A Virtual Machine Consolidation Framework for CloudStack Platforms

Tanasak Janpan, Vasaka Visoottiviseth
Faculty of Information and Communication Technology
Mahidol University
Nakorn pathom 73170, Thailand
bestnatade@gmail.com, vasaka.vis@mahidol.ac.th

Ryousei Takano
Information Technology Research Institute
National Institute of Advanced Industrial Science and Technology (AIST)
Tsukuba, Ibaraki 305-8568, Japan
takano-ryousei@aist.go.jp

***Abstract* – Virtual machine (VM) consolidation is an emerging solution for energy saving in cloud data centers. An appropriate VM consolidation policy must change accordingly, depending on the situation. To easily deploy a VM consolidation solution, this paper proposes a VM consolidation framework for Apache CloudStack, which is a popular open source cloud platform software suite. We separate the VM consolidation mechanism from the policy used to decide which VMs should be moved to somewhere more appropriate to run. This framework is a modular architecture that combines various resource monitoring systems, power control mechanisms, and VM packing algorithms. Using the proposed framework, we have demonstrated that a web application can consolidate and deconsolidate VMs so as to balance the load of CPU utilization among physical machines in response to the number of physical machines in operation. Experimental results reveal the proposed framework is preliminary, but provides the required functionality.**

Index Terms* – *VM consolidation, Cloud Computing, Energy-Efficient, Data Center, Green Computing

I. Introduction

In recent years, the development of the cloud computing model has tended towards to the establishment of large-scale virtualized data centers. Currently, the use of cloud-based applications by Internet users is increasing and thus tends to produce high traffic volumes. Cloud data centers have to add thousands of servers just to handle the growing resource demand from users. Those data centers certainly consume tremendous amounts of electricity for the additional computational power. High levels of power consumption are not only caused by computer components but also come from the requirements of the cooling system. Moreover, data centers also often face the problem of server sprawl and underutilization that is considered a major cause of energy inefficiency. Some running servers have a very low usage rate. They consume a lot of energy without being sufficiently used. These servers contribute significantly to rising operating costs, low energy efficiency, performance degradation, and increasing environmental harm.

Virtual machine (VM) consolidation promises to be a significant emerging solution to alleviate these problems in cloud data centers. Basically, the cloud system is built up of numerous physical servers and each of these servers can run multiple virtual machines. Theoretically, VM consolidation merges and re-allocates target VMs into as small a number of running physical servers as possible according to their resource demands. Underutilized servers should be switched to the sleep mode or switched off so that they consume no power. Due to static energy consumption by server components, running servers at the maximum utilization level is considered to be energy efficient. Much research shows that cloud data centers can achieve much better energy efficiency and could effectively reduce operating costs due to tremendous energy savings by applying VM consolidation. However, there is no one-fits-all solution. A suitable policy of VM consolidation depends on various factors, such as resource availability, the number of users, the characteristics of workloads, and so on.

This paper proposes a framework for VM consolidation in the CloudStack platform. The proposed framework is able to provide an extended application to CloudStack with the implementation of VM consolidation using live migration. The framework allows us to implement different policies for different purposes. This framework supports not only consolidating VMs but also deconsolidating VMs. The latter is required for avoiding performance degradation due to overload. We have demonstrated a simple VM packing algorithm based on the load balancing of machines.

The rest of the paper is organized as follows. Section II describes the background and structure of the framework. Section III describes the details of the VM consolidation framework. In section IV, we propose a simple and proof-of-concept VM packing algorithm. Section V presents the implementation and evaluation results. Section VI discusses the related work, and we conclude the paper in Section VII.

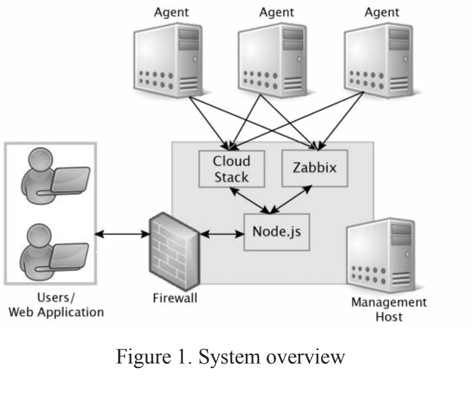

Figure 1. System overview

II. System Overview

A. Overview

VM consolidation remains a fascinating solution, as an effective power-saving technology that would satisfy a critical need for data centers. Cloud computing is thus an advanced computing model, which is being discussed mainly in the context of the future of the web. Figure 1 shows an overview of our VM consolidation system which combines three technologies together: CloudStack [1], Zabbix [2] and Node.js [1]. The system consists of the controllers and the agents. Each controller runs on the management host; each agent runs on the computing host. The pair of controller and agent works as a component. CloudStack is the main component of the system, and we utilize the CloudStack REST API to implement the live migration function. Live migration is one of the most important mechanisms of VM consolidation that will enable migration to perform in the background. VM consolidation requires that the policy should consider the status of the CPU, storage, memory, power consumption, and temperature. To acquire the above data, we utilize Zabbix, which is open source software for efficient performance monitoring. Zabbix offers various ways to monitor and track performance and availability of network servers. The outstanding feature of our VM consolidation framework is the ability to monitor and operate the consolidation process over the web application. In addition, our framework was actually inspired by existing research in the area of energy efficiency in the cloud data center. More details about seminal work will be discussed later in Section VI.

B. CloudStack and Live Migration

There are several open source infrastructure-as-a-service (IaaS) software suites used for building enterprise-based private cloud environments like Amazon cloud data centers. The best known are CloudStack, OpenStack, Eucalyptus, and OpenNebula. Among these technologies, CloudStack has been very fast-growing in popularity in the cloud software market, because it allows deploying and managing multi-tenant private clouds with large scale virtual machines. It also supports a number of popular hypervisors such as KVM, VMware, and XenServer. CloudStack is a turnkey solution for cloud infrastructure software that also has a comprehensive set of features. Users can easily manage their cloud via the web interface, command line tools and the RESTful API. CloudStack and KVM nicely support the parallel live migration of multiple virtual machines. Live migration refers to a process of transferring a running VM between different physical servers without disruption of services, including memory, storage, and network connectivity. Live migration is an important technology for VM consolidation when the goal is to migrate VM without interruption of services running on the VM, depending on the resources needed. CloudStack also includes the capability of checking whether any particular VM and destination server may be suitable for migration, according to the needs of each request.

C. Resource Monitoring Software

Zabbix provides many ways to monitor different aspects of IT infrastructure, and those tools can be characterized as a distributed monitoring system with a centralized management server. The main Zabbix system is installed on the management host; and the Zabbix agent is on the agent host. VM consolidation makes use of Zabbix as a data collector from each agent host in order to measure the status of devices with a provided threshold or a suitable policy. Zabbix is actually open source software that provides a high performance and a high capacity monitoring system, which is able to monitor hundreds of thousands of devices. The major functions of Zabbix are designed to monitor and track the status of various network services, servers, and other network hardware, which can be updated every minute. The data gathered resides in the Zabbix database and is presented in the form of easily understandable graphs through the Zabbix web front-end. In the case where the server devices support an IPMI interface, Zabbix can also monitor temperature and power consumption. The web front-end of Zabbix is an acceptable tool for making configuration changes to the Zabbix server. In addition, Zabbix also provides an API or programmable interface, which allows creating, updating and fetching Zabbix objects through the JSON RPC protocol.

D. Node.js and Web front-end

We have decided to use Node.js as the main component of the development environment. Node.js gains a lot of attention as a software platform used to build scalable network applications, especially on the server side. Node.js utilizes JavaScript as its scripting language using the V8 JavaScript runtime. Using Node.js, our entire framework was written in the JavaScript language. Major advantages of Node.js include non-blocking I/O and an event-driven model which allow handling high throughput with much better memory efficiency than the thread-based model. Node.js also leverages super fast compiling and transforming JavaScript into native code. Moreover, Node.js is an open source cross-platform suite that can run reliably on Mac OS X, Windows, and Linux operating systems.

III. VM Consolidation Framework

A VM consolidation framework provides programmers with the functionalities of monitoring VM status and controlling the VM placement underlying a cloud platform; and also provides various VM packing algorithms that run on top of the framework.

We have developed a VM consolidation framework on the CloudStack platform, which runs separately from the core of CloudStack services. The framework interacts with the core of the CloudStack service using the CloudStack API. In addition, it is stateless in term of databases which means it does not require the use of any particular database by itself, since the third party monitoring software will also have its own database and performs very well.

The proposed framework consists mainly of four components: a CloudStack operator, a monitoring operator, a mapping operator, and a power controller. They communicate with each other via the REST API.

The CloudStack operator is responsible for interacting with the CloudStack API in order to utilize the CloudStack service. Once the request is received in the form of the REST API, the request handler first checks the purpose of the request and invokes the request from the CloudStack via its API. The operator also handles functions to perform live migration. The request generally contains the information on VMs and destination hosts. Once the mapping operator determines the migration strategy, this operator submits requests to the CloudStack to migrate VMs. Each migration obtains a JOBID that can be used to check the status of the process. Moreover, the operator can also obtain monitoring information such as VM statistics and current CPU utilization.

The monitoring operator is responsible for collecting monitoring data from monitoring systems such as Zabbix. As Zabbix performs as a data collector, data collected is stored in the Zabbix central database for each minute. To obtain the historical data from Zabbix, it requires a pretty tricky process. We first request the Zabbix with its hostname to get the Zabbix *hostid*, then check the graph id of each host. After we get the *graphid*, we can get a *graphitem* by specifying the *graphid*. Finally, we can fetch the historical data with the *itemid* that is specific to each *graphitem*. We can also specify the aomunt of historical data or any parameter to be fetched by using the Zabbix API.

The power control operator is responsible for switching the host off and on to save energy. There are many methods available to perform power control such as direct SSH, IPMI, and Wake-on-LAN. In our case, the operator will open an SSH connection to apply a sleep mode to the target host by invoking a specific command, or to shutdown the target host for saving power consumption. When a host needs to be re-activated from the sleep mode, the operator will send a command via the SSH to the power management to invoke a command to start up a specific destination host.

The mapping operator is responsible for performing the VM packing algorithm in order to map data based on a suitable policy. This function is requested based on the status of hosts, user VMs, system VMs and target hosts. After the mapping process according to the VM packing algorithm is finished, the result is sent back to the main section of the framework.

We need to take care of not only the user VMs, but also the system VMs. Unlike user VMs, CloudStack automatically deploys various types of system VMs, for example, secondary storage VMs, virtual router VMs and console proxy VMs. Even though user VMs do not deploy on some hosts, we cannot shutdown those hosts, because system VMs may run on them. In that case, all system VMs need to be migrated. To migrate system VMs, a different API is required. The migration operation must be flexible enough to handle both user VMs and system VMs.

IV. VM Packing Algorithm

A. *Packing algorithm*

The purpose of the packing algorithm is to map VMs on the minimum number of physical machines. VM mapping can be considered as a bin-packing problem, where a bin represents a physical machine; items are the VMs that have to be allocated; bin sizes are the available CPU capacities of the machines; and weight corresponds to the CPU consumption on each VM. We have employed a simple packing algorithm based on load balancing of CPU utilization. This algorithm is similar to the First-fit decreasing (FFD) algorithm [4], which is widely used to solve bin-packing problems. The FFD algorithm provides a fast but non-optimal solution. The algorithm will sort items in decreasing order according to the weight, the first item will be placed into the first bin. If the current bin does not offer a fit, then the algorithm will try to put the item into the next bin. In our deconsolidation algorithm, we find the smallest bin size in which we can put the item rather than trying indefinitely. For the consolidation algorithm, we sort bins according to size. We remove one item from the first bin to put into a new reserved bin. Until the target bin is reached to the expected size, we continue to move the item to the next bin. Nonetheless, the computation cost is $O(N \log N)$, in which N is the number of VMs to be migrated.

B. *Mapping methods*

To demonstrate the feasibility of the proposed framework, we have designed an FFD-like VM packing algorithm that consists of two mapping methods. Use of these methods is assumed as follows. A consolidation method is used to remove hosts from the resource pool for maintenance. A deconsolidation method is used to add hosts after the maintenance.

Consolidation method: Figure 2 illustrates our idea of a consolidation method based on load balancing of CPU utilization. Initially, there are host B and C, each running three VMs. Host A is required to reactivate from the sleep mode. It consolidates one VM from B and one from C which is in keeping with the load balancing model. The pseudo code of this process is shown in Figure 3.

Before we perform the packing algorithm, we first obtain the resource information, including active hosts, user VMs, and system VMs, from the CloudStack operator. We also obtain a target host, which is the host that tries to consolidate VMs from other active hosts. After the system obtains all necessary information, then it selects the algorithm to use. In this case, our consolidation algorithm is interested only in user VMs. System VMs consume very low CPU utilization; thus we have no need to migrate them. However, we need to migrate system VMs only when the host that contains system VMs switches to the sleep mode. To get migration mapping for this algorithm, we first retrieve the host with the lowest CPU utilization from the active hosts to set the main threshold which will be considered as an indicator to determine when the consolidation should finish. After the threshold is established, then we find a source host, i.e., an active host, with the highest CPU utilization. We randomly select one VM from the source host and map it with the target host. This may be not an optimal solution, but it is certainly quite fast. Actually, the best process should take the most suitable VM for the current target host. Finally, we keep doing the above process until the target host has CPU utilization greater than the threshold. In other words, we will perform consolidation until we can balance the CPU utilization on all hosts.

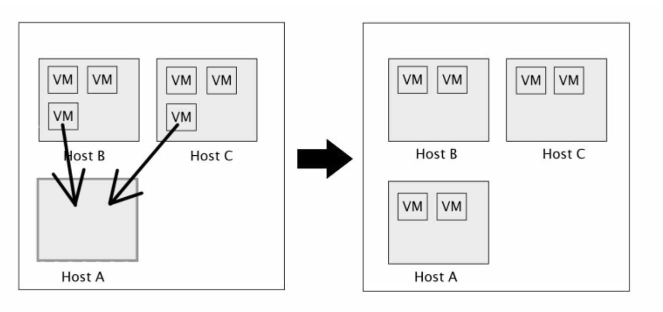

Figure 2. Consolidation method paradigm

```
consolidation (target_host) {
    hosts = get all hosts list from CloudStack
    vms = get all VMs from CloudStack

    min = get minimum utilization from hosts
    until target host utilization is greater than hosts[min] {
        max = get maximum utilization from hosts
        for each vm in vms {
            if vm.host match hosts[max] {
                mapping push vm and target_host together
                break the loop
            }
        }
    }
    return mapping
}
```

Figure 3. Consolidation algorithm based on CPU load balancing

Deconsolidation method: Figure 4 illustrates the idea of the deconsolidation method based on load balancing of CPU utilization. Host A contains four VMs which are required to be shutdown. It finally migrates VMs on to hosts B and C.

While the consolidation attempts to build a mapping of VMs from another host to the target host, deconsolidation does the opposite. It tries to decentralize mapping of VMs from the target host to other hosts. In this case, we attempt to switch a host to the sleep mode. Therefore, we need to migrate all user VMs and system VMs away from this host. Since the CPU usage of system VMs are negligible, we disregard them as we immediately map all system VMs to the host that has the minimum CPU utilization. After the system has received all resource information from the CloudStack operator and all system VMs are mapped, we then find a host with the minimum CPU utilization, and designate this host as the destination host. We randomly select one VM from the target host and map it with the destination host. We will keep doing the above process until all VMs are mapped. The pseudo code of the deconsolidation process is shown in Figure 5.

```
deconsolidation (target_host) {
    hosts = get all hosts list from CloudStack
    vms = get all VMs from CloudStack
    systemVMs = get all system VMs form CloudStack

    min = get minimum utilization from hosts
    mapping push systemVMs and host[min]

    for each vm in vms {
        if vm.host match target_host {
            min = get minimum utilization from hosts
            mapping push vm and hosts[min] together
        }
    }
    return mapping
}
```

Figure 4. Deconsolidation method paradigm

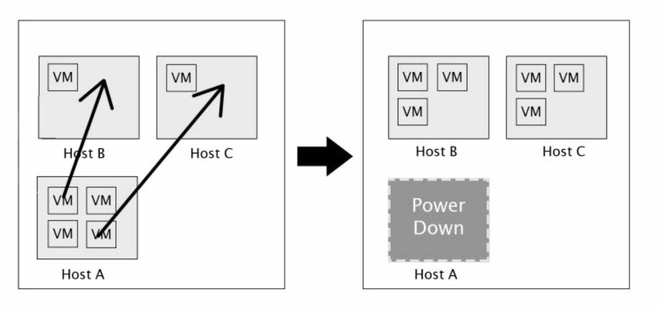

Figure 5. Deconsolidation algorithm based on CPU load balancing

V. IMPLEMENTATION AND EVALUATION

A. Implementation

Our system is implemented using the CoffeeScript and Node.js. CoffeeScript is a high productivity programming language that compiles down to JavaScript. This system runs as a web server by using Node.js on the management host. We also have installed add-in modules of Node.js to support CloudStack [5] and Zabbix [6], and a web application framework, Express.js [7].

B. Evaluation

All experimental results in this paper were performed on four physical machines with Ubuntu version 13.04 installed. One machine runs as the cloud management host, while the others are cloud agents. On the management host, we installed CloudStack version 4.0.2 and Zabbix version 2.0. On the agent host, we installed the CloudStack agent and the Zabbix agent. KVM is used as the hypervisor. The agent hosts are named khonmek-7, khonmek-9, and khonmek-10.

We have conducted experiments with three cases of consolidation and deconsolidation. In our experiments, we deployed 14 VMs running on CloudStack. Each VM has a simple loop program running just for simulating CPU utilization, where the CPU utilization is up to 13%.

Table I reveals the performance testing results of the consolidation and distribution algorithm. In this experiment, we conducted the three experimental cases shown below.

TABLE I. ELASPSED TIME OF VM CONSOLIDATION

Experiments	*Consolidation (seconds)*	*Deconsolidation (seconds)*
Case 1	26	38
Case 2	28	52
Case 3	28	73

Case 1: In the initial stage, there were 4, 4, and 6 VMs running on khonmek-7, khonmek-9, and khonmek-10, respectively. Khonmek-10 was shutdown, and then 5 VMs needed to be migrated to other hosts. It took 38 seconds for the migration process. Khonmek-10 was re-started, and then it consolidated 5 VMs from other hosts. This took 26 seconds. All migration was performed properly. Table II shows the CPU utilization of this case.

Case 2: In the initial stage, there were 5, 2, and 7 VMs running on khonmek-7, khonmek-9, and khonmek-10, respectively. Khonmek-10 was shutdown, and then 7 VMs needed to be distributed to other hosts. It took 52 seconds for the migration process. Case 2 took more time than Case 1 because Case 2 had two more VMs. Khonmek-10 was re-started, and then it consolidated 5 VMs from other hosts. It took 28 seconds for this case. All migration was performed properly. Table III shows the CPU utilization of this case.

Case 3: In the initial stage, there were 2, 2, and 10 VMs running on khonmek-7, khonmek-9, and khonmek-10, respectively. Khonmek-10 was shutdown, and then 10 VMs needed to be distributed to the other hosts. It took 73 seconds. This was time consuming because khonmek-10 was over-subscribed. Table IV shows that the CPU utilization of each VM on khonmek-10 at the initial stage could not reach 13% since the total CPU utilization exceeds the capacity of the host CPU, and thrashing occurs. Thus, some VMs could not perform at the maximum CPU speed. This could ensure the mapping process cannot work properly because of an inefficient CPU calculation. When VMs are migrated to other hosts, they can run at the maximum CPU speed. Khonmek-10 was re-started, and then it consolidated 5 VMs from other hosts. It took 28 seconds. All migration was performed properly. Table IV shows the CPU utilization of this case.

TABLE II. CPU UTILIZATION FOR CASE 1

Hosts	Initial CPU% (VMs)	Shut down CPU% (VMs)	Start up CPU% (VMs)
Khonmek-7	53.88 (4)	91.26 (7)	66.68 (5)
Khonmek-9	56.81 (5)	90.47 (7)	53.81 (4)
Khonmek-10	56.81 (5)	0 (0)	66.78 (5)

TABLE III. CPU UTILIZATION FOR CASE 2

Hosts	Initial CPU% (VMs)	Shut down CPU% (VMs)	Start up CPU% (VMs)
Khonmek-7	66.68 (5)	91.20 (7)	67.32 (5)
Khonmek-9	27.96 (2)	90.31 (7)	54.05 (4)
Khonmek-10	90.47 (7)	0 (0)	65.29 (5)

TABLE IV. CPU UTILIZATION FOR CASE 3

Hosts	Initial CPU% (VMs)	Shut down CPU% (VMs)	Start up CPU% (VMs)
Khonmek-7	27.00 (2)	99.94 (8)	66.19 (5)
Khonmek-9	26.69 (2)	70.31 (6)	53.65 (4)
Khonmek-10	99.97 (10)	0 (0)	65.86 (5)

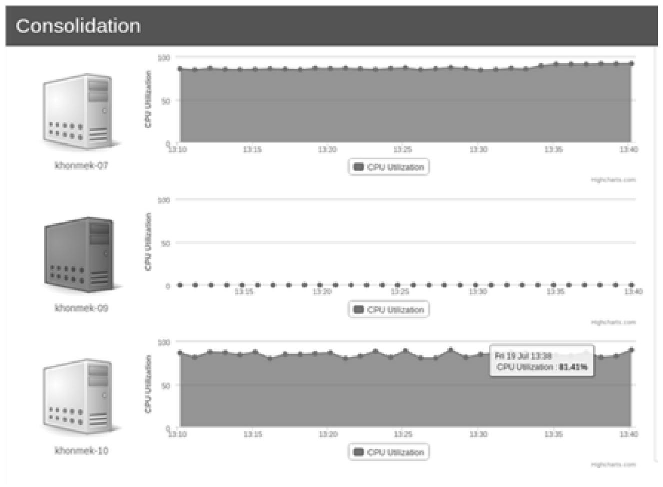

Figure 6. Screenshot of a web application simulating the consolidation process

Based on the results, we show that our VM consolidation only takes approximately one minute, where the migration time depends on the volume of VMs running on each host.

Figure 6 shows a screenshot of a web application simulating the consolidation process.

VI. RELATED WORK

VM consolidation is an important solution in cloud data centers to achieve energy efficiency, improving utilization of physical resources, reducing energy consumption, reducing operating costs, and decreasing environmental impact. Many attempts have been made at proposing different procedures to improve the performance of VM consolidation.

The OpenStack-neat [8] is a similar work aimed toward providing an extensible framework for VM consolidation. The target is OpenStack platforms, and it interacts with the

OpenStack core services through the REST API. To keep track of the resource usage of VMs, historical log data is stored in a MySQL database. This framework consists of three components. The first one is a global manager running on the management server. Its main task is to make global decisions such as mapping VMs, VM migration, and decisions involving the power controller. The rest of components are a local manager and a data collector which run on the agent host. The local manager is responsible for local decisions such as underload, overload, and selecting the VM to migrate, while the data collector is responsible for periodically collecting the resource usage of each VM and submitting the data to the central database.

Takeda, et al. [9], proposed a rank-based VM consolidation method. Each physical machine has a unique server rank value for representing the selection priority of a physical machine. Ranking may be designed for minimizing power consumption, perform autonomic remapping for saving power in datacenter using dozens or even hundreds of VMs. This method is similar to the First Fit Decreasing (FFD) algorithm that is widely used to solve bin-packing problems.

Feller, et al. [10], implemented fully decentralized dynamic VM consolidation to improve energy conservation in a cloud data center. This paper discussed the performance difference between static and dynamic VM consolidation, and also discusses many different types of mechanisms such as centralized, hierarchical, and ring-based approaches, which result in poor performance. They insisted that their proposed schema, which is a decentralized VM consolidation, provided good scalability and more packing efficiency. This technique is validated using three well-known algorithms: FFD, Sercon, V-MAN, and a novel migration-cost aware ACO-based algorithm.

Huang, et al. [11], proposed their MapReduce VM consolidation Framework. MapReduce is an acceptable and powerful distributed data analysis tool, which enables users to process a huge amount of data within a relatively short time period. Hadoop stands out distinctly as one of the most widely deployed MapReduce frameworks. This paper focuses on MapReduce frameworks similar to Hadoop.

Beloglazov, et al. [12], have conducted competitive analysis of the single VM migration and dynamic VM consolidation problems. This paper has found and proved competitive ratios for optimal online deterministic algorithms for these problems. They have concluded that this necessary to develop randomized or adaptive algorithms to improve upon the performance of optimal deterministic algorithms.

Unlike the above research, we focus on a mechanism to build the VM consolidation process in the CloudStack environment using its API and features such as live migration.

VII. Conclusion and Future Work

VM consolidation is a promising solution for cloud data centers to obtain efficient energy utilization and to reduce operating costs. For the purpose of monitoring a cloud data center and implementing VM consolidation, we believe a web application is most powerful and flexible tool. In this paper, we have presented an extensible framework for CloudStack to provide administrators of cloud data centers with the benefits of VM consolidation. We separate the VM consolidation mechanism from the policy deciding which VMs must be moved to somewhere else to run properly. Also, this framework allows web applications to run several VM packing algorithms on top of it. We have demonstrated a simple FFD-like VM packing application using the proposed framework with CloudStack and the Zabbix monitoring system. Experimental results in a real cluster environment confirm that our framework is preliminary, but provides sufficient functionality.

For future work, we plan to improve the framework to provide more flexibility by using Zabbix and the AIST power monitoring system [13] to acquire more monitoring data, including power consumption. Extensive empirical experiments also will be carried out with different system parameters.

Acknowledgment

This work was done during an internship at AIST, Japan. This work was partly supported by JSPS KAKENHI Grant Number 24700040.

References

[1] CloudStack, [Online], Available: http://www.cloudstack.org

[2] Zabbix, [Online], Available: http://www.zabbix.org

[3] Node.js, [Online], Available: http://www.nodejs.org

[4] G. Dósa, "The Tight Bound of First Fit Decreasing Bin-Packing Algorithm Is FFD(I) ≤ 11/9OPT(I) + 6/9," Springer LNCS 4614, pp.1-11 2007.

[5] Node-CloudStack, [Online], Available: https://github.com/Chatham/node-cloudstack

[6] Zabbix.js, [Online], Available: https://github.com/flexd/zabbix.js

[7] Express.js, [Online], Available: http://expressjs.com/

[8] A. Beloglazov, Openstack-neat, "A framework for dynamic consolidation of virtual machines in OpenStack clouds," [Online], Available: http://openstack-neat.org

[9] S. Takeda, T. Takemura, "A Rank-based VM Consolidation Method for Power Saving in Datacenters," IPSJ Transactions, vol. 3, pp. 88–96, 2010.

[10] Feller E, Morin C, Esnault A. "A Case for Fully Decentralized Dynamic VM Consolidation in Clouds," Proc. of 4th IEEE International Conference on Cloud Computing Technology and Science, 2012.

[11] Z. Huang, D. H.K. Tsang, J. She, "A Virtual Machine Consolidation Framework for MapReduce Enabled Computing Clouds," Proc. of the 24th Intedrnational Teletraffic Congress, 2012

[12] A. Beloglazov and R. Buyya, "Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers," Concurrency and Computation: Practice and Experience (CCPE), Vol.24, No.13, pp. 1397-1420, John Wiley & Sons, Ltd., New York, USA, 2012.

[13] R. Takano, T. Shimizu, H. Nakada, and T. Kudoh, "A Scalable and Distributed Electrical Power Monitoring System Utilizing Cloud Computing," Springer LNEE, Vol. 280, pp. 809-817, 2014