Smart-DRS : A Strategy of Dynamic Resource Scheduling in Cloud Data Center

Lei Xu, Wenzhi Chen, Zonghui Wang, Shuangquan Yang
College of Computer Science and Technology
Zhejiang University, Hangzhou, China
Email: {leixu, chenwz, zjuzhwang, sqyang}@zju.edu.cn

Abstract— **The biggest advantage of employing virtualization is the ability to flexibly remap physical resources to virtual servers in order to handle the resource redistribution. So virtual machine is the fundamental unit in cloud data center. However, the load of virtual machine constantly changes owing to the needs of applications. In order to improve the resource utilization and reduce power energy, data center needs an automatic, quick and dynamic resource scheduling strategy which treats virtual machine as a scheduling unit to balance load and consolidate servers.**

In this paper, we present a two-steps dynamic resource scheduling strategy, named Smart-DRS, which fits cloud data center well and strikes a balance between efficiency, cost and instantaneity. Firstly, we employ a prediction technique based on Single Exponential Smoothing algorithm. Then a novel and efficient migration algorithm based on Vector Projection was applied.

For evaluating the performance of Smart-DRS, we develop a complete resource management prototype system in which resource scheduling is just only a module. Then we build a cluster with 32 physical machines running with 3200 virtual machines to simulate datacenter environment. Experiment results tell us that Smart-DRS has a high forecast accuracy and also can deal well with load balancing and load consolidation.

Keywords- Dynamic Resource Scheduling, Load Balancing, Load Consolidation

I. INTRODUCTION

The ability of virtualization technology which is an important enabler for cloud computing brings immense benefits in terms of reliability, efficiency and scalability. Virtualization, coupled with migration capability, enables the cloud datacenters to balance load and consolidate servers. As we know, cloud computing has a high requirement about *QoS*, so we should well manage cluster resource to automatically balance load. Moreover, the energy consumption of data center is also an increasingly sharp problem. We need to release some physical machines (PM) by consolidating virtual machines (VM) together to other PMs while the load of cluster system is low. For achieving these goals, there should be an automatic management approach that could dynamically adjust VMs allocation to the right PMs. This approach is named dynamic resource scheduling (DRS) strategy which is directly related with the efficiency and performance of the cloud data center.

DRS strategy consists of two distinct parts. The first part is how to correctly estimate the VM resource requirements. This is a crucial and difficult step since VMs keep changing their resource requirements dynamically. After the resource requirements of VMs are properly estimated is the second part. It is how to apply a VM migration strategy to achieve efficient resource utilization of PMs. In this paper, we present a novel DRS strategy, named Smart-DRS, which could address the problem of *when* to initiate a migration and *where* to migrate the virtual machines. It employs prediction techniques based on Single Exponential Smoothing (*SES*) algorithm which is a kind of weighted moving average sequence data process method to judge whether PMs will overload. Upon prediction, it then employs a novel methodology based on Vector Projection (*VP*) arithmetic which is a good way to solve VM placement problem.

The major contributions of this paper can be summarized as follows:

- In the first step of Smart-DRS, we apply a more accurate prediction method to avoid a tiny and temporary load peak value triggering unnecessary migration.
- In the second step of Smart-DRS, we apply a novel vector projection method to decide how to place a VM with low cost and quick execution.
- We implement a prototype system to demonstrate that our strategy achieves better performance of load balancing and load consolidation.

The rest of this paper is organized as follows. Section 2 describes related work. In Section 3, we describe the framework of Smart-DRS, and Section 4 details the algorithms using in Smart-DRS. Finally, in Section 5, we implement a prototype system to evaluate the performance of Smart-DRS. A summary and plan of our future work are described in Section 6.

II. RELATED WORKS

Dynamic resource scheduling is a primary problem in virtual environment management. VMs, the minimal scheduling unit, are ceaselessly customized, produced and deployed. Meanwhile, the cluster resource utilization is changing accordingly. In order to achieve a high *QoS*, many methodologies have been researched in previous literatures.

Hermanier [1] discusses respectively about scheduling scheme making and executing. They treat VM deployment as a two-dimensional *Bin Packing* problem and use the two-dimensional *Bin Packing* dynamic programming algorithm to solve. This method traverses different number of backpacks to find the least number of backpacks that can accommodate all items. In addition, it employs pruning strategy to optimize the solution process. For example, it limits the scope of solution space and only explores the solution that decrease in the number of backpacks. However, in fact, VM scheduling problem is more complicated than the two-dimensional *Bin Packing* problem. That means deploying this method on cloud datacenter has several limitations.

Hyser et al. [2] propose that the problem of VM deployment is different with *Bin Packing* problem. *Bin Packing* starts with a clear state while VM deployment starts with an existing mapping state. Besides, this paper uses simulated annealing method to achieve optimal but the author doesn't give a detailed description of the algorithm.

Grit et al. [3] consider some VMs replacement issues for resource management policies in the context of Shirako [4], a system for on-demand leasing of shared networked resources in federated clusters. When a migration is not directly feasible, due to sequence issues, the VM is paused using suspend-to-disk. Once the destination node is available for migration, the VM is resumed on it. This paper presents a good approach to migrate VMs in federated clusters, but this method is only applied in the Shirako and can't work well in a common datacenter.

Verma et al. [5] propose an algorithm that dynamically packs the VMs running HPC applications. It uses dynamic consolidation and dynamic voltage scaling policies to reduce the power consumption of clusters. The placement is made to satisfy the CPU and memory requirements of each VM while reducing the number of migrations. The algorithm is an extension of the FFD heuristic and it migrates VMs located on overloaded nodes to under-exploited nodes. Nevertheless, this implies that the approach may fail to compute a new viable configuration or miss opportunities for savings when rearranging the VMs within the under-loaded nodes is essential to reach a viable configuration or enable more beneficial migrations.

Wood et al. [6] develop a VM scheduling system named Sandpiper which is a XEN based automated provisioning system for monitoring and detecting hotspots. Thus, it detects when a VM is under-provisioned and either allots more resources locally or migrates the VM to a new PM which is capable of supporting the VM. However, Sandpiper is easy to choose a wrong target PM, because Sandpiper takes VM migration decision based on a metric, which it refers to as *volume*. That means it converts the three dimensional resource information of PMs into a single dimension metric (which is volume) and then uses this single dimension metric for worst fit in a three dimensional scenario. In this process, the information about the shape of the resource utilization is lost.

There are several other literatures introduced how to select migration targets in virtual cluster environment. Random algorithm [7] is the most simple and widely used method. It does not take any system information into account, randomly select migration target node. Dinda [8] proposed cyclic algorithm which based on preset order of nodes to be selected as migration targets, we noticed cyclic algorithm is an improved version of random method. However these two methods are short of consideration of efficiency as well as stability. Central algorithm [9], which proposed by Zhou, design a special load information center LIC. The LIC centralized collecting and managing system load information. Centralized algorithm can effectively avoid the occurrence of nodes conflict, but it brings the problems of single node failure phenomena. It suffers from scalability and is not suitable for dynamic systems.

In addition to these various types of scheduling strategies, some companies and research institutions have developed several DRS products and projects, like: Vmware DRS [10], OpenNebula [11], Ganglia [12], Entropy [13] and so on.

III. SYSTEM ARCHITECHTURE

In virtual computing environment, it is inevitable to go through the process of dynamic VM migration from creating a VM to hibernate a VM. A complete migration process consists of four parts as shown in Figure 1: data monitoring, load predicting, migration scheme making and scheme executing.

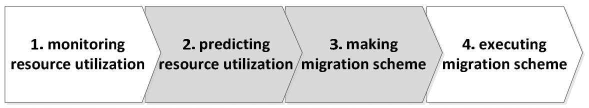

1. monitoring resource utilization
2. predicting resource utilization
3. making migration scheme
4. executing migration scheme

Figure 1. A complete scheduling process.

The major goal of our design is to evaluate the performance of Smart-DRS strategy while applied in data center environment. In order to avoid time-consuming and complicated implementation in code without knowing potential effects of the modification, we choose some mature open source projects in our system, such as Ganglia in monitor module and Xen Motion [14] in execution module. So in this paper, we just introduce our work about the parts with a gray logo in Figure 1. That is our Smart-DRS, an integrated scheduling algorithms, the detailed system architecture is described in Figure 2.

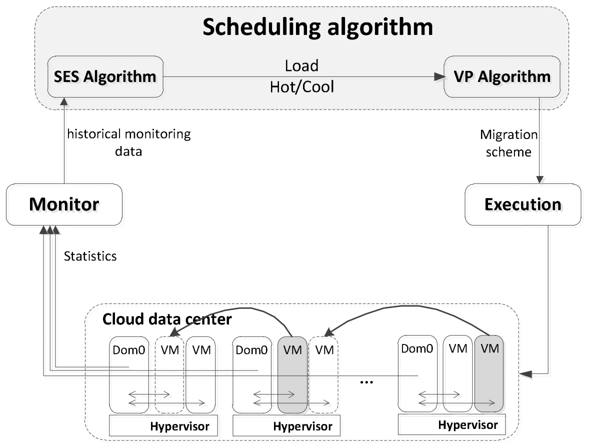


Figure 2. System architecture.

Firstly, our system needs to get the historical monitoring data of resource utilization, and then reasonably predict the upcoming time slots to load based on *SES* algorithm. According to the prediction value, system should be aware that whether system is overheating or overcool. We set upper threshold and lower threshold for each kind of resource, in order to determine the need for a migration. If a migration is triggered, we should make an optimized migration scheme based on *VP* algorithm, a novel and efficient scheduling algorithm. Then a mapping list about the VMs to be migrated to the potential target PMs would be generated. At last, executing module carries out the online migration based on the mapping list.

IV. Scheduling Algorithm

In this section, we concentrate on the methodologies of Smart-DRS. In terms of dynamic resource scheduling, Smart-DRS proposes an integrated solution that determines *when* and *where* to migrate VMs.

A. SES Algorithm for Predicting

SES algorithm is a widely used forecasting method and it's an efficient technique that can be applied to time series data, either to produce smooth data for presentation. *SES* algorithm is very suitable for our model, because in our system, the time series data are a sequence of historical monitoring data of resource utilization. And the observed phenomenon may be an essentially random process, or it may be an orderly, but noisy, process. Whereas in the *SES* the past observations are weighted equally, exponential smoothing assigns exponentially decreasing weights over time.

SES removes random perturbations of time series data, then the form of *SES* is given by the formulas (1), while the sequence of observations begins at time *t* = 0.

$$\overrightarrow{y_{t+1}} = \overrightarrow{y_t} + \alpha(y_t - \overrightarrow{y_t}) \tag{1}$$

Where $\overrightarrow{y_{t+1}}$ is the prediction value at time *t+1*, $\overrightarrow{y_t}$ is the prediction value at time *t*, while y_t is the real value at time *t*, α is the smoothing factor, and 0< α <1. Formulas (1) can change further to formulas (2).

$$\overrightarrow{y_{t+1}}=\alpha y_t + (1-\alpha)\overrightarrow{y_t} \tag{2}$$

Through the observation, the above equation is kind of a recursion equation, which can be expanded to formulas (3).

$$\overrightarrow{y_{t+1}}= \sum_{i=0}^{t-1} \alpha(1-\alpha)^i y_{t-i} + (1-\alpha)^t \overrightarrow{y_1} \tag{3}$$

As we can see from formulas (3), exponential smoothing forecast value is a weighted sum of all the previous real observational values. That means *SES* makes use of all the historical data, so it has more stability and regularity.

1) The Value of α *:* The value of α determine the degree of smoothing and how responsive the model is to fluctuation in the time series data. The value of α is arbitrary and is determined both by the nature of the data and the feeling by the forecaster as to what constitutes a good response rate. A smoothing constant close to zero leads to a stable model while a constant close to one is highly reactive. To our knowledge, it is good to set a small value of α in order to increase the weight of historical data when the time series data doesn't have fluctuations. On the contrary, it's good to set a big value of α to increase the weight of recent prediction value when the time series data has obvious fluctuations.

2) The Value of $\overrightarrow{y_1}$ *:* In fact, the smaller value of α, the more sensitive our prediction value will be on the selection of this initial smoother value $\overrightarrow{y_1}$. In our method, we define $\overrightarrow{y_1}$ to be initialized to $y_1$when the number of series data is more than an experiential value 15 [15]. While the number is less than 15, we define $\overrightarrow{y_1}$ to be the average value of the series data. As shown in formulas (4).

$$\overrightarrow{y_1} = \begin{cases} \sum_{i=1}^{k} y_i/k & , \quad k < 15 \\ y_1 & , \quad k \geq 15 \end{cases} \tag{4}$$

After the process mentioned above, we could calculate the prediction value which is a major basis of resource scheduling. Several notations are described in table 1.

TABLE I. NOTATION DESCRIPTION

Variable	Meaning
$preCpu_i$	the predicted value of Cpu utilization of i_{th} PM
$preMem_i$	the predicted value of Mem utilization of i_{th} PM
$preIO_i$	the predicted value of IO utilization of i_{th} PM
UC	upper threshold of CPU
UM	upper threshold of Mem
UIO	upper threshold of IO
LC	lower threshold of CPU
LM	lower threshold of Mem
LIO	lower threshold of IO

As long as one of the followed three formulas is true, which means there will be several load-imbalanced PMs. Then next step of Smart-DRS strategy should be carried out.

$$\begin{cases} preCpu_i \geq UC & or & preCpu_i \leq LC \\ preMem_i \geq UM & or & preMem_i \leq LM \\ preIO_i \geq UIO & or & preIO_i \leq LIO \end{cases} \tag{5}$$

B. VP Algorithm for Scheduling

VP algorithm, to the best of our knowledge, addresses many drawbacks existing in other methodologies [16]. In this algorithm, firstly, we choose three major resources available with the PM, namely CPU, Mem and IO. These resources form the three dimensions of an abstract object. We normalize the resources along each axis. Thus, the total available resource can be represented as a unit cube which is called Normalized Resource Cube (NRC). Secondly, we should express the resource related information of potential VMs and potential target PMs as a vector within the NRC, as shown in Figure 3.

The total capacity of PM is expressed as a vector from the origin of the cube (0, 0, 0) to point (1, 1, 1). This vector is identified as Total Capacity Vector (TCV). Resource Utilization Vector (RUV) represents the current utilization of resources of a PM The vector difference between TCV and RUV represents the Remaining Capacity Vector (RCV), which essentially captures how much capacity is left in the PM. The resource requirement of a VM is represented by Resource Requirement Vector (RRV) which is the vector addition of normalized resource requirement vectors of each resource type. So, to measure the degree of imbalance of resource utilization of a PM, we define the Resource Imbal-

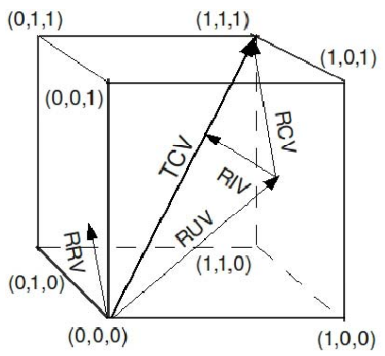


Figure 3. Normalized Resource Cube.

ance Vector (RIV) of PM, which is the vector difference between RUV's projection on TCV and RUV. RIV of a VM is defined in a similar way: it is the vector difference between RRV's projection on TCV and the RRV.

Then, we project NRC on a plane perpendicular to the principal diagonal of the cube as shown in Figure 4. It is easy to see that this would result in a regular hexagon on the said projection plane. After that, we take the projection of the resource vectors onto the projection plane to make the resource information of 3D space reduce to 2D plane.

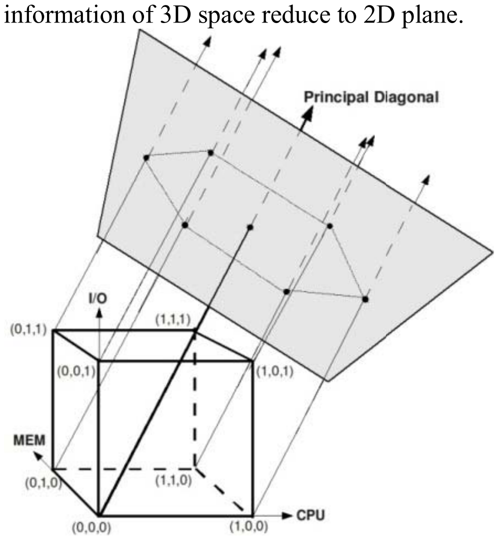


Figure 4. Resource Vector Projection.

By carefully analysis, we can know the regular hexagon on the projection plane is divided into six regular triangles. Shown in Figure 5, we named these triangles like that: ΔCI defines the region where the projection of tip of RUV whose CPU> Mem > IO. Likewise, ΔMC represents the region where the projection of tip of RUV whose Mem > CPU > IO. Other triangles can be identified by similar inequalities.

Further analysis shows that the VMs and PMs whose resource vectors projected in the same triangle or axis have an affinity.

Now, we group all of the potential VMs and potential target PMs based on the rules that the RRV of a VM and RCV of a PM are in the same triangle or the same axis and in the opposite direction. That means they have the characteristics of complementary resources.

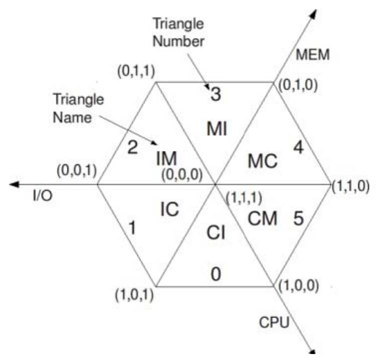


Figure 5. The Planar Resource Hexagon.

Our ultimate goal is to find a best complementary target PM for a potential VM in a group, whose RIV is of same magnitude as the VM's RIV and is in the opposite direction. The groups are shown in table 2. After comparison, we found that the triangle is imbalanced in the resource which the closest resource axis represents.

TABLE II. COMPLEMENTARY RESOURCES GROUPS

Number	Name	Resource Characteristics
0	CI	imbalanced in CPU, CPU-intensive
1	IC	imbalanced in IO, IO-intensive
2	IM	imbalanced in IO, IO-intensive
3	MI	imbalanced in Mem, Mem-intensive
4	MC	imbalanced in Mem, Mem-intesive
5	CM	imbalanced in CPU, CPU-intensive
6	CPU axis	especially imbalanced in CPU
7	Mem axis	especially imbalanced in Mem
8	IO axis	especially imbalanced in IO
9	CI/IC axis	imbalanced in CPU & IO
10	CM/MC axis	imbalanced in CPU & Mem
11	IM/MI axis	imbalanced in IO & Mem
12	Origin point	load balancing

If a triangle has no projection of tip of vectors, we will make it with adjacent triangles into a new group, because adjacent triangles are with similar characteristics of resources. Loop this process until every group has resource vectors.

C. Algorithm Elaboration

This segment details dynamic resource scheduling algorithm. Our VM placement algorithm can be chosen by one of the two goals:

- Load Balancing: Placing the new VM in such a manner that it helps in load balancing. For this purpose, the algorithm starts out with the PM which is the least loaded and has complementary resource usage with respect to the VM. The algorithm for load balancing is shown in Algorithm 1.
- Load Consolidation: Placing the new VM such that it helps in load consolidation and energy saving. In this case, the algorithm starts out with the PM with the highest load and has complementary resource usage with respect to the VM. The algorithm for load consolidation is shown in Algorithm 2.

When there is a VM whose resource requirements are not being fulfilled by the PM, on which it is hosted, thus leading to overload of the PM. Then a load balancing strategy as presented in Algorithm 1 is needed.

Algorithm 1 Load Balancing

```
1:  init_PotentialVMlist();
2:  init_PotentialPMlist();
3:  for all PMs will overload do
4:        do
5:        add_PotentialVM (); /*VM in this PM which has the
          most utilization*/
6:        delete the utilization of this VM and calculate the
          new_load of this PM
7:        while ( new_load is not overloaded);
8:  end for
9:  order_PotentialPMlist(); /* according to the remaining ca-
    pacity of every PM in ascending order. In the beginning,
    every PM is in the PotentialPMlist*/
10: named the triangle T which contains the PM in the top of
    PotentialPMlist
11: if no VM of PotentialVMlist locates T then
12:     T = T + left triangle of T + right triangle of T
13: end if
14: while PotentialVMlist !=NULL & PotentialPMl-
    ist !=NULL do
15:       choose the most complementary VM whose RIV is of
          the most closest magnitude ( is the most slightly
          less )as the PM's RIV and is in the opposite direction
16:       add_MigrationScheme(); /* add a record in migration
          scheme*/
17:       delete_PotentialVM();    /* current VM gets a target
          PM and pop it from the PotentialVMlist*/
18:       add the utilization of this VM and calculate the
          new_load of this PM
19:       if new_load is overloaded then
20:          add_PotentialVM();    /* current PM can't hold this
             VM and push it in the PotentialVMlist again*/
21:          delete_PotentialPM();
22:       end if
23: end while
24: if PotentialPMlist ==NULL & PotentialVMlist !=Null then
25:     Introduce a new PM; continue;
26: end if
27: if PotentialPMlist !=NULL & PotentialVMlist==NULL
    then
28:     return MigrationScheme; // end Algorithm
29: end if
```

When a PM runs in low utilization level, the VMs on it can be migrated to other PMs so that this PM can be taken offline. For achieve this goal, we present our algorithm shown in Algorithm 2.

Algorithm 2 Load Consolidation

```
1:  init_PotentialVMlist();
2:  init_PotentialPMlist();
3:  for all PMs will under lower_threshold do
4:      add_PotentialVM (); /* all VMs on this PM*/
5:  end for
6:  order_PotentialPMlist(); /* according to the remaining ca-
    pacity of every PM in descending order. In the beginning,
    every PM is in the PotentialPMlist*/
7:  named the triangle T which contains the PM in the top of
    PotentialPMlist
8:  if no VM of PotentialVMlist locates T then
9:      T = T + left triangle of T + right triangle of T
10: end if
11: while PotentialVMlist !=NULL & PotentialPMl-
    ist !=NULL do
12:     choose the most complementary VM whose RIV is of the
        most closest magnitude ( is the most slightly less )as the
        PM's RIV and is in the opposite direction
13:     add_MigrationScheme(); /* add a record in migration
        scheme*/
14:     delete_PotentialVM();    /* current VM gets a target PM
        and pop it from the PotentialVMlist*/
15:     add the utilization of this VM and calculate the new_load
        of this PM
16:     if new_load is overloaded then
17:        add_PotentialVM();    /* current PM can't hold this
           VM and push it in the PotentialVMlist again*/
18:        delete_PotentialPM();
19:     end if
20: end while
21: if PotentialPMlist ==NULL & PotentialVMlist !=NULL
    then
22:     Introduce a new PM; continue;
23: end if
24: if PotentialPMlist !=NULL & PotentialVMlist==NULL
    then
25:     return MigrationScheme; // end Algorithm
26: end if
```

V. PERFORMANCE EVALUATION

For evaluating the performance of Smart-DRS, we have implemented a prototype system as discussed in Section 3. Besides, we build a small-scale cluster whose detail information is listed as follows:

TABLE III. EXPERIMENTS ENVIRONMENT

PM Numbers	VM Numbers	CPU Frequency	Memory	Network	VMM Version
32	3200	3.3GHz	4GB	100Mb/s	Xen 3.0.3

A. Predicting Evaluation

In order to evaluate the performance of predicting algorithm, we need some of convictive host load samples. Fortunately, Dinda and O'Halloran from Carnegie Mellon University [17] offered us plenty of load samples by long-term tracing with many kinds of machines in a cluster system. We choose the day's collection of load time series on August 18, 2010 as our test samples.

In the first experiment, we continuously monitor 32 PMs for 21 minutes and record the real load value. After that, we respectively calculate the forecasting value for average when α= 0.1, 0.3, 0.5. As shown in Figure 6, the Mean Relative Error (MRE) is correspondingly 13.6%, 9.7% and 7.3%. That means *SES* algorithm has an acceptable predicting accuracy while applied in a small-scale datacenter and α= 0.5 is a better situation in our experiment environment.

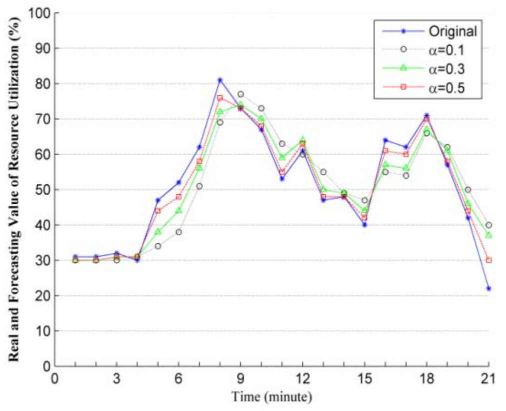
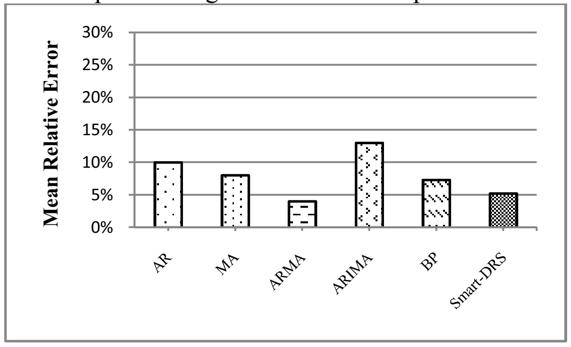
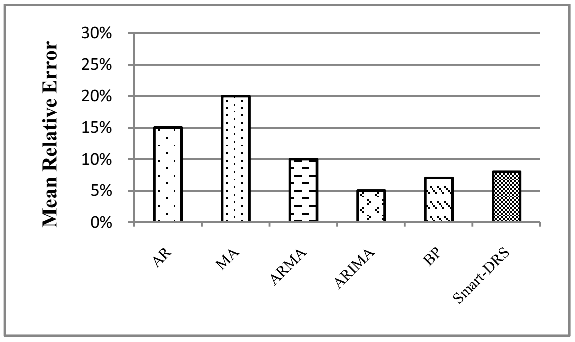


Figure 6. Real value and forecasting value.

In the second experiment, we compare Smart-DRS with other classic forecasting algorithms. Such as Autoregressive (AR), Moving Average (MA), Autoregressive Moving Average (ARMA), Autoregressive Integrated Moving Average (ARIMA) and Back Propagation (BP) neural network. To our knowledge, the prediction time interval has a certain influence on the forecasting value. In order to fully observe the predictive ability of various prediction models, we respectively set the prediction time interval as 1s and 15s to make a comparison. Figure 7 shows our experiment result.



(a) The time interval is 1 second.



(b) The time interval is 15 second

Figure 7. Mean relative error of various predicting models.

From the aforementioned experiments, we can know that: (1) some models work well with small interval, like MA and ARMA. Some other models work well with long interval, like ARIMA. However, some other models are not sensitive with the interval, like BP and Smart-DRS. (2) Even Smart-DRS is a relatively simple model, it has a relatively good performance, and the MRE of it is acceptable.

B. Scheduling Evaluation

In the third experiment, we try to evaluate the performance of scheduling. First of all, we should claim that we have defined $Balance_i$ to represent the degree of load balancing of each machine and $SYS_{balance}$ to represent the degree of load balancing of the whole system (a higher value is better).

$$Balance_i = \sqrt{\left[{\delta_{cpu}}^2 + {\delta_{mem}}^2 + {\delta_{io}}^2\right]/2} \quad (6)$$

$$\delta_{cpu} = cpu_i - cpu_{avg} \quad (7)$$

$$\delta_{mem} = mem_i - mem_{avg} \quad (8)$$

$$\delta_{io} = io_i - io_{avg} \quad (9)$$

$$SYS_{balance} = \sum_{i=1}^{n} Balance_i/n \quad (10)$$

This experiment compares the performance of Smart-DRS with two classical *Bin Packing* algorithms used in dynamic resource scheduling, *Best Fit Decreasing* (BFD) and *First Fit Decreasing* (FFD). We measure the execution time and $SYS_{balance}$ of each algorithm. As shown in Figure 8, the BFD and FFD consume much more execution time but achieve no better performance in$SYS_{balance}$ than Smart-DRS. The reason is that they have to sort all of the potential VMs and target PMs firstly according to their resource utilization.

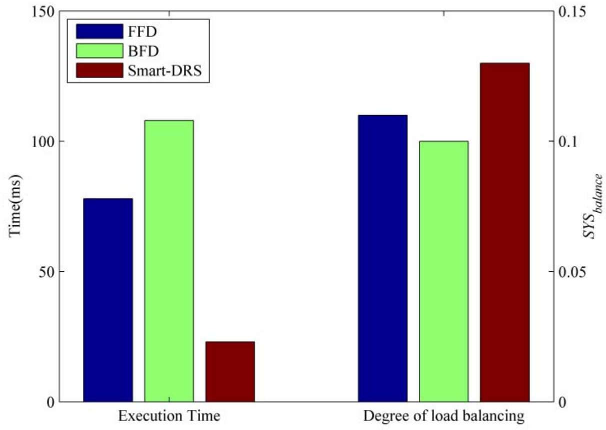
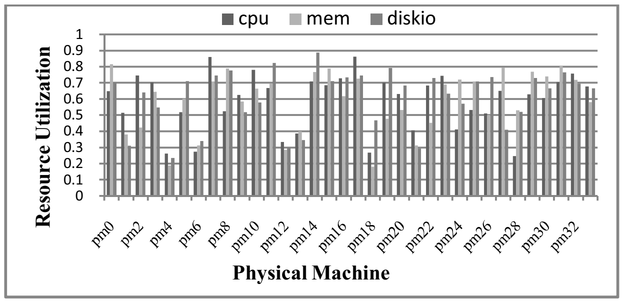
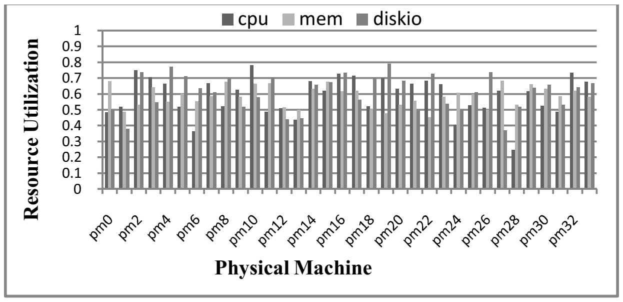


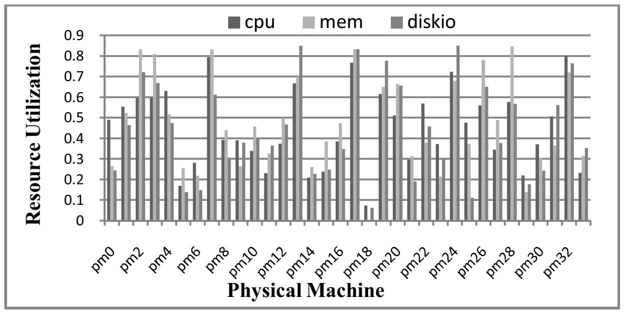
Figure 8. Performance comparison of various scheduling algorithms



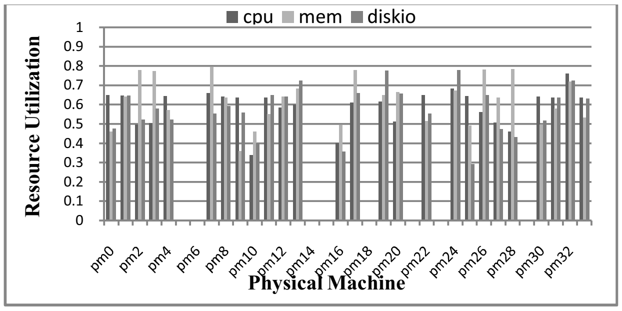
(a) Before scheduling (load balancing)



(b) After scheduling (load balancing)



(c) Before scheduling (load consolidation)



(d) After scheduling (load consolidation)

Figure 9. The contrast between before scheduling and after scheduling.

In our forth experiment, we assess the performance of Smart-DRS in load balancing and load consolidation. By continuously monitoring the resource utilization of cpu, mem, diskio of 32 PMs, we could observe the difference between before scheduling and after scheduling. In order to carry out this experiment, we have set 0.8 as the upper threshold and 0.2 as the lower threshold.

Figure 9 shows that our scheduling algorithm really works. It is clear that the load of system is imbalanced and the resource utilization of several machines exceeds 0.8 before scheduling as shown in Figure 9 (a). While in Figure 9 (b), after scheduling, the load of system is more balanced than before and none of resource utilization exceeds the upper threshold. Similarly, in Figure 9 (c) we can see the resource utilization of several machines is under 0.2 while in Figure 9 (d), these low utilization machines are taken offline and the others are load balancing.

VI. CONCLUSIONS

In this paper, we studied the server load balancing and power consumption problems in cloud datacenter environment and we have presented an integrated dynamic resource scheduling strategy named Smart-DRS. It employs *SES* algorithm to predict the resource utilization of PMs in order to avoid tiny and temporary load peak value triggering unnecessary migration. The experiment result shows that prediction value is pretty close with the real value. Then Smart-DRS employs *VP* algorithm to make the migration scheme. The reason why we choose *VP* algorithm is that it's a novel theory which can be used to make the process of choosing PMs easier and more appropriate. The experiment result tells us that it really works and it's a kind of low cost and efficient method.

Nevertheless, our strategy has some limitations that we plan to address in the future. We now don't consider the spending of migration, sometimes, the best match may bring much more spending while a common match is our best choice. In addition, various other measurements and optimization strategies will need to be explored in the future.

REFERENCES

[1]. F. Hermenier, X. Lorca, J.M. Menaud, G. Muller, and J. Lawall, "Entropy: a consolidation manager for clusters," in Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, 2009, pp. 41-50.

[2]. Hyser, C. and Mckee, B. and Gardner, R. and Watson, B.J, "Autonomic Virtual Machine Placement in the Data Center," Hewlett Packard Laboratories, 2007, pp.189-195.

[3]. L. Grit, D. Irwin, A. Yumerefendi, and J. Chase, "Virtual machine hosting for networked clusters: Building the foundations for "autonomic" orchestration," in First International Workshop on Virtualization Technology in Distributed Computing, 2006, pp.1–8.

[4]. D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, and K. G.Yocum, "Sharing networked resources with brokered leases," in Proceedings of the annual conference on USENIX Annual Technical Conference, Berkeley, CA, USA, 2006, pp.18-18.

[5]. A. Verma, P. Ahuja, and A. Neogi, "Power-aware dynamic placement of hpc applications," in Proceedings of the 22nd annual international conference on SuperComputing, New York, NY, USA, 2008, pp.175–184.

[6]. T.Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif,

"Black-box and gray-box strategies for virtual machine migration," in Proceedings of the 4th ACM/USENIX Symposium on Networked Systems Design and Implementation, 2007, pp.229-242.

[7]. L. Kleinrock and S. Lam, "Packet switching in a multiaccess broadcast channel: Performance evaluation," IEEE Transactions on Communications, vol.23, pp.410-423, April, 1975.

[8]. Peter A. Dinda and David R. O'Hallaron, "An evaluation of linear models for host load prediction," in Proceedings of the 8th international symposium on High Performance Distributed Computing, 1999, pp.87-96.

[9]. Songnian zhou, "A trace-driven simulation study of dynamic load balancing," IEEE Transaction on Software Engineering, vol.14, pp.1327-1341, Sept,1998.

[10]. VMware DRS. http://www.vmware.com/products/drs/

[11]. OpenNebula. http://opennebula.org/

[12]. Ganglia. http://ganglia.sourceforge.net/

[13]. Entropy. http://entropy.gforge.inria.fr/

[14]. XenMotion. http://support.citrix.com/article/CTX115813

[15]. Exponential Smoothing. http://en.wikipedia.org/wiki/ Exponential _smoothing

[16]. Mayank Mishra, Anirudha Sahoo, "On theory of VM placement: Anomalies in existing methodogies and their migration using a novel vector based approach," in IEEE 4th International Conference on Cloud Computing, 2011, pp.275-282.

[17]. P．Dinda, "Online prediction of the running time of tasks," in Proceedings of 10th IEEE international symposium on High Performance Distributed Computing, 2001, pp.383-394.

[18]. Bin packing problem. http://en.wikipedia.org/wiki/Bin packing problem

[19]. E. Arzuaga, "Quantifying load imbalance on virtualized enterprise servers," in Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering, 2010, pp.235-242.

[20]. G. Khanna, "Application performance management in virtualized server environments," in 10th IEEE/IFIP Symposium on Network Operations and Management, 2006, pp.373-381.

[21]. A. Singh, "Server-storage virtualization: integration and load balancing in data centers," in Proceedings of the 2008 ACM/IEEE conference on Supercomputing, 2008, pp.1-12.

[22]. T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy, "Predicting Application Resource Requirements in Virtual Environments," HP Laboratories, 2008, pp.122-128,.

[23]. Qingyi, G, "VirtualRank: A Prediction Based Load Balancing Technique in Virtual Computing Environment," in IEEE World Congress on. Services , 2011, pp.247-256.