DBpia

논문 17-42-08-11 The Journal of Korean Institute of Communications and Information Sciences '17-08 Vol.42 No.08
https://doi.org/10.7840/kics.2017.42.8.1562

OpenStack 클라우드에서 VM의 자원 이용도 기반 동적 마이그레이션을 위한 프레임워크

아팍 무하마드•, 송 왕 철°

Framework for Resource Utilization-Based Dynamic Migration of VMs in OpenStack Clouds

Afaq Muhammad•, Wang-Cheol Song°

요 약

가상화기술은 클라우드 컴퓨팅에서 여러 가지 필수적인 특성들을 제공한다. 그러한 한 가지 특성은 가상 머신(VM)들의 라이브 마이그레이션인데, 이는 서비스 중단 없이 물리적 호스트들 사이에서 하나의 VM에 대한 완전한 상태를 전송하는 프로세스이다. 이러한 기능은 시스템 재구성, 장애 허용성, 에너지 효율성, 부하 분산 및 시스템 유지 등의 목적으로 널리 쓰여오고 있다. 대부분의 기존 연구들에서는, 어느 결함있는 VM들이 어느 물리 머신으로 전달되어야 하는냐를 자원 이용성에 따라 결정한다. 하지만, 역동적인 VM 마이그레이션에 대해서는 아주 적은 구현사례만이 있을 뿐이다. 이러한 목적을 위해서, 본 논문에서 우리는 Openstack 클라우드에서의 역동적인 VM 마이그레이션을 위한 프레임워크를 제안한다. 이 프레임워크는 Openstack에서 역동적인 VM 마이그레이션을 위해서 제안한 과부화 검출, VM 선택, VM 할당 알고리즘들을 구현하고 있다. 실험을 통해서, 제안된 알고리즘들이 기존 알고리즘들보다 더 성능이 앞서고 있음을 보였다.

Key Words : Cloud Computing, Virtualization, OpenStack, Dynamic VM Migration, Resource Utilization

ABSTRACT

Virtualization technology provides several vital features in cloud computing. One such feature is live migration of virtual machines (VMs). It is a process of transferring the complete state of a VM between physical hosts without interruption of any service. This capability is being widely used for the purpose of reconfiguration and fault tolerance, energy efficiency, load balancing, and system maintenance. Most of existing studies make decision on which defective VMs should be transferred to which suitable physical machines (PMs) in terms of resource utilization. However, there are very few implementations available for dynamic VM migration. To that end, in this paper, we propose a framework for dynamic VM migration in OpenStack clouds. The framework implements the proposed overload detection, VM selection, and VM allocation algorithms for dynamic VM migration in OpenStack. Furthermore, with the help of experiments, we show that the proposed algorithms outperform the algorithms that are considered for the purpose of evaluation.

※ This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(NRF-2016R1D1A1B01016322).
• First Author : Jeju National University Department of Computer Engineering, afaq24@gmail.com, 학생회원
° Corresponding Author : Jeju National University Department of Computer Engineering, kingiron@gmail.com, 종신회원
논문번호 : KICS2017-05-146, Received May 16, 2017; Revised July 25, 2017; Accepted August 3, 2017

www.dbpia.co.kr

Ⅰ. Introduction

Cloud computing platforms such as IBM's Blue Cloud, Microsoft's Azure, and Amazon's EC2 host various distributed applications. Service level agreement (SLA) of these services is a major issue for determining service providers's profit and user loyalty[1], but SLA requirements are not easy to be satisfied because of the high variations of Internet characteristics and workloads. Deploying new storage elements and new servers are costly choice that may result in high system management costs and low server utilization. Migration of VMs across PMs has the ability to improve service SLA compliance by reducing resource contention.

The main challenge is that which VM should be transferred and where the transferred machine should be located. Most studies focus on resource utilization to make decision for migration action[2,3], but in this case migration action may result in performance degradation if not handled properly during the migration process. Different VMs have different workload characteristics and configurations [4] that lead to different migration costs.

OpenStack, one of the well-known cloud platforms for both public and private clouds, was announced in 2010[5]. Among several existing sub-projects, OpenSack Nova[6] is the core project of OpenStack, which provides infrastructure as a service (IaaS) on demand. An instance/VM can be launched by means of OpenStack Nova on the efficient compute node, which meets the customer's requirements. Although, OpenStack supports live migration technique, the administrator has to manually intervene within the appropriate time to migrate a VM instance from one compute node to another. This feature of OpenStack is useful whenever a compute node, where many VM instances are running, needs to redistribute or maintain load. However, the necessary requirement for conducting a dynamic live migration, which guarantees the QoS and SLA, is the VM migration decision taken at a suitable time. In additoion, due to a huge number of processes running on the compute nodes and VM instances, the load is dynamic in nature. This dynamic nature of load demands the process of VM migration to respond dynamically upon receiving the load statistics at each periodic interval. This lack of dynamic VM migration leads to search for an appropriate method to monitor and measure the load in order to migrate VMs to efficient compute nodes. This method should be adaptable with the resource usage requirement of on demand up/down scaling.

In addition, recent virtualization techniques do not provide enough performance isolation among the VMs[7]. The contention for physical resources among VMs leads to different performance impact level among the VMs. To that end, in this paper, we introduce an architecture and implementation of a framework for dynamic VM migration in cloud data centers based on the OpenStack platform. The deployment of the framework includes a controller node and multiple instances of compute nodes. The purpose of the framework is to provide dynamic live migration to adjust the VM instances on compute nodes to offload a number of VMs from an overloaded compute node. We address this issue by focusing on the following key points:

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

The remainder of the paper is organized as follows. In Section II, we discuss the related work. In Section III, the architecture of the proposed framework, and the proposed algorithms are presented. In Section IV, the framework is experimentally evaluated and the results are analyzed. In Section V, the paper is concluded with a discussion of scalability and future research directions.

www.dbpia.co.kr

Ⅱ. Related Work

One of the early works[8], which focuses on dynamic VM migration, was implemented to offload an overloaded physical machine. Although, the model designed for the optimization of the dynamic VM allocation has considered the cost of VM migration, the authors did not implement any algorithm for deciding when it was vital to perform the VM allocation optimization. The designed model was invoked periodically to adjust the VM allocation, which needs a supplementary performance any necessary demand for optimization operation.

A dynamic resource placement for OpenStack has been implemented in [9], which is based on a protocol to interact between peer servers. The operation was periodically performed, and the peer servers were chosen randomly without any requirement to exchange their state. Furthermore, performance degradation of migration operation was not taken into account.

In [10], authors proposed a technique for placement of VMs when the SLA of hosted applications was violated. With the same technique, authors in [11] examined the performance of VM placement decision by automating the response time which was described in the SLAs. This technique needed more time for VM placement and the duration of SLA violation was increased.

In [12], authors have used the threshold based on the utilization of resources such as network bandwidth, RAM, and CPU to decide when to migrate VM(s) from an overloaded PM to an efficient one. In [13], authors handled resource placement based on the response duration of QoS conditions at the server and the cluster level by implementing control loops. In [14], a remaining utilization-aware (RUA) algorithm has been proposed for VM allocation. In this work, the overloaded PM is first detected by means of the proposed algorithm, and then the migration for some VMs is conducted.

Authors in [15] proposed a system for managing resources of virtualized data centers by means of local and global policies. The system on the local level was responsible for implementing the power handling strategies for a guest host, whereas the global policies were responsible for managing the consolidation of VMs. However, the QoS requirements were not taken into account by the global policies.

In [16], authors proposed an algorithm for dynamic VM migration based on time-series analysis, and predicting of the resource demand. In [17], the authors devised the issue of VM placement as stochastic optimization issue with a constraint on the host overload probability, considering multiple resource constraints. A disadvantage of all the aforementioned approaches is that they are centralized, i.e., a single algorithm running on the master host (controller node in OpenStack) limits the global view of an overall system to enhance the VM instance placement. Furthermore, by the use of such a centralized approach, the scalability of the system is limited with an increase in the number of physical machines (compute nodes in OpenStack).

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

Ⅲ. System Model

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

www.dbpia.co.kr

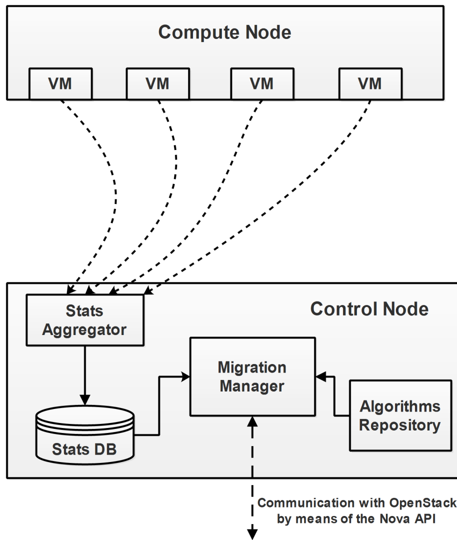


Fig. 1. Proposed Framework

efficient compute node.

Fig. 1 depicts the overall architecture of the proposed framework that is deployed in OpenStack to avoid an overloaded compute node by conducting dynamic VM migration at appropriate time. It includes four fundamental building blocks: (i) Stats Aggregator, (ii) Stats Database, (iii) Migration Manager, and (iv) Algorithms Repository. In the following, we will discuss each of the framework's components in detail.

3.1 Stats Aggregator

A component that is deployed on the control node and is responsible for collecting statistics on the resource utilization by VM instances and hypervisors and then forwarding it to the statistics database, which can also be shared with other components. The statistics are collected by means of libvirt's API[18] in the form of the RAM consumed by VM instances and hosts. For the collection of statistics related to CPU utilization of VM instances, we have installed and configured the Ceilometer[19] project of OpenStack on control node. The RAM and CPU statistics are both collected periodically and submitted to the Stats Database module of the framework.

3.2 Stats Database

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

3.3 Migration Manager

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

3.4 Algorithm Repository

This repository is deployed to store custom decision-making algorithms for dynamic VM migration, i.e., compute node overload detection, VM selection, and VM allocation algorithms. Based on these algorithms, the migration manager module determines the overloaded compute node, selects the VM instances that are to be migrated, as well as initiates VM migrations and makes VM placement decisions.

3.5 Proposed Structure and Algorithms

The migration process should be initiated offload the compute nodes based on a predefined threshold. A compute node is offloaded by VM migrations, which can make the load below the predefined threshold. Fig. 2 depicts the exchange of messages for handling a compute node overload situation. First, the migration manager detects an overload of

www.dbpia.co.kr

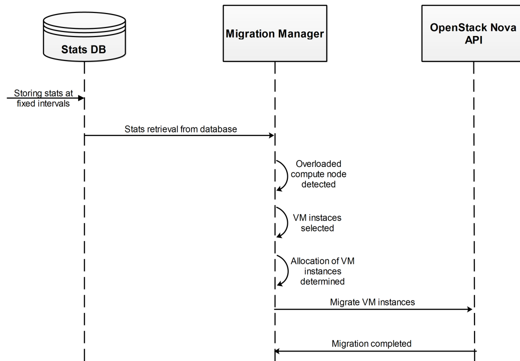


Fig. 2. Exchange of messages between components for VM instances migration

the compute node using the overload detection algorithm. Then, by means of proposed VM selection algorithm, the migration manager selects VM instances based on their CPU utilizations. Next, the migration manager initiates the VM allocation algorithm with the list of selected VMs along with their utilized resources and states of the compute nodes obtained from the stats database as arguments. Finally, based on the VM allocation generated by the algorithm, the migration manager requests the OpenStack Nova API for the appropriate VM live migration.

3.5.1 Overload Detection Algorithm

The overload detection algorithm shown in Algorithm 1 is a simple algorithm which detects an overloaded compute node if the average of the last n CPU utilization measurements is greater than the pre-defined threshold value.

Algorithm 1: Compute Node Overload Detection Algorithm

```
1 Input: n, utilization, threshold   Output: if the compute node is overloaded
2 if utilization received then
3 |   util ← previous n values of utilization
4 |   m ← calc_mean(util)
5 |   return m ≥ pre-defined threshold value then
6 return True
```

3.5.2 VM Selection Algorithm

Once an overloaded compute node is detected, it is necessary to determine what VMs are the best to be migrated. This issue is solved by algorithms for VM selection. One such example can be selecting a VM randomly from a pool of VMs assigned to the host. Alternatively, based on our proposed algorithm called minimum RAM maximum CPU utilization (minRmaxC) algorithm as shown in Algorithm 2, VMs with the maximum amount of RAM usage are first selected, and then out of these selected VMs, the VMs with the maximum CPU utilization averaged over the last n quantifications are selected.

Algorithm 2: Minimum RAM Maximum CPU utilization (minRmaxC) Algorithm

```
1  Input: n, vms_cpu_stats, vms_ram_stats   Output: a VM to select
2  min_ram ← min (values of vms_ram_stats)
3  min_cpu ← below pre-defined threshold value
4  selected_vm ← None
5  foreach vm, cpu in vms_cpu_stats do
6  |   if vms_ram_stats of a vm ≥ min_ram then
7  |       |continue
8  |   util ← previous n values of cpu utilization
9  |   mn ← calc_mean(util)
10 |   if mn ≥ pre-defined threshold value then
11 |       |selected_vm ← vm
12 return selected_vm
```

3.5.3 VM Allocation Algorithm

In order to get the efficient compute nodes for hosting VM instances, the VM allocation algorithm engages an OpenStack Nova-scheduler which conducts an overall allocation process. More precisely, it returns the efficient compute nodes on which the VM instances would be placed. Therefore, the algorithm first selects a list of the light load compute nodes for the heavy load VM instances. Then, this list is delivered to the OpenStack-Nova API in order to initiate the VM migration process between source and destination compute nodes. The pseudo-code for algorithm is shown in Algorithm 3, which explains the method of selecting the light load compute node for the heavy load VMs.

Algorithm 3: VM Instance Allocation Algorithm

```
1 Input: compute_node_utilization_stats, vm_utilization_stats
2 Output: allocation of VM instances to compute nodes
3 foreach vm in vm_utilization_stats do
4 |   while not allocated do
5 |   |   foreach compute_node in compute_node_utilization_stats do
6 |   |   |   if compute_node_cpu ≥ vm_cpu and compute_node_ram ≥
  |   |   |   vm_ram then
7 return allocating[vm_instances] ← compute_nodes
```


www.dbpia.co.kr

Ⅳ. Experiment and Results

The proposed architecture has been validated in the cloud computing platform-based testbed. OpenStack Mitaka release has been deployed on the testbed. The deployment includes one control node and four compute nodes, all running Ubuntu 14.04 as an operating system. The OpenStack control node runs the Identity service, Image service, management portions of Compute, management portions of Networking, various networking agents, and the dashboard. It also includes supporting services such as an Sequential Query Language (SQL) database, message queue, and (Network Time Protocol) NTP. Each compute node runs the hypervisor portion of Compute that operates instances.

In our scenario, Compute uses QEMU hypervisor [20]. The compute node also runs a Networking service agent that connects instances to virtual networks and provides firewalling services to instances via security groups. The testbed also consists of a separate machine that has ONOS installed in it. Each node of OpenStack (control node and compute nodes) is connected to ONOS[21] SDN controller via management network. Each compute node is connected to one another via a Data Tunnel Network, whereas to the Internet via External Network.

After the successful deployment, it is necessary to generate the work load in an appropriate way in order to reproduce a realistic data. For this purpose, we used a software called Lookbusy[22], which is a simple application for load generation on a Linux system. It allows to generate predictable, fixed loads on CPUs, and also maintain selected amounts of memory active.

We performed several experiments to evaluate the proposed algorithms. For this purpose, the proposed overload detection algorithm have been analyzed for two cases, i.e., the impact on an overall system without and with the execution of this algorithm. The proposed VM selection and VM allocation algorithms have been compared with random VM selection and random VM allocation algorithms. The proposed algorithms have been also compared with the existed algorithms in literature. We used a VM instance type with 32 MB amount of RAM allocated to it. We have launched 8 VM instances on compute node 1, 7 VM instances on compute node 2, 5 VM instances on compute node 3, and 3 VM instances on compute node 4. Load has been generated on random VM instances of compute node 1, which results in an increased CPU utilization of the VM instances.

During the experiment, the overload detection algorithm have been executed by the migration manager module, which detects an overloaded compute node as soon the CPU utilization surpasses the pre-define threshold value. The performance of our proposed overload detection algorithm has been compared for two scenarios: 1) overload situation without any algorithm used, and 2) direct overload detection[23]. The proposed algorithm detects an overloaded compute node if the average of the last n CPU utilization measurements is greater than the pre-defined threshold value, whereas the direct overload detection detects an overloaded compute node as soon as the CPU utilization surpasses the pre-define threshold value. The drawback in direct overload detection is that most of the times a compute node goes above the threshold for a very short period of time because there are always CPU utilization spikes in real time while performing tasks. In this case, a compute node would be detected as an overload node as soon as its CPU utilization is above the threshold for a very short period of time. The granularity can be selected according to the requirement; however, in most of the cases the system is not too sensitive to the CPU overloads. In the case when no overload detection algorithm is executed, the compute nodes stay overloaded, which may result in the degradation of an overall system/service. Fig. 3 depicts the detection of compute node for each run of the proposed overload detection and direct overload detection algorithms. It is obvious from the figure that the compute node is detected as an overloaded compute node by direct overload detection algorithm as soon as it surpasses the threshold value of 80% CPU utilization. On the other hand, the proposed

www.dbpia.co.kr

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

It is also obvious from Table 1 that a compute node is detected as an overloaded compute node for each run of the proposed overload detection algorithm as soon as it surpasses the threshold value of 80% CPU utilization. On the other hand, when the algorithm is not executed, there is no overload detection even if the CPU utilization surpasses the threshold value of 80%.

Once an overloaded compute node has been detected, the next step is to select the VM instances which are consuming most of the resource of that compute node. These VM instances can be selected randomly, but this may lead to performance degradation because the selected VM instances may have maximum unutilized RAM and minimum CPU utilization at the time of selection. We address this issue by proposing minRmaxC algorithm, which selects VM instances on the basis of minimum RAM and maximum CPU utilization. It accepts historical data on the resource usage by VM instances running on the compute nodes and returns a set of VM instances to be selected. The performance of the proposed minRmaxC algorithm is compared with random VM selection and

Table 1. Overloaded Compute node situation

CPU(%) without algorithm	CPU(%) direct threshold algorithm	CPU(%) proposed OD algorithm	Number of runs
62	40	46	1
65	52	43	2
72	90	56	3
77	72	72	4
87	54	84	5
93	50	87	6
96	45	90	7
89	55	83	8
85	68	73	9
83	84	67	10
84	77	54	11
89	63	46	12
91	53	44	13
90	48	43	14
82	44	43	15

maximum utilization[24] algorithms on the basis of the maximum CPU utilization. It is obvious from Fig. 4 that the VM instances with the maximum CPU utilization are selected by means of minRmaxC algorithm, whereas the random VM selection scheme selects VM instances regardless of the CPU utilization for each run. Although, the maximum utilization algorithm selects VMs which have higher CPU utilizations than the VMs selected by proposed minRmaxC algorithm, the former does not consider the RAM factor when selecting VMs, which may

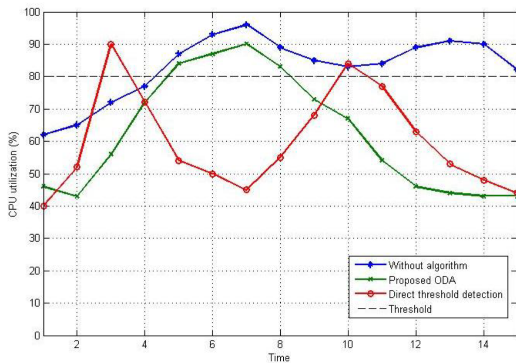


Fig. 3. Performance analysis of overload detection algorithm

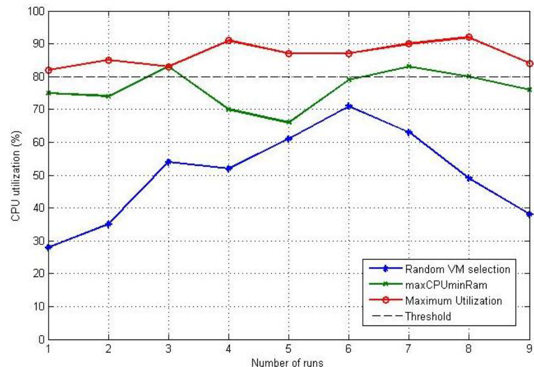


Fig. 4. Performance analysis of proposed minRmaxC VM selection algorithm

www.dbpia.co.kr

have effect on the minimum migration time required for migrating the selected VMs.

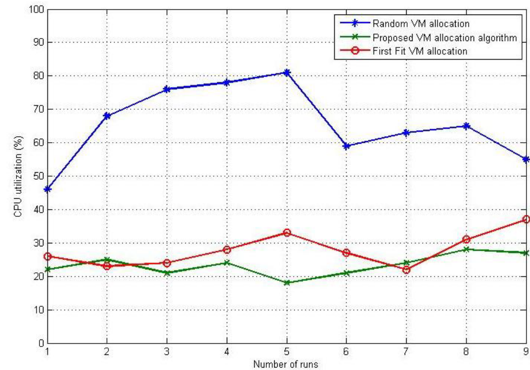
The same behavior is obvious from Table 2 in which the proposed minRmaxC VM selection algorithm selects VM instances on the basis of maximum CPU utilization, whereas the random VM selection algorithm selects VM instances regardless of the CPU utilization for each run.

After selecting the overloaded VM instances, the migration manager module runs the VM allocation algorithm to place the selected VM instances on an efficient compute node. The performance of the proposed algorithm is compared with random VM allocation scheme and first fit VM allocation algorithm[25]. The first fit algorithm starts with the first compute node and determines the availability of the required CPU resources. If it finds enough resources, it places the migrated VM on that compute node, otherwise goes to the next compute node. On the other hand, our proposed algorithm finds the most efficient compute node among all the nodes. For this purpose, it calculates the mean CPU utilization and mean RAM utilization to determine the most efficient compute node from a list of compute nodes. If the calculated mean values satisfy the requirements of the migrated VM, it selects that compute node for VM placement. In case of random VM allocation scheme, the VMs are placed on random compute nodes. Fig. 5 depicts the comparison of our proposed VM allocation

Table 2. VM selection algorithm

CPU(%) random VM selection algorithm	CPU(%) maximum utilization algorithm	CPU(%) proposed minRmaxC algorithm	Number of runs
28	82	75	1
35	85	74	2
54	83	83	3
52	91	70	4
61	87	66	5
71	87	79	6
63	90	83	7
49	92	80	8
38	84	76	9

Fig. 5. Performance analysis of the proposed VM allocation algorithm



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

It can also be clearly seen in Table 3 that the proposed VM allocation algorithm migrates the selected VM instances to the light load compute nodes in terms of CPU utilization, whereas the random VM allocation algorithm on the hand

Table 3. VM allocation algorithm

CPU(%) random VM allocation algorithm	CPU(%) First Fit algorithm	CPU(%) proposed VM allocation algorithm	Number of runs
46	26	22	1
68	23	25	2
76	24	21	3
78	28	24	4
81	33	18	5
59	27	21	6
63	22	24	7
65	31	28	8
55	37	27	9

www.dbpia.co.kr

migrates the selected VM instances to already overloaded compute nodes.

It is pertinent to mention here that the CPU utilization shown in Fig. 4 is higher compared to that of shown in Fig. 5 because the former shows the CPU utilization of those VM instances which are consuming most of the resources of a compute node and thus selected for migration, whereas the latter shows the CPU utilization of an efficient compute node which has resources available to accommodate new VM instances, and thus selected for migrated VM instances.

Ⅴ. Conclusions

In this paper, we have proposed a design and implementation of a framework for dynamic VM migration in OpenStack clouds. The framework addresses the issue of the dynamic VM migration by implementing the proposed overload detection, VM selection, and VM allocation algorithms. It can be easily deployed to the default OpenStack installation by communicating with it by means of the public APIs, and without the requirement of any alterations of OpenStack's configurations. The experiment results show that the proposed framework prevents compute nodes from getting overloaded, selects the VM instances which are consuming most of the compute node's resources, and migrates them to other efficient compute nodes. The proposed algorithms outperform the algorithms that are considered for the sake of comparison.

In the future, we aim to apply the proposed framework for further research on dynamic VM migration and large-scale OpenStack deployments to improve the utilization of resources.

References

[1] S. Bose, A. Pasala, D. Ramanujam A, S. Murthy, and G. Malaiyandisamy, "Sla management in cloud computing: A service provider's perspective," *Cloud Computing*, pp. 413-436, 2011.

[2] M. Bichler, T. Setzer, and B. Speitkamp. "Capacity planning for virtualized servers," in *WITS,* vol. 1, Milwaukee, Wisconsin, USA, 2006.

[3] G. Khanna, K. Beaty, G. Kar, and A. Kochut. "Application performance management in virtualized server environments," in *IEEE/IFIP NOMS 2006*, pp. 373-381, 2006.

[4] G. Jung, K. Joshi, M. Hiltunen, R. Schlichting, and C. Pu, "A cost-sensitive adaptation engine for server consolidation of multitier applications," *Middleware* 2009, pp. 163-183, 2009.

[5] O. Sefraoui, M. Aissaoui, and M. Eleuldj. "OpenStack: Toward an open-source solution for cloud computing," *Int. J. Computer Appl.,* vol. 55, no. 3, Jan. 2012.

[6] OpenStack Nova: http://nova.openstack.org/, 2011.

[7] J. Y. Choi, "Virtual machine placement algorithm for saving energy and avoiding heat islands in high-density cloud computing environment," *J. KICS,* vol. 41, no. 10, pp. 1233-1235, Oct. 2016.

[8] A. Verma, P. Ahuja, and A. Neogi, "pMapper: Power and migration cost aware application placement in virtualized systems," in *Proc. 9th ACM/IFIP/USENIX Int. Conf. Middleware,* pp. 243-264, 2008.

[9] F. Wuhib, R. Stadler, and H. Lindgren, "Dynamic resource allocation with management objectives—Implementation for an OpenStack cloud," in *Proc. Network and Service Management (CNSM), Int. Conf. and Workshop on Syst. Virtualiztion Management (SVM)*, pp. 309-315, 2012.

[10] S. Kumar, V. Talwar, V. Kumar, P. Ranganathan, and K. Schwan, "vManage: Loosely coupled platform and virtualization management in data centers," in *Proc. 6th ICAC,* pp. 127-136, 2009.

[11] W. Zheng, R. Bianchini, G. J. Janakiraman, J. R. Santos, and Y. Turner, "JustRunIt: Experiment-based management of virtualized data centers," in *Proc. 2009 USENIX Annu. Tech. Conf.,* pp. 18-33, 2009.

www.dbpia.co.kr

[12] X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, and D. Gmach, "1000 Islands: Integrated capacity and workload management for the next generation data center," in *Proc. ICAC,* pp. 172-181, 2008.

[13] X. Wang and Y. Wang, "Coordinating power control and performance management for virtualized server clusters," *IEEE TPDS,* vol. 22, no. 2, pp. 245-259, 2011.

[14] G. Han, W. Que, G. Jia, and L. Shu, "An efficient virtual machine consolidation Scheme for multimedia cloud computing," *Sensors,* vol. 16, no. 2, p. 246, 2016.

[15] R. Nathuji and K. Schwan, "VirtualPower: Coordinated power management in virtualized enterprise systems," *ACM SIGOPS Operating Syst. Rev.,* vol. 41, no. 6, pp. 265-278, 2007.

[16] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing SLA violations," in *Proc. 10th IFIP/IEEE Int. Symp. Integrated Netw. Management (IM),* pp. 119-128, Munich, Germany, 2007.

[17] B. Nandi, A. Banerjee, S. Ghosh, and N. Banerjee, "Stochastic VM multiplexing for datacenter consolidation," in *Proc. 9th IEEE SCC,* pp. 114-121, Honolulu, HI, USA, 2012.

[18] *Libvirt, the Virtualization API*, http://libvirt.org/ Accessed in May. 2017.

[19] OpenStack Community, *Ceilometer*, 2013, available at: https://wiki.openstack.org/wiki/Ceilometer/. accessed in: May 2017.

[20] QEMU - open source processor emulator, 2009. http://www.qemu.org

[21] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: towards an open, distributed SDN OS," in *Proc. HotSDN '14*, pp. 1-6, Aug. 2014.

[22] lookbusy -- a synthetic load generator

[23] A. Abdelsamea, et al., "Virtual machine consolidation challenges: A review," *Int. J. Innovation and Appl. Stud.,* vol. 8, no. 4, pp. 1504-1516, 2014.

[24] M. R. Chowdhury, M. R. Mahmud, and M. R. Rashedur, "Implementation and performance analysis of various VM placement strategies in CloudSim," *J. Cloud Computing,* vol. 4, no. 20, Dec. 2015.

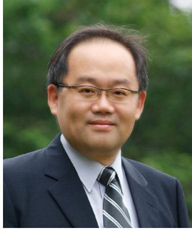
[25] M. R. Chowdhury, M. R. Mahmud, and M. R. Rashedur, "Study and performance analysis of various VM placement strategies," *IEEE/ACIS SNPD,* pp. 1-6, Jun. 2015.

아팍 무하마드 (Afaq Muhammad)

He received BS degree in Electrical Eng. from University of Eng. and Technology, Peshawar, Pakistan, and MS degree in Electrical Eng. with emphasis on Telecom from Blekinge Institute of Technology, Sweden in 2007 and 2010 respectively. Currently, he is pursuing his PhD degree as a KGSP (Korean Government Scholarship Program) scholar at Jeju National University. He has worked as a Research Associate in the Faculty of Comp. Sci. and Eng. at GIK institute of Eng. Sciences and Technology, Pakistan. His research interests are software defined networking, network function virtualization, wireless networks, and protocols.

www.dbpia.co.kr

송 왕 철 (Wang-Cheol Song)



He received B.S. degree in Food Engineering and Electronics from Yonsei University, Seoul, Korea in 1986 and 1989, respectively. And M.S. and PhD in Electronics studies from Yonsei University, Seoul, Korea, in 1991 and 1995, respectively. Since 1996 he has been working at Jeju National University. His research interests include VANETs and MANETs, Software Defined Networks, network security, and network management.

www.dbpia.co.kr