Average disk usage per second	243.022 B
Disk I/O read kb/second	0.00306 KB
Disk I/O write kb/second	1287 KB
Network I/O send kb/second	139.208 KB
Network I/O receive kb/second	141.868 KB

3.6.3 Idle Experiment 3

For the Idle experiment 3, the number of service nodes increased to 12 in order to handle 10 computes nodes. In this experiment, there is no VM. It is expected to get more impact on resource consumptions compare to scenario 1 with one single Compute Node and 0 VMs. Table 7 shows the OpenStack settings for the Idle Experiment with 0 VMs and 10 compute node.

Table 7. OpenStack settings in Idle Experiment 3

Features	
Controller Node: 4 cores	**Service Node:** 12 cores
Compute Node: 4 cores	**Number of Compute Nodes:** 10
Number of Virtual Machines: 0	**Measurement length:** 4000 seconds

The results show that CPU usage is increased compare to the scenario one. Besides that, disk I/O ratio is also more than scenario one, because of the number of Compute Nodes.

Table 8. Idle Experiment 3 Results

CPU total usage percentage (per second)	**18.17%**
Average Physical Memory usage	3445.190 MB
Average disk usage per second	822.220 B
Disk I/O read kb/second	0
Disk I/O write kb/second	1794 KB
Network I/O send kb/second	218.962 KB
Network I/O receive kb/second	229.328 KB

3.6.4 Idle Experiment 4

In the last idle experiment scenario compare to the previous one 100 VMs got started. Table 10 shows the OpenStack settings for the Idle Experiment with 100 VMs and 10 compute nodes.

Table 9. OpenStack settings in Idle Experiment 4

Features	
Controller Node: 4 cores	**Service Node:** 12 cores
Compute Node: 4 cores	**Number of Compute Nodes:** 10
Number of Virtual Machines: 100	**Measurement length:** 4000 seconds

This scenario's result has the most CPU usage compared to the previous ones since it has more VMs than scenario 3 and more compute nodes than scenario 2. In addition, the Disk I/O write is also more than the other scenarios

Table 10. Idle Experiment 4 Results

CPU total usage percentage (per second)	**24.39%**
Average Physical Memory usage	3465.184 MB
Average disk usage per second	943.781 B

Disk I/O read kb/second	0
Disk I/O write kb/second	1871 KB
Network I/O send kb/second	304.976 KB
Network I/O receive kb/second	306.674 KB

3.6.5 Comparison and Conclusion

We compared the results of four experiments based on the mentioned parameters. The following figures are showing the differences of the scenarios based on each resource. Each scenario is distinguished based on the number of Compute Nodes and VMs which is shown like **(C,V)** where **C** is number of compute nodes and **V** is number of VMs.

According to the figure 6, with an increase in VMs the CPU consumption at the controller node increases. Besides that, the number of compute nodes also affects the CPU usage. The two experiments with 10 compute nodes have slightly more CPU usage than the other two.

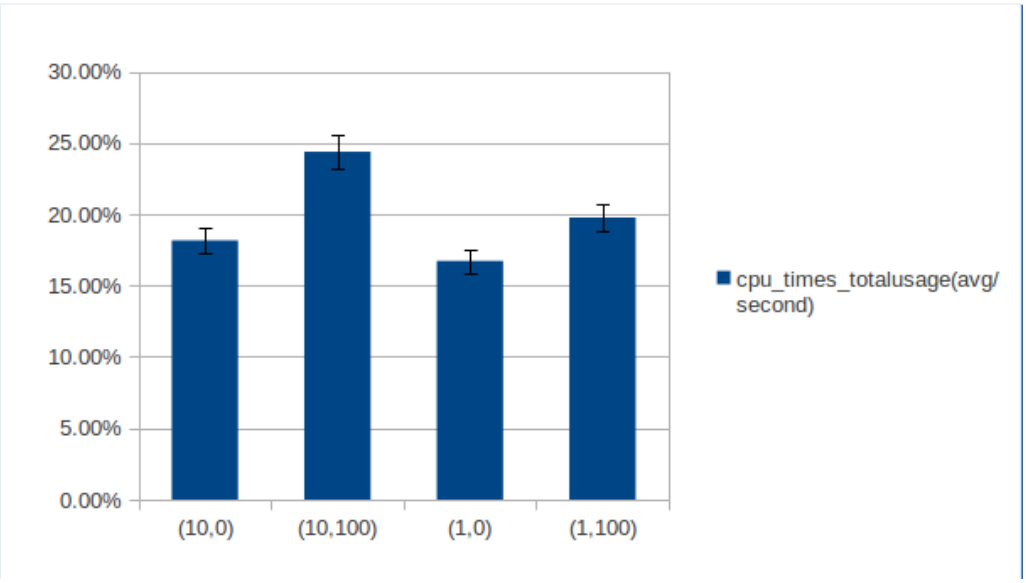

Figure 6, CPU total usage percentage (per second)

In the figure 7, it is clarified that creating VMs does not affect the physical memory usage of the Controller node. However, the more compute node OpenStack setup has, more physical memory usage it will produce.

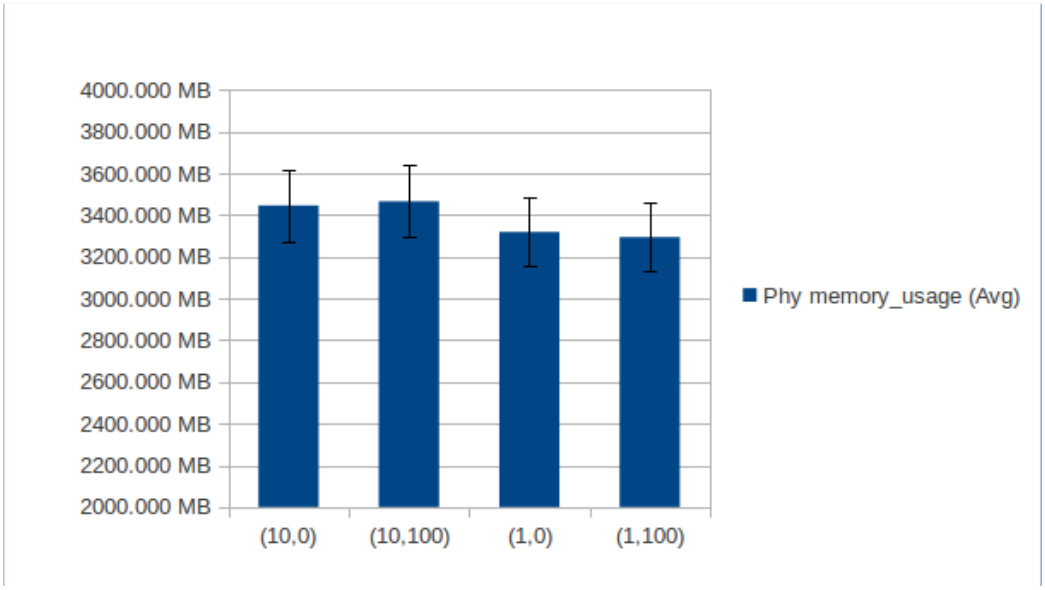

Figure 7, Physical Memory usage (Average per second)

As it is demonstrated in figure 8, both number of compute nodes and number of VMs affect average disk memory usage per second. In addition, the number of compute nodes has more impact on the memory usage ratio than VMs.

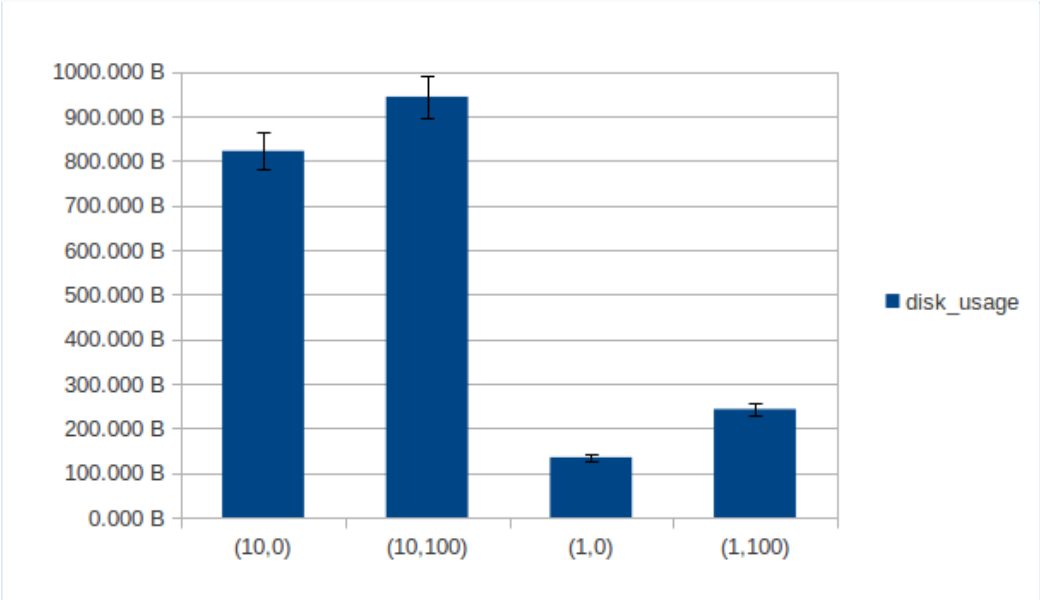

Figure 8, Disk Memory usage (Average per second)

In general, for Disk I/O services, which are shown in figure 9, single compute node has more ratio than 10 compute nodes. When the OpenStack setup get 100 VMs, the ratio of disk I/O reads

per second get close to zero. However, in write, send and receive, the ratio increases compare to the state without any VMs.

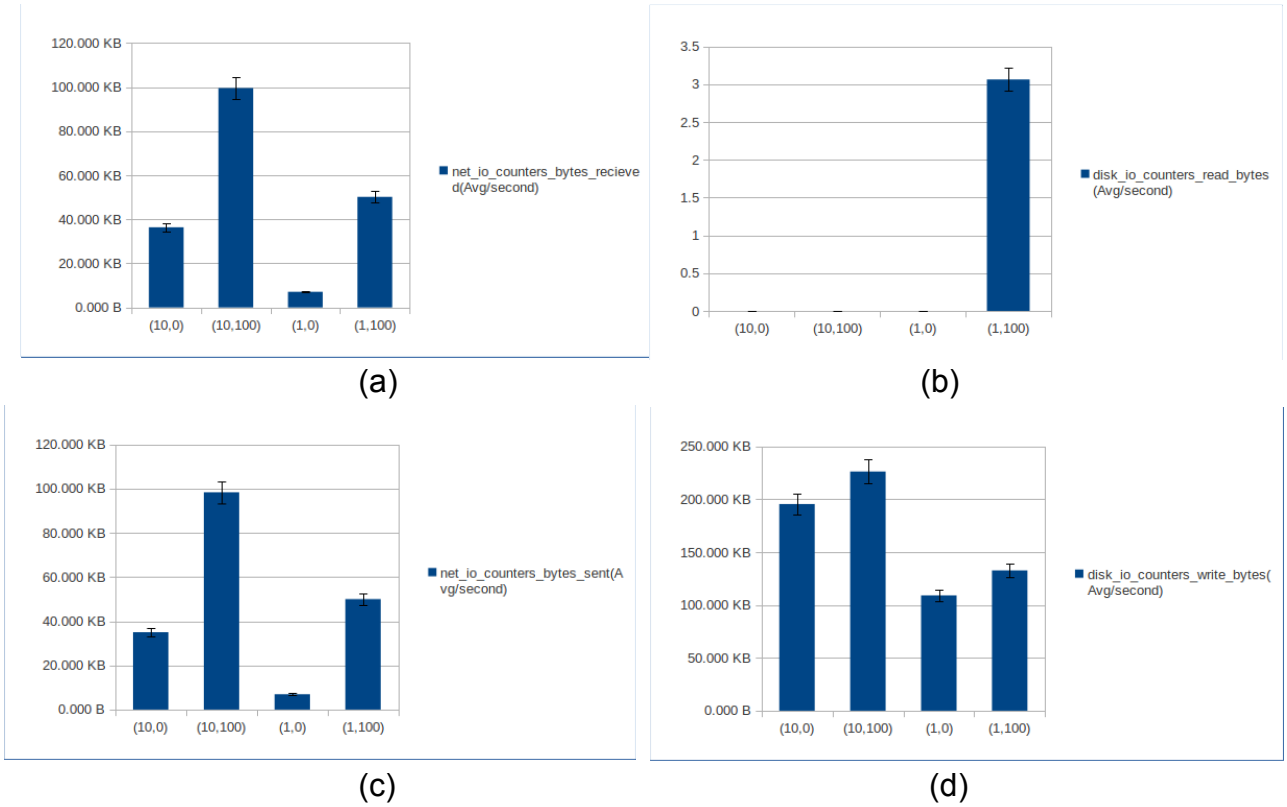

Figure 9, (a) Network I/O receive KB/second, (b) Disk I/O read KB/second
(c) Network I/O send KB/second, (d) Disk I/O write KB/second

3.6.5 Comparison based on consuming processes

In addition to the previous experiments, the top consuming processes are extracted from each experiment for comparing the scenarios. Here is the list of top eight consuming processes:

1) nova-conductor [18]: It implements a new layer on top of the nova-compute, which enables OpenStack to access the database without using the compute nodes.

2) nova-api: It sends and receives API calls from end users.

3) mysqld: It is the main program that runs MySQL server.

4) mongod: It is the key process of MongoDB (NoSQL DB) system.

5) cinder-api: It is getting the API requests and send them to cinder-volume in order to interact with block storage service

6) ceilometer-column: It is a telemetry collector, which runs on central management server(s).

7) ceilometer-agent: It is an agent, which handles the ceilometer operations

8) **beam.smp:** It is RabbitMQ which is an open source message oriented middleware.

In figure 10, there is a comparison between one compute node and 10 compute nodes with or without VMs. It is understandable from the figure 10 that nova_conductor, beam.smp and mysqld are more in 10 compute nodes than single compute node. The reason is that each compute node has a metadata overhead, which sends to the controller. Thus, as the number of computing nodes increases, the overhead data will increase too.

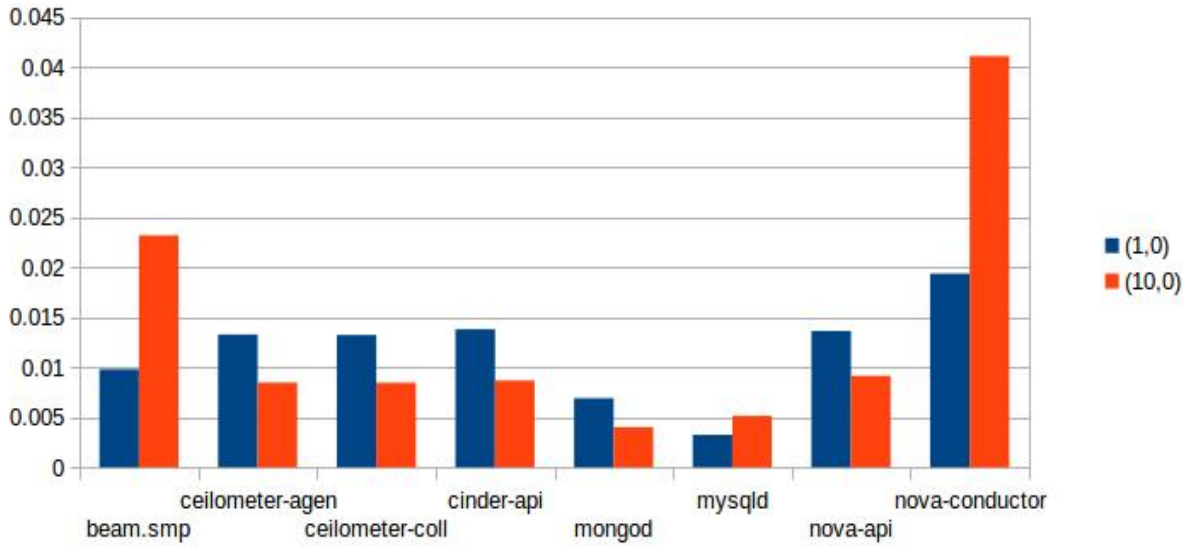

Figure 10, Top process usage comparison with no VMs

For the next comparison, which is shown in figure 11, 100 VMs are started. Similar to the previous comparison nova-conductor, beam.smp mysqld are more in the setting with 10 compute nodes than in single compute node. In theory, the number of compute nodes is expected to be independent from other processes like ceilometer ones in theory. However, because of noise and OpenStack's unsteady behavior in idle mode, changing the number of compute nodes affects the processes.

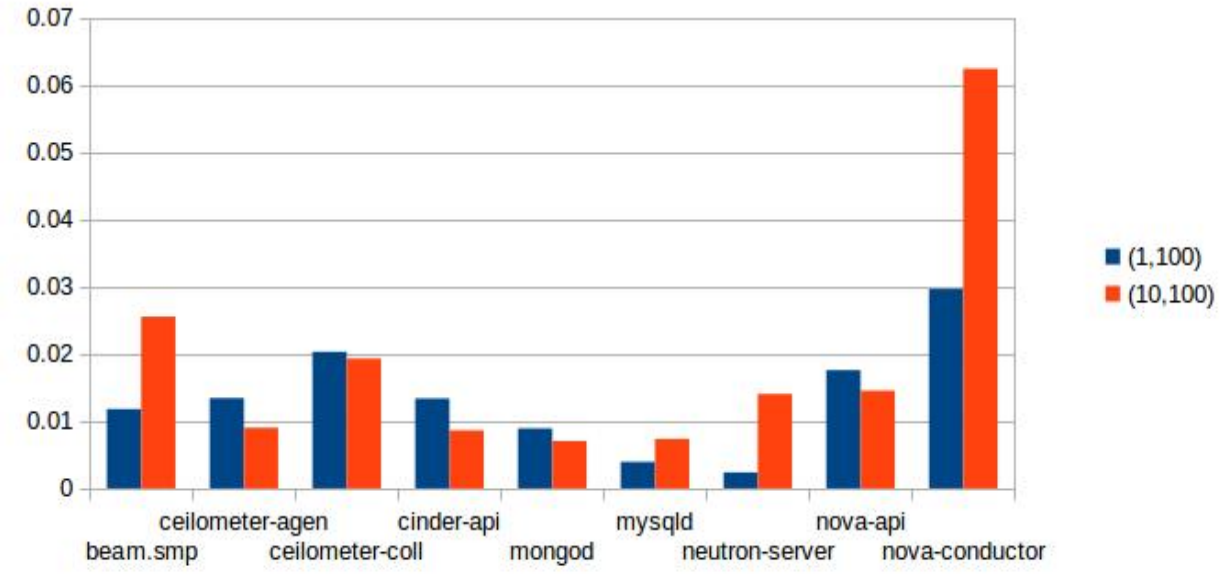

Figure 11, Top process usage comparison with 100 VMs

I also compared each OpenStack set-up individually based on their number of VMs. From Figure 12 and 13, it can be concluded that the process usage is increased by number of VMs. Therefore the two scenarios with 100 VMs have more process usage than the once without Virtual Machines.

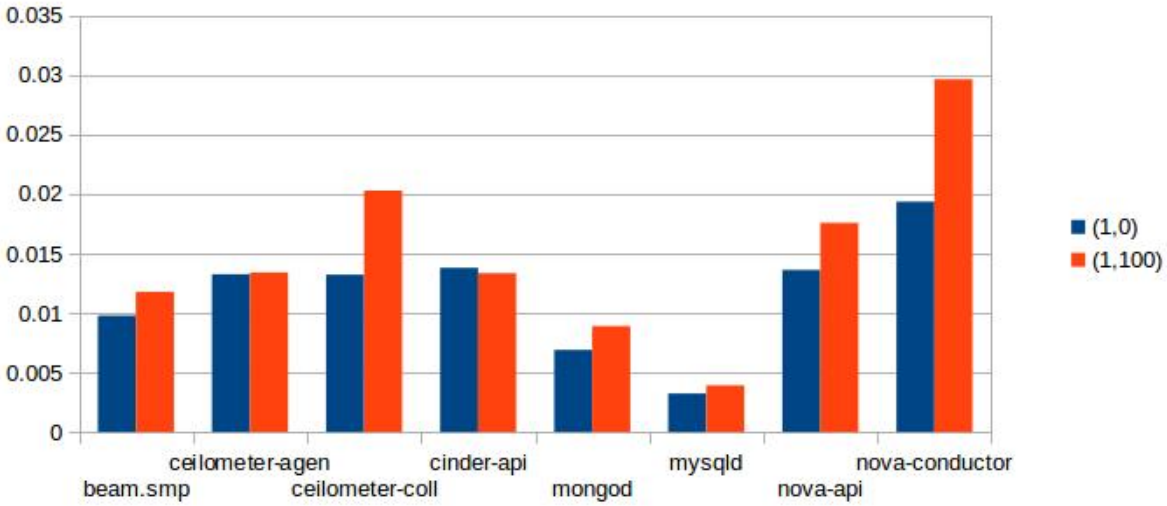

Figure 12, Top process usage comparison with single compute node

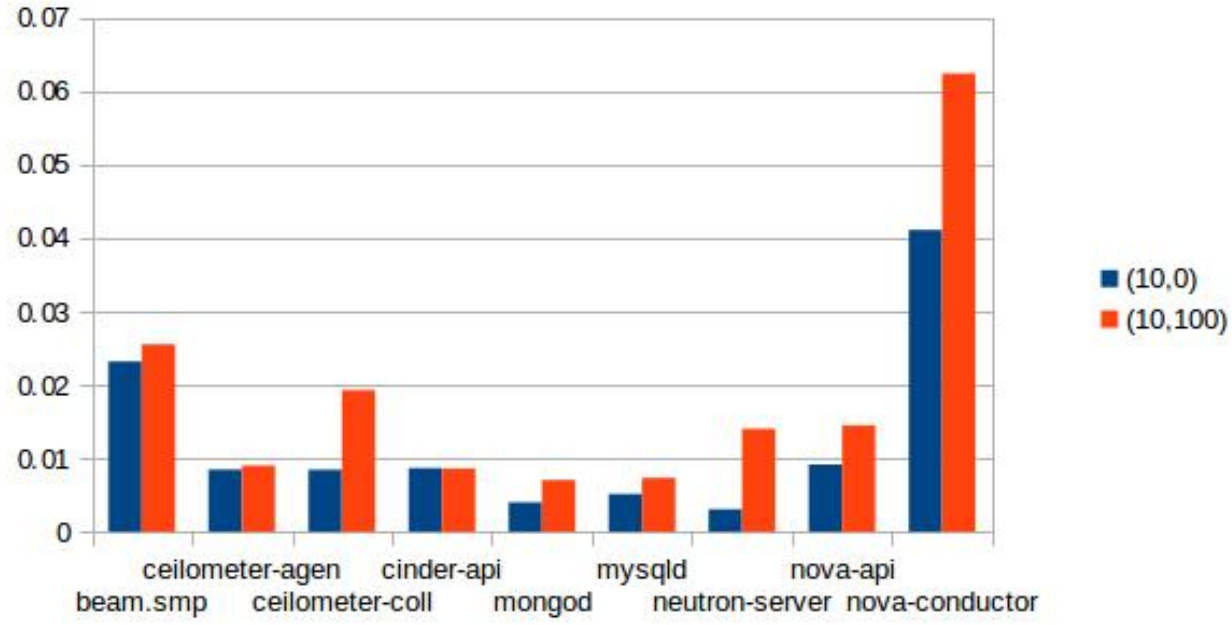

Figure 13, Top process usage comparison with ten compute nodes

3.7 Per API call Experiments

In this set of experiments, OpenStack is monitored based on different API calls. We collect the data by monitoring the API calls after calling them for a specific period of time. After collecting the data, the idle part got extracted to get the pure and accurate information regarding the effect of API calls on the CPU.

There are several API calls to measure, due to the limited time and scope of the thesis, it was impractical to measure and experiment all of them. Thus, I focused on working the most important API calls, which are creating or starting a VM, deleting or stopping a VM.

3.7.1 Per API Experiment 1

Table 11 shows the OpenStack settings for the Per API Experiment with 0 VMs for two different settings with single compute node and 10 compute nodes.

In the Experiment 1, it is shown how the CPU usage got affected via different API calls. The Top 5 API calls which have the most effect on the CPU usage are: Create VMs, Attach Volumes to

VMs, Detach Volumes from VMs, Delete VMs and Delete Volumes (Figure 14). Other API calls didn't have considerable effect on CPU consumption. It is also demonstrated in figure 14 that API calls have more effect on CPU consumption in Scenario 2 with 10 compute nodes than with single compute node.

Table 11. Per API Experiment 1

Scenario 1 (1, 0) Features	
Controller Node: 4 cores	**Service Node:** 4 cores
Compute Node: 4 cores	**Number of Compute Nodes:** 1
Number of Virtual Machines: 0	**Measurement length:** 30 seconds
Scenario 2 (10, 0) Features	
Controller Node: 4 cores	**Service Node:** 12 cores
Compute Node: 12 cores	**Number of Compute Nodes:** 10
Number of Virtual Machines: 0	**Measurement length:** 30 seconds

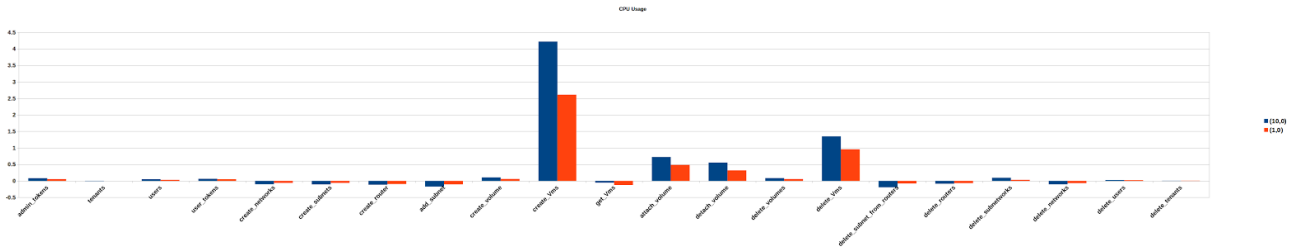

Figure 14, CPU Usage Per API Call With 0 VMs

To get a better understanding of the API calls, I also extract the resource consumption of the top processes in each API calls, which are demonstrated in Figures from 15 to 19.

Therefore, as it is demonstrated in figure 15 for create-VM API call, nova-conductor and neutron-server processes are the two main with the highest consumption.

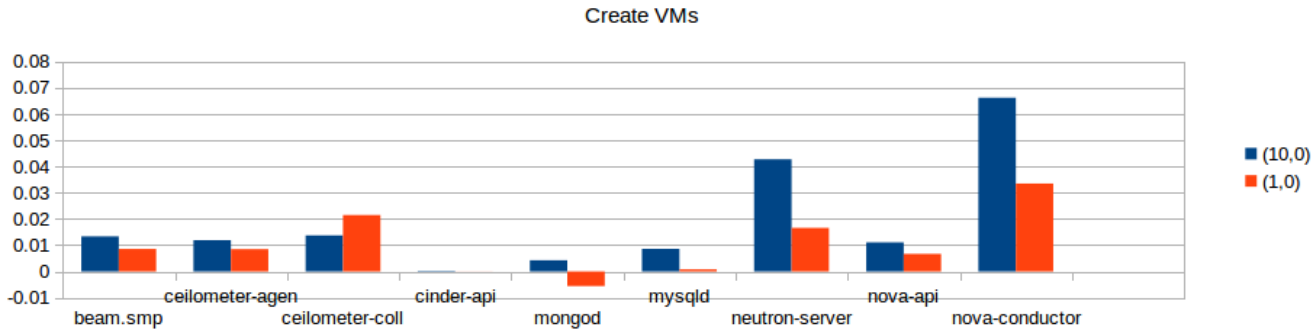

Figure 15, Top process usage comparison with 0 VMs based on Create VMs

As it is shown in figure 16, for stopping or deleting a VM, similar to creating a VM nova-conductor plays an important role in resource consumption. However, in deleting VM, ceilometer-coll is also consuming a significant part of the whole delete-vm API call resource.

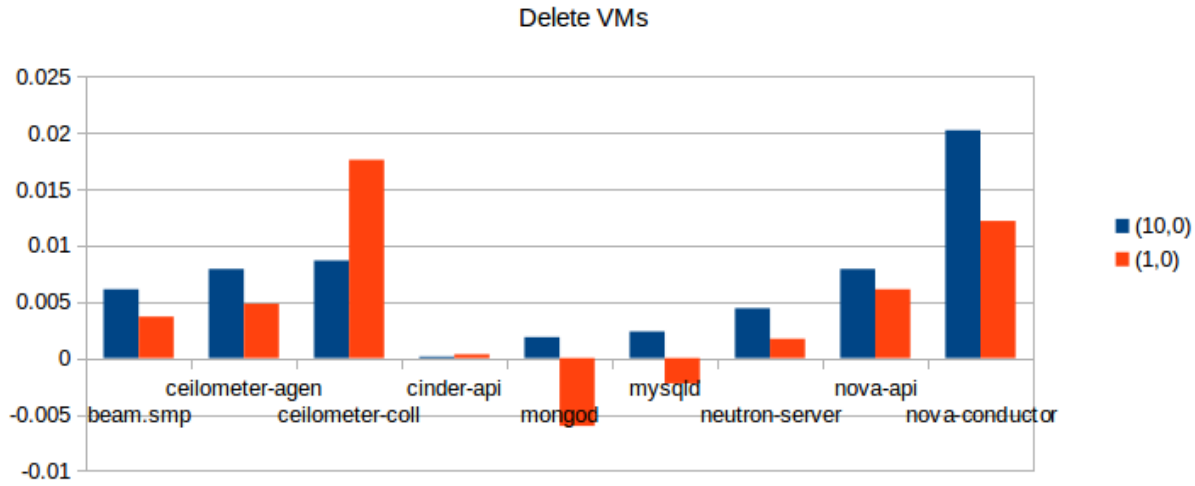

Figure 16, Top process usage comparison with 0 VMs based on deleting VMs

As it is shown in figure 17, attach volume to VM API call involves cinder, ceilometer and nova processes the most.

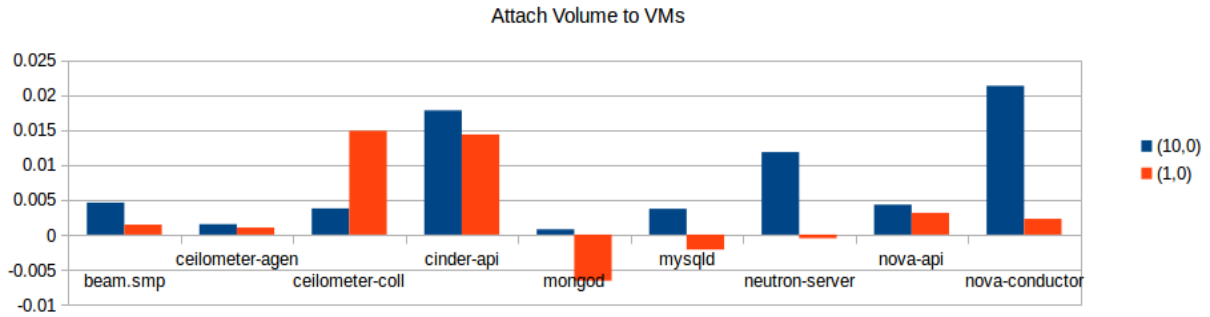

Figure 17, Top process usage comparison with 0 VMs based on attach volumes to VMs

In addition, for detaching volume from VM same OpenStack components consume the most, which is shown in figure 18.

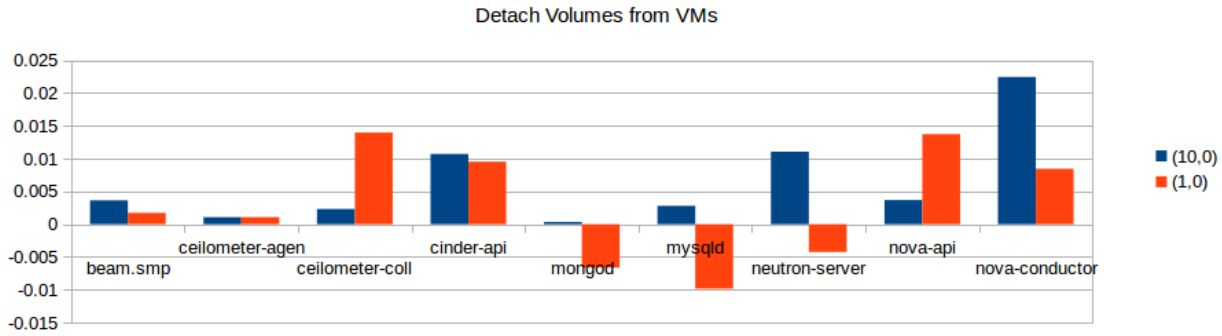

Figure 18, Top process usage comparison with 0 VMs based on detach volumes from VMs

In figure 19, there are some negative numbers, which in theory they are wrong, but as the experiments have some errors, they appeared in the results. The reason is that OpenStack is not stable in idle phase. Therefore, on the time of measuring OpenStack behavior in idle phase, some processes periodically consume resources, which may not have been appeared on the time of the API call measurements.

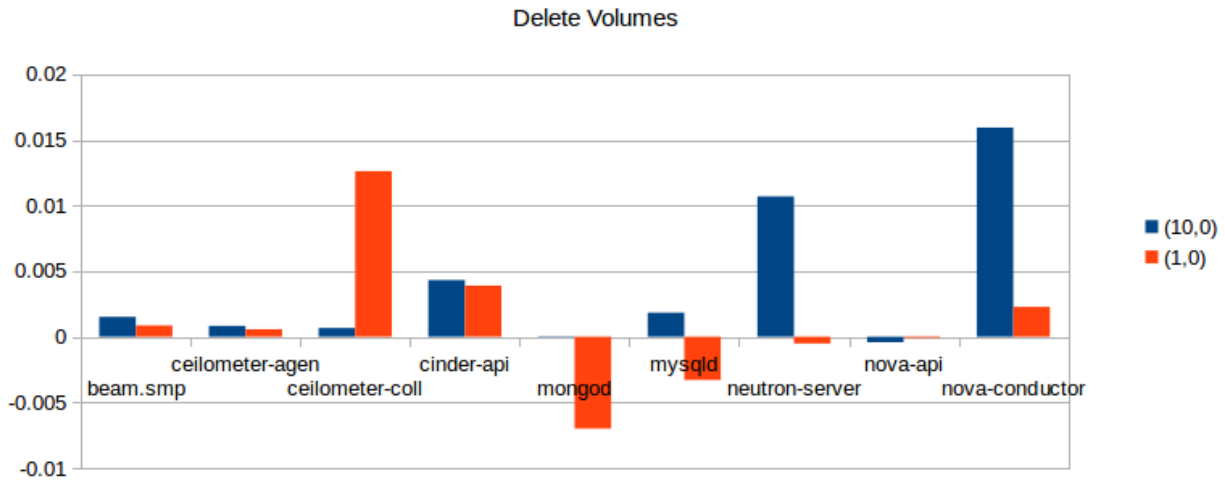

Figure 19, Top process usage comparison with 0 VMs based on deleting Volumes

3.7.2 Per API Experiment 2

Table 12 shows the OpenStack settings for the Per API Experiment with 0 VMs and 10 compute nodes.

Table 12. Per API Experiment 2

Scenario 3 (1, 100) Features	
Controller Node: 4 cores	**Service Node:** 4 cores
Compute Node: 4 cores	**Number of Compute Nodes:** 1
Number of Virtual Machines: 100	**Measurement length:** 30 seconds
Scenario 4 (10, 100) Features	
Controller Node: 4 cores	**Service Node:** 12 cores
Compute Node: 12 cores	**Number of Compute Nodes:** 10
Number of Virtual Machines: 100	**Measurement length:** 30 seconds

Experiment 2 is similar to the previous experiment with the difference of number of VMs in the OpenStack setting. The Top 5 API calls which have the most effect on CPU usage are: Create

VMs, Attach Volumes to VMs, Detach Volumes from VMs, Delete VMs and Delete Volumes (Figure 20)

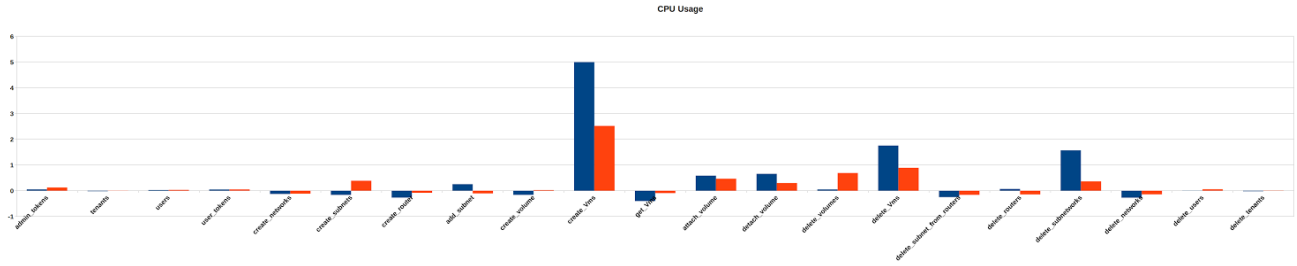

Figure 20, Usage Per API Call with 100 VMs

Similar to the previous experiment, the comparisons of the top processes, which consume most in each API calls, are shown in Figures from 21 to 25.

In figure 21, it shows how Creating a VM API affects different processes in CPU consumption. Neutron-server and nova-conductor are the two most consuming resources. As it is shown in figure 21, the scenario 4 processes are higher than scenario3 ones because of the number of VMs.

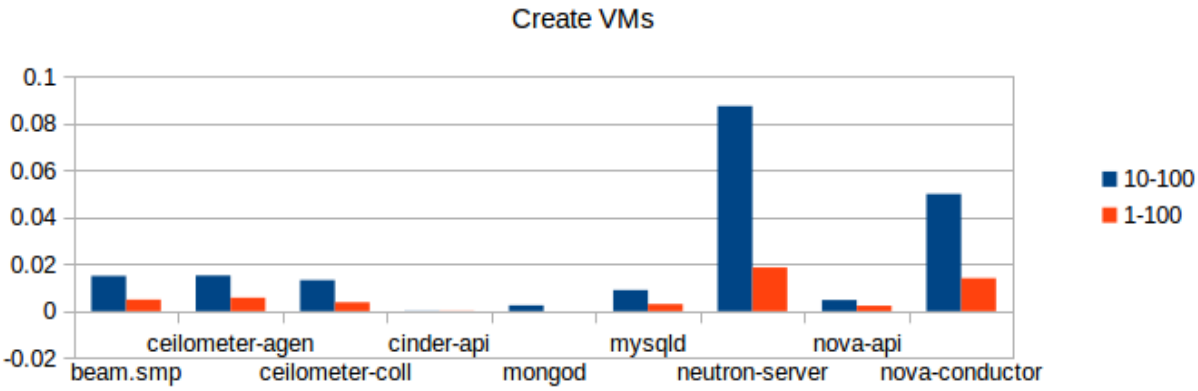

Figure 21, Top process usage comparison with 100 VMs based on attach volumes to VMs

For deleting a VM, nova-conductor has the highest impact on resource consumption. Similar to create a VM API call, number of compute nodes also increases the resource consumption.

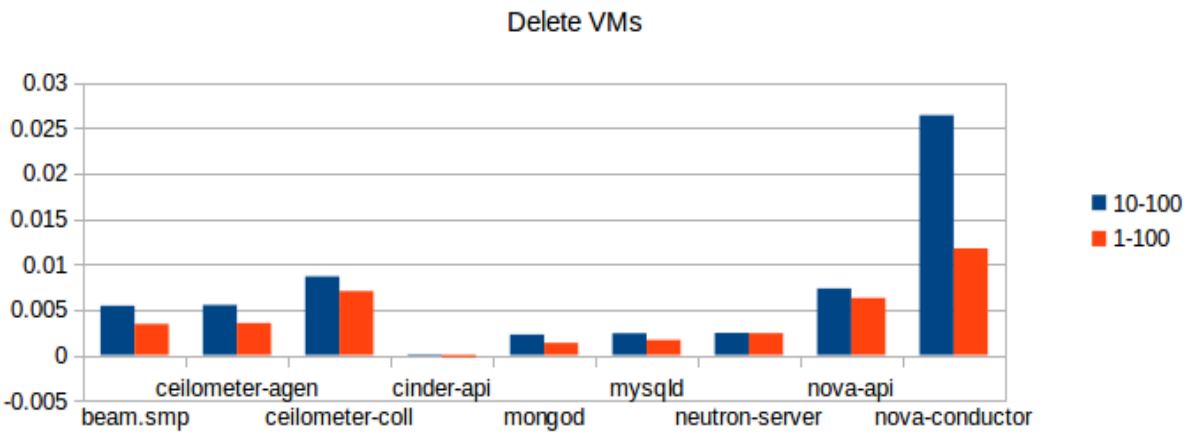

Figure 22, Top process usage comparison with 100 VMs based on deleted VMs

As it is shown in figure 23 and figure 24, results for attach and detach volume API calls are not quite what are expected in theory. All the processes of scenario 4 should have more impact than scenario 3. However, due to the noise and complicated behavior of OpenStack, the results are not accurate.

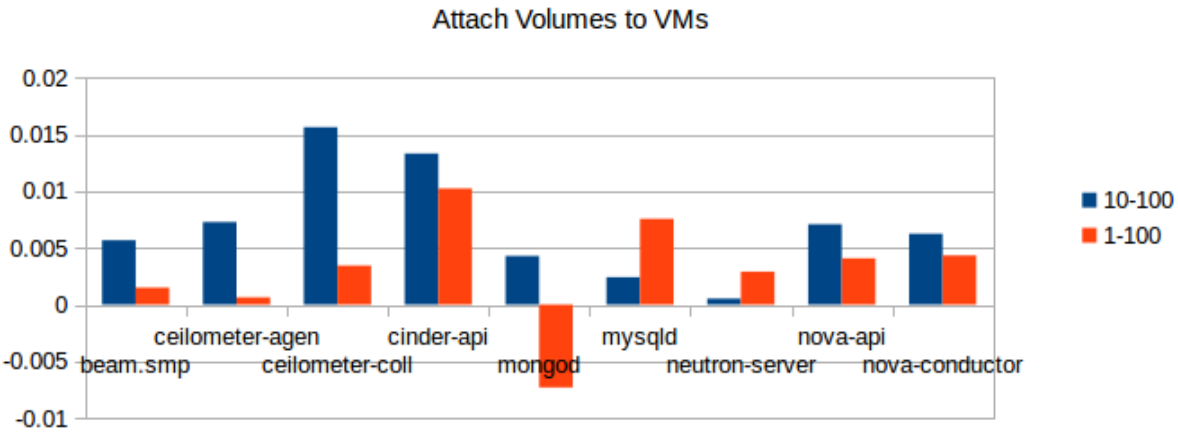

Figure 23, Top process usage comparison with 100 VMs based on attach volumes to VMs

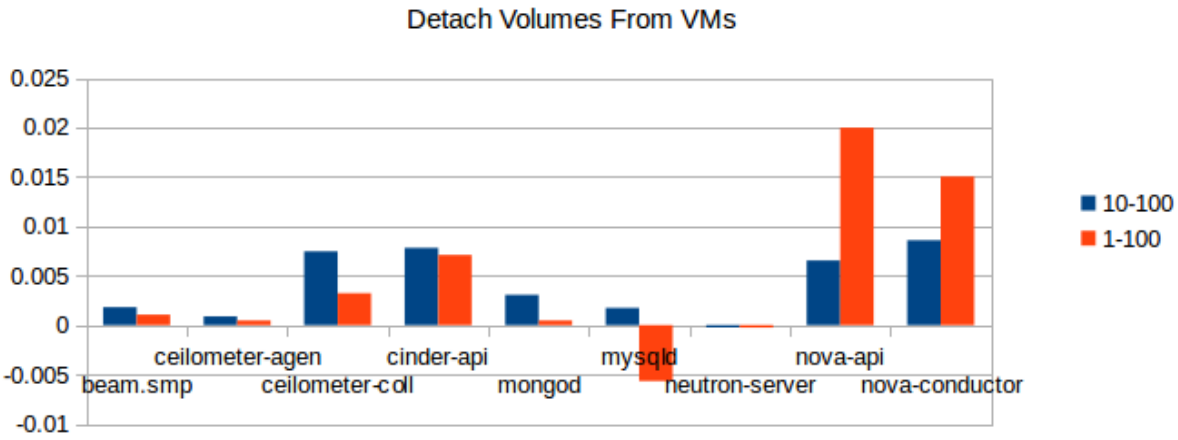

Figure 24, Top process usage comparison with 100 VMs based on detach volumes from VMs

For delete volumes API that is shown in figure 25, ceilometer and nova components have the most impact in CPU utilization.

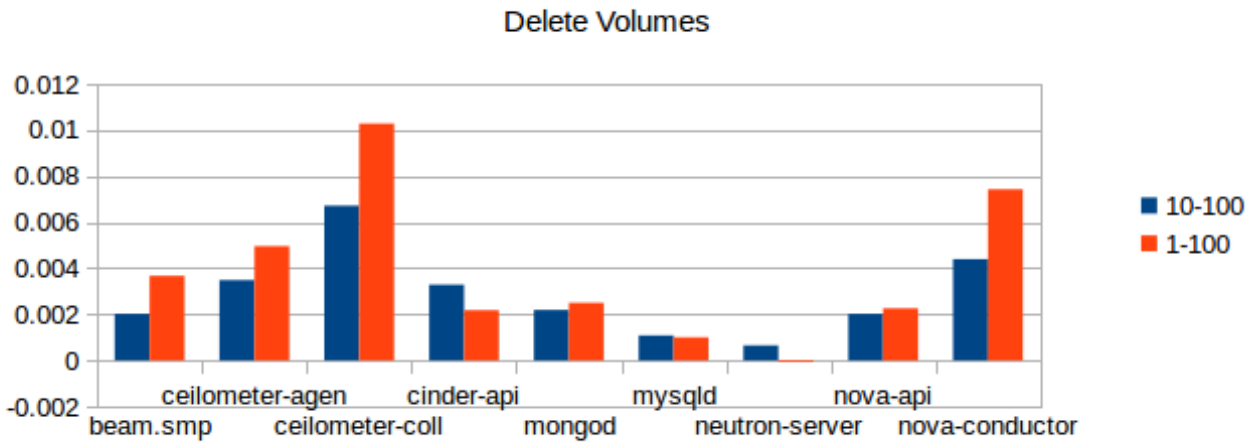

Figure 25, Top process usage comparison with 100 VMs based on deleted Volumes

4 Analysis and Evaluation of results

In the evaluation section, we use the data from the experiments to analyze and model the results. The idle approach to get the optimum results from the experiments is to run as many experiments as possible for different values of compute nodes and VMs and then measure how much the idle resource utilization changes in each scenario and also measure how much each API call of interest consumes when it is being called. In the end, when you have the assumption data points, you can do the curve fitting and get a function to estimate the resource consumption. However, due to the limitation of thesis scope, the evaluation and modeling is based on simple scenarios that are described in previous section.

Thus, after experimenting with different API calls, I decided to measure the start VM and stop VM API calls since they have larger impact on resource utilization and also it is easier to detect them. However, other API resource were masked under the noise of the OpenStack processes, so they didn't get included in the measurements

The experiments are measured on 3+n OpenStack Deployment. The results are shown in figure 26, 27 and 28. In figure 26, it shows that how much resource has been consumed when the system is in idle mode. Thus, as it is been shown, the resource utilization increases as number of VMs and compute nodes increase. In theory, it is expected to have linear correlation between idle resource consumption, VMs, and compute nodes, but due to the error, it has a slight curve. Therefore, the mathematical curve fitting [29] approach has been applied for the graphs for better understanding the behavior. Therefore, to calculate the curve fitting, MATLAB's curve-fitting library [30] is used.

After doing the curve fitting on the idle experiments the approximated linear curve is formulated as:

$$\boldsymbol{Idle(C,V)} = 0.00779 * \boldsymbol{C} + 0.0007699 * \boldsymbol{V} + 0.1224$$

C : # of Compute Nodes

V : # of Virtual Machines

This curve is illustrated in a 3D format in the figure 26. For Idle phase the linear model is used because the results had linear behavior. One of the ways to choose the model is to check the sum

of squared errors of prediction (SSE) [31], which shows the accuracy of the model. The SSE for the linear model was 0.03119, which is close to zero. If the SSE is more than one, the linear model for this experiment wouldn't be suitable.

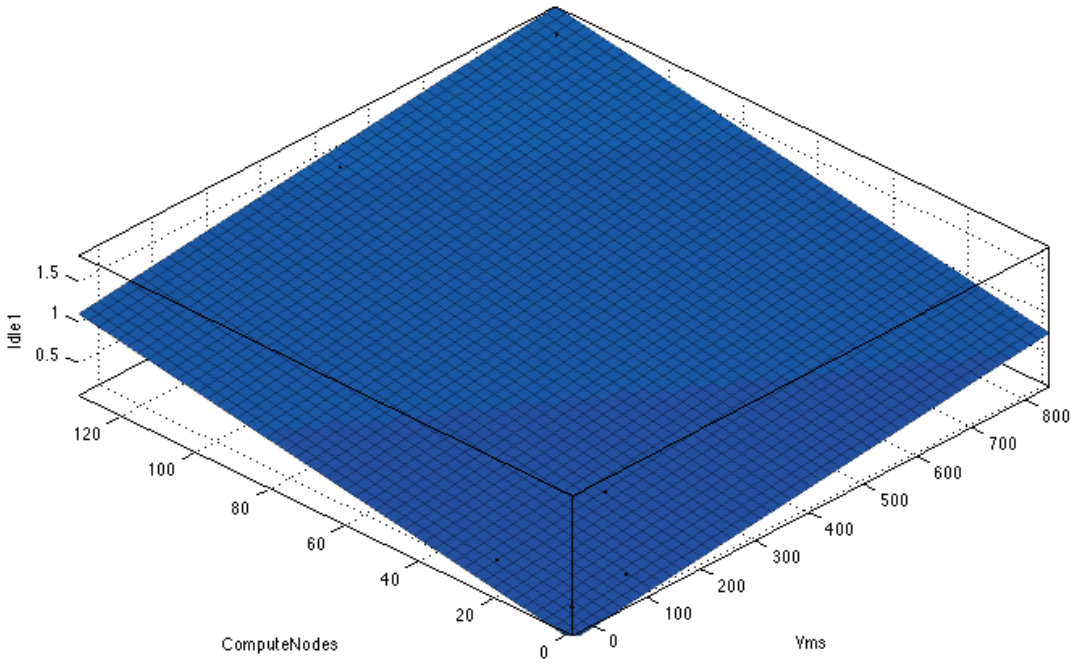

Figure 26, Idle experiments for modeling

For starting a VM API call, the curve fitting approach provides the second-degree formulation as follows, since it had more complicated behavior and linear model was not a suitable model for that because of high error. The SSE for the linear model is 3.075, however, for the polynomial model is 0.0002639. After doing the curve fitting on the idle experiments, the result is:

$$\boldsymbol{StartVM}\ (\boldsymbol{C},\boldsymbol{V}) = \mathbf{2.397} + \mathbf{0.0792} * \boldsymbol{C} - \mathbf{0.00396} * \boldsymbol{V} - \mathbf{0.0005435} * \boldsymbol{C}^{\mathbf{2}} + \mathbf{4.823}\boldsymbol{e}^{-\mathbf{5}} * \boldsymbol{C} * \boldsymbol{V} + \mathbf{7.4}\boldsymbol{e}^{-6} * \boldsymbol{V}^{\mathbf{2}}$$

C : # of Compute Nodes

V : # of Virtual Machines

Where in figure 27 it shows that how much resource has been consumed when the Start VM API is called.

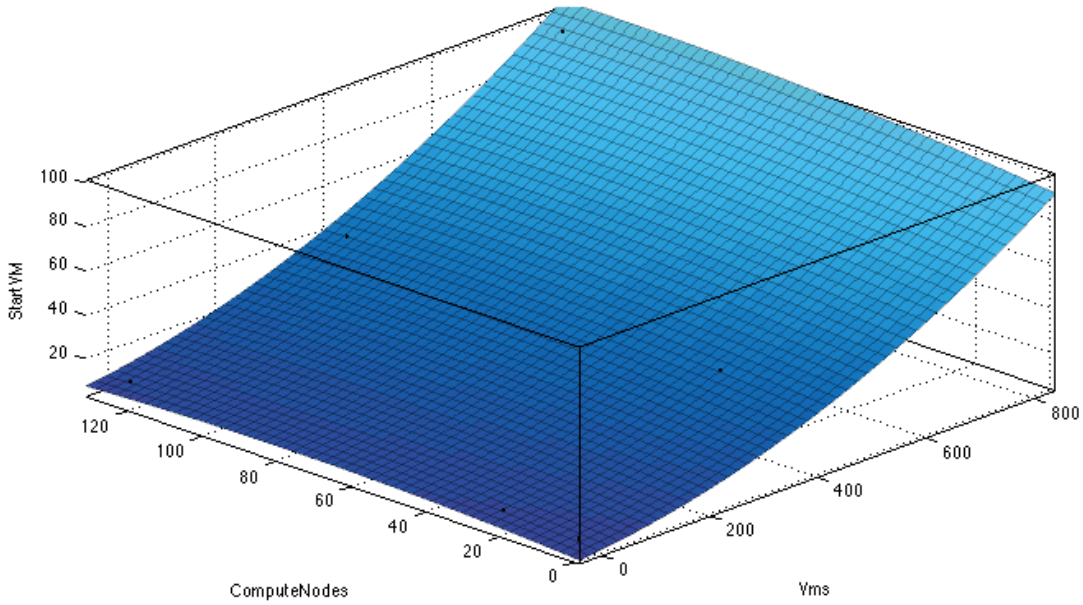

Figure 27, Start VM API call experiments for modeling

For stopping VM, the best model was a polynomial model with the second-degree formulation. However, due to the noise the results are not highly accurate and it had a considerable error, which was 0.9803. To get more accurate results, more experiments were needed to measure, but because of the thesis scope and time limit, it didn't happen. Therefore, the final model after curve fitting is:

$$\boldsymbol{StopVM}\ (\boldsymbol{C},\boldsymbol{V}) = -1.127 + 0.05721\ *\boldsymbol{C} + 0.01556\ *\boldsymbol{V} + 0.0002112\ *\boldsymbol{C^2}\ +\ \boldsymbol{4}.445\boldsymbol{e^{-5}} * \boldsymbol{C} * \boldsymbol{V} + 1.5e^{-5} * \boldsymbol{V^2}$$

C : # of Compute Nodes

V : # of Virtual Machines

Therefore, in figure 28 it shows that how much resource has been consumed when the Stop VM API is called.

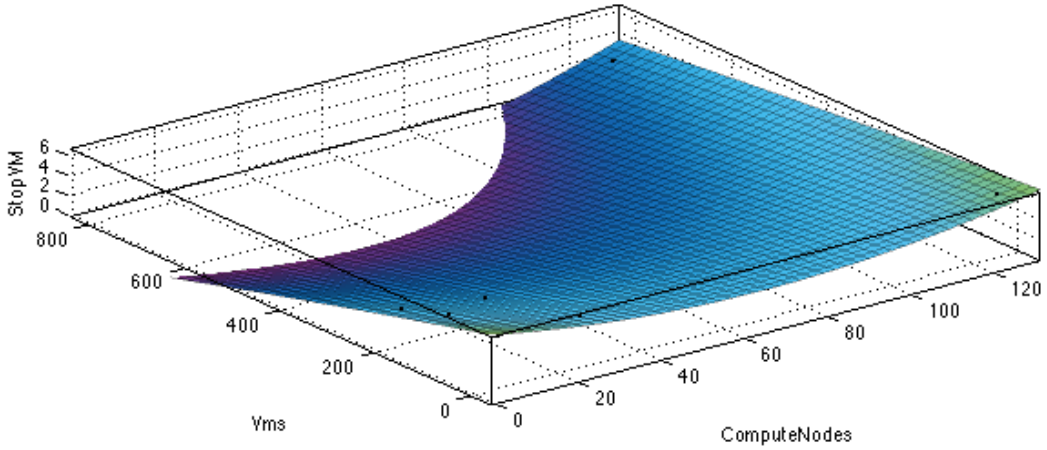

Figure 28, Stop VM API call experiments for modeling

Thus, for the idle approach we have relatively solid model and results because of linear method and low SSE. For Start VM and Stop VM scenarios, polynomial curve fitting approach is more suitable because of low SSE. In order to get more accurate results, the number of scenarios and experiments should increase.

5 Conclusion and Future work

The purpose of this Master thesis project is to model the CPU resource utilization in OpenStack Controller. For this purpose, a lot of research has been made to find a way to calculate the resource consumption and create a model out of it. To be able to tackle the problem, after careful study of setup and installation of Open Stack Juno on Virtual environment, the relevant parameters of the main component has been changed and the output has been monitored in details. The CPU resource consumption has been selected to model due to the less noise measures comparing other outputs. It has been found out that starting and stopping VM has the highest effect on CPU resource consumption. The CPU resource consumption has been selected to model due to the less noise measures comparing other outputs. It has been found out that starting and stopping VM has the highest effect on CPU resource consumption. Among those, starting and stopping the VM had the highest effect on CPU consumption. Therefore, more experiments were done to measure them in different OpenStack settings. At the final stage of this work, a mathematical model of curve fitting with MATLAB both linear and nonlinear has been proposed for the CPU resource consumption behavior based aggregated data

The model takes as input the test-bed measurement results and produces an analytical model that can predict the resource demand of the various OpenStack components under different operating conditions. With the help of this model, it is possible also to have different configurations given an envelope operating condition for a specific cloud deployment. Many different approaches to utilize the resource consumption have been studied as a part of technical background to model the Controller node of OpenStack and then predict the needed resources such as number of VMs and number of compute nodes based on the model.

This approach is still a very simplified solution to predict OpenStack controller resources, due to the limited resources of this thesis work to a specific OpenStack configuration. Besides that, the level of possible errors of the experiment based on VMs instead of real instruments should also take in to the account of this work. Although there are some test cases where the system does not seem to give correct results or show a *False Positive*, it is safe to say that the overall performance of the application is acceptable in Idle, StartVm and StopVM API calls scenarios. The naive answer to the question "how accurate the prediction of the model is?" is “it can be

better". Of course there is always room for more improvement. If in some way the limitations of the system such as a relatively small dataset can be removed the efficiency would increase by several times.

Considering this Master thesis project a proof of concept, there is a lot to be discussed at part of the Future work and improvements. For now, the model is just created based on some small experiments and is not tested. One of the future works can be focused on testing and improving the model. The model should be self-trained based on experiments. In addition to that, the experiments should also be executed in the physical environment.

References

[1] Furht, Borivoje, and Armando Escalante. Handbook of cloud computing. Vol. 3. New York: Springer, 2010.

[2] Armbrust, Michael, et al. "A view of cloud computing." Communications of the ACM 53.4 (2010): 50-58.

[3] Sehgal A. Introduction to OpenStack. Running a Cloud Computing Infrastructure with OpenStack, University of Luxembourg. 2012.

[4] Jackson K, Bunch C, Sigler E. OpenStack cloud computing cookbook. Packet Publishing Ltd; 2015 Aug 19.

[5] Sefraoui O, Aissaoui M, Eleuldj M. OpenStack: toward an open-source solution for cloud computing. International Journal of Computer Applications. 2012 Jan 1;55(3).

[6] Takako,P., Estcio,G., Kelner,J., Sadok,D. (2010) . A Survey on Open-source Cloud Computing Solutions.WCGA - 8th Workshop on Clouds, Grids and Applications.Gramado:28 May, 3-16. Teyssier,S. (2010).

[7] Rosado T, Bernardino J. An overview of openstack architecture. InProceedings of the 18th International Database Engineering & Applications Symposium 2014 Jul 7 (pp. 366-367). ACM.

[8] Kumar R, Gupta N, Charu S, Jain K, Jangir SK. Open source solution for cloud computing platform using OpenStack. International Journal of Computer Science and Mobile Computing. 2014 May;3(5):89-98.

[9] Song Y, Kang J. Analysis of Immersive Media Platforms focus in OpenStack. International Journal of Advancements in Computing Technology. 2013 Sep 1;5(13):452.

[10] Toraldo G. OpenNebula 3 Cloud Computing. Packt Publishing Ltd; 2012.

[11] Nasim R, Kassler AJ. Deploying openstack: Virtual infrastructure or dedicated hardware. InComputer Software and Applications Conference Workshops (COMPSACW), 2014 IEEE 38th International 2014 Jul 21 (pp. 84-89). IEEE.

[12] Beloglazov A, Buyya R. OpenStack Neat: a framework for dynamic and energy-efficient consolidation of virtual machines in OpenStack clouds. Concurrency and Computation: Practice and Experience. 2015 Apr 10;27(5):1310-33.

[13] Sobeslav V, Komarek A. Opensource automation in cloud computing. InProceedings of the 4th International Conference on Computer Engineering and Networks 2015 (pp. 805-812). Springer International Publishing.

[14] García ÁL, del Castillo EF, Fernández PO. ooi: OpenStack OCCI interface. SoftwareX. 2016 Jan 22.

[15] Lei, Qing, Yingtao Jiang, and Mei Yang. "Evaluating Open IaaS Cloud Platforms Based upon NIST Cloud Computing Reference Model." Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on. IEEE, 2014.

[16] Nasim, Robayet, and Andreas J. Kassler. "Deploying openstack: Virtual infrastructure or dedicated hardware." Computer Software and Applications Conference Workshops (COMPSACW), 2014 IEEE 38th International. IEEE, 2014.

[17] Ericsson Research, Kista, Stockholm, Sweden

[18] Jackson K, Bunch C, Sigler E. OpenStack cloud computing cookbook. Packt Publishing Ltd; 2015 Aug 19.

[19] Yamato Y, Katsuragi S, Nagao S, Miura N. Software Maintenance Evaluation of Agile Software Development Method Based on OpenStack. IEICE TRANSACTIONS on Information and Systems. 2015 Jul 1;98(7):1377-80.

[20] Anne Håkansson, Portal of Research Methods and Methodologies for Research Projects and Degree Projects. WORLDCOMP'13 - The 2013 World Congress in Computer Science, Computer Engineering, and Applied Computing, 22-25 July, 2013 Las Vegas, Nevada; USA

[21] García ÁL, del Castillo EF, Fernández PO. ooi: OpenStack OCCI interface. SoftwareX. 2016 Jan 22.

[22] Fifield T, Fleming D, Gentle A, Hochstein L, Proulx J, Toews E, Topjian J. OpenStack Operations Guide. " O'Reilly Media, Inc."; 2014 Apr 24.

[23] Mell, Peter, and Tim Grance. "The NIST definition of cloud computing." (2011).

[24] Plouffe J, Davis SH, Vasilevsky AD, Thomas III BJ, Noyes SS, Hazel T, inventors; Oracle International Corporation, assignee. Distributed virtual machine monitor for managing multiple virtual resources across multiple physical nodes. United States patent US 8,776,050. 2014 Jul 8.

[25] Gong S, Sim KM. CB-Cloudle and cloud crawlers. InSoftware Engineering and Service Science (ICSESS), 2014 5th IEEE International Conference on 2014 Jun 27 (pp. 9-12). IEEE.

[26] Wuhib F, Yanggratoke R, Stadler R. Allocating compute and network resources under management objectives in large-scale clouds. Journal of Network and Systems Management. 2015 Jan 1;23(1):111-36.

[27] Cernak J, Cernakova E, Kocan M. Performance testing of distributed computational resources in the software development phase.

[28] Iyer EK, Krishnan A, Sareen G, Panda T. Analysis of dissatisfiers that inhibit cloud computing adoption across multiple customer segments. InProceedings of the European Conference on Information Management & Evaluation 2013 May 13 (pp. 145-152).

[29] Lancaster, Peter, and Kestutis Salkauskas. Curve and surface fitting. Academic press, 1986.

[30] Schulz MN, Landström J, Hubbard RE. MTSA—A Matlab program to fit thermal shift data. Analytical biochemistry. 2013 Feb 1;433(1):43-7.

[31] Montgomery DC, Peck EA, Vining GG. Introduction to linear regression analysis. John Wiley & Sons; 2015 Jun 29.

[32] OpenStack document, http://docs.openstack.org/icehouse/training-guides/content/operator-getting-started.html

[33] Sefraoui O, Aissaoui M, Eleuldj M. OpenStack: toward an open-source solution for cloud computing. International Journal of Computer Applications. 2012 Jan 1;55(3).