Optimization of Computer Resources Using OpenStack Private Cloud

Agustinus Noertjahyana
Petra Christian University
Siwalankerto 121-131
Surabaya, Indonesia
+6282233442989
agust@petra.ac.id

Henry Novianus Palit
Petra Christian University
Siwalankerto 121-131
Surabaya, Indonesia
hnpalit@petra.ac.id

Daniel Kuntani
Petra Christian University
Siwalankerto 121-131
Surabaya, Indonesia

ABSTRACT
The rapid development of computer technology and applications has caused the increasing need for computer use in an organization. The performance of personal computers has increased rapidly, but they cannot be utilized properly. Therefore, personal computers can be used together to perform parallel processing, in order to shorten data processing time. This paper discusses Private cloud, which could be utilized as an alternative solution to provide cloud services to organizations that are in the same network. The service provided was Infrastructure as a Service (IaaS), where users can request certain types of services and use certain operating systems.

Most universities have many computers in the laboratory that have considerable resources for a superb processing potential if they optimized properly. Supported by high specification on each computer, the laboratory is suitable for private cloud implementation. The use of the OpenStack framework can produce sufficient performance to perform parallel data processing. The test results showed that the parallel processing performance worked as expected.

CCS Concepts
Computer systems organization → Architectures → Distributed architectures → Cloud computing: 500

Keywords
Computer Resources; IaaS; OpenStack; Private Cloud.

1. INTRODUCTION
Application development is in line with the development of computing technology, which is very fast. As a result, distributed applications and parallel processing require large computer resources. Cloud computing is a system that combines several computers into a unified system with the availability of more processors and memory [1]. Although the technology is up-to-date, according to IBM Research Report, CPU usage at various Data Centers is very low, ranging from 7% to 25% with an average of 18%. This means that computers in various Data Centers use more processor specifications to perform activities on the Data Center [2].

In a university, computers with the latest specifications are also not utilized optimally. Therefore, computer resources can be used to perform parallel processing using cloud computing. The OpenStack framework, as an alternative to cloud computing, is an excellent tool to test computer performance simultaneously[3-4].

WSSE 2019, September 20–23, 2019, Wuhan, China

ACM ISBN 978-1-4503-7213-8/19/09…$15.00
DOI: https://doi.org/10.1145/3362125.3362138

2. THEORY
2.1 Cloud Computing
Cloud computing is a computational model enabling scattered, convenient and on-demand computer network access to share configurable computer resources, such as networks, servers, storage, applications and services [5]. Cloud computing has 3 service models offered: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) [4]. Computers in the same network can perform parallel processing using the OpenStack framework; OpenStack provides Infrastructure as a Service (IaaS).

2.2 OpenStack
OpenStack is a cloud system software platform that acts as a middleware. Middleware is a software layer that provides programming abstraction and combines the various layers below, such as networks, hardware, operating systems and programming languages [6]. This means that OpenStack is a service that provides hardware capabilities and is a part of Infrastructure as a Service (IaaS) OpenStack consists of various components that communicate with each other to provide services to users.

2.3 Keystone
Keystone provides identity service and access policy services to all components in OpenStack. Keystone also implements REST-based APIs as identity.

Keystone provides authentication and authorization for all OpenStack components. Authentication verifies that the actual request is coming from the correct source. There are two authentication methods, namely username / password-based and Token based. Authorization verifies user permissions to the requested service [7]. Components inside the Keystone consist of tenant, service endpoint, region, user and role [7].

2.4 OpenStack System
Registered users can access the IaaS feature provided by the cloud system through the user interface of the website. Users can request a virtual machine by selecting a flavor (specs such as CPU, RAM and disk capacity) and image (operating system) [8].

The controller node, where the website is located, will perform the scheduling, which selects the IP address for the virtual machine

and compute nodes that meet the criteria according to the capacity requested by the cloud user. At this stage, various OpenStack services will work together to check user rights over hardware, image, network and storage before running an instance (virtual machine).

A number of instances that run will be on a virtual network that is formed on the host where the virtual machine is operating. It is as if the number of instances is a set of nodes in a cluster.

The user already has a virtual environment created from a collection of virtual machines. Each virtual network formed is connected to the network bridge on the host's physical interface so that it can connect to the network outside the virtual network, in this case, the private network belonging to Petra Christian University.

The user as the instance owner must access the instance through the console interface provided on the website only at the first instance of the instance. This step is the stage where the cloud users do the initial setup of the operating system, as well as the installation of other programs if desired.

Users can enable connections on certain ports according to the security group, a type of OpenStack's firewall. Furthermore, a direct connection can be done to the instance without going through the web interface.

3. SYSTEMS DESIGN

OpenStack required 2 networks: Management Network and Public Network. Management Network was used to provide a line of communication systems and cloud administration. Public Network provided access from cloud users; in case of private cloud, this network was a private network (or also called Flat Network). Flat network used the available arrangement, which was a private network (192.168.32.0/248). The network (172.16.1.0/24) was then shared into a management network and partly for use on a virtual machine that would operate on a cloud system, which was 172.16.0.0/24. The network design can be seen in Figure 1.

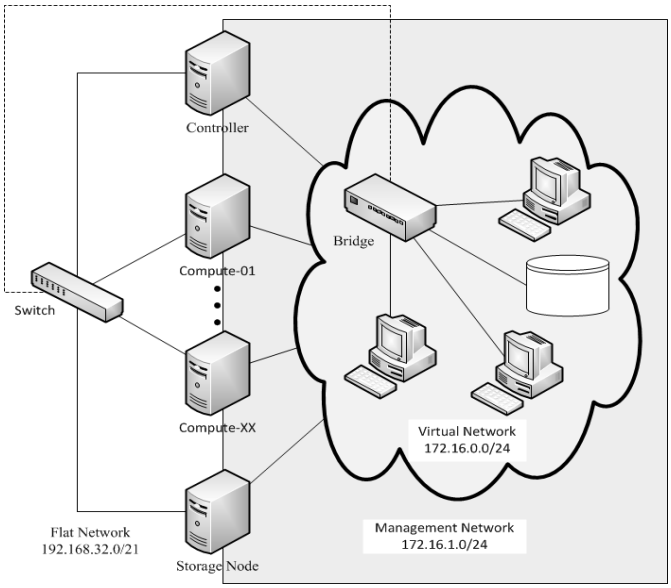

Figure 1. Network Design.

3.1 Basic Environment

The basic environments needed to build an OpenStack system were nodes with Ubuntu Server 14.04 LTS operating system. For each node, the hardware specification was as follows:

Processor: Intel Core i5-3340 @ 3.1 GHz

(2 cores / 4 threads)

RAM: 16 GB

Disk: 250 GB

Connection: 1 interface 100Mbps Ethernet

In addition, the installation process required the basic components needed by OpenStack. The basic components were as follows:

OpenStack Packages

For OpenStack packages installation, Juno cloud repository should be added to the source-list of Advanced Package Tool (APT). The APT on each node had to be updated and upgraded.

MariaDB

The Database Management System (DBMS) was mandatory because it was used to store information for every service running on OpenStack. The DBMS installation was performed on the node controller. MariaDB was an open source DBMS that was suggested in making the cloud using OpenStack since the Juno version.

MariaDB had many performance optimizations compared to MySQL in performing simple query execution. So MariaDB was able to handle more SQL commands in units of time.

RabbitMQ

RabbitMQ was a message broker, which received and continued messages. The RabbitMQ installation was performed on the node controller. In RabbitMQ, all components of OpenStack (services) acted as guest. The settings on RabbitMQ were not changed, except for the guest password.

Network Time Protocol (NTP)

NTP installation was done to all nodes residing on the private cloud system. However, the role of the node controller differed from that of the other nodes. NTP on the node controller had a function, to provide timing to all other nodes in a private cloud system. For that reason, the NTP setting on the node controller must use itself as the time provider and answer the NTP request from another node.

On the nodes other than the controller, the NTP was directed to adjust timing settings to the time provided by the node controller, so all nodes in the private cloud system had the same time settings as the node controller.

3.2 Application Network

The framework used was Juno's OpenStack version, a number of components that operated on OpenStack-based cloud systems. These components were separated into several nodes: a controller node with multiple compute nodes and additional nodes such as storage nodes and network nodes. Node design can be seen in Figure 2.

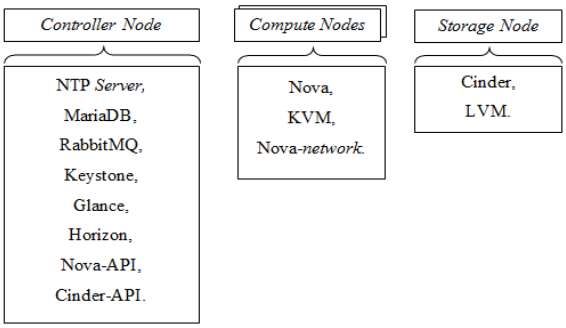

Figure 2. Node Design.

3.3 Compute Node and Storage Node

The compute node acted as the host for the virtual machine run by the user. For that, the compute node had a hypervisor, KVM, a most widely used and tested virtual machine manager (VMM) for the OpenStack system.

Another arrangement was made to form the nova-network that assigns the IP address to the virtual machine. The Nova API was located on the node controller, such that the network and work settings and distribution were not performed on the compute node.

Storage node was an additional node that was added as a storage medium used as the virtual hard drive in a virtual machine. Before the arrangement took place, a partition was required as a storage medium. The partition provided on each computer was 250 GB, so the partition created was 200 GB by providing the remaining 50 GB for the Ubuntu system including root and swap.

The partition could be shared with the user and served as a removable disk between virtual machines. For its use, the user would need to allocate some capacity to be a virtual disk. Then, the virtual disk could be attached to a VM and the user could access the virtual disk via NFS using a VM in the form of a physical disk.

3.4 Virtual Machine Image

The designed private cloud had to have an image provided to the user. The image contained the operating system provided by the developer of the operating system and had been verified by OpenStack as a decent operating system, functioning as a virtual machine. The selected images were:

Windows Server 2012 R2 Standard Evaluation Edition
Ubuntu Server 14.04
Debian (CLI)
Windows XP 32-bit
Flavor was a template provided as a virtual machine specification. Flavor included CPU, RAM and disk capacity. The list of the Flavor can be seen in table 1.

Table 1. List of Flavor

Flavor	CPU Core	Memory Capacity (GB)	Disc Capacity (GB)
Tiny	1	1	20
Small	1	2	40
Medium	2	4	60
Large	2	8	100
XLarge	4	14	200

4. SYSTEM IMPLEMENTATION

Installation Process

The Implementation of OpenStack Framework used 8 computers, namely: Cont1 as Controller Node, Sto1 as Storage Node and Cm1, Cm2, Cm3, Cm4, Cm5 and Cm6 as Compute Node.

Each node included a Node Controller using 2 connected networks, for settings and virtual interfaces. The IP Address on the computer used the network configuration in Fig. 2. The IP address of the main interface was 192.168.38.xxx, where xxx was the computer number. The IP address of the virtual interface was 172.16.1.xxx, where xxx was the computer number.

For each node to recognize another node on the network, the IP address had to be logged into the IP Address-Hostname pair list in the /etc/hosts file for each node.

The OpenStack installation process was done by following the steps provided by OpenStack document. The OpenStack installation manual contained a complete guide to the installation process [8].

4.1 Error Handling

In the implementation process, there were several problems:

* Nova Service

Nova Service had a dependency on the database (MariaDB). Problems were found on the Controller Node. Nova services (including nova-cert, nova-consoleauth, nova-scheduler and nova-conductor) were first run by the operating system before the database service was successfully executed. This caused Nova Services to fail and stop. The solution was to change the script at the initializing service by the operating system (Ubuntu 14.04). The location of the script was in the /etc/init/directory. The addition of the script made Nova Service wait for the database before running the service.

* Cinder Error

Cinder or block storage divided a partition into small parts according to the user and gave it to the user. When the request was submitted to the cinder, the application could not share the partition and, thus, returned the error message.

This happened because the app did not have permissions to change the hardware settings. The solution was to provide cinder access as a super user, by editing the /etc/sudoers file on Storage Node (Sto1) at the end of the file with the command:

$$\text{Cinder ALL} = (\text{ALL}) \text{ NOPASSWD: ALL} \quad (1)$$

4.2 System Result

Testing was based on the base cache functionality that occurred in the system. Nodes that had cache for base image, needed spawning time between 5 to 10 seconds, while nodes that did not have cache took time depending on the size of the image used. The time required to run each image without using the cache can be seen in table 2.

Table 2. Image, Size and Spawning Time without Cache

Image name	Size (MB)	Spawning Time
Cirros-0.3.3-x86_64	13	10 – 15 sec
Ubuntu 14.04 *Trusty*	251	30 – 40 sec
Debbian Jessie 64-bit	447	40 – 60 sec
Windows XP	1.699	2 – 3 min
Windows *Server* 2012	16.780	25 – 30 min

The placement of the instance (Virtual Machine) could form several host sharing scenarios. The test used only 2 scenarios:

- An instance in a host
- Multiple instances in a host

Using these two scenarios, testing was done by calculating the time required by the system to run an instance until the operating system ran. Test results can be seen in Table 3.

Table 3. Test results from multiple instances

Image	Scenario 1 (s)	Scenario 2	
		Instances	Time (s)
Debian Jessie	31	4	60
Windows XP	30		90
Ubuntu 14.04	180		240
Windows Server 2012	720	2	660

4.3 System Benchmark

For performance testing using 3 benchmarks, namely: Sysbench, HPCC and Blender.

4.3.1 Sysbench

Sysbench is a benchmark tool and is used to calculate CPU, memory and I / O capabilities. Testing was done by making an instance closest to a node using the largest flavor (m1.xlarge). Testing aimed to determine the ability achieved by an instance compared with the actual host. Therefore, testing was performed against a host and an instance. Test results on CPU can be seen in Table 4.

Table 4. Sysbench result on CPU

Parameter	Instance	Host
Maximum prime number checked in CPU test	20000	20000
Test execution summary		
Total time	7.4611s	6.8509s
Total number of events	10000	10000
Total time taken by event execution	29.8209	27.3939
Per-request statistics:		
Min	2.63ms	2.69ms
Avg	2.98ms	2.74ms
Max	22.76ms	14.76ms
approx.95 percentile	2.90ms	2.75ms
Threads fairness:		
Events (avg/stddev)	2500.0000/37.93	2500.0000/15.78
Execution time (avg/stddev)	7.4552/0.00	6.8485/0.00

4.3.2 HPCC

HPCC (High Performance Computer Challenge) was a benchmark software used to measure the performance of a distributed system (cluster). HPCC is another name for LINPACK Benchmark, a benchmark software provided by top500.org that provides high-performance computer related statistics.

Before running the benchmark, the host used should form a cluster. The first step was to make SSH authentication without a password from one host to all other hosts in the cluster. After the cluster was formed, the installation process could proceed, and the test could be performed.

Testing was done by comparing the benchmark results using physical instances and hosts. The results of the measurement and performance comparison between instances and physical hosts generated using HPCC Benchmark can be seen in Table 5.

Table 5. HPCC benchmark result

No. Test	# Node	Core/ Node	Result (Gflops)		Overhead (%)
			Instance	Physical Host	
1	1	1	2.7670	2.7830	0.575%
2	1	2	4.6320	4.6600	0.601%
3	2	1	1.5180	1.5340	1.043%
4	1	4	7.6600	7.7780	1.517%
5	2	2	0.9660	1.0320	6.395%
6	4	1	0.8419	0.9171	8.199 %
7	2	4	1.1030	1.1720	5.887 %
8	4	2	0.7225	0.7478	3.383 %
9	8	1	0.7823	0.8512	8.094 %
10	4	4	0.5649	0.5951	5.074 %
11	8	2	0.5596	0.6031	7.212 %
12	8	4	0.5782	0.6130	5.676 %

Based on the data in Table 5, the performance of an instance / virtual machine was slightly lower against the physical host. On the other hand, each benchmark result using multi-nodes produced a low performance compared to a host. This means that the use of multi-nodes through the network did not improve application performance.

4.3.3 Blender

Blender is a software for rendering animations and can be used as a cluster render farm. Blender is a loose-couple software, which means breaking the animation into several frames separated by its rendering process on many computers.

Testing was done by using animation consisting of 50 frames. The number of nodes used was 5 pieces of virtual machine instance. The configuration used 1 master node and 4 slave nodes. The results of the tests performed can be seen in Table6.

Table 6. Blender benchmark result

No. Test	Node Slave	Rendering Time (s)
1	1	5,400
2	2	10,474
3	3	6,566
4	4	4,756

5. CONCLUSION

Implementing private cloud by utilizing computer resources has satisfactory results in several ways:
A good computer network is capable of supporting the performance of private cloud, although it still takes a long time to deliver an image. Image caching feature can help in solving this problem.

An instance on the system is able to achieve performance close to physical host performance, so the CPU capacity can be used optimally.

By using OpenStack as a private cloud storage, the existing computer can utilize the remaining storage capacity to be able to store images from several Operating Systems and other data.

In the next research, the private cloud that has been developed will be used to store user data so that it can optimize the storage capacity of all client computers.

The disadvantage of this private cloud is on network capabilities that do not support applications such as HPCC. These tight-coupled apps have dramatically reduced performance when run on more than a computer/node.

6. ACKNOWLEDGMENTS

This research project was sponsored by the Directorate General of Higher Education Indonesia on the scheme "Penelitian Terapan Unggulan Perguruan Tinggi" in 2019. Many thanks to Center of Research, Petra Christian University, Surabaya, Indonesia for the great wonderful piece.

7. REFERENCES

[1] J. Glanz. The Cloud Factories: Power, Pollution and the Internet. URI=http://www.nytimes.com/2012/09/23/technology/data-centers-was-te-vast-amounts-of-energy-belying-industry-image.html. 2012.

[2] R. Birke, L. Y. Chen, and E. Smirni. Data Centers in the Wild: A Large Performance Study. URI= http://domino.research.ibm.com/lib-rary/cyberdig. 2012.

[3] Li Z, Li H, X. Wang, and Li K. A generic cloud platform for engineering optimization based on OpenStack. Adv Eng Softw [Internet]. 2014. pp:42–57.

[4] O. Litvinski, and A. Gherbi. Experimental Evaluation of OpenStack Compute Scheduler. Procedia Comput Sci [Internet]. 2013. pp:16–23.

[5] P. Mell, and T. Grance. The NIST Definition of Cloud Computing. Gaithersburg, United State: National Institute of Standards and Technology. 2011.

[6] G. Coulouris, J. Dollimore, Kindberg, and G. Blair. Distributed Systems Concepts and Design. Boston: Addison-Wesley. 2012.

[7] A. Jha, D. Johnson, K. Murari, M. Raju, V. Cherian, and Y. Girikumar. OpenStack Beginner's Guide (for Ubuntu - Precise). 2012.

[8] OpenStack Foundation. OpenStack Installation Guide for Ubuntu 14.04. URI=http://docs.openstack.org/openstack-ops/openstack-ops-manual-.pdf. 2014