Server Consolidation for Internet Applications in Virtualized Data Centers

Bo Wang
Department of Computer Science and Technology, Xi’an Jiaotong University
Xi’an, China 710049 & SKL Computer Architecture, ICT, CAS
wangbo2012@stu.xjtu.edu.cn

Ying Song
SKL Computer Architecture, ICT, CAS
Beijing, China 100190
songying@ict.ac.cn

Yuzhong Sun
SKL Computer Architecture, ICT, CAS
Beijing, China 100190
yuzhongsun@ict.ac.cn

Jun Liu
SPKLSTN Lab, Department of Computer Science and Technology, Xi’an Jiaotong University
Xi’an, China 710049
liukeen@mail.xjtu.edu.cn

ABSTRACT
Server consolidation based on virtualization technology simplifies system administration and improves energy efficiency by improving resource utilizations and reducing the physical machine (PM) number in contemporary service-oriented data centers. The elasticity of Internet applications changes the consolidation technologies from addressing virtual machines (VMs) to PMs mapping schemes which must know the VMs statuses, i.e. the number of VMs and the profiling data of each VM, into providing the application-to-VM-to-PM mapping. In this paper, we study on the consolidation of multiple Internet applications, minimizing the number of PMs with required performance. We first model the consolidation providing the application-to-VM-to-PM mapping to minimize the number of PMs as an integer linear programming problem, and then present a heuristic algorithm to solve the problem in polynomial time. Extensive experimental results show that our heuristic algorithm consumes less than 4.3% more resources than the optimal amounts with few overheads. Existing consolidation technologies using the input of the VM statuses output by our heuristic algorithm consume 1.06% more PMs.

Author Keywords
Elasticity; Internet application; server consolidation; virtualization

1. INTRODUCTION
Virtualization technology, such as VM, has been ubiquitously used in cloud computing for resource management. It offers opportunities not only to better isolation and manageability but also to on-demand resource provision for server consolidation. There are many efforts focusing on virtualization, such as resource virtualization, dynamic deployment of virtual machines (VMs), and on-demand resource allocation among the hosted VMs. These works lead to improvements in the performance of virtualization and resource utilizations. Server consolidation based on VM simplifies system administration and improves energy efficiency by improving resource utilizations and reducing the used physical machine (PM) number.

Server consolidation remaps VMs and PMs when the resources needed by the VMs change to minimize the used PM number or energy consumption. While almost all of the existing works [1–10] addressed the consolidation problem of the applications of which the corresponding VMs are fixed in number at runtime. There are some problems to apply these works to Internet applications, such as e-commerce and web services, whose instances each of which corresponds to exactly one VM can be tuned in number at runtime. The most major one is deciding the number of application instances before consolidation. If the instance number is too large for an application, there would be many underutilized VMs when the load is low, which will increase the used PM number. While if the number is too small, the performance requirement of the application will not be satisfied when the load is high. Thus, the used PM number is affected by not only the resource needed by every VM but also the VM number. So, it should be considered to adjust the number of instances for Internet applications when consolidating them.

Some elasticity managements [11–29] applied one or the combination of vertical scaling (resizing a VM) and horizontal scaling (adding/deleting VMs) to provide the mapping between elastic applications and VMs leased from a public cloud for minimizing the rent cost in the perspective of cloud users. While they do not consider the placement of VMs on PMs, which is critical for improving energy efficiency. A few of elasticity managements [30–32] minimized one or more of
SLA (service level agreement) penalty cost, rented hardware cost, software cost, and action (e.g., load balancing) cost for service providers. While, all of these works did not take into account energy efficiency which is one major goal of efficient operations in virtualized data centers [33]. Except that, they considered that the VMs with same configuration had identical performance for an application, which is not true for heterogeneous PMs.

In this paper, to our best knowledge, we make the first attempt to consolidate multiple Internet application for improving energy efficiency by minimizing PM number from the perspective of a service provider using its owned cloud. We first model the consolidation providing the application-to-VM-to-PM mapping to minimize the number of PMs as an integer linear programming (ILP) problem, and then present a heuristic algorithm to solve the ILP problem in polynomial time. In brief, the contributions of this paper can be summarized as follows:

1. We model the consolidation into an ILP problem which provides the application-to-VM-to-PM mapping minimizing PM number guaranteeing required performance.

2. To solve the ILP problem in polynomial time, we propose a heuristic algorithm. Its basic idea is respectively assigning an available PM and the VM instance type both of which provide the best ratios between performance and resource amount to the application with maximum relative difference between required performance and provided performance and then allocating the rest of this PM’s resources to applications in the same way.

3. We conduct extensive experiments using various benchmarks to investigate the effectiveness and efficiency of the proposed heuristic algorithm. The experiments results show that our heuristic algorithm consumes only about 4.3% more resources than the optimal amounts with few overheads and that two existing consolidation technologies using the input of the VM statuses output by our heuristic algorithm consume 1.06% more PMs.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 presents our model and heuristic algorithm. Section 4 evaluates our heuristic algorithm and Section 5 concludes this paper.

2. RELATED WORKS

2.1. VM Consolidation

The power consumption of a PM when it is powered on but idle is above 50% of that when it is busy (100% resource utilization) [34]. This motivates server consolidation which increases resource utilizations and energy efficiency by consolidating multiple applications concurrently running on fewer PMs.

Existing server consolidation algorithms [1–10] provided a target VM-to-PM mapping minimizing the PM number or with other objectives, e.g., VM migration cost and consumed energy, and switched the current mapping to the target mapping by VM migration and resource reallocation (vertical scaling) when some VMs have changes in their required resources. While, to our best knowledge, no existing work has considered consolidating Internet applications whose performance can be tuned by not only vertical scaling but also horizontal scaling for improving energy efficiency. Besides, all of these existing works changed the VM-to-PM mapping by migration leading to non-negligible performance loss and energy overhead [4, 35–37]. In this paper, we consolidate Internet applications taking both vertical and horizontal scaling into account.

2.2. Elasticity Management

In a cloud, customers request resources provided by the providers in the form of VMs each of which has a price. Cloud customers pay for the requested resources. Cloud providers should pay for SLA penalty cost, rented hardware cost, software cost and action cost [31, 32]. To minimize the cost for cloud customers or providers, a plenty of works [11–32] have studied on scaling the application horizontally and/or vertically.

A few works [11–16] studied on the horizontal scaling which tunes the number of VM instances depending to workload variations for an application. Compared to the vertical scaling, horizontal scaling adjusts allocated resources in coarse granularity. Horizontal scaling is supported by most enterprise clouds [38].

Some existing works [17–23] studied on the vertical scaling which reconfigures VMs. Vertical scaling in comparison to horizontal scaling allows to allocate resources with lower overhead in terms of time and cost [20]. While, vertically scaling up a VM can cause costly migration if its host has no enough resource. Vertical scaling is widely used for dynamic consolidation in data centers [18–20].

To more effectively manage the elasticity of applications, some works [24–29] combined horizontal scaling and vertical scaling. These works changed the current VM set into the target VM set which provided the required performance with minimal financial expenditure for customers. These works did not take into account the placement of VMs on PMs.

From the service providers’ perspective, a few works [30–32] studied on application-to-VM-to-PM mappings to minimize the cost operating in a per-application level. For example, SmartSLA [32] horizontally and vertically tuned VMs according to the average SLA penalty cost predicted using machine learning to minimize SLA penalty cost, rented hardware cost, and action cost. Jung et al. [30] predicted the behavior of workloads employing an autoregressive moving averages (ARMA) model and then tuned VMs to minimize SLA penalty. These works did not take advantage of consolidating multiple applications for improving resource utilizations and energy efficiency.

In addition, all of these above elasticity managements assumed that VMs with same configuration had identical performance for applications, which is not true for heterogeneous PMs.
On the contrary, our work studies on the server consolidation for Internet applications in the perspective of service providers. Our work provides application-to-VM-to-PM mapping to minimize the PM number satisfying the required performance.

3. SERVER CONSOLIDATION

In a virtualized data center providing Internet applications, as shown in Fig. 1, a request is distributed to an instance of the corresponding application which has multiple instances each of which is deployed on a VM hosted on a PM, by corresponding load balancer (LB). The designs of LBs are out of scope of this paper. In this paper, we focus on the server consolidation which minimizes PM number guaranteeing required performance for multiple Internet applications.

In this section, we first present the ILP model of the server consolidation (Section 3.1.) and then describe the heuristic algorithm solving the ILP model in polynomial time (Section 3.2.) in details.

3.1. Modelling

The goal of the consolidation is to solve the optimization problem (OP) that is to minimize the number of used PMs while the provided performance (e.g., throughput) satisfies the requirement for each application.

We take $A$ applications, $R$ types of resources, $P$ heterogeneous PMs and $V$ VM configurations $(v_1, \ldots, v_V)$ into account. The available amount of resource $j$ ($j = 1, \ldots, R$) on PM $k$ is $r_{j,k}$. The required performance of application $i$ ($i = 1, \ldots, S$) is $\mu_i$. For resource $j$, the configured amount $(l = 1, \ldots, V)$ is $v_{j,l}$ in VM instance type $l$. On PM $k$, the performance provided by a VM instance with type $l$ for application $i$ is $\mu_{i,k,l}$. We define the variables $x_{i,k,l}$, $i = 1, \ldots, S$, $k = 1, \ldots, P$, $l = 1, \ldots, V$, where $x_{i,k,l} = m$ if there are $m$ VM instances hosted on PM $k$ with type $l$ to provide application $i$, and the binary variables $z_k$, $k = 1, \ldots, P$, where $z_k = 1$ if PM $k$ is used and $z_k = 0$ if not. Table 3.1. summarizes these notations used in this paper.

We formulate the problem of server consolidation as an ILP problem as follows:

\[
\text{Minimize } \sum_{k=1}^{P} z_k, \quad (1)
\]

subject to:

\[
\sum_{i=1}^{A} \sum_{l=1}^{V} (v_{j,l} \cdot x_{i,k,l}) \leq r_{j,k} \cdot z_k, \quad \forall j = 1, \ldots, R, \forall k = 1, \ldots, P; \quad (2)
\]

\[
\sum_{k=1}^{P} \sum_{l=1}^{V} (\mu_{i,k,l} \cdot x_{i,k,l}) \geq \mu_i, \quad \forall i = 1, \ldots, A; \quad (3)
\]

\[
x_{i,k,l} \geq 0, \quad \text{and is integer,} \quad \forall i = 1, \ldots, A, \forall k = 1, \ldots, P, \forall l = 1, \ldots, V; \quad (4)
\]

\[
z_k \in \{0, 1\}, \forall k = 1, \ldots, P. \quad (5)
\]

The decision variables are $x_{i,k,l}$ ($i = 1, \ldots, A$, $k = 1, \ldots, P$, $l = 1, \ldots, V$) and $z_k$ ($k = 1, \ldots, P$). The objective (1) of this model is minimizing the PM number. Constraints (2) ensure that the aggregate amount of any resource required by all applications deployed on any PM does not exceed its available amount. Constraints (3) guarantee that the provided performance of any application satisfies the corresponding requirement. Constraints (4) and (5) represent the integrality and binary requirements, respectively, for decision variables. After solving this model, we achieve the application deployments,
Algorithm 1 The Heuristic Algorithm

A: the set of Internet applications, \(|A|\) 2-tuples: (an application, required performance);

\(P\): the set of available PMs, |P| (R+1)-tuples: (an available PM, the available amounts of resource 1, ..., R);

\(V\): the set of available VM instance types, \(|V|\) (R+2)-tuples: (an available VM instance type, the configured amounts of resource 1, ..., R, price in public cloud);

\(PV\): the set of the performance of every application running on a VM instance with each type hosted on each PM, \(|A|\cdot|V|\cdot(|P|+1)\) 4-tuples: (a, v, p, the performance), where \(a \in A, v \in V, \) and \(p \in P\);

\(M\): the set of application deployments, |M| 4-tuples: (a mapping, a, v, p, the provided performance), where \(a \in A, v \in V, \) and \(p \in P\);

Input: \(A, P, V, PV\)

Output: \(M\)

1: while \((P \neq \emptyset)\) do
2: \(\wedge(\exists a) (a \in A) \land \sum_{m \in M \land m(2)=a} m(5) < a(2)) \) do
3: \(\forall i \in m(i) \text{ is ith element in tuple } m(i)\)
4: \(R2P(a, p) = \max \{R2P(app, p) ; p \in P\}\)
5: \(R2P(app, p) = \max \{R2P(app, p') ; p' \in P\}\)
6: \(C7(a, v, p) = \min \{C7(\{\} ; v, p') \}
7: \) If \(vm = \text{null} \text{ then goto line 1}\)
8: \) end if
9: \) end while
10: \) end while

Step 2. 3MAX allocates a VM to the application with maximum \(RPR\) (line 6) on the PM selected in Step 1. 3MAX allocates a VM with the type giving the best performance for this application (line 7). If there are multiple PM instance types giving the best performance, 3MAX selects the VM instance type configured minimal amounts of resources (C7 in line 7).

Step 3. 3MAX repeats Step 2 until there is no available resource on the selected PM for any application (lines 8-10).

Step 4. 3MAX repeats Step 1-3 until the provided performances of all applications are satisfying their respective requirements or there is no available PM (line 1).

Table 2. The configuration of PMs for hosting VMs.

| CPU (cores) | MEMORY | #PM |
|-------------|--------|-----|
| Intel(R) Xeon(R) CPU E5410 @ 2.33GHz (8) | 8GB | 10 |
| Quad-Core AMD Opteron(tm) Processor 2378 (8) | 8GB | 10 |
| Dual-Core AMD Opteron(tm) Processor 2216 (4) | 4GB | 10 |

4. PERFORMANCE EVALUATION

In this section, we introduce our testbed and experiment design. And then we discuss the experimental results.

4.1. Testbed and Experiments Design

In our testbed, the configurations of PMs used as servers for applications are shown in Table 2. Each PM is configured with two 1000Mbps Network Interface Cards (NICs).

We select five applications from four benchmarks for our experiments, as follows.

- **TPC-W** [40] is a transactional web e-Commerce benchmark. The specification defines three different mixes of performance \((RPR_i = \mu_i / \sum_{k=1}^{P} \sum_{v \in V} \mu_{i,k,l} \cdot x_{i,k,l})\) and respectively assigning PM and the VM instance type both of which provide best (maximal) performance to the application. This is why we call it 3MAX. The details, outlined in Algorithm 1, are presented as follows.

3.2. The Heuristic Algorithm

As ILP is NP-hard problem [39], the methods exactly solving the ILP problem, such as enumeration or branch-and-bound, are not feasible to analysing the large scale systems because of their exponential time complexities. Thus, we provide 3MAX heuristic algorithm to find a near-optimal solution with low overhead.

The basic idea of 3MAX is selecting the application with maximum ratio between the required performance and provided PMs, \(x_{i,k,l} = \mu_{i,k,l} / \sum_{v \in V} \mu_{i,k,l} \cdot x_{i,k,l}\) and the used PMs, \(x_{i,k,l} = \mu_{i,k,l} / \sum_{v \in V} \mu_{i,k,l} \cdot x_{i,k,l}\).
web interactions, each varying the ratio of browse to buy activities. We use TPC-W with the options, the TPC-W Shopping Mix and 1.0 think time, to generate workloads. The performance is measured by Web Interactions Per Second (WIPS).

- Yahoo! Cloud Serving Benchmark (YCSB) [41] is a performance measurement framework for cloud serving systems. Six core workloads (Workload A-F) are provided. A tool YCSB Client is developed to execute the YCSB benchmarks. We chose Workload A, which has 50 percent reads and 50 percent updates, as the workload generator. One million records are loaded into each database server. Performance is evaluated by throughput (operations per second).

- Apache Benchmark (ab) [42] is a tool for benchmarking HTTP server. We design two applications by the benchmark, abk and abm. They transmit a fixed size file: 1KB, 1MB, which are representative log sizes in current data center [43], to their requests, respectively. Additional, to reduce the disk readings for increasing the performance, files are cached in the buffers in advance. Performance is evaluated by throughput (finished requests per second).

- SysBench [44] is a modular, cross-platform and multi-threaded benchmark tool for evaluating operating system (OS) parameters. We use the CPU performance benchmark which is one of the most simple benchmarks in SysBench. In this mode each request consists in calculation of prime numbers up to a specified value (20000 in the paper). Events (i.e. finished requests) per second (EPS) is the performance metric.

The configurations for VM instances have 4, 4, and 5 options for CPU (1-4 virtual CPUs (VCPUs)), memory ((1-4)×0.5GB), and NIC ((1-5)×200Mbps), respectively. Thus there are 80 different VM instance types. We do not consider the disk resource because the disk resource is never the bottleneck in all of our experiments. We assume that SysBench is an Internet application for which its processing results are the data returned to users for the requests.

We use the trace collected from the 1998 World Cup Web site [45] at the five days from May 28, 1998 to June 1, 1998 to generate the workloads of the five applications, respectively, in the following experiments. For an application, we scale the average request number per second within 15 minutes of the trace data by a factor so that the maximal scaled value is equal to the maximum throughput, $mt_i$, which is two fifths of the value of aggregating throughputs provided by all PMs when running the application on a VM with type provisioning best performance, $mt_i = \frac{2}{5} \sum_{k=1}^{P} \max_{1 \leq j \leq V} \mu_{i,k,l}$, and set the scaled values as the required throughputs. Our consolidation algorithm, 3MAX, runs every 15 minutes.

In the following experiments, we pin each VCPU of hosted VMs on a CPU core because performance loss of virtualization can be reduced by core pinning. On a server, the aggregated number of VCPUs of all hosted VM is not larger than the number of CPU cores to avoid the additional overhead of overcommitment.

Next, we first evaluate the performance of our heuristic algorithm on the accuracy (Section 4.2.), minimizing PM number (Section 4.3.) and scalability (Section 4.4.), and then experimentally study on the sensitivity of our algorithm on the accuracy of workload evaluations (Section 4.5.).

### 4.2. Accuracy

We measured the performance of 3MAX using the relative differences between the total throughputs ($\hat{\mu}_i$) achieved by 3MAX and the required throughputs ($\mu_i$), $RD_i = (\hat{\mu}_i - \mu_i)/\mu_i, i = 1, ..., A$, and the overall relative difference, $ORD = \frac{1}{A} \sum_{i=1}^{A} (\sum_{t} |RD_i|/t)$, where $t$ represents the experiment time intervals. The closer to 0 $RD_i$ and $ORD$ are, the less resources are wasted and thus the better our algorithm performance is, when $RD_i \geq 0, i = 1, ..., A$. $RD_i < 0$ indicates that the required performance is not satisfied for application $i$.

Figure 2 shows the relative differences of applications, provided by our consolidation algorithm. As shown in this figure, $RD_i, i = 1, ..., A$ are all close to 0, which means that our consolidation algorithm always has a high accuracy all the time. $RD_i, i = 1, ..., A$ are always slightly larger than 0, which means that the performance achieved by the consolidation is always satisfying the required performance, guaranteed by the termination condition that all required performances of applications are satisfied (line 1 in Algorithm 1). The $ORD$ is about 0.043, i.e., using our consolidation algorithm consumes only about 4.3% more resources than the optimal amounts.

### 4.3. Performance in Minimizing PM Number

In this section, we experimentally study on the performance of 3MAX in minimizing PM number by two existing consolidation algorithms. We respectively run these consolidation algorithms with the VM statuses output by 3MAX. If the PM number is not reduced by these algorithms, then PM number is minimized by 3MAX in practice is proved. These two consolidation algorithms respectively are First Fit Decreasing (FFD) [5] and Least Loaded (LL) [46]. The commonly used FFD packing algorithm places the largest VM on the first physical server on which it will fit. If there is no such server, the VM is placed in a new empty server. Least loaded (LL) algorithm [46] assigns the current VM to the used PM with least-load or a new server when there is no room for the VM in the used PMs. When running these algorithms in our
In this section, we evaluate the scalability of $3MAX$.

4.4. Scalability

In this section, we evaluate the scalability of $3MAX$ in consumed CPU time and ORD. We first scale the PM number by a factor ($f_{PM}$) ranging from 1 to 100 to examine the scalability as the PM number increases, and then scale the application number in the same way ($f_{APP}$) with 3000 PMs to study on the scalability as the number of applications increases. For example, $3MAX$ consolidating 5 applications on 30 PMs when $f_{PM} = 1$, same as the original system decreased in Section 4.1., and consolidating 5 applications on 3000 PMs when $f_{PM} = 100$. We scale the required throughputs of applications in original system by the factor, the ratio of the numbers of PMs and applications, and set them as corresponding required throughputs in the scaled system. We use the average application performance measured in the original system as that in the scaled systems. The results of running on a Quad-Core AMD Opteron(tm) Processor 2378 core are respectively shown in Fig. 4.

Figures 4a and 4b show a pattern of ORD change, ORD decreases with increasing the workloads of applications. The reasons are as follows. We consider that the resources are allocated to the VMs on a PM in a discrete way, such as CPU is allocated in a granularity of cores, as done in most of clouds, such as EC2 [38] and OpenStack [47]. The absolute difference between provided performance and required performance is less than the performance provided by a VM with minimal resources using $3MAX$. Therefore, the relative difference is decrease with increasing the workload for an application.

As shown in Fig. 4a, the time consumed by $3MAX$ increases quadratically with the PM number, which is consistent with the analysis in Section 3.2. As the application number increases, the consumed time, as shown in Fig. 4b, increases slightly because the consumed time depends largely on the PM number due to that the number of applications are much less than that of PMs. Our algorithm can make a decision only about 0.22 seconds which is much less than the decision-making periods (tens or hundreds of seconds) in most of clouds, even in the case of consolidating 500 applications on 3000 PMs.

4.5. Sensitivity

In this section, we experimentally study on the impact of the accuracy of workload evaluation on the accuracy and fluctuation of our consolidation algorithm. We use two workload evaluation methods, $\chi^2$ and $F$. These methods first test whether the workload of an application is changed by $\chi^2$-test [48] and $F$-test [48], respectively, using the data of current time window containing the time intervals of last requests, and evaluate the workload as the average value within current time window if the workload is tested to be changed. We set the size of time windows and the level of significance of tests as 1000 and 0.01, respectively.

As $F$-test has the assumption that the time intervals of requests follow exponential distribution [48], we generate workloads for evaluating workload evaluation methods and their impact on our consolidation algorithm as follows. We first generate 100 random numbers and realignment them as 20 combinations each of which contains 5 numbers respectively set as the workloads of the five applications. Then, we sample the exponential distributions with means of the generated 100 random numbers and take the samples as the input of evaluation methods. We set that each combination lasts 100 seconds.

The accuracy of workload evaluation is evaluated by the overall relative difference between the estimated workload $\hat{\lambda}$ and the real workload $\lambda$, $ORD = \sum_t (|\hat{\lambda} - \lambda|/\lambda)/t$. The closer to 0 $ORD$ is, the more accuracy an evaluation method is. Figure 5 shows the overall relative differences of these two evaluation methods and our consolidation algorithm respective using the workloads evaluated by these two methods.

As shown in Fig. 5, ORDs of $\chi^2$ and $F$ are respectively 0.027 and 0.041, which degrade 0.043 of the ORD of $3MAX$ into 0.0674 and 0.0662, respectively, i.e., these workload evaluation methods have only a little influence on the accuracy of our consolidation algorithm.
The numbers of evaluated workload changes (#E), application deployment changes (#T), and consolidations without changing deployment (#CNT).

The workloads of the five applications change 19 times, while these two evaluation methods respectively change the evaluated workloads 154 and 75 times, all of which are much more than the actual value, as shown in Fig. 6. This is because the workloads would be re-evaluated even when only one workload are judged to change by the test, which dramatically increases the fluctuations of the evaluation methods. Our consolidation algorithm reduces the fluctuations in two ways. The first is that two combinations of the workloads which have a small difference may correspond to a same application deployment, which reduces the number of application deployment changes. The second is that the current deployment may still satisfy the requirement when the evaluated workloads have small changes, which reduces the number of consolidations. Thus the numbers of application deployment changes, 85 and 55, shown in Fig. 6, are only about half of the numbers of evaluated workload changes, respectively.

3MAX alleviates fluctuations of evaluation methods while does not eliminates them. The deployment change numbers of $\chi^2+3\text{MAX}$ and $F+3\text{MAX}$ are more than that of $3\text{MAX}$, as shown in Fig. 6. Thus, more accuracy of workload evaluation method helps $3\text{MAX}$ to be more practical.

5. CONCLUSION

In this paper, to our best knowledge, we make the first attempt to study on consolidating multiple Internet applications in virtualized data centers. We model the consolidation into a ILP problem providing a three-tiered mapping, application-to-VM-to-PM, to minimize the PM number satisfied the required performances of applications. To solve the ILP problem in polynomial time, we propose a heuristic algorithm. Extensive experiments have been conducted to study on the effectiveness and efficiency of our heuristic algorithm. The experiments results show high accuracy of our heuristic algorithm having little sensitivity to the accuracies of workload evaluation methods with good scalability.

ACKNOWLEDGMENT

The authors are grateful to the anonymous reviewers’ comments. This work was supported in part by the project of NSFC under grants 61202060, 912183001, 61173112 and 61221062, the National High-Tech Research and Development Program (863) of China under grant 2013AA01A212.

REFERENCES

1. Ligang He, Deqing Zou, Zhang Zhang, Chao Chen, Hai Jin, and Stephen A. Jarvis. Developing resource consolidation frameworks for moldable virtual machines in clouds. Future Generation Computer Systems, 32(0):69 – 81, 2014.
2. Gergö Lovázs, Florian Niedermeier, and Hermann de Meer. Performance tradeoffs of energy-aware virtual machine consolidation. Cluster Computing, 16(3):481–496, 2013.
3. Hao Liu, Xin Qi, Shuo Yang, and S. Midkiff. Workload-Driven VM Consolidation in Cloud Data Centers. In Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International, pages 207–216, May 2015.
4. Fei Xu, Fangming Liu, Linghui Liu, Hai Jin, Bo Li, and Baochun Li. iAware: Making Live Migration of Virtual Machines Interference-Aware in the Cloud. IEEE Transactions on Computers, 63(12):3012–3025, 2014.
5. Akshat Verma, Puneet Ahuja, and Anindya Neogi. pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems. In Middleware 2008, pages 243–264, 2008.
6. Benjamin Speitkamp and Martin Bichler. A Mathematical Programming Approach for Server Consolidation Problems in Virtualized Data Centers. Services Computing, IEEE Transactions on, 3(4):266–278, 2010.
7. Ron C. Chiang and H. Howie Huang. Profiling-Based Workload Consolidation and Migration in Virtualized Data Centers. IEEE Trans. Parallel Distrib. Syst., 26(3):878–890, March 2015.
8. Tiago C. Ferreto, Marco A.S. Netto, Rodrigo N. Calheiros, and Csar A.F. De Rose. Server Consolidation with Migration Control for Virtualized Data Centers. Future Generation Computer Systems, 27(8):1027 – 1034, 2011.
9. Shekhar Srikantaiaah, Aman Kansal, and Feng Zhao. Energy Aware Consolidation for Cloud Computing. In Proceedings of the 2008 Conference on Power Aware Computing and Systems (HotPower’08), 2008.
10. Xiaocheng Liu, Chen Wang, Bing Bing Zhou, Junliang Chen, Ting Yang, and A.Y. Zomaya. Priority-Based Consolidation of Parallel Workloads in the Cloud. Parallel and Distributed Systems, IEEE Transactions on, 24(9):1874–1883, Sept 2013.
11. Jing Jiang, Jie Lu, Guannquan Zhang, and Guodong Long. Optimal Cloud Resource Auto-Scaling for Web Applications. In Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on, pages 58–65, May 2013.
12. Junliang Chen, Chen Wang, Bing Bing Zhou, Lei Sun, Young Choon Lee, and Albert Y. Zomaya. Tradeoffs Between Profit and Customer Satisfaction for Service Provisioning in the Cloud. In HPDC ’11, pages 229–238, New York, NY, USA, 2011. ACM.
13. Emiliano Casalicchio and Luca Silvestri. Mechanisms for SLA provisioning in cloud-based service providers. Computer Networks, 57(3):795 – 810, 2013.
14. Zhipiao Liu, Shangguang Wang, Qibo Sun, Hua Zou, and Fangchun Yang. Cost-Aware Cloud Service Request Scheduling for SaaS Providers. The Computer Journal, 57(2):291–301, 2014.
