

Load Balancing Virtual Machines Deployment Mechanism In SDN Open Cloud Platform

Wen-Kuei Hsieh^a, Wen-Hsu Hsieh^{b,c}, Jiann-Liang Chen^c, Feng-Yi Chou^c, Yung-Sheng Lee^c

^a General Education Center, De Lin Institute of Technology, New Taipei, Taiwan

^b Department of Computer and Communication, De Lin Institute of Technology, New Taipei, Taiwan

^c Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan

wkhsieha@gmail.com, wnhsieh742@gmail.com, lchen1215@gmail.com, feng420@gmail.com, yslee@uch.edu.tw

Abstract— As a cloud data center may be located over many regions and the network environment within a cloud data center may differ from traditional ones, how Virtual Machines (VMs) are deployed will influence service performance. This study, based on the Eucalyptus cloud computing and Software-Defined Networking platform, proposes a load balancing scheduling mechanism that works on the current network status between users and associated VMs to improve the cloud services. This study also set up a node controller on the same subnet and different subnet. Study results indicate that in the same subnet of normal network, the VMs deployed with the proposed mechanism improve transmission time by 2.64% and 68.72% compared to that of the Round Robin and Greedy mechanisms consecutively. In the different subnet of normal network scenario, the proposed mechanism improves transmission time by 22.01% and 52.11% compared to that of the Round Robin and Greedy mechanisms respectively.

Keywords— Load Balancing, Virtual Machines, Eucalyptus, OpenFlow, Cloud Computing, Software-Defined Networking

I. INTRODUCTION

As a new application service, many benefits can be obtained from cloud computing. Whether one builds the hardware platform or software application services or provides a large data storage center, cloud computing has gradually replaced traditional methods. One important feature is its ability to serve as an enormous data center with high-performance computing capability.

The availability and reliability of a cloud data center service is very important for all users. However, a user's behavior has been transformed by various devices such as smartphones and tablet PCs. Unfortunately, as access to the data center may be distributed over many countries or different devices, the status of each network connection is distinctive and unpredictable.

Virtual Machines (VMs) have an important role in providing remote services to users of a cloud data center. However, the deployment of VMs within a cloud computing architecture is very crucial to maintain the predictable quality of cloud data center services. Logically, each user is associated with one or several distributed VMs behind a cloud

data center. Network topology of cloud computing would differ depending on the cloud infrastructure type.

The Eucalyptus cloud computing infrastructure was designed as a hierarchical topological model, in which the cloud controller becomes its parental node and serves as a cloud façade. To serve users, Eucalyptus cloud computing deploys VMs in its topological infrastructure. Some VM deployment mechanisms exist with the Eucalyptus cloud, namely, Round Robin mechanism (RoRoM), Greedy mechanism (GrM) [1]. However, these mechanisms lack any preference for network connectivity between VMs and associated users. This consideration is very important to guaranteeing Quality of Service (QoS) of cloud data centers and fulfilling cloud computing Service Level Agreement (SLA). Therefore, this study solves the problem of how to deploy efficiently VMs in the Eucalyptus cloud computing platform in order to maximize network connection utilization between users and VMs. The proposed approach implements an agent-based controller to monitor network status between controllers and cloud node controllers. The controller schedules the deployment of VMs according to information gathered from the agent. Simultaneously, it sends the information to the OpenFlow switch to set the flow table, flow connectivity between users and their associated VMs. This study applies the novel Load Balancing VMs Deployment (LBVMD) mechanism to establish optimally VMs deployment in the Eucalyptus cloud computing infrastructure. The proposed LBVMD mechanism can promote system network throughput.

The remaining part of this paper is organized as follows. Section 2 surveys the related background knowledge of this study. Section 3 describes the proposed LBVMD mechanism. In Section 4, the implementation environment and the performance analysis are presented. Finally, Section 5 gives the conclusion and future work.

II. BACKGROUND KNOWLEDGE

According to the definition of cloud computing from the National Institute of Standards and Technology (NIST), cloud computing has five characteristics, three service models, and

four deployment types [2]. Five essential characteristics of cloud computing are identified, including on-demand self-service, broadband network access, resource pooling, rapid elasticity or expansion, and measured service. Three popular models are Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS).

Cloud computing is making a dramatic effect on cloud technology. Currently, various open source applications are available for cloud computing, such as Eucalyptus, OpenStack, OpenNebula, CloudStack and so on. In this paper, the test platform is Eucalyptus [3].

Eucalyptus, an acronym for “Elastic Utility Computing Architecture for Linking Your Programs to Useful Systems,” was developed by a research group at the University of California, Santa Barbara (UCSB) [4]. Eucalyptus is Linux-based open-source for cloud computing that pertains to the IaaS service model. The Eucalyptus uses each high-level system component as a standalone web service, which is a language-agnostic Application Programming Interface (API) that is described by Web Services Description Language (WSDL) documents [5]. Figure 1 presents the Eucalyptus architecture.

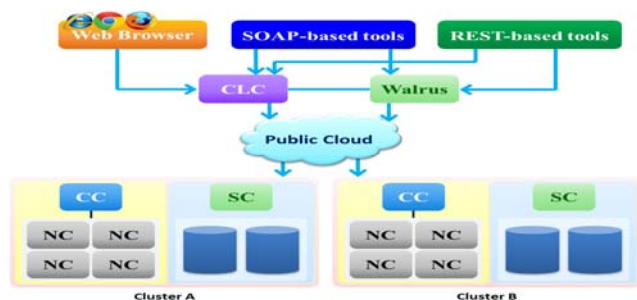


Figure 1. Eucalyptus architecture

The Eucalyptus system components are introduced as below:

- Node Controller (NC): The NC runs on every node which is designated for hosting VM instances. The NC queries the node to discover its physical resources including the number of cores, the size of memory, the available disk space, and learn about the state of VM instances.
- Cluster Controller (CC): The CC executes on a cluster front-end machine, which connects to both the NC and Cloud Controller (CLC). The CC collects and reports information about the resources of NCs to the CLC, and receives requests from CLC and schedules the execution of VM on specific NCs.
- Cloud Controller (CLC): The CLC is the entry-point of Eucalyptus to end-users and the interface to the management platform. The responsibility of CLC is exposing and managing the underlying virtualized resources via Amazon EC2 API and a web-based user interface [6].
- Storage Controller (Walrus): A Walrus is a data storage

service that implements the standards REST and the SOAP interfaces that compatible with Amazon's Simple Storage Service. Walrus acts as a VM image storage and management service, it provides a mechanism for storing and accessing VM and user data.

A. Cloud Virtualization

This subsection introduces the main virtual technology in the test platform. A Kernel-based Virtual Machine (KVM) is full virtualization for Linux on x86 hardware including Intel VT or AMD-V. A KVM allows running VMs using Linux or Windows disk images. A normal Linux system has two modes of execution: user mode and kernel mode [7, 8]. A KVM adds a third mode and the division of the three modes is as follows.

- Guest mode: Guest mode executes (QEMU-I/O) guest code.
- Kernel mode: The Kernel mode handles any existing guest mode due to normal I/O.
- User mode: User mode provides normal I/O for a user, but cannot use special instructions.

B. OpenFlow

OpenFlow is an open standard that is based on an Ethernet switch, with an internal flow table and a standardized interface, for adding flow entries to, and removing flow entries from a network environment [9].

Figure 2 presents the OpenFlow architecture, which comprises at least three parts. (1) An OpenFlow switch is composed of one or more flow tables and a group table, which perform packet lookups and forwarding. (2) An OpenFlow channel, which communicates with the controller and the controller, manages the switch using the OpenFlow switch protocol. (3) The controller can add, update, delete and manage flow entries in flow tables using the OpenFlow switch protocol [10].

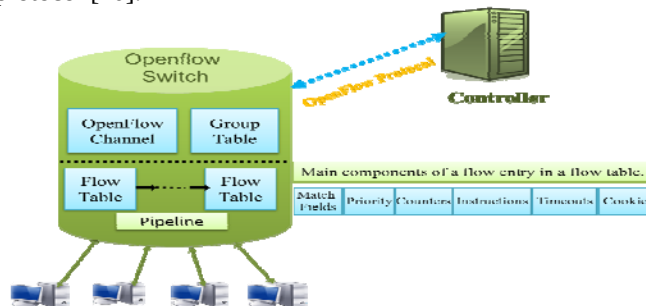


Figure 2. OpenFlow architecture

Matching begins at the first flow table and may proceed to other flow tables of the pipeline. Flow entries match packets in priority order. If a matching entry is found, then the instructions that are associated with the particular flow entry are executed. If no match is found in a flow table, then the consequence depends on the configuration of the table-miss flow entry. Instructions that are associated with each flow entry either contain actions or modify pipeline processing. Actions that are included in instructions are packet forwarding,

packet modification and group table processing. Pipeline processing instructions allow packets to be sent to subsequent tables for further processing [11]. Figure 3 presents the packet flow through an OpenFlow switch (v1.3.3).

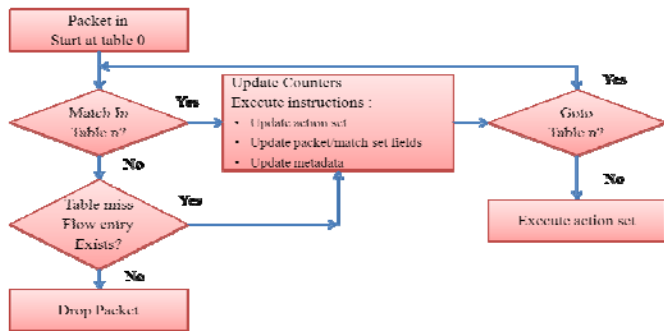


Figure 3. Packet flow through an OpenFlow switch (openflow-switch-v1.3.3)

C. OpenFlow Controller

A controller is responsible for controls and adjusts all of the OpenFlow switches; creates a virtualized network, and forwards packets. Regardless of whether the control is in-band or out-of band, the network manager must define the policy that governs the controller. Figure 4 presents the connection between the OpenFlow switch and the controller.

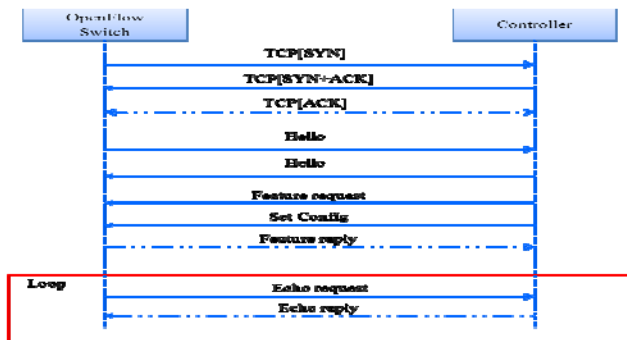


Figure 4. Connection Establishment between Switch and Controller

In this research, OpenDaylight [12] is used as the controller. The OpenDaylight controller is an open-source development platform, which can be expanded using Java. OpenDaylight is a combination of components that include a fully pluggable controller, interfaces, protocol plug-ins and applications. In this work, OpenDaylight was used to control the platform.

III. PROPOSED LOAD BALANCING VMs DEPLOYMENT (LBVMD) MECHANISM

The proposed LBVMD mechanism is applied to determine the appropriate node controller to run the VM. The system consists of the Eucalyptus cloud system, the Agent, and the OpenFlow Switch. The Eucalyptus cloud system comprises a cloud controller, cluster controller, storage controller, walrus, and node controller. In the Eucalyptus cloud system, each controller can be set in a single machine. The Agent is located at the controller which monitors network utilization from the cluster controller to each node controller and the client. The

OpenFlow switch directs access flow to the node controller. Figure 5 shows the proposed system architecture.

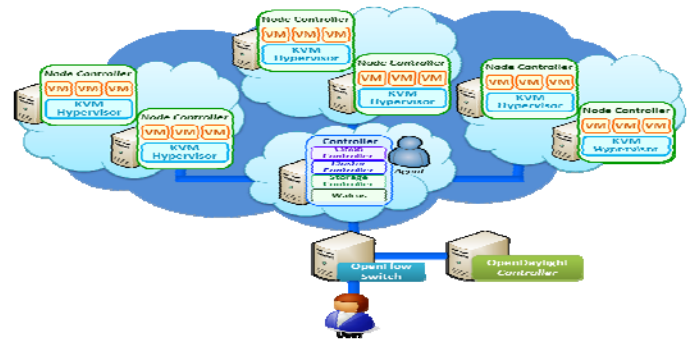


Figure 5. The system architecture of LBVMD

A. VM Scheduling Algorithm

In the Eucalyptus cloud system, the allocation scheduling of VMs is based on two policies: the Round Robin mechanism (RoRoM) and the Greedy mechanism (GrM).

- 1) **RoRoM:** The VMs scheduling mechanism is based on simple the RoRoM that fairly deploys VMs depending on the number of available node controllers.
- 2) **GrM:** The VMs are deployed on the first available node controller. Subsequent VMs are deployed on the next available node controller when all resources are occupied.

B. LBVMD mechanism

Figure 6 shows a topological architecture of the Eucalyptus cloud system. The module is simply a hierarchical tree-based topology with the cluster controller on the top of the topology. Node controller (NC) denotes as physical machine that is implementing VMs. On implementation, an NC could be implemented on different subnets. An NC can even reside at the same virtual layer of the Cluster Controller (CC). Through such a topological consideration, each network link from an NC to a CC would differ in terms of bandwidth utilization. For each user accessing the CC, the available link capacity is denoted by C_{u-cc} . We assume there is set of NCs given by $N = \{N_1, N_2 \dots N_i\}$, where each link's capacity is denoted by $CN(i)$. The set capacity link for each NCs is given by $CN = \{CN(1), CN(2), \dots, CN(i)\}$. Between the CC and certain NCs, the flow may pass more than one subnet. Thus, the CC is not the front end of the Eucalyptus computing topology. In network implementation, end users can directly access the CC.

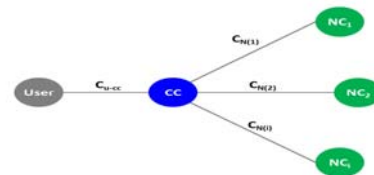


Figure 6. Eucalyptus cloud system topology

This section provides a detailed explanation of the LBVMD mechanism in selecting the most appropriate NC for VMs implementation for network QoS. The proposed mechanism is

based on a basic objective of avoiding all possible bottlenecks along the flow. The main feature of this mechanism is that it calculates the equilibrium between Cu-cc and CN(i). The initial value to acquire is the average value of the maximum link capacity between the user and Cluster Controller (CC),

denotes as $\overline{C_{u-cc}}$. Then, based on the agent measurement, all link capacities of CN(i) are collected, where the mean is given

by $\omega = \frac{\sum C_{CN(i)}}{i}$. Based on those two values, the rules in choosing an NC are as follows:

- If $\overline{C_{u-cc}} > \omega$; the NC with $\max\{CN\}$ will be selected,
- If $\overline{C_{u-cc}} < \omega$; the NC with $\max(\{CN\} < \omega)$ will be selected,
- If $\overline{C_{u-cc}} \approx \omega$; the NC with $\min(\{CN\} > \omega)$ will be selected.
- Figure 7 shows the flowchart for selecting an NC to run a VM instance. The NC selection strategy is as follows:

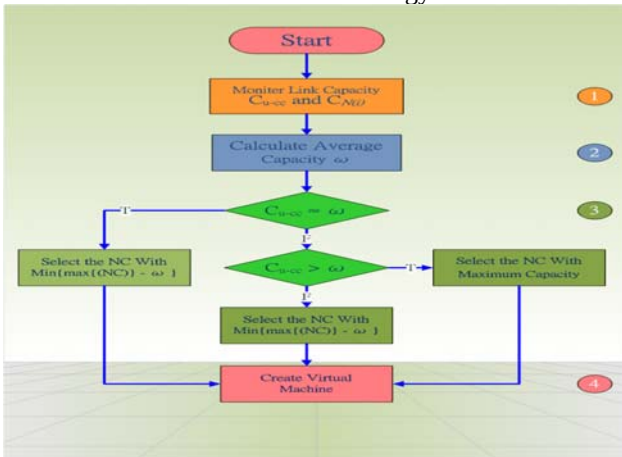


Figure 7. Flow chart of VM scheduling algorithm

- Step 1: When the agent obtains request from the CC, it starts monitoring network utilization status from the CC to every NC (CN(i)), and from users to client CC which denotes as (Cu-cc).
- Step 2: After the agent obtains all the results, it calculates the average capacity of all CN(i), denoted as ω .
- Step 3: The agent then compares ω with Cu-cc and then selects the associated NC based on the process.
- Step 4: The information regarding the selected NC by the agent is executed by the CC to deploy the VM.

C. NC Selection Mechanism

Figure 8 shows the VM deployment process. First, when a client wants to access services from Eucalyptus cloud computing, it sends a runInstance() request to the Cloud Controller (CLC). The CLC chooses the first CC that has enough free resources by querying the CC through describeResources() to obtain the current resource condition. In the same manner, CC queries each NC through

describeResources(). The NC responds to the CC with available resource information by describeResourceResponse(). Simultaneously, the CC also sends a describeNetwork() request to the agent to query about network capacity. After the agent receives the request, it sends the detectNetwork() request to collect network information between the CC with NC and CC with client users. Then the CC executes operations for selecting the NC that will be appropriate for implementing the VM according to the network-constrained scheduling algorithm. Meanwhile, the agent sends the setFlowtable() message to the OpenFlow switch to facilitate access flow from client to the chosen NC.

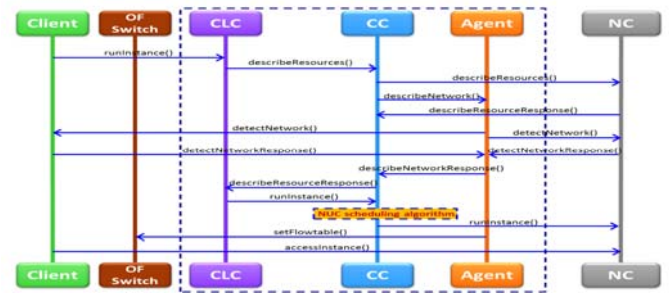


Figure 8. VM Deployment Process

IV. SYSTEM DESIGN AND PERFORMANCE ANALYSIS

This section first introduces the emulation environment and then analyses the performance of the proposed mechanism.

A. System Implementation

This study uses four machines to implement the Eucalyptus system. Table 1 shows the Eucalyptus cloud computing configuration. Figure 9 shows the system configuration.

Table 1: Devices and Eucalyptus system configuration

Functionality	Controller CLC, CC and Walrus	Node1 NC	Node2 NC	Node3 NC
IP Address	192.168.1.120	192.168.1.3	192.168.1.4	192.168.1.5
HostName	Cluster	EucNC1	EucNC2	EucNC3
CC Address	192.168.1.120			
Gateway	192.168.1.1	192.168.1.2	192.168.1.2	192.168.1.2
Instances Public Addresses	192.168.1.200-192.168.1.220			
Cluster Name	euccluster			
Hardware Device	PC(Intel Core2 Duo CPU E7500@2.93GH, 4 GB RAM, Intel I350 T2(NIC))			
Software Device	OS : Ubuntu 10.04, Cloud Software : Eucalyptus 1.6.2			

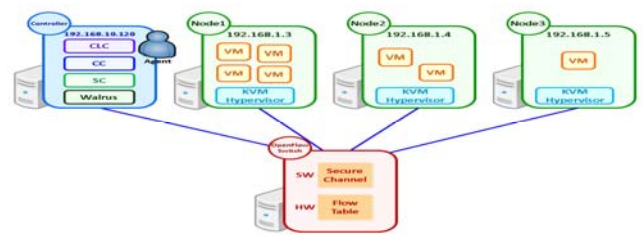


Figure 9. System configuration at the same subnet

B. System Emulation

Figure 10 shows the Eucalyptus management interface. In the credentials page, the administrator and users can download

the credentials to use Eucaly2ools. The images page lists all uploaded image files, including the kernel, root file system, and ram disk image. The user page shows users administrator information. The most important page is the configuration page that the administrator uses to configure the CLC, and VM types. The services page show information related to the Eucalyptus system and the extras page list some Eucalyptus-certified images and Eucalyptus-compatible tools.

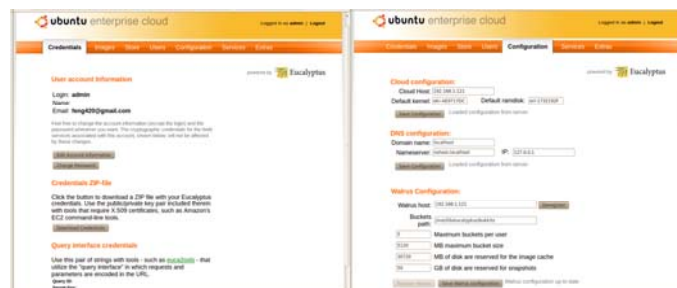


Figure 10. Eucalyptus management interface

To obtain the link capacities of both CN(i) and Cu-cc, the agent is built at the controller where CLC and CC reside. Figure 11 shows the operation of the agent. The agent measures all connected links from the CC to obtain all link conditions of available NCs.

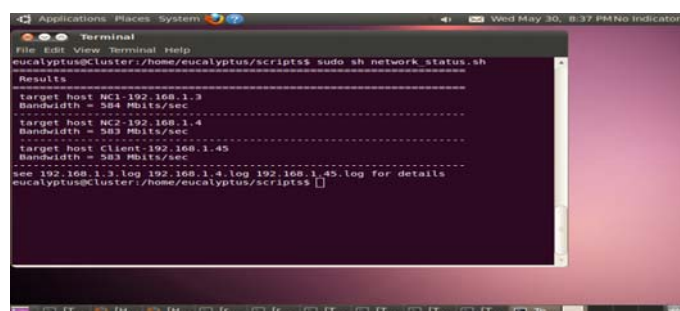


Figure 11. Agent operations

C. Performance Analysis

The performance in four cases was analyzed. Case 1- Node controller at the same subnet; Case 2- Node controller at the same subnet with a bottleneck; Case 3- Node controller at the different subnet; and Case 4- Node controller at the different subnet with a bottleneck. Each case is run with three scheduling mechanisms: the RoRoM, GrM, and proposed LBVMD mechanism. This study creates four VMs to identify the placement of each VM and then upload the same image file to the VM to determine how VM placement influences performance. The image files are uploaded to the VMs every 10 seconds. At the beginning, upload the file to VM1; after ten seconds, upload the file to VM2; at 20 seconds, upload the file to VM3, and last, upload the file to VM4 at 30 seconds. Analytical results for the four cases are as follows.

● Case 1- Node controller at the same subnet

Figure 12 shows the placement of VMs with different scheduling mechanisms. The first is the RoRoM. The VMs are

fairly deployed on every NC, VM1 and VM4 are located at the first NC, and VM2 and VM3 are located at the second NC and third NC, respectively. With the GrM, four VMs are deployed on the same NC. The last is the proposed LBVMD mechanism. The number of VMs in the NC is the same as that with the RoRoM, but the sequence differs.

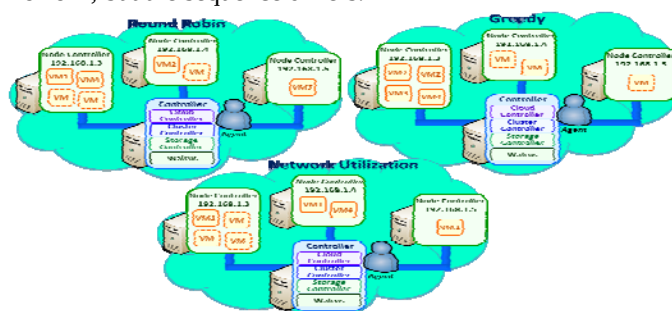


Figure12. VM placements in Case 1

Figure 13 lists the completion time for uploading the image file in Case 1. The completion time with the RoRoM and proposed LBVMD mechanism are approximately the same, but the LBVMD mechanism is faster than the RoRoM, decreasing 1.89 seconds in average. Compared to that of the GrM, the average completion time of proposed LBVMD mechanism decreases 153.41 seconds.

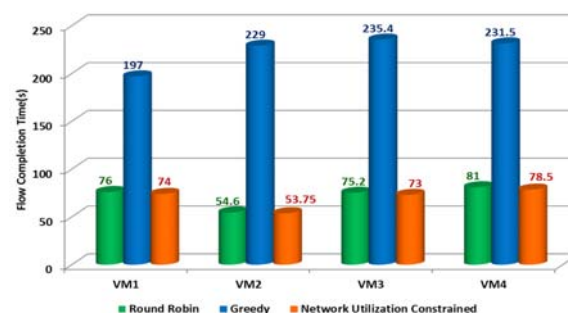


Figure 13. Flow completion time of each VM in Case 1

● Case 2: Node controller at the different subnet

In this case, one NC and the controller are located at the same subnet and two NCs are located at the other subnet. Figure 14 shows the placement of VMs with different mechanisms. First, with the RoRoM, the VMs are fairly deployed on every NC. VM2 is located at the second NC, which is at the same subnet with the controller; Both VM1 and VM4 are located at the first NC, which is at the other subnet; and VM3 is located at the third NC, which is at the same subnet with first NC. In the GrM, four VMs are deployed on the same NC, which is at the other subnet without the controller. With the LBVMD mechanism, VM1 and VM2 are located at the second NC, which is at the same subnet with the controller, and VM3 and VM4 are located at the first and third NC, respectively.

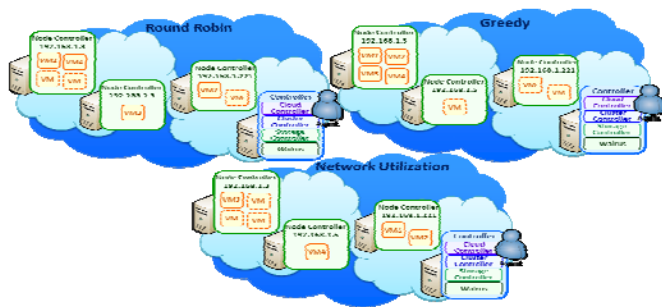


Figure 14. VM placements in Case 2

Figure 15 describes the completion time for uploading the image file in Case 2. With the RoRoM, because VM1, VM3, and VM4 are at the same subnet without the controller, average transmission time is 207.40 seconds. Notably, VM2 is at the subnet with the controller, such that transmission only takes 60.80 seconds. Compare to the GrM in Case 1, the average transmission time of the GrM increases to 278.10 seconds from 223.23 seconds, because the NC is located at the other subnet. The LBVMD mechanism puts VM1 and VM2 at the same subnet with controller, so average transmission time is 99.80 seconds, and VM3 and VM4 are located at the other subnet without controller, and the average transmission time is 166.55 seconds.

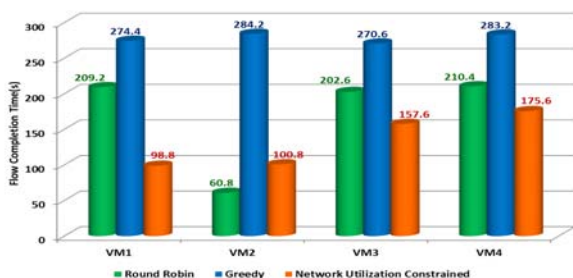


Figure 15. Flow completion time of each VM in Case 2

D. Comparison of Completion Time

Table 2 show the comparison of completion time between RoRoM and LBVMD. Table 3 show the comparison of completion time between GrM and LBVMD.

Table 2: The comparison of completion time between RoRoM and LBVMD

Test Case	RoRoM				RoRoM AVE	NCVMD				NCVMD AVE	RoRoM - NCVMD
	VM1	VM2	VM3	VM4		VM1	VM2	VM3	VM4		
Case 1: node controller at the same subnet	76.00	54.60	75.20	81.00	71.70	74.00	53.75	73.00	78.50	69.81	1.89
Case 2: node controller at different subnet	209.20	60.80	202.60	210.40	170.75	98.80	100.80	157.60	175.50	133.18	37.58

Table 3: The comparison of completion time between GrM and LBVMD

Test Case	GrM				GrM AVE	NCVMD				NCVMD AVE	GrM - NCVMD
	VM1	VM2	VM3	VM4		VM1	VM2	VM3	VM4		
Case 1: node controller at the same subnet	197.00	229.00	235.40	231.50	223.23	74.00	53.75	73.00	78.50	69.81	153.41
Case 2: node controller at different subnet	274.40	284.20	270.60	283.20	278.10	98.80	100.80	157.60	175.50	133.18	144.93

V. Conclusion

Due to the rapid development of cloud computing technology, the number of users has increased dramatically. One key service of cloud computing technology is determined by deployment of VMs. Traditional VM deployment mechanisms only depend on the resources of physical machines, such as CPU processing capacity and memory size. This study applies a novel network-constrained scheduling algorithm to select the appropriate NC to deploy the VM. The system includes the Eucalyptus cloud infrastructure, an agent and an OpenFlow switch. The agent monitors network utilization and then sends the information to the CC to choose an appropriate NC to create a VM. The implementation result illustrates that the VM created by the network-constrained scheduling mechanism provides better performance than existing mechanisms. Openflow and the Eucalyptus cloud computing configuration are used in the testbed. The authors also use Software-Defined Networking (SDN) to improve the operational efficiency of cloud data centers, prompting Internet service providers, telecommunications operators and centers of Internet service technology research and development to conduct research into related technologies, with the goal of improving the operation of existing wireless and broadband networks using SDN technology.

REFERENCES

- [1] Chung Pan Tang, Patrick P. C. Lee, and Tsz Yeung Wong, "A Tunable Version Control System for Virtual", *Proceedings of the IEEE Transactions On Services Computing*, pp.1-14, December 2013.
- [2] Cynthia Harvey, "60 Open Source Apps You Can Use in the Cloud," *Datamation*, May 2014.
- [3] E. Sousa, F. Lins, E.Tavares, P. Cunha, and P. Maciel, "A Modeling Approach for Cloud Infrastructure Planning Considering Dependability and Cost Requirements" *2014 IEEE International Conference on Systems, Man, and Cybernetics*, pp. 2168-2216 September 2014
- [4] A.M. Lonea, D.E. Popescu and O. Prosteian, "A Survey of Management Interfaces for Eucalyptus Cloud," *Proceedings of the 7th IEEE International Symposium on Applied Computational Intelligence and Informatics*, pp.261-266, May 2012.
- [5] Debora Souza, Rubens Matos, Jean Araujo, and Vandi Alves, Paulo Maciel, "EucaBomber: Experimental Evaluation of Availability in Eucalyptus Private Clouds", *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pp.4080 – 4085 October 2013
- [6] V. Amiry, S.Z. Rad, M.K. Akbari, and M.S. Javan, "Implementing Hadoop Platform on Eucalyptus Cloud Infrastructure," *Proceedings of the 3PGCIC Seventh International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pp.74-78, November 2012
- [7] Iustina Melinte, Alexandru Bujor, Razvan Dobre, and Alexandru Herisanu, "Running KVM Virtual Machines in Batch Systems," *Proceedings of the 2013 19th International Conference on Control Systems and Computer Science*, pp. 106-112, May 2013
- [8] Alexandru-Catalin Bujor, and Razvan Dobre "KVM IO Profiling," *Proceedings of the 2013 RoEduNet International Conference 12th Edition*, pp. 26-28 September 2013
- [9] Adrian Lara, Anisha Kolasani, and Byrav Ramamurthy "Network Innovation using OpenFlow: A Survey" *Proceedings of the IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, First Quarter 2014
- [10] OpenFlow Switch Specification, Version 1.3.3(Dec. 18, 2013). [Online]. Available: <https://www.opennetworking.org/sdn-resources/onf-specifications>
- [11] OpenFlow Switch Specification, Version 1.3.4(Mar. 27, 2014). [Online]. Available: <https://www.opennetworking.org/sdn-resources/onf-specifications>

- [12] A. Vishnio, R. Poddar, V. Mann and S. Bhattacharya, "Effective Switch Memory Management in OpenFlow Networks," *Proceedings of the ACM International Conference on Distributed Event-Based Systems* pp. 177-188, May 2014



Wen-Kuei Hsieh was born in Taiwan on January 15, 1961. He received his Ph.D. degree from Florida International University, USA, in 1999 and M.S. in Computer Science from University of Oklahoma City, USA, in 1990. He has been with Computer Center of Tamsui Oxford College in 1991. Since August 1992, he works with De Lin Institute of Technology, where he is an associate professor now. His research interests include computer related applications



Wen-Hsu Hsieh was born in Taiwan on February 9, 1963. He received the master degree in Computer Science from the University of Oklahoma City, U.S.A. in May 1994. He was an instructor of the Department of Computer Center, De Lin Institute of Technology from August 1994 to July 2007. He is an instructor of the Department Computer and Communication Engineering now. His research interests include Computer Network, the application of cloud computing, mobile communication and SDN. Currently, Professor Hsieh also is the PhD student of the Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan, R.O.C.



Jiann-Liang Chen Prof. Chen was born in Taiwan on December 15, 1963. He received the Ph.D. degree in Electrical Engineering from National Taiwan University, Taipei, Taiwan in 1989. Since August 1997, he has been with the Department of Computer Science and Information Engineering of National Dong Hwa University. Prof. Chen joins the Department of Electrical Engineering, National Taiwan University of Science and Technology, as a full professor now. His current research interests are directed at cellular mobility management, digital home network and personal communication systems.



Feng-Yi Chou is the master student of the Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan, R.O.C. She received the master degree in July 2012. Her research interests include Openflow, the application of cloud computing and computer network.



Yung-Sheng Lee (M'92) received the B.S. and M.S. degrees in Electrical Engineering from National Taiwan University in 1986 and 1988, respectively. He is currently working toward the Ph.D. degree at the Department of Electrical Engineering, National Taiwan University of Science and Technology, Taiwan. His research interests include wireless communications, network architectures, automation systems and renewable energy systems.