Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2013, Article ID 878542, 10 pages
http://dx.doi.org/10.1155/2013/878542

Research Article

A Decentralized Virtual Machine Migration Approach of Data Centers for Cloud Computing

Xiaoying Wang, Xiaojing Liu, Lihua Fan, and Xuhan Jia

Department of Computer Technology and Applications, Qinghai University, Xining, Qinghai Province 810016, China

Correspondence should be addressed to Xiaoying Wang; xiaoyingwang.paper@gmail.com

Received 3 June 2013; Accepted 19 July 2013

Academic Editor: Yuxin Mao

As cloud computing offers services to lots of users worldwide, pervasive applications from customers are hosted by large-scale data centers. Upon such platforms, virtualization technology is employed to multiplex the underlying physical resources. Since the incoming loads of different application vary significantly, it is important and critical to manage the placement and resource allocation schemes of the virtual machines (VMs) in order to guarantee the quality of services. In this paper, we propose a decentralized virtual machine migration approach inside the data centers for cloud computing environments. The system models and power models are defined and described first. Then, we present the key steps of the decentralized mechanism, including the establishment of load vectors, load information collection, VM selection, and destination determination. A two-threshold decentralized migration algorithm is implemented to further save the energy consumption as well as keeping the quality of services. By examining the effect of our approach by performance evaluation experiments, the thresholds and other factors are analyzed and discussed. The results illustrate that the proposed approach can efficiently balance the loads across different physical nodes and also can lead to less power consumption of the entire system holistically.

1. Introduction

Recently, as a newly emerged technology, cloud computing [1] becomes a new paradigm for dynamic provisioning of various services. It provides a way to deliver the infrastructure, platform, and software as services available to consumers in a pay-as-you-go manner [2]. Such typical commercial service providers include *Amazon*, *Google*, and *Microsoft*. In cloud computing environments, large-scale data centers [3] are usually the essential computing infrastructure, which are comprised of plenty of physical nodes with multiple virtual machines running upon them.

The virtualization technology enables a novel model such that customized virtual environments could be created upon the physical infrastructure [4]. The use of virtualization techniques provides great flexibility with the capability to consolidate multiple virtual machines on a single physical node [5]. In this way, the resource capacity allocated to different virtual machines could be resized, and virtual machines could also be migrated [6] across different physical nodes on demand to achieve various purposes. Recently, most modern virtualization technologies products have realized the notion of live or seamless migration of virtual machines that involve extremely short downtimes ranging from tens of milliseconds to a second [7]. Thus, the migration of virtual machines has emerged as a promising technique to be utilized by resource management algorithms to rapidly solve the problems in virtualized data centers.

To fully utilize the underlying cloud resources, the provider has to ensure that the services can be flexibly delivered to meet various consumer requirements which are usually specified by (service level agreements) SLAs [8], while keeping the consumers isolated from the underlying physical infrastructure. However, since the workloads of the different applications or services fluctuate a lot as time elapses, it is a challenge to adaptively manage the resource allocation and make appropriate decisions. Thus, the potential benefits of migrating virtual machines provide the opportunity to address the issues of high performance concern of the service provider.

On the other hand, as green computing [9] gains a lot of attention due to significant energy consumption of large-scale

data centers, the focus of the researcher is gradually shifted from optimizing the pure performance to optimizing the energy efficiency while maintaining high quality of services. Consequently, the energy costs become an important part of the (total cost of ownership) TCO, which needs to be suppressed from the point of the providers' views. Under such consideration, the proper migration of virtual machines could also lead to more consolidation and thus release some underutilized nodes, in order to further save energy costs.

A number of approaches addressing the issues of resource management for cloud computing have been proposed [10–17]. Many of them are based on centralized architectures, which are known to be not very scalable and might suffer from fault-tolerant issues. For example, the crash of the centralized resource arbiter will disable all of the later possible adaptive resource management actions, leading the whole system into a static state. Under the demand of autonomy, a truly decentralized solution is preferable, which brings improved scalability and naturally fault tolerant.

Given this analysis above, the main objective of our work is to design a decentralized virtual migration approach for data centers in cloud computing environment. The aim of the approach is to dynamically adjust the resource allocation amount by migration and reduce possible energy wastes at the same time. The main contributions of this paper include the following: (1) the definition of system models and power models for the cloud computing infrastructure discussed in this paper; (2) the design and development of the autonomic and decentralized mechanisms for dynamic virtual machine management to satisfy service quality requirements and reduce energy consumption as much as possible; (3) comprehensive performance evaluation results which illustrate the effect and efficiency of the proposed approach, from the aspects of SLA violation, power consumption, load-balancing effects, and so on.

The rest of this paper is then organized as follows: in Section 2 some related work in this area are presented and discussed; the system models of the target data center we studied is described in Section 3; in Section 4 we propose the decentralized virtual machine management approach; performance evaluation results are illustrated in Section 5; finally, in Section 6, we conclude the paper with some final remarks and future directions of this work.

2. Related Work

2.1. Virtualization-Based Resource Management for Cloud. As the employment of virtualization facilitates the fine-grained resource allocation in cloud environment, a number of researchers have made efforts on the study of virtualization-based resource management for cloud. Iqbal et al. [18] implemented a prototype that actively monitors the response time of each VM and adaptively scales up the application to satisfy the SLA promise. Maniymaran and Maheswaran [19] present a centralized heuristic algorithm to solve the VM creation and location problem, using a local search technique. Campegiani [20] proposed a genetic algorithm to find the optimal allocation of virtual machines in a multitier distributed environment. Almeida et al. [21] modeled each VM in the system as an M/G/1 open queue and applied Markov's Inequality to estimate the SLA violation possibility. Also, in our recent work [22], we have exploited model-free methodologies to adaptively manage the resources and energy consumption in virtualized environments.

However, from an architectural point of view, the resource manager in the above research usually lies on a central node as a single module, which is vulnerable to potential failures. Hence, we turn to exploit decentralized management approaches which could be a possible solution to address the availability issues of the central control unit.

2.2. Virtual Machine Migration. Since most major virtualization platforms support live migration within a local area network (LAN), some work has been done to study the migration mechanism and strategies of virtual machine migration inside the data center. Abdul-Rahman et al. [7] have surveyed relevant work in the area of migration-based resource manager for virtualized environments and also discussed several types of management algorithms. Liu et al. [23] have investigated the performance and energy cost for live VM migrations from both theory and practice. Choi et al. [24] have presented a framework that autonomously finds the VM migration thresholds at run time, using the history resource utilization. Park et al. [25] proposed an automated strategy for virtual machine migration in a self-managing virtualized environment, as well as an optimization model based on linear programming.

In contrast, in our work we employ a two-threshold VM migration strategy to balance the loads across different physical nodes in the data center, in order to meet the varying demands of user applications.

2.3. Power-Aware Resource Allocation Approaches. Besides performance, energy consumption becomes another critical design parameter in modern data center, and enterprise environments, because it directly impacts both the power deployment and operational cost. Hence, plenty of work has focused on energy-aware and power-aware resource allocation approaches. Krioukov et al. [26] presented an energy agile cluster that is power proportional and exposes slack. Zhu et al. [27] proposed several power-aware storage cache management algorithms and also an online power-aware cache replacement algorithm. Petrucci et al. [28] presented a dynamic configuration support for specifying and deploying power management policies in a platform running multiple application services. Chen et al. [29] designed a server provisioning and load dispatching algorithm which aimed to save energy without sacrificing user experiences.

Similarly, as designing the decentralized VM migration scheme, our work also considers power consumption as an important metric too. Specifically, by migrating out VMs, underutilized physical nodes could be released to save more energy.

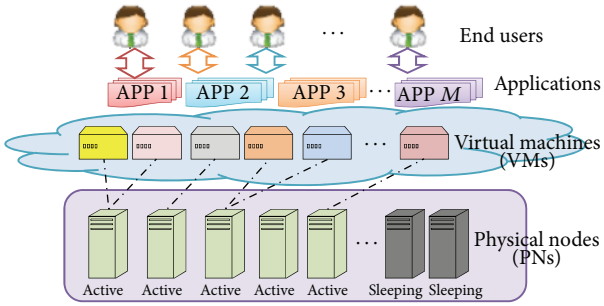

Figure 1: Data center architecture.

3. System Models

In this section, the basic architecture of the target system is described in detail as well as other model definitions. Then, the decentralized mechanism will be introduced.

3.1. Large-Scale Data Centers for Cloud. The target cloud environment we discuss in this paper is mainly a large-scale data center, which is usually comprised of a great number of physical nodes (PNs). The high-level system architecture is shown in Figure 1. Upon the physical infrastructure, virtual machines (VMs) are widely used to host many third-party applications. Multiple VMs can be dynamically started or stopped on a physical node according to incoming workloads, sharing the resources from the same physical devices. These VMs can run applications based on different operating system environments on a single physical node, providing usability and flexibility for cloud end users. At the upper level, end users obtain services from applications deployed in multiple virtual machines, which are residing on the underlying physical infrastructure.

Since the incoming workload varies significantly, the resource demands of each VM will fluctuate a lot too. To consolidate these workloads and release some underutilized resources, VM can be dynamically migrated across different physical nodes. In this way, some physical nodes could be turned off or into sleep mode in order to save extra energy. In Figure 1, we use "active" and "sleeping" to describe two types of node states, which are represented in light green and grey colors, respectively.

Besides, in the following problem discussion, we assume there are N physical nodes in the data center and K virtual machines running upon these nodes.

3.2. Power Model. Here, we present the power consumption model used in this paper. Since CPU usually consumes much more energy than the other parts of the computer, hereafter we focus on managing the power consumption and usage of CPU resources.

Most modern CPUs support Dynamic Voltage and Frequency Scaling (DVFS) techniques to dynamically change its own frequency to reduce energy wastes. Hence, we consider that the CPU utilization is typically proportional to the workload intensity, and the power consumption of a physical node is mainly impacted by its current CPU utilization. However, an idle physical node even with 0% utilization could still consume a plenty of power. Let α be the fraction of power consume by an idle node compared to a full utilized node and θ the current CPU utilization of the node. Then, we use the power model defined as follows to compute the power consumption of PN i:

$$P_i = \alpha \cdot P_i^{\mathrm{MAX}} + (1 - \alpha) \cdot \theta \cdot P_i^{\mathrm{MAX}}, \quad (1)$$

where P_i^{MAX} is the power consumption of PN i when it is fully utilized (i.e., it reaches 100% of CPU utilization).

3.3. Decentralized Mechanism. To manage the resources inside a large-scale data center, a central resource manager is usually designed and implemented to adjust the systemwide resources and make appropriate decisions. However, such centralized manner is vulnerable facing single-point failure, which might lead to an unmanaged status of the whole system. Here, we propose a decentralized mechanism to address such issues and provide guarantees for availability of the VM management actions in various cases.

As shown in Figure 2, each active node will send a load index of itself to some other nodes during each control interval. At the same time, it will receive some load indexes from other active nodes. The sending targets are randomly picked at each interval and will possibly change in the next interval. Each node will add the load information it received into its own load vector. Then, the average length of the load vectors over all nodes will be equal to the number of load indexed sending times, which is denoted as η in this paper.

In this way, the nodes are sending and receiving information to each other in a decentralized manner, without a central manager on the upper level. Such exchange will not be impacted even if some of the nodes fail to run or crack due to some unpredicted reason. On the other hand, the network flow would also be distributed and dispatched among different nodes in this case, rather than concentrated to a common node which receives all the information.

4. Decentralized Virtual Machine Migration Approach

In this section, the decentralized VM migration approach and resource management scheme will be presented, including the details of how to select the target VM and how to determine the destination node.

4.1. Load Vector Establishment. In order to make VM migration decisions, load information have to be collected first on each physical node. According to the decentralized working mechanism, each physical node maintains a load vector to receive load indexes from other peer nodes. Here, we use a tuple to represent the load index LI, defined as follows:

$$\mathrm{LI} = \langle src, dest, util \rangle, \quad (2)$$

where *src* indicates where the load index information comes from, *dest* indicates the ID of the target physical node that is

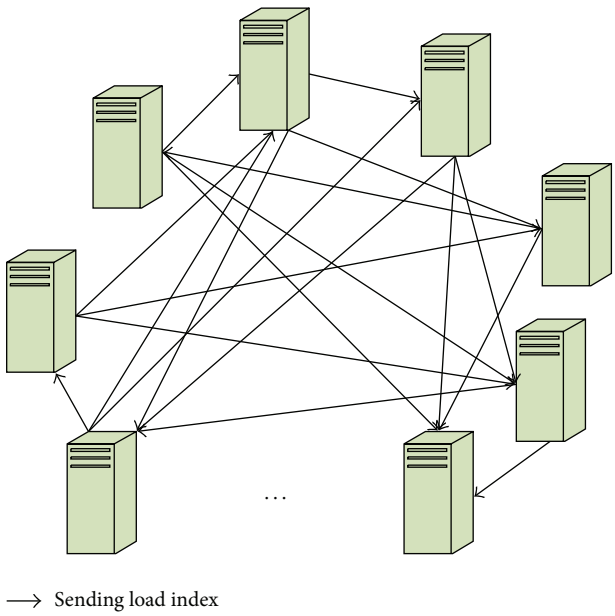

Figure 2: Decentralized load index exchange.

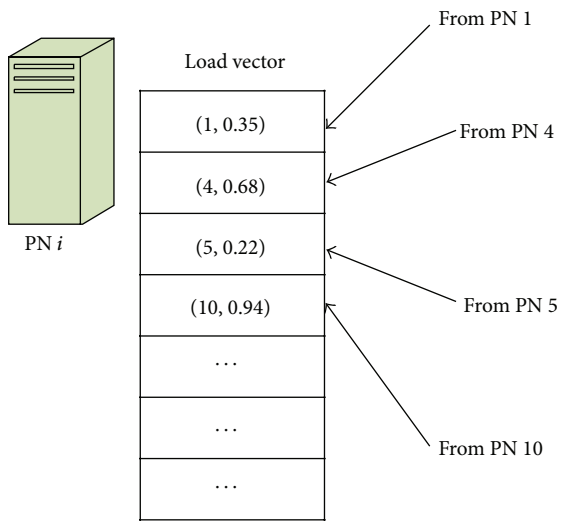

Figure 3: Load vector establishment.

going to receive this load index, and *util* denotes the current CPU utilization of the source node (as *src* indicates).

As shown in Figure 3, PN *i* will receive load indexes from other PNs and add them into the load vector of itself, which could be implemented by an array or a queue. Each element of the load vector contains information about the ID of the source node and its current CPU utilization. After all the load indexes in the current control interval have been received, the load vector of PN *i* could be established, which provides necessary directions for later migration decisions.

4.2. Selecting a VM to Migrate. Since the VMs are hosting different applications with varying workloads, the CPU utilization of the physical nodes will change a lot over time. When the CPU utilization is too high, exceeding a certain level, some VMs should be migrated to other PNs to release some resource capacity. On the other hand, if the CPU utilization is too low, below a certain level, we regard the node as "underutilized", and the VMs on it should also be migrated out to clear the load of this node. In this case, the blank node with no jobs could be turned into sleeping state so that more energy could be saved.

Hence, we introduce a double-threshold VM migration strategy, with two predefined threshold values called "lower threshold (LT)" and "upper threshold", respectively. The aim of setting the upper threshold is to preserve extra CPU capacity for unpredicted workload rises and to prevent SLA violations as much as possible. The objective of setting the lower threshold is trying to switch more physical nodes which are not fully utilized into sleep mode, leading to much less energy consumption than idling.

The pseudocode for the algorithm is presented in Algorithm 1, which describes how to select a VM to migrate on PN *i* which is overloaded. Lines 1~2 are the initialization of getting the current utilization of PN *i*. Lines 3~32 are loop to select VMs one by one to migrate out until the current utilization becomes less than the predefined upper threshold.

When the current utilization of PN *i* is higher than the upper threshold, there are three scenarios as shown in Figure 4. In case (a), we can find a VM that if it is migrated out, the resulted utilization of PN *i* will be reduced to a value lower than the upper threshold and higher than the lower threshold. In this case, we only need to choose this VM to migrate and end the selecting procedure, as shown in lines 6~16 of Algorithm 1. Otherwise, if we did not find any VM meeting the requirements in case (a), the reason is perhaps that all VMs utilize relatively little amount of CPU capacity, as shown in case (b). In this case, several VMs will be migrated out until the resulted utilization drops below the upper threshold, as shown in lines 17~29. The last scenario is that we cannot find any VM in case (a) and (b). That is probably because some VM occupies too much CPU capacity so that if it is selected, the utilization of PN *i* will fall down below the lower threshold, as shown in case (c). In this case, the VM will not be selected, since the migration of this VM will also cause overutilization on the target node. The corresponding code is as shown in line 30. At last, line 31 is a function to find a destination for the previously selected VM, which will be elaborated in the next section.

If the utilization of a physical node is below the lower threshold, we regard it as "underutilized". Then, all the VMs residing on the PN will be selected to migrate. The detailed algorithm is omitted here due to space constraint.

4.3. Decide the Destination. After some VM is selected to be migrated, the next necessary step is to find a migrating destination for it. Here, we use a function called *findDest*(*vmToMig*) to conduct the destination decision, which will find a proper physical node for *vmToMig* according to the *BestFit* strategy. The pseudo-code of the detailed algorithm is shown as Algorithm 2.

```
(1)  double util=this.utilization; //get the current utilization of PN i
(2)  VirtualMachine vmToMig=null;
(3)  while (util > UT)
(4)  {
(5)     vmToMig=null;
(6)     for each (vm in this.VMlist)
(7)     {
(8)        if (vm.toBeMigrated) continue; //skip the VMs that have been marked
(9)        double vmutil=calcutilz(vm); //get the CPU utilization of vm
(10)        if (vmutil > util-UT && vmutil <util-LT)
(11)        {   //case (a)
(12)           util = util – vmutil;
(13)           vmToMig=vm;
(14)           break; //find a VM to migrate
(15)        }
(16)     }
(17)     if (vmToMig==null)
(18)     {
(19)        for each(vm in this.VMlist)
(20)        {
(21)           if (vm.toBeMigrated) continue;
(22)           double vmutil=calcutilz(vm);
(23)           if (vmutil <util-LT)
(24)           {   //case (b)
(25)              util = util – vmutil;
(26)              vmToMig=vm;
(27)              break; //find a VM to migrate
(28)           }
(29)        }
(30)     if (vmToMig==null) break; //case (c)
(31)     findDest(vmToMig); //find a destination for the current VM to migrate to
(32)  }
```

Algorithm 1: Selecting a VM to migrate on PN *i* which is overloaded.

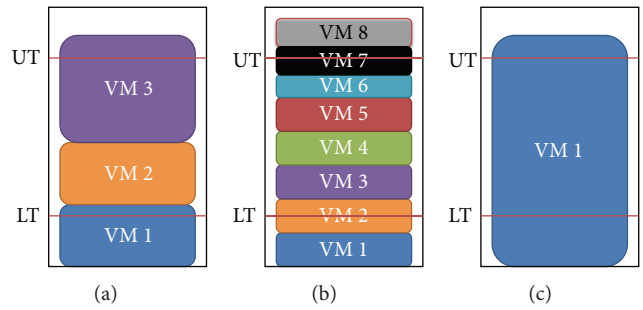

Figure 4: Some typical scenarios on overutilized nodes.

First, the utilization of the selected VM has to be computed as shown in line 1. Then, the PN traverses all the load indexes in its own load vector and try to find a target so that if the VM is migrated there, the resulted utilization will be between the lower threshold and the upper threshold. Among such nodes, the algorithm is prone to choose the one with minimum utilization currently, as shown in lines 2~13.

As shown in lines 18~24, if there is no such proper node meeting the above requirements, we continue to find a target with little resource utilization that its future utilization will still be below the lower threshold even with an additional VM.

Furthermore, if there is still not any target found during the last two rounds, it means that there are no suitable nodes which can accept an extra VM (maybe every active node is too over-utilized). Then, the source node of *vmToMig* will attempt to require a sleeping node to wake up and receive the VM to be migrated, as shown in lines 25~30. If it fails to require any sleeping node successfully, the whole procedure

```
(1)  calculate the utilization of vmToMig as vmutil
(2)  double minUtil=1.0;
(3)  int bestTarget=−1;
(4)  for each (li in LoadVector)
(5)  {   if (vmutil + li.util >=LT
(6)          && vmutil + li.util <= UT)
(7)      {   if (li.util < minUtil)
(8)          {
(9)            minUtil=li.util;
(10)           bestTarget=li.PNid;
(11)         }
(12)     }
(13) }
(14) if (bestTarget >0) //find target successfully
(15) {   vmToMig.dest=bestTarget;
(16)     return 0;
(17) }
(18) for each (li in LoadVector)
(19) {
(20)     if (vmutil + li.util <=LT)
(21)     {   vmToMig.dest=bestTarget;
(22)         return 0;
(23)     }
(24) }
(25) int getSleepNodeID=requireSleepNode();
(26) if (getSleepNodeID < 0) return −1;
(27) else
(28) {   vmToMig.dest=getSleepNodeID;
(29)     return 0;
(30) }
(31) return −1;
```

Algorithm 2: Function *findDest*(vmToMig).

of finding destination will be terminated, and *vmToMig* will not be migrated but still remain on its original physical node.

5. Performance Evaluation

In this section, we conduct a series of performance evaluation experiments of the decentralized VM migration approach proposed in Section 4. Since it is extremely difficult to conduct repeatable large-scale experiments on real-world infrastructure, we chose simulation methods to evaluate the performance of the proposed approach. We used C#.NET to develop an event-driven simulation environment, which could simulate the workload variation, application behaviors, task completion status, and energy consumption. Simulation parameter settings are first described and then the results will be illustrated later.

5.1. Parameter Settings. We have simulated a data center comprising 50 homogeneous physical nodes. Each node is modeled to have a CPU capacity of 750 MIPS. Power consumption is defined according to the model presented in Section 3.2, where P_i^{MAX} is set to 259 W according to the SPECpower benchmark [30], and α is set to 50%. Then, a physical node consumes 129.5 W with 0% CPU utilization and consumes 259 W with 100% CPU utilization. Upon the underlying physical infrastructure, there are 150 heterogeneous VMs hosting different kinds of applications. The workload and CPU demand of each VM varies as time elapsed. The initial CPU demand, minimum and maximum demand, and the variation amount of the VMs are set randomly according to a uniformly distributed variable, which simulated the independent fluctuation of different types of applications.

Furthermore, during the experiments, VM migration decisions have to be made across a constant control interval time, which is set to 60 seconds. Besides, we set the delay from sending the load information to receiving the information as 1 s, the VM migration time is set to 5 s, and the wakeup time of a sleeping physical node is set to 15 s. The simulation time of the entire experiment is 1440 minutes in total, which simulates a whole day effect of system running.

Notably, the length of the load vector of each physical node is set to 10, if not specified explicitly.

5.2. Threshold Analysis. In this subsection, we intend to examine the performance of the decentralized VM migration approach when setting different lower and upper utilization thresholds.

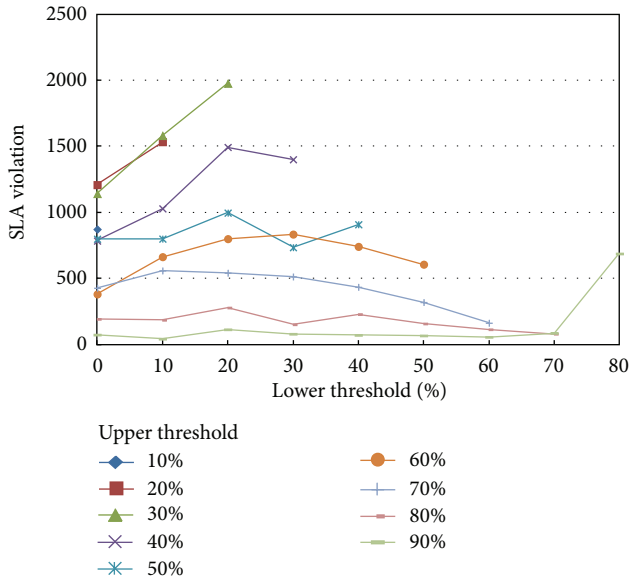

Figure 5: SLA violation analysis with different lower thresholds and upper thresholds.

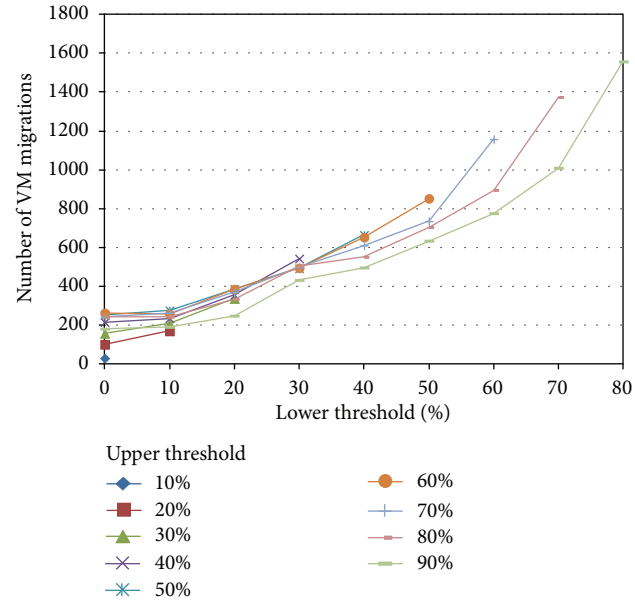

Figure 6: Number of VM migrations analysis with different lower thresholds and upper thresholds.

First, the average SLA violation amount is recorded and illustrated in Figure 5. As shown, we can see that increasing the upper threshold beyond 40% helps to reduce SLA violation remarkably. However, the impact of the lower utilization threshold is not regularly noticeable. The best result occurs at LT = 10% and UT = 90%, and we will use this setting for the following experiments if not specified explicitly.

Furthermore, we also investigate the number of VM migration times with different lower and upper utilization thresholds, and the results are illustrated in Figure 6. It can be observed that as the lower threshold increases, the number of VM migration times rises obviously. The reason is that as the lower threshold increases, more physical nodes will judge themselves as "underutilized" and more VM migrations will be triggered to eliminate resource waste. On the other hand, with the same lower threshold, higher upper threshold leads to the reduction of migration times. This is because the that as the upper threshold increases, the physical nodes are allowed to hold more VM demands, and then fewer migrations will be triggered due to overutilization.

When we jointly consider the results of Figures 5 and 6 together, it can be seen that although most results are relatively good at the aspect of SLA violation when UT = 90%, higher LT will trigger more VM migration times, which incurs heavier overhead to the whole system. Synthetically, we regard the combination of LT = 10% and UT = 90% as a possible appropriate choice for later experiments.

5.3. Load Balancing Effect. Here, we intend to examine the load balancing effect of the proposed decentralized approach. The experiments are repeated using three different strategies as follows.

(1) *Static*: In the initialization stage, the virtual machines are allocated onto the physical nodes as long as the utilization of each node does not reach 100%. Then, the VM placement scheme will not change during the system execution procedure.

(2) *RR*: In the initialization stage, the virtual machines are allocated onto the physical nodes one by one in a round robin manner, in order to balance the load among multiple physical nodes. VM migration is not supported in this strategy.

(3) *DVM*: As described in Section 4, load information is collected in a decentralized manner between different node pairs. Virtual machines will be dynamically migrated according to the load distribution among the physical nodes.

In this group of experiments, we compared the standard deviation across all of the 50 physical nodes. The results are shown in Figure 7. As it can be observed, the *Static* strategy leads to large deviation value since the VMs are distributed in an unbalancing way. When using *RR* strategy, although the deviation is small during the first several rounds due to the evenly distributed VMs in the initialization stage, the load becomes remarkably unbalanced in later time periods. In comparison, by dynamically migrating VMs among different physical nodes using *DVM* strategy, the resource utilization values are kept relatively more balanced, leading to the least deviation among all nodes.

5.4. Energy Consumption. In this subsection, we focus on the energy consumption of our approach compared to *RR* strategy. The number of physical nodes and VMs are set to 50 and 100, respectively, in order to simulate a relatively lighter workload scenario.

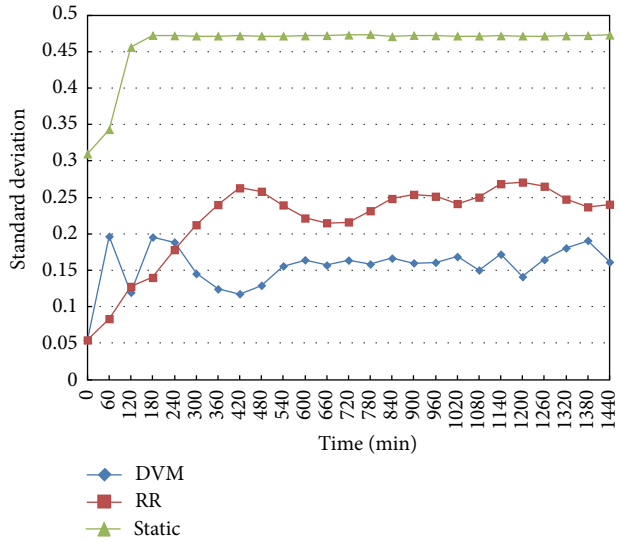

Figure 7: Standard deviation of all physical nodes with three different strategies.

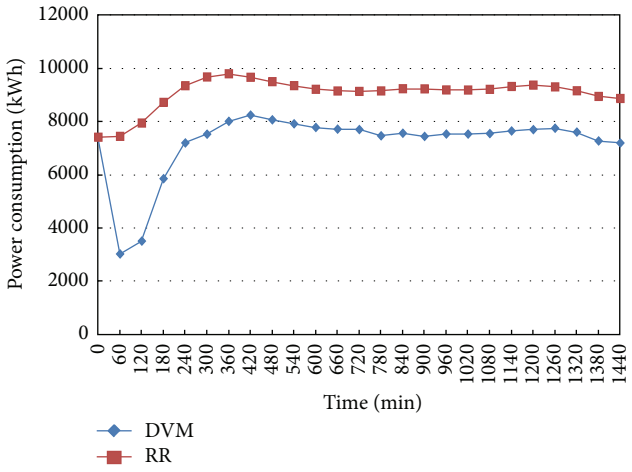

Figure 8: Power consumption comparison of DVM and RR strategies (no SLA violation).

The experimental results are shown in Figure 8. It is notable that the *DVM* strategy achieves less power consumption, leading to more than 20% energy savings. The reason is that our approach considers migrating VMs from the physical nodes whose utilization is under the predetermined lower threshold. In this way, the underutilized physical node could be released and be turned into sleeping status, which incurs much less power consumption than the idling state.

5.5. Impact of Load Vector Length. At last, we attempt to investigate the impact of the load vector length on the performance of our approach. The length of the load vector determines how many load indexes will be transferred during the system execution. Longer load vector may provide more information for the current physical node, but will also bring more overheads at the same time. We repeated several experiments with the same lower threshold and upper threshold but different length of the load vector, and the results are illustrated in Figure 9.

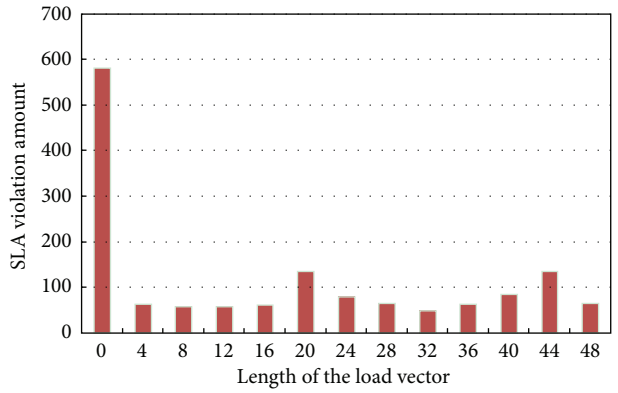

Figure 9: Power consumption comparison of DVM and RR strategies (no SLA violation).

It is notable that a large value of the load vector length will not always lead to better performance, even though for each node it gets more information from the other nodes. The reason is that a node will choose the most light-loaded node from its load vector as the destination target. However, a possible scenario is that multiple nodes choose the same node as the target but do not know that situation from each other. As a result, a light-loaded load might be selected as the target for many times, which makes it overloaded in the next interval and leads to much SLA violation. In other words, the performance is not proportional with the increase of the load vector length due to the decentralized mechanism.

From another point of view, shorter load vector could also achieve better performance which benefits from incomplete messages among different node pairs. Besides, the smaller value of the length could also reduce the network overhead for sending and receiving load indexes. Thus, we found 8 to 12 is an appropriate value for the load vector length in a data center comprised of 50 physical nodes.

6. Conclusions and Future Work

In this paper, we have proposed a decentralized resource management approach for data centers which use virtual machines to host many third-party applications. The system models are defined and described in detail. Then, we present the design of the decentralized VM migration approach, which considers both load balancing and saving of energy costs by turning some underutilized nodes into sleeping state. The VM migration decisions are made according to the two thresholds predetermined for the system, and several load indexes of one node will be sent to another several nodes randomly chosen according to the load vector length. Performance evaluation results of the simulation experiments illustrate that our approach can achieve better load balancing effect and less power consumption than other strategies. Besides, we also examine and discuss the impact of some key

factors in our approach on the final performance. The benefit of the decentralized approach is to eliminate the fatal problem of single-point failure, which helps improve the availability of the entire system.

As part of ongoing work, we plan to incorporate the proposed methods into our realistic cloud environment and examine its effect and efficiency when putting into real-world usage. Also, we are considering combining the centralized management and decentralized management approach together to further utilized their advantages in different aspects.

Acknowledgments

This paper is granted by the National Natural Science Foundation of China (no. 61363019) and the Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology (no. 2011-1).

References

[1] B. Hayes, "Cloud computing," *Communications of the ACM*, vol. 51, no. 7, pp. 9–11, 2008.

[2] M. Armbrust, A. Fox, R. Griffith et al., "Above the clouds: a berkeley view of cloud computing," Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[3] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[4] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, "Resource allocation using virtual cluster," in *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09)*, pp. 260–267, May 2009.

[5] X. Wang, D. Lan, G. Wang et al., "Appliance-based autonomic provisioning framework for virtualized outsourcing data center," in *Proceedings of the 4th International Conference on Autonomic Computing (ICAC '07)*, p. 29, Fla, USA, June 2007.

[6] M. Rosenblum and T. Garfinkel, "Virtual machine monitors: current technology and future trends," *Computer*, vol. 38, no. 5, pp. 39–47, 2005.

[7] O. Abdul-Rahman, M. Munetomo, and K. Akama, "Live migration-based resource managers for virtualized environments: a survey," in *Proceedings of the 1st International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING '10)*, pp. 32–40, 2010.

[8] P. Patel, A. Ranabahu, and A. Sheth, "Service level agreement in cloud computing," in *Proceedings of the Cloud Workshops at OOPSLA*, pp. 1–10, 2009.

[9] W.-C. Feng, X. Feng, and R. Ge, "Green supercomputing comes of age," *IT Professional*, vol. 10, no. 1, pp. 17–23, 2008.

[10] X. Wang, Z. Du, Y. Chen, and S. Li, "Virtualization-based autonomic resource management for multi-tier Web applications in shared data center," *Journal of Systems and Software*, vol. 81, no. 9, pp. 1591–1608, 2008.

[11] X. Wang, Y. Xue, L. Fan, R. Wang, and Z. Du, "Research on adaptive QoS-aware resource reservation management in cloud service environments," in *Proceedings of the IEEE Asia-Pacific Services Computing Conference (APSCC '11)*, pp. 147–152, December 2011.

[12] X. Wang, Z. Du, Y. Chen et al., "An autonomic provisioning framework for outsourcing data center based on virtual appliances," *Journal of Networks Software Tools and Applications*, vol. 11, no. 3, pp. 229–245, 2008.

[13] X. Wang, H. Xie, R. Wang, Z. Du, and L. Jin, "Design and implementation of adaptive resource co-allocation approaches for cloud service environments," in *Proceedings of the 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE '10)*, pp. V2484–V2488, August 2010.

[14] 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.

[15] H. N. Van, F. D. Tran, and J.-M. Menaud, "Autonomic virtual resource management for service hosting platforms," in *Proceedings of the ICSE Workshop on Software Engineering Challenges of Cloud Computing (CLOUD '09)*, pp. 1–8, May 2009.

[16] R. Urgaonkar, U. C. Kozat, K. Igarashi, and M. J. Neely, "Dynamic resource allocation and power management in virtualized data centers," in *Proceedings of the 12th IEEE/IFIP Network Operations and Management Symposium (NOMS '10)*, pp. 479–486, April 2010.

[17] H. N. Van, F. D. Tran, and J.-M. Menaud, "SLA-aware virtual resource management for cloud infrastructures," in *Proceedings of the IEEE 9th International Conference on Computer and Information Technology (CIT '09)*, pp. 357–362, October 2009.

[18] W. Iqbal, M. Dailey, and D. Carrera, "SLA-driven adaptive resource management for Web applications on a heterogeneous compute cloud," *Lecture Notes in Computer Science*, vol. 5931, pp. 243–253, 2009.

[19] B. Maniymaran and M. Maheswaran, "Virtual clusters: a dynamic resource coallocation strategy for computing utilities," in *Proceedings of the 16th IASTED International Conference on Parallel and Distributed Computing and Systems*, pp. 53–58, November 2004.

[20] P. Campegiani, "A genetic algorithm to solve the virtual machines resources allocation problem in multi-tier distributed systems," in *Proceedings of the 2nd International Workshop On Virtualization Performances: Analysis, Characterization and Tools (VPACT '09)*, 2009.

[21] J. Almeida, V. Almeida, D. Ardagna, C. Francalanci, and M. Trubian, "Resource management in the autonomic service-oriented architecture," in *Proceedings of the 3rd International Conference on Autonomic Computing (ICAC '06)*, pp. 84–92, June 2006.

[22] X. Wang, Z. Du, and Y. Chen, "An adaptive model-free resource and power management approach for multi-tier cloud environments," *Journal of Systems and Software*, vol. 85, no. 5, pp. 1135–1146, 2012.

[23] H. Liu, C.-Z. Xu, H. Jin, J. Gong, and X. Liao, "Performance and energy modeling for live migration of virtual machines," in *Proceedings of the 20th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC '11)*, pp. 171–181, June 2011.

[24] H. W. Choi, H. Kwak, A. Sohn, and K. Chung, "Autonomous learning for efficient resource utilization of dynamic VM migration," in *Proceedings of the 22nd ACM International Conference on Supercomputing (ICS '08)*, pp. 185–194, June 2008.

[25] J.-G. Park, J.-M. Kim, H. Choi, and Y.-C. Woo, "Virtual machine migration in self-managing virtualized server environments," in *Proceedings of the 11th International Conference on Advanced*

Communication Technology (ICACT '09), pp. 2077–2083, February 2009.

[26] A. Krioukov, S. Alspaugh, P. Mohan, S. Dawson-Haggerty, D. E. Culler, and R. H. Katz, "Design and evaluation of an energy agile computing cluster," Tech. Rep. UCB/EECS-2012-13, EECS Department, University of California, Berkeley, 2012.

[27] Q. Zhu, F. M. David, C. F. Devaraj, Z. Li, Y. Zhou, and P. Cao, "Reducing energy consumption of disk storage using power-aware cache management," in *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, pp. 118–129, February 2004.

[28] V. Petrucci, O. Loques, B. Niteroi, and D. Mossé, "Dynamic configuration support for power-aware virtualized server clusters," in *Proceedings of the WiP Session of the 21th Euromicro Conference on Real-Time Systems*, Dublin, Ireland, 2009.

[29] G. Chen, W. He, J. Liu et al., "Energy-aware server provisioning and load dispatching for connection-intensive internet services," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pp. 337–350, 2008.

[30] K.-D. Lange, "Identifying shades of green: the SPECpower benchmarks," *Computer*, vol. 42, no. 3, pp. 95–97, 2009.