An Anti-Overload Model for OpenStack Based on an Effective Dynamic Migration

Al-moalmi Ammar[1], Juan Luo[1*] and Zhuo Tang[1]
[1]College of Computer Science and Electronic Engineering, Hunan University
[e-mail:al_moalmi,juanluo,ztang@hnu.edu.cn, juanluo@hnu.edu.cn]
*Corresponding author: Juan Luo

Received January 29, 2016; revised May 25, 2016; revised July 2, 2016; accepted August 4, 2016; published September 30, 2016

Abstract

As an emerging technology, cloud computing is a revolution in information technology that attracts significant attention from both public and private sectors. In this paper, we proposed a dynamic approach for live migration to obviate overloaded machines. This approach is applied on OpenStack, which rapidly grows in an open source cloud computing platform. We conducted a cost-aware dynamic live migration for virtual machines (VMs) at an appropriate time to obviate the violation of service level agreement (SLA) before it happens. We conducted a preemptive migration to offload physical machine (PM) before the overload situation depending on the predictive method. We have carried out a distributed model, a predictive method, and a dynamic threshold policy, which are efficient for the scalable environment as cloud computing.
Experimental results have indicated that our model succeeded in avoiding the overload at a suitable time. The simulation results from our solution remarked the very efficient reduction of VM migrations and SLA violation, which could help cloud providers to deliver a good quality of service (QoS).

Keywords: Cloud computing, Virtualization, OpenStack, Live migration

This work is partially supported by Program for The National Natural Science Foundation of China (61370094,61300219).

1. Introduction

Cloud computing attracts significant attention from public and private sectors, nowadays[1]. Cloud computing providers deliver their services according to several fundamental models such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Infrastructure as a Service (IaaS) was the focus of this study.IaaS is a provision model in which physical resources can be provided to organizations to support operations, including storage, hardware, servers and networking components. Thepay-as-you-use model for cloud computing IaaS, which enables a convenient on-demand provisioning of theelastic computing resource, has attracted a diverse interest from individual users as well as organizations.Customers can connect their provision resources by different type of terminal devices. Transmit data depends on the available throughput between customers and cloud which should be increased for a good response. The traffic load between the terminal devices and cloud should be offloaded to increase the response time[2, 3].Some other techniques, as mobile edge computing (MEC), solve the low latency requirements to overcome the disadvantages of cloud computing. However, the overloaded MEC system needs to cope with a high traffic which leads to a system failure[4].The power consumption of cloud computing has increased in recent years, which depends on the resource usage, especially the CPU consumption. The concern of power consumption motivated a lot of researchers to study the power consumed by CPU[5], network[6], etc. Whereas other researchers focus on the consolidation of VM to decrease the number of active PMs to reduce the power consumption, but most of these researchers did not consider the negative effects on performance to guarantee a good service to customers. The cloud computing providers introduce the cloud service in two ways: a) guaranteed service class (reserves the physical resource for VM demand), and b) best effort class (shares the cloud platform and guarantees the effort)[7]. The cloud providers give guaranteed service class a high priority by reserving the physical resource for customers' instance. Even though the performance and delivery of services are guaranteed, more secure, and has high QoS, this kind of service is expensive. On the other hand, the second option guarantees the effort in sharing cloud platform among cloud customers. Therefore, this type of service acquires lower cost than the first one. However, cloud computing in terms of QoS and SLA still experiences significant challenges concerning the performance, availability, energy consumption, and economic costs for the cloud operators. Consequently, service providers need to manage the physical resource tightly to guarantee a provision of high QoS without SLA violation[8]. In IaaS, the overloaded physical machine or aggressive consolidation of VMs on the same PMis the cause of SLA violation. Hence, a fit methodology is required by the cloud providers to manage the resource in a dynamic way as to guarantee QoS for the customers. Thus, we believe that the most common process which causes an overloaded machine is an aggressive consolidation which can be avoided by making live migration at a suitable time, as well as taking into account the cost of migration.

A highly recognized cloud platform among the open source cloud projects for both private and public clouds is OpenStack which was announced in July of 2010[9]. Currently, OpenStack project contains several sub-projects, and the core project of OpenStack is Nova, which provides infrastructure as a service upon demand. The OpenStack Nova project can launch an instance on the efficient compute node which meets the customer’s requirements by a part of nova project called Scheduler[10]. Even though OpenStack supports live migration method, the default actor and the action are for the administrator to intervene manually within the appropriate time to avoid the overload for any PM in the cloud system. The administrator can move a VM instance from one computer to another. This feature is useful wheneverthecompute noderequires maintenance or load redistribution where many VM instances are running on a specific PM. However, the importance of taking the decision at a suitable time which guarantees the SLA and QoS is the necessity for conducting a dynamic live migration process. The lack of a dynamic way motivates us to search for a suitable method to monitor, measure, and predict the load; hence, we can make right decisions at a suitable time to migrate VMs to efficient PMs. This method should be compatible with the requirement of the resource usage to scale up or down depending on the demand.

To achieve this goal, we have tried to improve the working mechanism of OpenStack by providing extension into the OpenStack implementation. This extension provides a dynamic live migration to adjust the VMs on PMs to obviate an overloaded PM. Therefore, we took various factors into consideration, such as the cost of live migration, time of migration, and destination of migration.

The solution proposed here was to make a dynamic live migration by distributed algorithms on the nodes to avoid an overloaded PM at a suitable time. First, it is critically essential to minimize the downtime during the entire live migration process. Second, we must detect the proper VMs and destination PM for the migration process because it is crucial to choose light VMs that comply with migration policies, and the PM which has efficient resource space with a light load. In order to address these problems properly, we devised the solution based on the following key considerations:

1. We adopted the architecture of distribution management system, which was essential for large-scale cloud providers. The importance of scalability of distribution management system is to enable natural scaling of the system to thousands of compute nodes such a cloud computing environment.
2. We tried to detect the overload before it happened by using the proper indicator to predict the overload and compare it with an upper dynamic threshold. For a dynamic workload environment, it is a good mean to compare the dynamic threshold with it to detect the overload situation. This way, overload can be avoided before SLA violation occurred, and achieve a migration before the machine developed an overloaded case.
3. We took into account the performance degradation of migration. The effective live migration downtime can be minimized by preparing shared storage, and searching for the lightest VM from the set of VMs that can satisfy the migration policy. If there is none of VM comply with the policy, a number of VMs are to be migrated on a continuous fashion to reduce the load. Consequently, we added an efficient algorithm to pick the best candidate of VMs to be migrated. This approach is the best to reduce migration downtime and the extra load.
4. To consider an appropriate PM where the VM is to be migrated, we have devised algorithms to migrate suitable VMs at an appropriate time. Then, another algorithm submitted the heaviest VM from the migratory list to the carefully chosen PM among cloud computing PMs.

In Section 2, we reviewed previous works in detecting and avoiding an overload in the cloud environment. Section 3 presents the distributed model, predictive method, and proposed algorithms. We described the adjustments to the experimental environment and the results in section 4. Section 5 presents the discussion and conclusion including the limitation of this research and discussed our future work.

2. Related Work

There have been a number of related works on overload detection and management on cloud data centers. The data from previous approaches for detecting an overload can be divided into two categories: periodical adjustment of VM placement (no overload detection), and threshold-based heuristics.

2.1 Periodical Adjustment

This kind of approaches is based on the idea of redistributing the VMs on PMs at periodic times. One of the early works, in which dynamic VM redistribution was applied to obviate the load from an overloaded physical machine was performed as part of a job by Verma et al[11]. Although the designed model for optimizing the placement of VMs by the dynamic way as a bin-packing problem has taken into account the cost of VMs migration, the authors did not apply any algorithm for determining when it was necessary to conduct the VM placement optimization. The proposed model was periodically invoked to adjust the placement of VMs which requires an additional performance without any condition for optimization operation.

FetahiWuhib[12]implemented a dynamic resource allocation for OpenStack, which was based on gossip protocol to react between peer servers. The operation was conducted periodically, and the peer servers were randomly chosen without any condition to exchange their state (including the list of running VMs and their predictive demands). In addition, performance degradation of migration operation was not considered.

Zhenget. al[13]tested the efficiency of a reallocation decision by automating the response time which was specified in the SLAs. With the same approach,Kumar et al[14]proposed a reallocation of theresource under the condition of violating the SLA of hosted applications. This approach had required more time for VM reallocation and the time of SLA violation was increased.

2.2 Threshold-based Heuristics

This kind of approaches isbased on the idea of setting the utilization threshold for PMswhile keeping the total utilization of CPU below the threshold.

Zhu et al[15]applied a static CPU utilization with 85% threshold like a condition for determining an overload PM. In a recent work, Gmach et al[16] used both periodic and reactive threshold to control migration process. In addition, VMware[17]distributed power management based on the same idea with 81% static threshold. However, static threshold heuristic is unsuitable for an unknown workload system because it could not be adaptedto a transient change of workload. As different types of applications are able to share a physical resource, hence, the system should automatically adjust its behavior that is exhibited by the application, depending on the workload patterns.

Beloglazov[18] used a static and dynamic heuristic threshold as benchmark algorithm to determine an aggressive consolidation and energy consumption in his experiment. Althoughthe author had used a dynamic threshold that is suitable for unknown workload, he did not use a prediction method to perform a migration process at a suitable time to decrease the effect of migration on the PMperformance.

Maury and Sinh[19]used the adaptive dynamic threshold depending on the utilization of multi-resources such as CPU,RAM, and network bandwidth to determine when to invoke the migration to redistribute VMs among the PM. However, the only dynamicthreshold was not enough to avoid the overload before it happened within an unknown workload. Wang et al[20]managed resource allocation under the response time of QoS conditions at the cluster and the server level by applied control loops. All these works were based on instantaneous values of system performance and did not use the observed history of system state to predict the system behavior in the future to avoid overload case.

Xu[21]used a prediction method and set an upper and lower threshold for each kind of resources, in order to determine the need for a migration. On the contrary, this threshold made a migration process for any transient fluctuation and did not react for a dynamic workload. Guenter et al[22]applied a weighted linear regression to predict the future workload and optimized the resource allocation proactively to implement an energy-aware dynamic VM consolidation system. Beloglazov[18], used the same algorithm to predict the future workload and adopted this algorithm to make a dynamic threshold, among other methods he had used for the dynamic threshold.

Han[23]proposed a remaining utilization-aware (RUA) algorithm for VM placement. In this work, the algorithm detected the overloaded PM and then conducted a migration for some VMs. However, the authorshave useda constant threshold to determine the overload situation and did not use any predictive way to make preemptive migration toward reducing the time where the PM is located under the impact of the overloaded situation.Nathuji and Schwan[24]proposed a management system for virtualized data centers where resource management was divided into local and global policies. On the local level, the system was responsible for applying the power management strategies for a guest operating system. Consolidation of VMs was handled by global policies applying live migration to reallocate VMs. However, the global policies were not discussed in detail in term of considering QoS requirements. In addition, our work had focused on local policies to predict and consider when we can achieve a live migration to minimize the SLA violation.

In contrast to the aforementioned studies and based on the best of our knowledge, there has not been a study on a comprehensive solution that combines a prediction method and a dynamic threshold to optimize the VMs allocation on PMs. The algorithms regulated the actions according to the experimental performance characteristics of VMs. Hence, we proposed a strategy based on an adaptive threshold while predicting the load. The prediction method estimates the overload before it happened based on a single exponential smooth (SES)[25]. Furthermore, we conducted aware live migration prior to an overload PM to prevent performance degradation and SLA violation. The list of migratory VMs was carefully chosen to minimize the downtime of migration. Furthermore, the idea of estimating the least-affected PMs for reallocating the migratory VMs before the live migration process took place, helped to make an aware live migration.

The comprehensive method has been applied in a real environment for the most popular cloud computing software known as OpenStack. Apparently, this approach has advocated effectively in obviating the overload and preventing SLA violation.

3. System Model

3. 1 Deployment Distributed Model

The target is an IaaS environment, and the system model consists of some parts which are distributed on the equipment such as controllers and compute nodes. End-users submit the requests of provisioning VMs that are categorized according to the requirements of a CPU performance like MIPS (Million Instruction per second), RAM, memory, and network bandwidth. The proposed model for the cloud OpenStack software that comprises three main components is:

3.1.1 Global Manager (Controller)

The deployed components on the controller are responsible for making the global management decision, such as remapping VM instances to PMs. The controller receives a list of migration requests from local manager then initiates the VMs migration process to enhance the VM placement.

3.1.2 Local Manager (Compute Node)

The local manager, which is deployed on compute nodes, is a part of VM monitor which is responsible for monitoring the PM load continuously and deciding which VM should be migrated from the PM and when to migrate it.

3.1.3 VMM (Virtual Machine Monitor)

The deployed components on every compute node are responsible for data collection on used resources by the VM instance, then storing those data locally, which are employed by the local manager algorithms.

3.1.4 Shared Storage

The compute node servers do not have a direct-attached storage while the storage is provided by the technique of shared storage such as a Network Attached Storage (NAS), Storage Area Network (SAN), or other available technology to enable live VM migration.

Fig. 1 depicts the distributed system model that is deployed in OpenStack to avoid an overload PM by live migration at a suitable time.

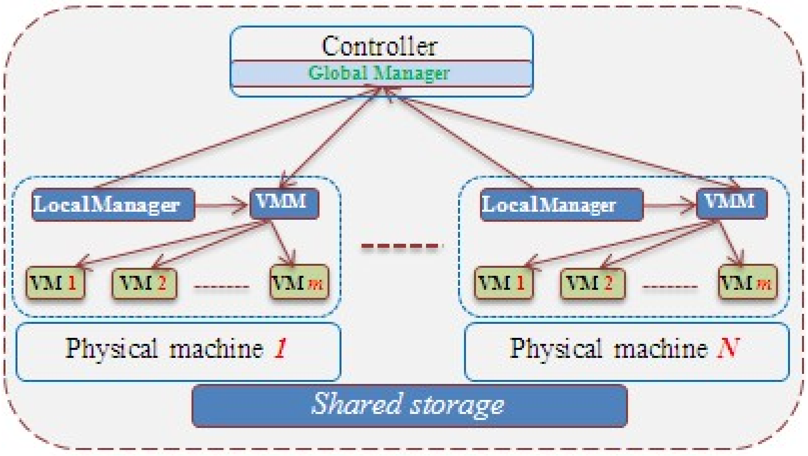

Fig. 1. Distributed System Model

We split the problem into three sub-problems as PM overload detection, VM selection, and VM placement. The distribution model and problem segmentation are suitable for expanded environment such as cloud computing. The local manager that is running on an overloaded PM invokes the VM selection algorithm to pick the suitable VM which generates the offload, then sends the migration list to the global manager to carry out the migration operation before the PMbecame overloaded. This method is helpful for on-time overload prevention to retain sufficient time to carry out the migration process. This operation will try to keep the total load under the threshold, and then the global manager initiates the migration process by invoking the placement algorithm to find a suitable PM to migrate the VM. Moreover, we have tried to ensure that the migration performs only for the necessity of alleviating the overload. After that, the local manager sends the migration request to Nova API to achieve a VM movement depending on the migration map. In this model, we have illustrated the following topics in theory.

- **Cost of VM Live Migration**

  Live migration operation transfers VMs between the hypervisors on PMs without suspension or long downtime. This process depends on the total amount of memory provided to VM during running time and the available network bandwidth between the source and destination in the shared storage environment. However, conducting a live migration process increases the performance of source node and impacts on the performance of VM and applications running on it during a migration. Therefore, the performance degradation and downtime depend on the behavior of VM applications such as how many memory pages the application to be updated during its execution phase. This was investigated by Voorsluys et al [26], who studied the value of this impact and found a way to model it. Consequently, we calculate the migration time and the performance degradation during the migration operation for VM_j as shown:

$$MT_{vm_j} = \frac{Memory_{vm_j}}{Bandwith_{vm_j}}, PD_{vm_j} = 0.1 \times \int_{t_0}^{t_0+MT_{vm_j}} u_{vm_j}(t)dt, \tag{1}$$

  where MT_{vm_j} is the migration time of vm_j , $Memmory_{vm_j}$ is the amount of memory used by vm_j , and $Bandwith_{vm_j}$ is the available network bandwidth for migration. PD_{vm_j} is the performance degradation by VM_j, t_0 is the starting time for migration, u_{vm_j} is the CPU utilization by VM_j.

- **Service Level Agreement Violation**

  SLA between a cloud provider and their customers is vital for cloud computing environments. Therefore, cloud providers have to tackle the QoS requirements with minimized energy consumption. The SLA violation occurs due to the overload situation or aggressive VM consolidation. SLA is defined as a provision of 100% of the performance for application inside the VM at any time. Beloglazov[18]introduced two metrics for measuring the level of SLA violations in an IaaS environment. The first measurement calculates the fraction of time for SLA violation, known as Overload Time Fraction (OTF) during the active PMs time, which was experienced by a 100% CPU workload utilization. The second metric calculates the overall performance degradation due to VMs migration, the Performance Degradation Migration (PDM):

$$OTF = \frac{1}{N}\sum_{i=1}^{N}\frac{T_{CPUfull_i}}{T_{hostActive_i}}, \tag{2}$$

$$PDM = \frac{1}{M}\sum_{j=1}^{M}\frac{EPD_j}{VCPU_j}, \tag{3}$$

  Where N is the number of PMs, M is the number of migratory VMs, $T_{cpufull}$ is the total time of the PM $_i$ has experienced 100% utilization. $T_{hostActive}$ is the total time of the PM $_i$. EPD is the estimate performance degradation of the VM_j . $VCPU$ is the total CPU utilization requested by the VM_j . Both OTF and PDM metrics are independent.

In our experiment, we defined the fraction of time for SLA violation as the duration of active PMs time which experiences CPU utilization workload greater than the adaptive threshold.

$$SLAV_{total} = OTF \times PDM, \tag{4}$$

- **Monitoring Historical Data**

  The system must continuously monitor the resource utilization and dynamically redistribute the VMs between the active hypervisors to satisfy the VMs requirements such as CPU, memory, and network bandwidth. Monitoring the CPU utilization and estimating the overload before it happens is helpful to achieve the migration process. Theprediction gives us a significant gain in time to minimize the impact of the migration process. In other words, the PM load and the migration load preferred to not exceed the overload threshold. There are various ways to calculate CPU load and VMs load. The modern server CPU consists of a multi-core CPU with *n*cores and each has *f* frequency as a single-core CPU with the total capacity of *nf*. The VMs and applications are not to be assigned to a specific core but is assigned to an arbitrary core by a time-shared scheduling algorithm. Therefore, the only limitation is that VM must be less or equal to the capacity of a single core. The CPU utilization for the PM is calculated as the summation of the fraction provided to VMs running on that CPU. As shown,

$$U = nf \sum_{i=1}^{n} f_i u_i, \tag{5}$$

  wherethe*n*is the number of the core on multi-core CPU, each having a frequency *f* ismodeled as a single core CPU with *nf* frequency, $f_i$is the fraction frequency submit for VM, and $u_i$is the CPU utilization specific for VM. The server utilization is obtained through an operating system interface (/proc) and VM utilization can be obtained through the*libvirt* library. The output of this monitoring method is stored and consumed by the algorithms.

3. 2 Proposed Structure and Algorithms

We need an algorithm to predict the load of the PM before an overload occurred. Furthermore, the migration process should be invoked to mitigate the VMs based on the threshold policy. This threshold policy is vital to prevent the violation of SLA that helps to ensure a good QoS to the customers who are sharing physical resources on PMs. Consequently, when an overload is detected, it is obviated by VM migrations which can make the load below the static or dynamic threshold. In order to address this problem properly, we have devised a solution based on the following key considerations:

- Measuring historical data and forecasting the future load: we have tried to detect an overloaded machine before it happened by using a good indicator to predict the overload and compared it with the dynamic threshold. If in case the migration increased the load for PM source, then this load is not likely to pass the threshold. The good effect is the avoidance of overload prior to SLA violation.
- Effective live migration: we have tried to minimize the downtime of VM by preparing shared storage and choosing the best fit lightweight VM to be migrated from the VMs

which can relax the load under the threshold. In the case of none of VM can calm down the load under the threshold, a number of VMs would be migrated on a continuous fashion to reduce the load. Therefore, the algorithm has to pick the one with light consuming of CPU, which can relax the load, as the best candidate to be migrated.

- Remapping VMs to PMs: Once the appropriate list of the migratory VMs has been chosen, it is necessary to consider the right target onto which the migration is going to take place. Here, the main concern is to make the best list of VMs to be migrated, and then choose the efficient destination machine with the least possible operational cost. Therefore, we estimated the load of targeted machine after adding the load of migratory VM to all PMs, then choose the PM of minimal impact before the actual migration is conducted.The reason is to implement an avoiding overload PM for the OpenStack cloud software and achieve dynamic migration, instead of the manual migration which is currently achieved by the administrator, the steps and algorithm were arranged this way. The **Fig. 2** explains the component interaction to achieve dynamic live migration on OpenStack.
-

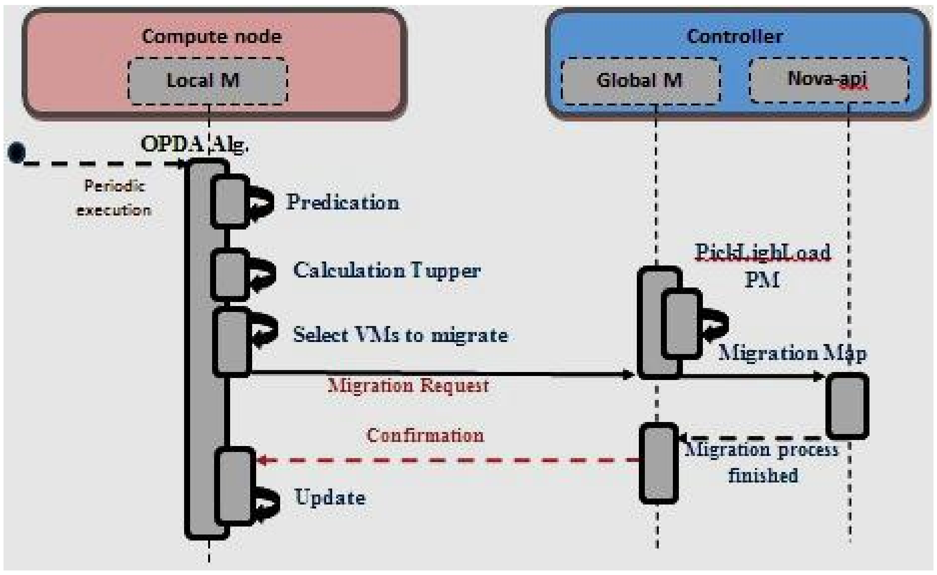

Fig. 2. Interaction of achieving dynamic Live migration

3.2.1 Overload Prediction and Detection Algorithm (OPDA)

This algorithm is distributed on all compute nodes to check if there is an overload situation. The algorithm uses the dynamic adaptive threshold policy. The range of threshold is restricted to be below upper utilization value for PMs. The total utilization of CPU should not exceed this threshold. Under normal circumstances, it is possible for CPU utilization to exceed the limit we have set earlier. When such situation occurred, the system would maintain the limit for the threshold to reduce the overload by migrating some VMs from the PM. In addition, to avoid an overload condition, migration of VMs also relieves the PM from having performance degradation which causes SLA violation, in order to increase QoS for end users. The good side of this algorithm is that the overloaded situation is detected and obviated before it has actually occurred. There are three key tasks to be accomplished to decide which VM to be migrated and when to migrate them. Accurate prediction of performance is acquired to see if an overload situation would be expected and work towards eliminating the overload before it happened. We have employed dynamic upper threshold policy to assess the performance of PMs. After getting the predicted utilization, we compared it with the threshold policy to see if it or the real load exceeds the threshold policy. The complete procedure of OPDA is presented inAlgorithm 1.In the following section, we have presented a discussion on the keys of computing activities for predicting, adaptive dynamic threshold and then making a comparison.

Algorithm 1: OPDA

```
1    Input:α , UPL , LPL
2    Output: migration list
3    Foreach host in hostlistdo {
4        Read resource usage data.
5        Predict the next CPU utilization load for thehost.
6        Calculate the threshold policy.
7        if predictive  value ≥ threshold policy OR real value ≥ threshold policy then {
8        Select VMs for migration.
9        Send the migrated list request to the controller.
10       If controller confirms the migration then {
11        Remove the migrated VMs from VMs list
12        calculate the new utilization load of host}}
13   Continue monitoring.
```

- **Predicting:**We have used a time series data model called SES to make a prediction. SES is a good forecast technique for generating a periodical list of observation values which are collected chronologically for monitoring the resource utilization. However, the previous observations in SES were measured equally, and exponential averaging (smoothing) assigned weights that decreased exponentially over time. In other words, SES is a kind of a weighted moving average sequence data process method to forecast whether the PM would be overloaded or not. The sequence of observation started from time zero and end at time t, where t is the time for the final observation value. The expression for SES is given by formula (6):

$$S_{t+1} = S_t + \alpha(x_t - S_t), \tag{6}$$

where S_{t+1} is the prediction value at time t+1, S_t is the prediction value at time t, x_t is the real value at time t, α is the smoothing factor, and $0 < \alpha < 1$. Formula (6) can be further reduced and expressed as in formula (7) below:

$$S_{t+1} = \alpha \times x_t + (1-\alpha)s_t. \tag{7}$$

If we notice carefully, the above equation is a kind of recursion equation, which can be expanded to formula (8) as follows:

$$S_{t+1} = \sum_{i=0}^{t-1} \alpha(1-\alpha)^i x_{t-i} + (1-\alpha)^t S_1 \tag{8}$$

As we can see from the formula (8),the exponential smoothing predictive value is a weighted sum of all the previous real observational values. This means SES uses all the historical data, so it has more stability and uniformity. The value of α indicates the degree of smoothing and how responsive the model is to frequent changes in the time series data. There are two factors that determine the arbitrary value of α, which are the characteristics of the data and condition that suit the prediction as to what gives rise to good response rate.If the observation values have stable rates, the smooth value is a constant value close to zero, but if observation values have fluctuated rates, the smooth value is constantly close to one. In order to increase the weight of chronological data when the time series remain uniform, we have chosen a small value of α. Larger α helps to increase the weight of recently forecasted values when the time series data keeps on showing clear fluctuation. The selection of this initial smooth value in this method, defined as S_1, is to be initialized to x_1 when the number of data series(k)is more than 15, which is the experimental value. While, if k is less than 15, we define S1 as the average value of the data series. As shown in formula (9),

$$S_1 = \begin{cases} \sum_{i=0}^{k} \frac{x_i}{k}, k < 15 \\ \\ x_1, k \geq 15. \end{cases} \tag{9}$$

- **An Adaptive Utilization Threshold**

  Setting a constant value for the resource utilization threshold might be unsuitable for a computing environment of dynamic workload. Such environment involves different types of application which share physical resources like cloud computing. To this effect, the core idea is to auto-adjust the dynamic value of the upper threshold based on closer examination of previously collected data throughout the lifetime of VMs[27]. These data are mainly based on CPU usage; CPU utilization here is the amount of time that is used to process the instruction of a program. Based on the utilization threshold that we have already set, we can determine if the CPU is overloaded or not. The adaptive threshold called Tupper is computed for each computes machine in the distributed model as follows,

$$Sum = \sum Uvm = \frac{totalrequestedMIPS}{totalMIPSVM} \tag{10}$$

$$sqr = \sqrt{\sum Uvm^2} \tag{11}$$

$$Tupper = 1 - [(UPL - LPL) * sqr]. \tag{12}$$

Where Uvm is the VM utilization, Sum is the summation for all VM utilization,sqr is the square root of the sum of squares of all the VMs utilization.UPLrefers to upper probability limit, and LPL is the lower probability limit for each PM, where we preserve the amount of CPU capacity by UPL and LPL probability limits. This method will help us to adjust the dynamic threshold to keep the PM without an overloaded situation.The PM was judged, whether overloaded or not depending on the comparison between the upper threshold and the predictive value. Therefore, we have to migrate some of its VMs to avoid the overload. This is explained by the mathematical expression below:

$$PM = \begin{cases} S_{t+1} \geq Tupper \ or \ hutil \geq Tupper \rightarrow overload \\ \\ otherwize \rightarrow normal. \end{cases} \tag{13}$$

Where $hutil$ is the real host utilization. S_{t+1} is the prediction value at time t+1.The comparison with real host utilization was conducted to take the probable miss of the prediction approach.

3.2.2 Selection of VM for Migration

We are now able to identify the overloaded PM, and later we need to make a careful selection for the list of migratory VMs. This selection has a direct impact on the source of the overloaded machine and the target machine to which the chosen VM is going to be migrated. Therefore, choosing the appropriate VM is a key task to be handled before the actual migration takes place, and hence to reduce the downtime of migratory VM. The migration time is described as the least possible time required to complete the migration process for aparticular VM. This time is predicted as the quotient of the size of RAM that had been utilized by VM, and the free bandwidth that is available for the migration process. Therefore, we will look for the lightest VM from the set of VMs that can satisfy the condition of relaxing the load under the threshold policy (if the VM utilization is greater than the difference of current PM utilization, or the predictive utilization, and upper threshold value). If none of VM complies with this condition, we have to migrate more than one VM to take the load under the overload threshold. This way minimizes the migration time and performance degradation which has a good impact on source PM and a good reduction of delay on the user side. Furthermore, this way facilitates the reallocation the migratory VMs to active PMs. On the contrary, choosing the heavy VMs would increase the performance degradation.It is worthy to mention that we would not conduct a block live migration, which costs a high performance, wastes the system resources, and makes a long suspension at the user’s side. We have adopted the shared-storage-based live migration. In this case, all hypervisors have to access shared storage, which obviates the need to migrate disk storage. OpenStack can support a shared technique like an OpenStack Gluster connector and a distributed replication storage using GlusterFS. The OpenStack system copies the base image from the image store to a local disk (in directory /var/lib/nova/instances) which is used as the first disk of the instance (vda). This directory would be stored in the shared storage to support a live migration. Therefore, we do not need to migrate the block storage.

The algorithm starts to look for VMs that have the least amount of CPU consumption, among those who satisfy the relaxation condition, to reduce the performance degradation. The algorithm tries to choose the quick-migrating VMs among those that would give good relaxation to an overloaded PM as well as minimizing the number of the migratory VMs. In the case of none of this option, a number of VMs have to be migrated as much as possible until the situation is mitigated and the CPU utilization is lower than the threshold. In this case, the algorithm would select the VM, removes it from the list of VMs, and proceeds to a new iteration. The new iteration would take again one of the previous two cases (there are a set of VMs satisfy the condition, or there are none of theVM satisfy the condition). The pseudo-code for the algorithm is presented in Algorithm 2. The complexity of the algorithm depends on the number of VMs allocated to this PM. By the same way of comparing a predictive load withan upper threshold, we have compared between the current utilization and the threshold policy in case the real utilization exceeded the threshold policy as an unexpected case. We do not provide the pseudo-code for this case, as it is similar to the algorithm 2 presented earlier.

Algorithm 2: PickLightestfitVMs

```
1   Input: VMList, predictivevalue, realutilization,Tupper
2   Output: MigrationList
3   VMList. sortDescendingUtilization( )
4   $predict \leftarrow predictivevalue$
5   $bestfitutil \leftarrow \max$
6   While $predict \geq Tupper$ do {
7   foreachvm in VMList do {
8           if vm.getUtil() > predict -  Tupper  then {/* the VMutil mustbe higher than the
                                                 difference between the predict and the Tupper.*/
9   $t \leftarrow \mathrm{vm.getUtil()} - \mathrm{predict} + \mathrm{Tupper}$
10  if $t < bestfitutil$ then    /* = Max it the beginning
11  { $\mathrm{bestfitutil} \leftarrow t$
12  { $\mathrm{bestfitvm} \leftarrow vm$} }
13  Else {
14  if $bestfitutil = \max$ then
15  $bestfitvm \leftarrow vm$
16  break }}
17  $predict \leftarrow predict - bestfitvm.getUtil()$
18  migrationList.add( $bestfitvm$ )
19  VMList.remove( $bestfitvm$ ) }
20  Return MigrationList
```

3.2.3 Placement Algorithm

The request of migration from local nodes, which has a list of requirements, is received by the controller. In return, to get the efficient computing nodes for hosting VMs, this algorithm employs an OpenStack Nova-scheduler which performs the entire placement process in two main phases: filtering and weighting[10]. The scheduler returns the sufficient computing nodes on which the VMs would run. Hence, the algorithm would estimate the load of adding migratory VMs to all PMs before the migration, and then pick the least impact PM to produce an efficient map between the migratory VMs and PMs.We used Best Fit Decreasing (BFD)[28] algorithm with an additional condition to reach this goal. The algorithm would produce a mapping list of heavy migratory VMs to the lightest PMs Load. The appropriate list was submitted to the OpenStack Nova-API to perform the migration process between source machine and destination machine. Then, global manager monitors the VMs migration to determine when the migration is completed during the movement of the migratory VMs memory page between source and destination with nearly zero downtime. The following algorithm explains the way of picking the light load PMs for the heavy load VMs.The pseudo-code for the algorithm is presented in Algorithm 3. The complexity of the algorithm is proportional to the product of the number of migrated VMs and the number of the sufficient PMs.

Algorithm 3: PickLightLoadPM

```
1      Input: sufficientListHost, VMLlist
2    Output: MapList
3      VMList.sortDecreasingUtilization()
4    foreach vm in VMList do   {
5    sufficientListHost.sortincreasingUtilization()
6    minload      ← Max
7       allocatedHost   ← Null
8    foreachhost in sufficienthostList do {
9    load     ← current load of host + load of vm
10   if load < minload then {
11               allocatedHost ← host
12   minload        ← load }}
13   if allocatedHost ≠ Null then {
14          MapList .add (vm,allocatedHost)
15          allocatedHostload= minload.
16          VMList.remove (vm) }  }
17   Return MapList.
```

4. Experiment and Results

4.1 Experimental Setup

Table 1. Experiment hosts specification

Host	Details
CPU	processor 4122, 2 CPU (4 cores) 2.2GHz
RAM	8GB DDR3
Hard Disk	500 GB * 2
NIC	1 G

The testbed,that is used for performance evaluation of the system, consisted of the hardware specifications as shown in**Table 1**.First,we installed LinuxCentOS 6operating system on all PMs. The Centos installation followed the standard process described in detail in the Red Hat Enterprise Linux 6 installation guide[29]. In this experiment, we adopted the shared storage to implement live migration. Therefore, it was necessary to use one of the shared storage techniques such as GlusterFS[30].Second, we deployed the OpenStack component on a distributed model. In this step, we used the Essex version of OpenStack and followed the documentation deploying OpenStack on CentOS using the KVM hypervisor and GlusterFS distributed file system[31].**Fig. 3** shows the current integration of OpenStack and GlusterFS.Finally, we deployed the OpenStack Neat project [32]. The purpose of the OpenStack Neat framework is to provide an extensible software framework to conduct research. The following part explains the deployment steps on one controller node and three compute nodes:

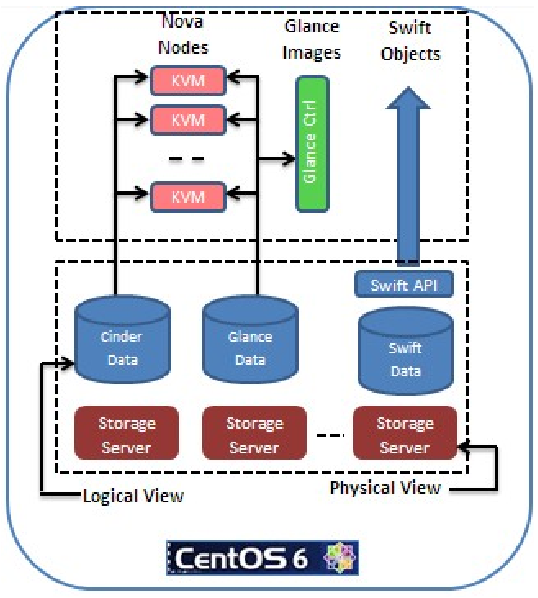

Fig. 3. OpenStack and GlusterFS - Current Integration

- **Controller Node:**Controller node represents the global state and interacts with all other components. An API Server acts as the web services front end for the compute nodes. In this node, we deployed the basic component as Nova-API, Nova-scheduler, Nova-network, Nova-volume, Nova-objectstore, Keystone service, Glance service, MySQL server, AMQP message broker, VNC server, and others. We have to mention that the controller node runs all service except for Nova-compute. Therefore, controller node does not host any VMs. Then, we added the additional component, that is responsible for making VM placement decision and initiating a VM migration. The global manager would call the VM placement algorithm for the requested list of migratory VMs. Hence, the algorithm produces an appropriate map between the VM and the PM. After that, the result's map would be sent to the OpenStack Nova-API to perform the migration. The global manager exposes a REST web service (REST-API) for processing VM migration requests that have been sent by local managers. The service Uniform Resource Locator (URL) defines according to the configuration options specified in /etc/neat/neat.conf.
- **Compute nodes:**Compute nodes are responsible for running the customers' instances and providing the physical requirements. We have installed the Nova-compute, Nova-network, and Nova-API packages in these nodes. We had chosen a GlusterFS technique rather than other shared storage technique due to its advantages such as no single point of failure. If a data replica is available on the PM, VM instances access the data locally rather than remotely over a network improving the I/O performance. During the installation of the operating system, we have specified a distinctive volume group which contains a single lv_gluster partition. In this partition, we stored the directories (/var/lib/nova/instances).These directories must be mounted and shared between the controller and all the compute nodes to enable live migration of VMs. The additional components in the proposed distribution system of compute nodes are responsible for making the local decision by executing periodically a function to determine whether it is necessary to reallocate VMs of the PMs to avoid overloaded machines. We deployed a monitoring service, an overload

prediction, a detection and selection VM algorithm which have specified the configuration file for the OpenStack. During the experiment, the configuration parameters are modified to be favorable for the algorithms. **Table 2** exposes the way of setting of important parameters for algorithms.

•

Table 2. Configuration parameters of algorithms for OpenStack

Configuration option	**Description**
algorithm_overload_detection_parameters= {"omega": 0.8, "UPL": 0.90, "LPL": 0.85 }	JSON encoded parameters to be parsed and passed to the specified overload Prediction and detection algorithm factory.
algorithm_vm_selection_factory=neat. locals. PickLightestfitVMs_factory.	The fully qualified name of a Python factory function that returns a function implementing a VM selection algorithm
algorithm_vm_placement_factory=neat.globals.PickLightLoadPM_factory	The fully qualified name of a Python factory function that returns a function implementing a VM placement algorithm.

- **Load Generation:**Once the installation has been completed, it is crucial to generating the workload in a reliable way to make a renewable experiment which would guarantee repeatable experiments as we need. To reproduce a realistic data, we need to use the workload traces collected from a real system which is better than the artificially generated data. The monitoring data of PlanetLab infrastructure[33]was provided as a part of the CoMon project. This data included a workload trace on the CPU utilization which had been collected every 5 minutes from more than a thousand of VMs which had been deployed on servers located in more than 500 places around the world. The set of selected trace and filtering script are available online[34]. After all, VMs had been launched on the compute nodes, and we have assigned a unique workload trace to each VMs. Look busy software[35] is a simple application for generating a load on a Linux system that was developed by Devin Carraway under the GPL license.The tool accepts the set of selective workload trace values which derive from PlantLab infrastructure as an optional value of CPU utilization. These selective values contain the level of CPU utilization numbered in the range [0,100] representing percentages. Lookbusy goes through the sequence of CPU utilization levels and generates each CPU utilization level for the specified time interval.

•

4.2 Results and analysis

We have conducted the experiment to evaluate the proposed algorithms by allocating as many VMs on compute nodes as possible to lead the compute nodes to an overloaded situation. We used a VM instance type with the minimum amount of RAM sufficient for Ubuntu 12.04. The minimum required amount of RAM was determined to be 128 MB.We have allocated 40 VMs on compute node1, 20 VMs on compute node2, and 10 VMs on compute node3. Forty CPU utilization traces had been randomly selected to generate the load on compute node1. During the experiment, the overload prediction and detection algorithm have been run three times to handle the variable random factors for the prediction algorithm such as the degree of smoothing, and the upper and lower limitation for an adaptive threshold. We have calculated the average of the predictive value when α(0.8,0.5,0.3). The real CPU utilization values and predicting values are shown in **Fig. 4**. The Mean Relative Error (MRE) was, correspondingly, 1.3%, 1.9%, and 3.7%. That remarks that SES algorithm has an acceptable predicting accuracy. The use of α =0.8 has produced a good experimental outcome which was associated with the best predictive value that determines the future load. The predictive value gave a convenient time to achieve a preemptive migration before the occurrence of the real overload. The compute node had used the prediction model to invoke the selection algorithm when the PM was expected as an overload. Then, the compute node sent a request for migration to the controller, and the migratory requests that were processed by the controllertook on averageroughly between 17 and 37 seconds, which was mostly determined from the time taken to the end of migration.

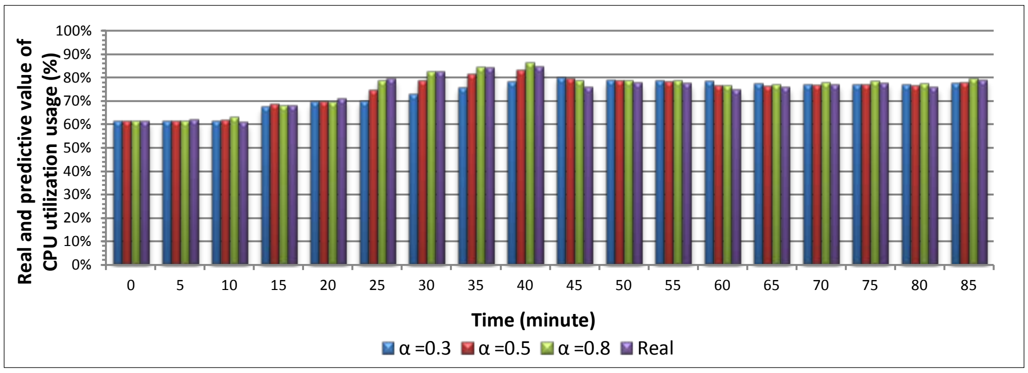

Fig. 4. Real and Predictive Value

According to the data that was stored by the data collector during the experiment, we evaluated the SLA violation by calculating the OTF and PMD metrics. The overload time fraction was calculated when PMshave experienced an overload time according to the overload threshold. Compute node1 was activated to serve the VMs 24 hours. Depending on the calculated maximum time to process the migration request, the obtained values of OTF and PDM were (1.42×10^{-4},0.1) respectively, and the SLA violation was 1.09×10^{-4}.Therefore, we succeeded in the elimination of an overload situation. The algorithms were able to adequately reduce SLA violation while moving VMs which have taken the load under the specified threshold. **Fig. 5**exhibits the real load (before, during, and after migration), and the predictive load. the peak of real load curve shows the real load during the migration process which did not exceed the upper threshold. The result reflected the efficiency of the algorithms that performed the live migration, depending on the prediction technique at a suitable time. Time saved, between the real load and predictive load, has been used to achieve a preemptive live migration before the overloaded case, and keep the load under the threshold even with migration load. Consequently, the obtained results imply a good effect on managing the resources which guarantee good QoS to customers.

We succeeded in the implementation of the proposed algorithms on OpenStack platform to achieve the goals. However, it is extremely difficult to conduct repeatable large-scale experiments on a real infrastructure which is required to cover the evaluation of all aspects of the proposed algorithms by comparing it with other algorithms. Therefore, to ensure the repeatability of experiments, simulations were chosen as a way to evaluate the performance of the proposed algorithms.

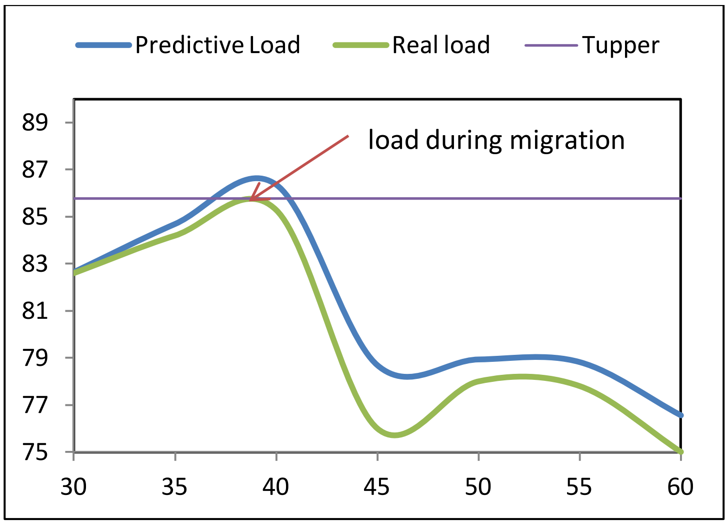

Fig. 5. Load during Live Migration

We chose the CloudSim toolkit[36]as a simulation platform. In this experiment, we had worked with one data center comprising 100 PMs. Each node was formed to have one CPU core with the performance equivalent to 800, 1200, 2000 MIPS, 8 G of RAM. Then, we submitted 300 VMs on the simulated datacenter. Each VM required one CPU core with 250, 500, 750 and 1000 MIPS respectively, with 128 MB of RAM. We have conducted several experiments to evaluate the model that was used in this work and set the simulation limit to 24 hours. Comparisons on the prediction model were made with other classic prediction methods such as Autoregressive (AR), Moving Average (MA), Autoregressive Integrated Moving Average (ARIMA) and Back Propagation (BP) neural network. The prediction time interval was set to 300s and then we repeated the same experiment with 600s to make the comparison.**Fig. 6** shows the experiment results.We can conclude that the prediction time interval has an influence on the predictive value. Consequently, we choose SES because it is not too sensitive for interval time, and it is a relatively simple model giving a persuasive performance. The MRE of SES is acceptable as compared to the other models, like MA that works well with small interval and ARIMA which works well with a long interval. The threshold policy was evaluated subsequently with different values of the probability of upper and lower limitation to determine the best interval values of threshold in QoS presented by SLA violation. The simulation results have been presented in **Fig 7**. The result shows that with the growth of the threshold, the percentage of SLA violation increases. This is due to the fact that high utilization threshold allows more aggressive consolidation of VMs which increased the risk of SLA violation. We have repeated the same experiment with the prediction model and compared it with the previous result. The result shows that the prediction model can potentially reduce the SLA violation for all threshold intervals at an average rate of 32.5%. This is associated with the time convenience offered in the prediction model prior to the time that the system actually gets an overloaded machine. This time was exploited for migration and decreased the violation time. We have also investigated the number of migratory VMs with different threshold policies, and the results are illustrated in **Fig. 8**.

In this figure, we can observe that the algorithms succeed in minimizing the number of migratory VMs by picking appropriate VM for migration. Moreover, when the threshold is increased, the number of migratory VMs is decreased. On the other hand, we repeated the same experiment with a PickLightloadPM algorithm that remapped heavy migratory load

VMs to light loadPMs. The results exhibited that the number of migrated VMs with PickLightloadPM algorithm was less than the number of VMs that migrated without it by an average 15.12% for all intervals.

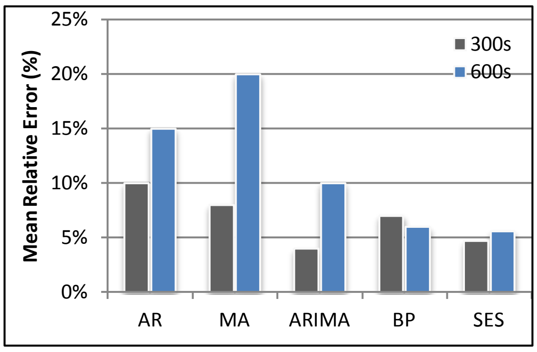

Fig. 6. Relative Error of Various Predicting Models

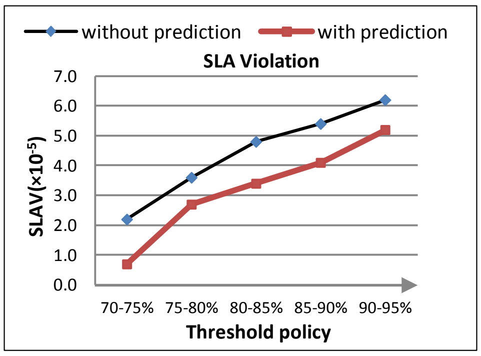

Fig. 7. SLA violation of different probability of threshold interval.

That is due to the algorithm estimation for the load before the migration started, which causes a new overload on other PMs. In addition, the algorithm has produced an efficient mapping between the heavy VMs and light load PMs. When we jointly consider the previous results together, we have chosen the threshold policy 85-90% which has an acceptable value of SLA violation (4.1×10-5). Furthermore, it hadan acceptance of provision resources and reduced the number of VM migration. Therefore, we have regarded the 85-90% threshold interval as a possible appropriate for later experiments. We moved further on evaluating the overload prediction and detection algorithms and compared it with other methods. We have chosen the probability interval 85-90% to conduct this comparison with experimental benchmarks.We have used the NonPower-Aware (NPA) policy and DVFS for the benchmark experimental results. These policies do not apply any optimizations or adaptation of the VM allocation at run-time and imply that all hosts run at 100% CPU utilization. Moreover, we also compared our work with ST (static threshold) and RUA. The mainresults are presented in **Fig. 9**, and **Fig. 10**,which show the effective reduction of SLA violation and number of migratory VM.From the presented results the OPDA algorithm brought down SLA violation to the minimum comparing to other algorithms specially RUA, that's because OPDA made preemptive migration which minimized the overloaded time hence reduced the SLA violation. Furthermore, the RUA resulted in a smaller number of migratory VMs due to the prediction method of OPDA not always producing the accurate result for next interval workload which sometimes causes unnecessary preemptive migration. However, thepriority of conducting a preemptive migration has a significant impact in reducing SLA violation and provides a high quality of service.

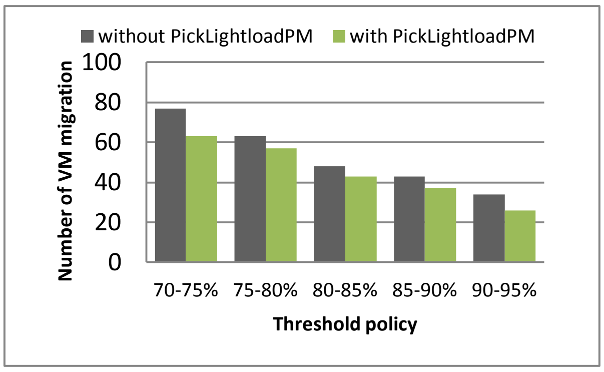

Fig. 8. Number of VM Migration With Our Algorithm

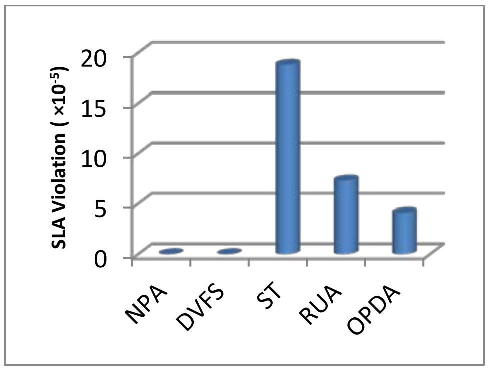

Fig. 9. SLA violation with other policies

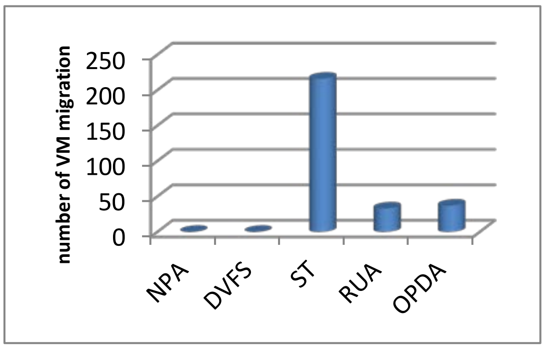

Fig. 10. Number of VM migration with other policies

5. Conclusion

Cloud computing becomes a powerful computing paradigm that enables the computing services like a software, infrastructure, and platform to be provisioned to users as per their desire on a pay-as-you-use. OpenStack is one of the robust open source software infrastructures consisting of several projects in it. However, the implementation of Nova has an aggressive consolidation ratio, because the scheduler submits VMs to the least PM capacity.

This study provides an efficient dynamic live migration to optimize the mechanism of OpenStack. The method of our research has avoided overload effectively and reduce SLA violation.The forecast method that we used is useful to predict future changes and mitigate to conserve the system’s stability. Furthermore, precise selection of VM for migration has an appreciable effect in preserving the resources which were degraded during the migration process. The prerequisite conditions for selecting the target machine obviate an unnecessary migration process, where the migration takes place only after it has ensured that the target machine is an appropriate and lightest node. The results show that our model makes important minimization for the number of VMs as well as minimizes SLA violation on the system. Moreover, the experimental results show a significant reduction in the cost of the migration process.

Taking only one factor that is CPU utilization to calculate adaptive threshold, as well as overload situation was one of the limitations of this study. However, there were other factors like bandwidth and RAM which may affect PM load somehow. An application may fail due to insufficient RAM whereas insufficient CPU may just slow down the execution of the application. Moreover, CPU capacity allocated for a VM was fixed to be less or equal to the capacity of one core and removing this constraint would require the VM to be executed on more than one core in parallel. On the other hand, in the current implementation, we assumed only one controller PM which may have limited the scalability and created a single point of failure. Further research is needed to investigate in details of all these limitations in order to aim for perfect results.

References

[1] E. Arianyan, H. Taheri, and S. Sharifian, "Novel energy and SLA efficient resource management heuristics for consolidation of virtual machines in cloud data centers," *Computers & Electrical Engineering*, 2015. Article (CrossRef Link)

[2] M. Jo, T. Maksymyuk, B. Strykhalyuk, and C.-H. Cho, "Device-to-device-based heterogeneous radio access network architecture for mobile cloud computing," *Wireless Communications, IEEE*, vol. 22, no. 3, pp. 50-58, 2015. Article (CrossRef Link)

[3] Y. W. Ahn, A. M. Cheng, J. Baek, M. Jo, and H.-H. Chen, "An auto-scaling mechanism for virtual resources to support mobile, pervasive, real-time healthcare applications in cloud computing," *IEEE Network*, vol. 27, no. 5, pp. 62-68, 2013. Article (CrossRef Link)

[4] D. Satria, D. Park, and M. Jo, "Recovery for overloaded mobile edge computing," *Future Generation Computer Systems*, 2016.Article (CrossRef Link)

[5] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future generation computer systems*, vol. 28, no. 5, pp. 755-768, 2012. Article (CrossRef Link)

[6] J. Luo, J. Hu, D. Wu, and R. Li, "Opportunistic routing algorithm for relay node selection in wireless sensor networks," *Industrial Informatics, IEEE Transactions on*, vol. 11, no. 1, pp. 112-121, 2015. Article (CrossRef Link)

[7] D. Serrano, S. Bouchenak, Y. Kouki, T. Ledoux, J. Lejeune, J. Sopena, L. Arantes, and P. Sens, "Towards QoS-Oriented SLA Guarantees for Online Cloud Services," pp. 50-57. Article (CrossRef Link)

[8] J. A. Wickboldt, R. P. Esteves, M. B. de Carvalho, and L. Z. Granville, "Resource management in IaaS cloud platforms made flexible through programmability," *Computer Networks*, vol. 68, pp. 54-70, 2014. Article (CrossRef Link)

[9] "OpenStack Open Source Cloud Computing Software," Article (CrossRef Link).

[10] O. Litvinski, and A. Gherbi, "Openstack scheduler evaluation using design of experiment approach," in *Proc. of 16th IEEE International Symposium on Objec (ISORC 2013)*, pp. 1-7, 2013. Article (CrossRef Link)

[11] A. Verma, P. Ahuja, and A. Neogi, "pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems," in *Proc. of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pp. 243–264, 2008. Article (CrossRef Link)

[12] F. Wuhib, R. Stadler, and H. Lindgren, "Dynamic resource allocation with management objectives—Implementation for an OpenStack cloud," in *Proc. of Network and service management (cnsm), international conference and workshop on systems virtualiztion management (svm)*, pp. 309-315, 2012.Article (CrossRef Link)

[13] W. Zheng, R. Bianchini, G. J. Janakiraman, J. R. Santos, and Y. Turner, "JustRunIt: Experiment-Based Management of Virtualized Data Centers," in *Proc. of the 2009 USENIX Annual Technical Conference*, pp. 18–33., 2009.Article (CrossRef Link)

[14] S. Kumar, V. Talwar, V. Kumar, P. Ranganathan, and K. Schwan, "vManage: Loosely Coupled Platform and Virtualization Management in Data Centers," in *Proc. of the 6th International Conference on Autonomic Computing (ICAC)*, pp. 127–136., 2009. Article (CrossRef Link)

[15] 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. of the 5th International Conference on Auto-nomic Computing (ICAC)*, pp. 172–181, 2008. Article (CrossRef Link)

[16] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, and A. Kemper, "An integrated approach to resource pool management: Policies, efficiency and qual-ity metrics," in *Proc. of the 38th IEEE International Conference on Dependable Systems and Networks (DSN)*, pp. 326–335, 2008. Article (CrossRef Link)

[17] VMware, "VMware distributed power management concepts and use," 2010. Article (CrossRef Link)

[18] 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*, vol. 24, no. 13, pp. 1397-1420, 2012. Article (CrossRef Link)

[19] K. Maury, and R. Sinh, "Energy Conscious Dynamic Provisioning of Virtual Machines using Adaptive Migration Thresholds in Cloud Data Center," *International Journal of Computer Science and Mobile Computing*, vol. IJCSMC, Vol. 2, Issue. 3, March 2013, pg.74 – 82, 2013. Article (CrossRef Link)

[20] X. Wang, and Y. Wang, "Coordinating Power Control and Performance Management for Virtualized Server Clusters," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 22, no. 2, pp. 245–259, 2011. Article (CrossRef Link)

[21] L. Xu, W. Chen, Z. Wang, and S. Yang, "Smart-DRS: A strategy of dynamic resource scheduling in cloud data center," in *Proc. of Cluster Computing Workshops (CLUSTER WORKSHOPS), 2012 IEEE International Conference on*, pp. 120-127, 2012. Article (CrossRef Link).

[22] B. Guenter, N. Jain, and C. Williams, "Managing Cost, Performance, and Reliability Tradeoffs for Energy-Aware Server Provisioning," in *Proc. of the 30st Annual IEEE International Conference on Computer Communications (INFOCOM)*, pp. 1332–1340., 2011. Article (CrossRef Link).

[23] G. Han, W. Que, G. Jia, and L. Shu, "An Efficient Virtual Machine Consolidation Scheme for Multimedia Cloud Computing," *Sensors*, vol. 16, no. 2, pp. 246, 2016. Article (CrossRef Link)

[24] R. Nathuji, and K. Schwan, "VirtualPower: coordinated power management in virtualized enterprise systems," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 265-278, 2007.Article (CrossRef Link).

[25] S. Makridakis, S. C. Wheelwright, and R. J. Hyndman,*Forecasting methods and applications*: John Wiley & Sons, 2008.Article (CrossRef Link).

[26] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation," in *Proc. of the 1st International Conference on Cloud Computing (CloudCom 2009)*, Beijing, China, 2009.Article (CrossRef Link).

[27] R. Sinha, N. Purohit, and H. Diwanji, "Power aware live migration for data centers in cloud using dynamic threshold," *International Journal of Computer Technology and Applications*, vol. 2, no. 6, 2011.Article (CrossRef Link).

[28] Y. MINYI, "A simple proof of the inequality FFD (L)< 11/9 OPT (L)+ 1,for all l for the FFDbin-packing algorithm," *ActaMathematicae Applicatae Sinica (English Series)*, 1991. Article (CrossRef Link).

[29] R. Landmann, J. Reed, D. Cantrell, H. D. Goede, and J. Masters, "Red Hat Enterprise Linux 6 Installation Guide," 2012.Article (CrossRef Link).

[30] E. J. Qaisar, "Introduction to Cloud Computing for Developers," in *Proc. of Information Technology Professional Conference (TCF Pro IT), IEEE TCF 2012*, 2012. Article (CrossRef Link).

[31] S. F. P. Anton Beloglazov, Mohammed Alrokayan, and Rajkumar Buyya, "Deploying OpenStack on CentOS Using the KVM Hypervisor and GlusterFS Distributed File System," in *Proc. of Cloud Computing and Distributed Systems (CLOUDS) Laboratory*, August 2012. Article (CrossRef Link).
[32] A. B. R. Buyya, "OpenStack Neat: A Framework for Dynamic Consolidation of Virtual Machines in OpenStackClouds – A Blueprint," 2012. Article (CrossRef Link).
[33] K. Park, and V. S. Pai, "CoMon: A Mostly-Scalable Monitoring System for PlanetLab," in *Proc. of ACM SIGOPS Operating Systems Review*, 2006.Article (CrossRef Link).
[34] A. Beloglazov. "The workload data," Article (CrossRef Link).
[35] "lookbusy -- a synthetic load generator," Article (CrossRef Link).
[36] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, "CloudSim: A Toolkit for the Modeling and Simulationof Cloud Resource Management and Application Provisioning Techniques," *Software: Practice and Experience*, January 2011. Article (CrossRef Link).

Al-moalmi Ammar received the Bachelor's degree in computer science from Sana'a university, Sana'a, Yemen, in 2005, and Master of engineering degree in Computer Science and technology, Hunan University, china, in 2014. He worked in Yemen as a lecturer for a computer science department at Sana'a University. He is a reviewer for some journals. Currently, he is pursuing the Ph.D. degree at the College of Computer Science and Electronic Engineering, Hunan University. His research interests include cloud computing.

Juan Luo received the bachelor's degree in1997 from National University of Defense Technology, Hunan, China. Sheachieved master's and Ph.D. degrees in communication and informationsystem in 2000 and 2005, respectively, both from Wuhan University, Hubei,China. She is currently a professor and doctoral supervisor at the College ofComputer Science and Electronic Engineering, Hunan University, Changsha,Hunan, China. From 2008 to 2009, she was a visitingscholar at the University of California at Irvine. She has published more than 60 papers. Her research is focused on wireless networks, cloud computing,and wireless sensor networks. She is a member of the IEEE andSIGCOM, a member of theACM, and a senior member of CCF.

Zhuotang received the Ph.D. in computer science from Huazhong University of Science and Technology, China, in 2008. He is currently an associate professor of the College of Computer Science and Electronic Engineering at Hunan University, and he is the associate chair of the department of computing science. His majors are distributed computing system, cloud computing and the parallel process for big data, include the security model, parallel algorithms, and resources scheduling and management in these areas. He is a member of ACM and CCF.