Automated Cloud Migration based on Network Traffic Dependencies

Jargalsaikhan Narantuya, Hannie Zang, and Hyuk Lim
School of Electrical Engineering and Computer Science
Gwangju Institute of Science & Technology
Gwangju, Republic of Korea
{jargalsaikhan.n, hanizang, hlim}@gist.ac.kr

***Abstract*—We propose a cloud migration strategy that migrates applications and services running on multiple virtual machines (VMs) from one cloud environment to another. Current migration techniques mostly focus on migrating VMs between physical servers that are placed inside a single cloud. In this work, we consider an automated cloud migration system that enables a cloud migration of multiple VMs between different cloud infrastructures. However, if multiple VMs are used to constitute a multi-tier web application, the cloud migration must consider any dependency among the VMs. Otherwise, it could cause a significant increase in downtime of the application during the migration. To reduce the migration downtime of applications, we propose a VM grouping scheme using principal component analysis (PCA) based on traffic dependencies. For our experiments, a cloud environment is built with OpenStack, which is an open source software for creating and managing VM-based cloud computing environments.**

***Keywords*—Cloud migration, OpenStack, traffic dependency, virtual machine, PCA, open-source software**

I. Introduction

The cloud lock-in problem is a situation where customers are dependent on a single cloud provider and cannot move to a different cloud environment [1]. One solution to the cloud lock-in problem is cloud migration, which enables the moving of applications of customers in one cloud to another cloud. However, current migration techniques mostly focus on migrating VMs between physical servers that reside in a single cloud. In this paper, we consider an automated cloud migration system that enables the migration of VMs between different clouds. Such a system can be used by customers to change cloud providers if they can take the advantages of moving to a new cloud environment.

Applications in the cloud are generally deployed across multiple VMs, and these VMs are often part of multi-tier web applications. If we migrate the VMs without considering traffic dependencies, the downtime of the applications could be increased during the cloud migration. Thus, migrating applications between different clouds may require joint migration of dependent VMs [2]. Based on traffic dependencies between the VMs, we propose a VM grouping strategy using PCA for joint migration of dependent VMs.

Most research work on VM migration has focused on the improvement of performance in a single cloud through techniques such as load balancing based migration, fault tolerance, and energy-efficient VM placement. Ghribi *et al.* [3] introduced an energy-aware VM migration strategy for reducing energy consumption in the datacenter. They proposed a strategy to reduce the number of physical servers used by letting idle physical servers go into sleep mode. In [4], Wang *et al.* proposed a VM migration plan in the datacenter that reduced the total migration time in software-defined networking (SDN) environments. Their strategy allows multiple VMs to be migrated simultaneously through multiple routing paths with the help of the SDN environment.

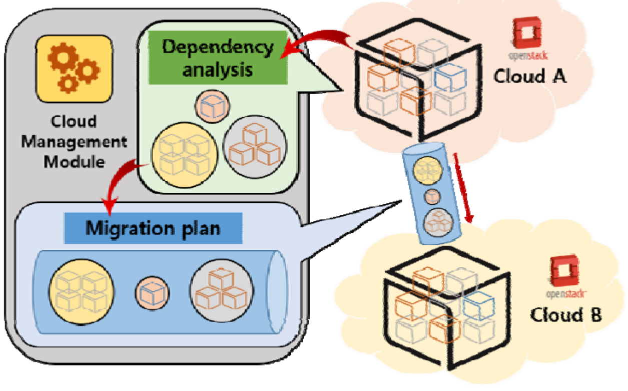


Fig. 1. Cloud Migration

Lu *et al.* [7] introduced VM grouping strategy based on minimum-cut and k-means algorithms for inter-cloud live migration. They attempted to reduce the amount of traffic between the clouds during the inter-cloud migration. In [8], Tziritas *et al.* proposed a service migration scheme for minimizing network overhead in a cloud. They introduced a strategy of deciding at what point in time the service must be migrated to another cloud for reducing network overhead.

VM migration techniques within a single cloud are developed, as can be seen by tools such as VMWare vMotion [9] and live migration in OpenStack [10]. VMware vMotion enables the migration of running VMs from one physical server to another inside a VMware vSphere environment. However, there are some limitations of VMware vMotion. For instance, the VM being migrated must remain within a single datacenter, and the source and destination servers must have access to the same data store. The OpenStack live migration tool enables the movement of VM instances from one compute node to another. This migration scheme supports the migration of VM instances to a single cloud environment only, and requires shared storage of compute nodes like vMotion.

II. System Overview

As shown in Figure 1, we built two private clouds using OpenStack. OpenStack is an open source software platform for building and managing cloud computing environments [10]. This platform provides fundamental functions required for building cloud based services, such as computation, networking, and storage [5]. For our automated cloud migration system, we developed a new framework consisting of three main functionalities: dependency detection, VM migration, and transmission speed maximization.

A. *Dependency Detection*

We measured the traffic between VMs that were in the process of running cloud applications in order to detect dependencies between them. One of the important operations when detecting traffic dependencies of VMs is the capturing of traffic between VMs in the cloud. Based on a port mirroring method, we implemented software to monitor east-west traffic in the cloud. Port mirroring is a well-known technique, in which packets are replicated based on their incoming or outgoing ports, and then forwarded to other ports. For each interval of time during which the cloud applications were being run, the total number of bytes of all packets transferred between VM pairs were summed up and recorded as traffic volume. The obtained traffic information was then used for detecting dependencies between the VMs. Further analysis and grouping of the VMs using this dependency information was then computed using our proposed VM grouping strategy.

B. *Migrating VMs*

We developed an automation framework to migrate the VMs between different OpenStack-based clouds. In the framework, there are two main modules: one which works at the source, sending VMs to their destination, and another which works at the destination, receiving inbound VMs. We have named the modules “sender” and “receiver,” respectively, and their operation is described as follows:

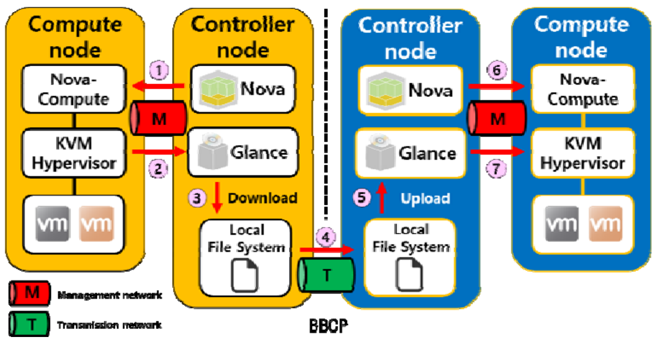


Fig. 2. VM migration steps

At the source cloud, we first obtain the network configuration information of the VMs by using the Nova and Neutron APIs. The sender then conveys the obtained network configuration information to the receiver at the destination cloud. After sending the configuration information, the sender starts to transmit the VMs to their destination based on our migration strategy. As shown in Figure 2, the steps one through four illustrate the process of sending VMs from the source cloud. First, a control message is sent to initiate the copying of the current state of the VM. Based on the control message, the cloud system copies the current state of the VM during step two. As a result of step 2, a new VM image is created in Glance, which is downloaded as an image file to the local system in step 3. Lastly, in step 4, the downloaded image file is transferred from the source to the destination. In this manner, all of the VMs are able to be sent from source to destination. After sending all of the VMs, the sender transmits a control message to the destination to let the receiver know that sending has completed.

The receiver method works at the destination, receiving and configuring the inbound VMs. To receive the VMs, the receiver method uses an *inotify* event handler that signals the arrival of a new file at the destination. *Inotify* is a Linux kernel subsystem that acts to extend filesystems, allowing them to notice changes that are made, and to report those changes to applications [11]. First, the receiver receives the network configuration information of the VMs from the sender. Next, based on the received network configuration information, the receiver uses Neutron APIs to recreate the network configuration at the destination, including the creation of floating IPs identical to the floating IPs of the VMs in the source cloud. As shown in Figure 2, steps four to seven illustrate a process of receiving a VM. With the help of the *inotify* event handler, the receiver receives an image of the VM from the sender in step 4. After receiving the image, the receiver uploads the image from local storage to Glance in step 5. Next, in step 6, a control message is sent to create a VM from the received image file and its related network configuration. In step 7, based on the control message, the cloud system creates a VM with its relevant configurations.

C. *Maximizing Transmission Speed*

As shown in Figure 2, the step 4 of the migration performs the transmission of a VM image between the clouds. The file transmission speed between the clouds is inversely proportional to the migration time [6]. In order to reduce the migration time, we connect the two clouds through a 10G network. To efficiently utilize the 10G network, we use the BBCP file transmission tool. BBCP is a multi-streaming file transmission tool that splits the files into multiple streams, each of which is then transferred simultaneously [12]. As a result, the file transmission speed of BBCP is faster than single-streaming file transmission tools such as SCP and SFTP. We compared the performance of BBCP and SCP by transmitting a 100 GB file through a 10 Gb/s Ethernet link, and found the transmission speed of BBCP to be almost twice that of SCP, with an average speed of 2.095 Gb/s versus 1.188 Gb/s, respectively.

III. Proposed Cloud Migration

A. *Background*

Generally, applications in a cloud are hosted across multiple VMs, and the proper running of these applications requires that all related VMs to be run together. However, it is impossible to migrate the VMs from one cloud to another without stopping them, even if only for a short time. Therefore, a cloud migration will always impact the applications that are running on that cloud. During a cloud migration, an application becomes unavailable when the first VM of that application stops, even though other VMs the application relies on may still be running. Therefore, service downtime due to a cloud migration is measured by the time beginning when the first VM in the source cloud stops, and ending when the last VM in the destination cloud restarts.

The downtime of the application during migration of multiple VMs fluctuates depending on the migration strategy. An inappropriate migration might increase the number of VMs to be migrated until all the VMs belonging to the application have been migrated. To decrease the application downtime, we categorize the VMs into affinity groups, and migrate together those VMs that belong to the same affinity group.

B. Grouping VMs

Assume there are n virtual machines in a cloud, and that these VMs should be migrated to another cloud. Let the set of VMs be represented by V = {$V_{1,} V_{2,} V_{3\,.....}\,V_n$}. Let "i" denote an index of the VMs such that i = {1, 2 … n}. By using the network traffic dependency detection software that we developed, we obtain traffic information of VMs in the cloud. The obtained traffic information is used for creating a traffic information matrix W. Let W be an n-by-n traffic information matrix, with each of its elements $w_{i,j}$ representing a traffic weight between the VMs V_i and $V_{j.}$

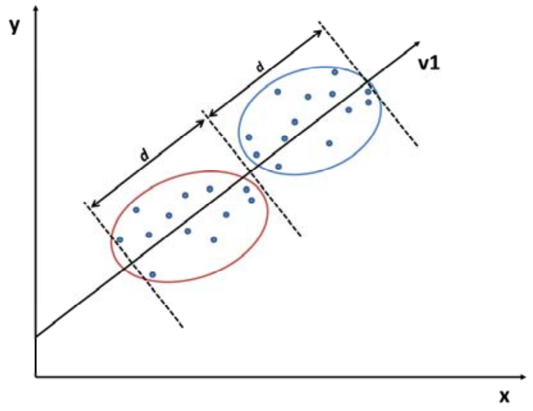


Fig. 3. Finding similarity group of the VMs using PCA

Our proposed strategy categorizes VMs into affinity groups based on the traffic dependencies of the VMs. To compute the affinity groups, our proposed algorithm uses principal component analysis (PCA). PCA converts a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. PCA projects these observations into a new, uncorrelated space of smaller dimension [13]. Using PCA, the traffic information matrix W is reduced to an m-by-n matrix, meaning the n VMs are clustered into "m" groups of related VMs. The details of the grouping strategy using PCA are explained in the following paragraph.

As shown in Figure 3, the VMs have been clustered into two affinity groups using PCA. To achieve this, we first compute a covariance matrix of the traffic information matrix W. Next, we find the eigenvalues and eigenvectors of the covariance matrix. After finding the eigenvalues and eigenvectors, we select the eigenvector with the highest eigenvalue as a principle component of the data set. The selected eigenvector is illustrated in Figure 3 as the vector labelled v1. The XY coordinates of the data are then projected onto a single point on v1 and clustered into two groups, depending on the distance between the points.

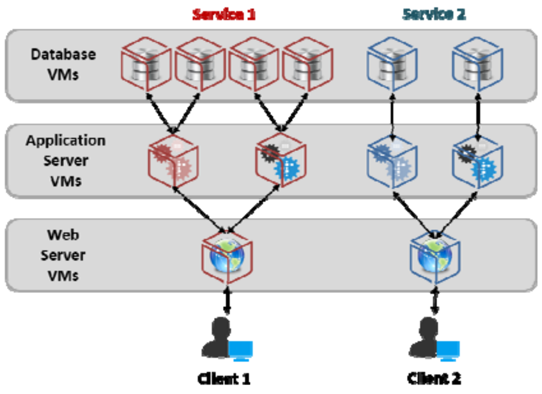


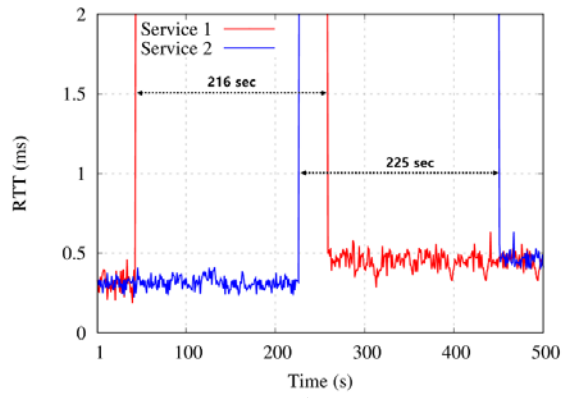
Fig. 4. 3-tier architecture of web services

IV. Experiment Result

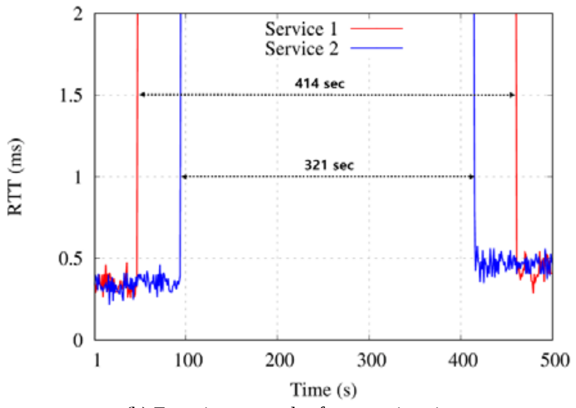
As mentioned in the System Overview section, we built two independent clouds using OpenStack, named cloud A and cloud B. The physical machines have the following specifications: The controller nodes have an Intel Xeon W3565 3.2 Ghz CPU, with 48 GB RAM, and 1 TB SSD. The compute nodes have an Intel Xeon E-5 1680v4 3.4 GHz CPU, with 128 GB RAM, and 1 TB SSD. We built a 3-tier architecture of web-based services in cloud A. As shown in Figure 4, there were two different services: one consisting of seven VMs, the other consisting of five VMs. The VMs that were used for building these services were configured with Ubuntu 14.04.5 LTS OS, and provisioned with 1 vCPU, 2 GB RAM, and 20 GB storage.

To evaluate the performance of our proposed migration strategy, we conducted a migration from cloud A to cloud B of the two services mentioned above, while providing two clients, named client 1 and client 2, access to the services. We compared the performance of our proposed migration strategy against a naïve migration strategy. The naive migration of the cloud randomly decides the migration order of the VMs, without considering any dependencies between the VMs.

Figure 5(a) shows an experimental result of our proposed migration strategy. Before starting the migration, both services responded normally, from cloud A, to the requests of the clients. We started the cloud migration at t = 35, using our proposed migration strategy, and service 1 became unavailable at t = 43. Service 2 continued to respond normally to its client during this time. After 216 seconds of downtime, service 1 finished restarting and began to respond, from cloud B, to its client, at t = 259. Service 2 continued to respond normally during the migration until t = 226. This means our proposed migration strategy successfully found the dependencies between the VMs, and migrated the dependent VMs together. As a result, the VMs belonging to service 1 were migrated before t = 226, and the VMs belong to service 2 were migrated after t = 226. At t = 451, service 2 completed restarting and began to respond, from cloud B, to client 2. Accordingly, the downtime of service 2 is measured by the difference in time between 226 and 451.



(a) Experiment result of proposed migration



(b) Experiment result of naïve migration

Fig. 5. Round Trip Time w.r.t Service Request Time

Figure 5(b) presents an experimental result of the naïve migration strategy. Compared to our proposed strategy, the downtime of both services is considerably larger for the duration of the migration. The downtime of service 1 increased to 414 seconds, and the downtime of service 2 increased to 321 seconds. The reason for the increase in downtime is that the dependencies between the VMs were not considered. As a result, VMs belonging to the same service not being migrated together.

V. Conclusion

In this work, we implemented an automated cloud migration system based on network traffic dependencies. For the detection of dependencies among the VMs, we proposed a PCA-based VM grouping strategy based on gathered network traffic information. Experimental results show that our proposed cloud migration strategy outperforms a naïve migration strategy. The cloud environment used for this work was built with OpenStack, which is a set of software tools for constructing and managing cloud computing platforms. Additionally, we plan to extend our work to enable cloud migration between public clouds such as Amazon, Google Cloud, and Azure. The source code and configuration of this work are available in [14] and are provided under the Apache license.

Acknowledgment

This work was partly supported by GIST Research Institute (GRI) grant, and by NRF grant (2017R1A2B2010478) and IITP grants (No. R7117-16-0218, Development of Automated SaaS Compatibility Techniques over Hybrid / Multisite Clouds, and No. B0101-15-0557, Resilient CPS Research) funded by the Korea government (MSIP).

References

[1] J. Opara-Martins, R. Sahandi, and F. Tian, “Critical analysis of vendor lock-in and its impact on cloud computing migration: A business perspective,” *Journal of Cloud Computing*, vol. 5, no. 1, pp. 1-18, 2016.

[2] E. Keller, S. Ghorbani, M. Caesar, J. Rexford, “Live migration of an entire network (and its hosts),” *ACM Workshop on Hot Topics in Networks*, pp. 109-114, 2012

[3] C. Ghribi, M. Hadji, D. Zeghlache “Energy efficient VM scheduling for cloud data centers: exact allocation and migration algorithms” *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 671-678, 2013.

[4] H. Wang, Y. Li, Y. Zhang, D. Jin. “Virtual machine migration planning in software-defined networks,” *IEEE Conference on Computer Communications (INFOCOM)*, pp. 487-495, 2015.

[5] J. So, D. Kim, H. Kim, H. Lee, and S. Park, “LoRaCloud: LoRa platform on OpenStack,” *IEEE NetSoft Conference and Workshops (NetSoft)*, pp. 431-434, 2016.

[6] S. Akoush, R. Sohan, A. Rice, A. W. Moore, and A. Hopper, “Predicting the performance of virtual machine migration,” *IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 37-46, 2010.

[7] T. Lu, M. Stuart, K. Tang, and X. He, “Clique migration: Affinity grouping of virtual machines for inter-cloud live migration,” *IEEE International Conference on Networking, Architecture, and Storage (NAS)*, pp. 216-225, 2014.

[8] N. Tziritas, S. Khan, T. Loukopoulos, S. Lalis, C.Z. Xu, K. Li, and A. Zomaya, “Online inter-datacenter service migrations,” *IEEE Transactions on Cloud Computing*, 2017.

[9] http://www.vmware.com/pdf/vmotion_datasheet.pdf

[10] https://www.openstack.org/

[11] http://man7.org/linux/man-pages/man7/inotify.7.html

[12] https://www2.cisl.ucar.edu/resources/storage-and-file-systems/bbcp

[13] https://en.wikipedia.org/wiki/Principal_component_analysis

[14] https://github.com/K-OverCloud/Composable-VM-Migration