Future Generation Computer Systems 50 (2015) 62–74

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Energy efficient scheduling of virtual machines in cloud with deadline constraint

Youwei Ding, Xiaolin Qin *, Liang Liu, Taochun Wang

College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, 210016, China

HIGHLIGHTS

- We develop a new VM scheduler to reduce energy cost for the cloud service providers.
- We deduce that there exists an optimal frequency for a PM to process certain VMs.
- We define the optimal performance–power ratio to weight the heterogeneous PMs in the cloud.
- The deadline constraint is satisfied by the definition of required resource of each VM.
- We achieve over 20% reduction of energy and 8% increase of processing capacity in best cases.

ARTICLE INFO

Article history:
Received 14 May 2014
Received in revised form
18 November 2014
Accepted 1 February 2015
Available online 11 February 2015

Keywords:
Energy efficiency
Virtual machine scheduling
Dynamic voltage and frequency scaling
Cloud computing

ABSTRACT

Cloud computing is a scale-based computing model, and requires more physical machines and consumes an extremely large amount of electricity, which will reduce the profit of the service providers and harm the environment. Virtualization is widely used in cloud computing nowadays. However, existing energy efficient scheduling methods of virtual machines (VMs) in cloud cannot work well if the physical machines (PMs) are heterogeneous and their total power is considered, and typically do not use the energy saving technologies of hardware, such as dynamic voltage and frequency scaling (DVFS).

This paper proposes an energy efficient scheduling algorithm, EEVS, of VMs in cloud considering the deadline constraint, and EEVS can support DVFS well. A novel conclusion is conducted that there exists optimal frequency for a PM to process certain VM, based on which the notion of optimal performance–power ratio is defined to weight the homogeneous PMs. The PM with higher optimal performance–power ratio will be assigned to VMs first to save energy. The process of EEVS is divided into some equivalent schedule periods, in each of which VMs are allocated to proper PMs and each active core operates on the optimal frequency. After each period, the cloud should be reconfigured to consolidate the computation resources to further reduce the energy consumption. The deadline constraint should be satisfied during the scheduling. The simulation results show that our proposed scheduling algorithm achieves over 20% reduction of energy and 8% increase of processing capacity in the best cases.

1. Introduction

Cloud computing has been developed to store, manage and analyze the massive data. The initial aims of cloud computing are high performance, scalability, capacity, cost of infrastructure and so on, not including energy. With the growth of the number and the size of data centers, energy consumption becomes a challenge for both companies and governments. It is shown that the cost of energy consumed by a server during its lifetime will exceed the cost of server itself [1]. A report of US Environmental Protection Agency (EPA) indicated that IT infrastructures in USA consumed about 61 billion kWh for a cost of 4.5 billion dollars in 2006 [2]. This electricity consumption is about 1.5% of the total US electricity consumption, and is more than double of that consumed by IT in 2000. It was also noticed in [3] that servers consume 0.5% of the world's electricity produced, and it will quadruple by 2020 if the trend continues. However, the utilization of a typical data center is around 20%–30% [4], which means a large amount of energy will be wasted.

Virtualization is an important technology typically adopted in cloud to consolidate the resources and support the pay-as-you-go service paradigm. It has been reported that virtual machines could be used for scientific applications with tolerable performance

* Corresponding author.
E-mail addresses: dingyouwei@nuaa.edu.cn (Y. Ding), qinxcs@nuaa.edu.cn (X. Qin), liangliu@nuaa.edu.cn (L. Liu), wangtc@nuaa.edu.cn (T. Wang).

http://dx.doi.org/10.1016/j.future.2015.02.001
0167-739X/© 2015 Elsevier B.V. All rights reserved.

punishment, and could provide desirable, on-demand computing environments for any users. Adapting virtualizations in cloud computing, the platform often provides various virtual machine templates, the jobs will be allocated with preconfigured virtual machines once they arrive at the cloud, and then the virtual machine is started at proper physical machines, finally it will be shut down if the job is finished.

Virtual machine scheduling is one of the most important and efficient technologies of reducing energy consumption in cloud. Beloglazov et al. [5] reported a survey of energy efficient data centers and cloud, they classified the research into hardware, operating system, virtualization and data center levels, and introduced the techniques used to save energy consumption for each level.

The main idea of scheduling VMs energy efficiently is placing them on only part of the physical machines and transforming the other ones into low power state (sleep or off). Since the service providers own all the details of the physical machines in the cloud and the resource requirements of VMs from the users, they can place VMs to proper physical machines to minimize the energy and thus maximize their profit. Existing scheduling methods mainly focus on minimizing the number of physical machines used to run the VMs. However, less attention is paid to energy saving technologies in the hardware level though the information of hardware in the cloud is known to the provider.

Dynamic voltage and frequency scaling (DVFS) is an efficient technology to reduce processor energy consumption. Manufacturers of processor have developed their patents on DVFS to make their products operate on several frequencies with different supply voltages. DVFS has been mainly used to achieve energy efficiency in embedded, multicore and multiprocessor systems. However, DVFS technology is rarely adopted in the virtualized cloud systems to save energy. Since DVFS technology is adopted for the processors, it is most efficient for computation-intensive VMs, but does not suit for I/O-intensive or network-intensive VMs. We only focus on the computation-intensive VMs in this paper.

Existing VMs scheduling methods using DVFS to reduce total energy are mostly developed in homogeneous clusters, and only the power of processors are measured. For example, a power-aware algorithm of VMs scheduling [6] was proposed to allocate the virtual machines in a DVFS-enabled cluster. They use as low as possible frequency for the processor to run the VMs, and all physical machines are homogeneous. The criterion of selecting physical machines for each VM is the power of the processors, which does not fit the practical cloud, because the processors only consume about 25% of the total energy of the server [7].

Example 1. Given two computation-intensive VMs vm_1 and vm_2, and two servers n_1 and n_2 with the same processors but different other components. Suppose the power of the processor is P_{cpu} on fixed frequency f, and the power of the other components of n_1 and n_2 are $P_{s1} = P_{cpu}$ and $P_{s2} = 0.5P_{cpu}$ respectively. The servers operate on frequency f unless they are powered down, and the execution time of vm_1 and vm_2 are t_1 and t_2. Obviously, there are four solutions for this case: (a) vm_1 for n_1 and vm_2 for n_2, (b) vm_1 for n_2 and vm_2 for n_1, (c) vm_1 and vm_2 for n_1, and n_2 is powered down, (d) vm_1 and vm_2 for n_2, and n_1 is powered down. Therefore the energy of n_1 and n_2 for processing the two VMs are $E_a = (2t_1 + 1.5t_2)P_{cpu}$, $E_b = (1.5t_1 + 2t_2)P_{cpu}$, $E_c = 2(t_1 + t_2)P_{cpu}$, $E_d = 1.5(t_1 + t_2)P_{cpu}$. It can be seen that solution d is the optimal one in terms of energy consumption. However, on the condition that the deadline is less than $t_1 + t_2$, solution a is the optimal one if $t_1 < t_2$, otherwise solution b is the optimal one.

We can see that heterogeneities and the total power of the physical machines are the main challenges of energy efficient scheduling in cloud computing, while the adoption of DVFS technology is another challenge for the scheduling of virtual machines.

This paper focuses on dynamic scheduling of virtual machines to achieve energy efficiency and satisfy deadline constraints in the cloud with heterogeneous physical machines. The main contributions of this paper are as follows. We conduct that there exists optimal frequency for a physical machine to process certain virtual machines, and each PM should operate on at least the optimal frequency. Then the notion of optimal performance–power ratio is defined to weight the heterogeneities of the physical machines, VMs will be allocated prior to the PMs with higher optimal performance–power ratio. The scheduling is divided into some equivalent periods, and the cloud will be reconfigured after each period to consolidate the computation resource to further reduce the energy consumption. Finally, the deadline constraint is maintained by the definition of required resource, VM can be completed on time as long as it is allocated successfully to a PM.

The rest of this paper is organized as follows: Section 2 reviews the existing research on energy efficient cloud computing and scheduling of virtual machines. Section 3 defines the problem and describes the power and virtual machine models used in this paper. In Section 4, we present an energy efficient algorithm to schedule virtual machines in cloud computing using DVFS technology. Section 5 shows the simulation results for the proposed algorithm, and Section 6 concludes the paper and points out future work.

2. Related works

Energy efficient scheduling of tasks in cloud is studied widely. Jacob Leverich et al. [8] proposed a strategy of selecting part of the physical machines in a Hadoop cluster to execute the tasks while powering down other ones to reduce the power consumption. However, Willis Lang et al. [9] indicated that using all physical machines to run the workloads and then powering down them simultaneously can save more energy. Both the two methods cannot work well if workloads are data-intensive, because powering down some physical machines will result in data unavailability while using all ones will cause frequent data migrations. A replication scheme named Chained Declustering was used in [10] to ensure data available when powering down partial physical machines in a cluster, and it also guaranteed load balance between the active machines. Since the replication scheme is the basis of powering down physical machines and load balance, it is not suitable for the deployed clusters. Considering the energy consumption during both execution time and idle periods, powering down fractional machines is an accepted method to make clusters energy efficient.

Virtualization is widely used in cloud computing to fully utilize the resources and improve the performance. Various VM scheduling methods [11–13] have been proposed to dynamically allocate and consolidate the VMs in cloud computing environment. The allocation algorithms can be mainly divided into two types, allocating VMs onto PMs and assigning PMs to VMs. The consolidation is typically achieved by VM migrations. Energy consumption was not considered in traditional VM scheduling in cloud computing.

Energy efficient VMs scheduling in data centers mainly focuses on fully utilizing each physical machine to reduce energy consumption. They take a physical machine as a whole and use the formula $P = P_{idle} + (P_{\max} - P_{idle})^{*}u$ to compute the power of each node, where u is the utilization of a physical node, P_{idle} and $P_{\max}$ are the idle and peak power of the node. It is an experimental formula, and has been tested in many data centers for a period of time. It is the simplest to estimate the total energy consumption of the data center using this formula, but may be not the optimal solution for the computation-intensive VMs.

It is reported that we can save energy via appropriate VM scheduling [14]. Energy efficient virtual machine scheduling is often viewed as an allocation or mapping problem, which is an optimization problem. It was abstracted to multi-dimensional space partition model in [15], based on which a VM placement algorithm EAGLE was proposed. But it needed two predefined parameters balance factor and satisfaction factor, which is hard for users without expert knowledge. Gergő Lovász et al. [16] modeled the energy-optimal VM allocation as a variant of multidimensional vector packing problem, and proposed a model to predict the performance of VM consolidation considering the tradeoff between power consumption and service performance. It was also abstracted as the combination of bin packing and quadratic assignment problems in [17] and a geedy algorithm was proposed to improve resource utilization and reduce the number of active PMs.

The issue can also be solved using different heuristics such as genetic, and simulated annealing. Nguyen et al. [18] proposed a genetic algorithm to power-aware allocate the VMs in a private cloud, which is a static algorithm. Xiaofei Liao et al. [19] proposed a heuristic method based on simulated annealing to dynamically remap VMs to a set of physical machines to form a green cluster. However, these methods require iterations for scheduling a VM and they cannot suit for the case of real-time applications.

Anton Beloglazov [20] set two utilization thresholds for each processor and defined a VM placement algorithm and three migration policies. Meanwhile, five open research challenges for energy efficient management of cloud computing environment were also presented. And a modification of [20] is proposed in [21], physical machines are placed in racks, which are coarser granularity of scheduling VMs. Nakku Kim et al. [22] revealed that billing users only based on processor time or the number of virtual machine instances in cloud is not sufficient, and proposed a model for estimating the energy consumption of each virtual machine using the in-processor events. And then a VM scheduling method is developed according to the energy budget. The thresholds are experimental values and may be different for various applications, hence it is hard for users to set the thresholds.

There are two main challenges for energy efficient VM scheduling in cloud environment, heterogeneous PMs and practical energy consumption of the PMs. The assumption of homogeneous PMs is often adopted in energy efficient VM scheduling in cloud computing, which is not actual in practice. The energy of processors is often used to replace the energy of PMs for scheduling of computation-intensive VMs, while low energy of processors does not mean low energy of physical machines especially for heterogeneous PMs.

In addition, modern hardware provides some opportunities to save energy. The most popular technology is DVFS, which is used in almost all processors nowadays. DVFS technology provides the finer grain of controlling the power of processors, which results in another way to reduce the energy consumption of the cloud. However, it is adopted in only few researches on energy efficient scheduling of VMs in cloud.

One of the first studies of energy efficient VMs scheduling using DVFS in clusters was proposed in [6]. Only the energy consumed by the processors is measured, which is not enough to reflect the energy efficiency of a cluster, because processors only consume about a quarter of total energy consumption, and the ratio will be even smaller when the operating frequency is low. Christine et al. [23] extended Xen's default credit scheduler to support DVFS scaling operations, which is a way of reducing power consumption in a virtualized cluster while guaranteeing the performance. But these researches do not pay enough attention to the heterogeneity of the physical machines.

There have been some researches on the energy cost of VM migration. Haikun Liu et al. [24] constructed two application-free models for live migration of VMs to estimate the performance and energy costs based on the knowledge learned from the history. Anja Strunk et al., [25] experimentally investigated that the factors affecting the energy consumption of virtual machine migration are the size of the virtual machine and the available network bandwidth. The performance overhead [26] and the policies [27] of VM migration were also studied, but they are not the keynotes in this paper.

In this paper we study how to schedule computation-intensive VMs in cloud to improve the energy efficiency. Physical machines in the cloud are heterogeneous, and the energy of all components of each PM is considered. Meanwhile, the DVFS technology is adopted in the scheduling to further reduce the energy consumption of PMs. We make an assumption in this paper that the energy and performance penalty are ignored.

3. System model

This section describes the models of the cloud and the virtual machines. These models are the basis of the scheduling algorithm in Section 4. We only deal with the computation-intensive virtual machines, hence all the components of an active physical machine except for processor can be viewed as a whole, and their power consumption keeps constant.

3.1. Power model

The power consumption of modern processors can be divided into two parts, dynamic power and static power, $P_{cpu} = P_{static} + P_{dynamic}$. The dynamic power $P_{dynamic} = aCV^2f$, where a is the switching activity, C is the physical capacitance, V is the supply voltage and f is the operating frequency. The values of switching activity and capacitance can be viewed as constants because they are only determined by the low-level system design. It is assumed that supply voltage V is in proportion to the operating frequency f, then we get $P_{dynamic} \propto f^3$. However, the dynamic power is not strictly in proportion to the cubic of the operating frequency in practice, so we assume $P_{dynamic} = \alpha f^h$, $(\alpha, h > 0)$ in this paper, and $P^f_{dynamic} = (f/f_{\max})^h P^{\max}_{dynamic}$, where $P^f_{dynamic}$, $P^{\max}_{dynamic}$ are the dynamic power of the processor operating on frequencies f and $f_{\max}$ respectively.

The static power, also called leakage power, is caused by leakage currents which are present in any active circuits. The static power is mainly determined by the type of transistors and the process technology. It is reported that the idle power of a processor may sometimes exceed 50% of the peak power, and the main part of the idle power is the static power. Therefore, we suppose that $P_{static} = \beta P^{\max}_{dynamic}$, $(0 < \beta \leq 1)$.

The total power of a PM is mainly consumed by the processor, memory, disk and other components. When processing the computation-intensive tasks, the power consumption of a PM is assumed to be $P = P_{cpu} + P_s$ in this paper, where P_{cpu} and P_s are the power of processor and all other components including memory, disk and so on. Additionally, we assume that P_s is fixed when the VMs are computation-intensive, while P_{cpu} is changed dynamically according to the number of active cores and their operating frequencies. For the simplicity of descriptions, we assume that $P_s = \gamma P^{\max}_{dynamic}$, $(\gamma > 0)$. Then we can get the total power and energy of a PM, $P = P_{dynamic} + P_{static} + P_s$, $E = (P_{dynamic} + P_{static} + P_s)t$, where t is the execution time for the PM to process the computation-intensive VMs. The execution time is affected by the computation of the VM and the operating frequency of the core, i.e. $t = w/f$, where w is the computation of the VM. So the energy of the PM will be $P = \left(\left(\frac{f}{f_{\max}}\right)^h + \beta + \gamma\right) P^{\max}_{dynamic} \frac{w}{f}$.

It is noticed that the processor or PM is active if it is used to process the VMs, and the cores on which VMs run are called active cores. The cores except the active ones of a processor are called inactive cores or idle cores, they will sleep until some VMs are allocated to them. The active cores are assumed to be 100% load, and it will sleep as soon as it finishes its workload. If the cores of a processor can be controlled independently, the dynamic power of an active core operating on frequency f is $P^{f}_{dynamic}/nc$, where nc is the number of cores of the processor, while that of any idle core will be 0. The static power of both active and idle cores is assumed to be P_{static}/nc unless they are powered down. Hence the power of an active processor cpu is $P_{cpu} = P_{static} + \frac{1}{nc}\sum_{i=1}^{nc} x(i)\, P^{f(i)}_{dynamic}$, where $P^{f(i)}_{dynamic}$ is the dynamic power of processor running on frequency $f(i)$, and $x(i) = 1$ if the ith core is active, otherwise $x(i) = 0$. The inactive processor will be powered down, and $P_{cpu} = P_{static} = P_{dynamic} = 0$.

The energy of the whole physical machine is

$$E = E_s + E_{cpu} = (\gamma + \beta)\, P^{\max}_{dynamic} \max_{i=1}^{nc}\{t_i\} + \frac{1}{nc} P^{\max}_{dynamic} \sum_{i=1}^{nc} x(i) \left(\frac{f}{f_{\max}}\right)^{h} t_i \qquad (1)$$

where t_i is the execution time of the ith core of processor cpu, and $t_i = 0$ if the core is idle. Obviously, the energy caused by P_s and P_{static} is determined by the maximum execution time of the cores, since the processor and other components should be activated as long as any core is active.

3.2. Physical machines model

In this paper we focus on the cloud with heterogeneous physical machines, and it is assumed that all PMs support DVFS and cores of any processor can be controlled independently. Each physical machine in the cloud, also called node, may be personal computers or servers, and typically contains several cores which can be operated on multiple frequencies. The set of heterogeneous PMs is $PM = \{pm_1, pm_2, \ldots, pm_n\}$, where n is the number of PMs. Each PM is divided into two parts, the processor and other components, and the power of the former P_{cpu} varies on different frequencies, while that of the latter P_s keeps stable.

The processor of each PM is defined as $cpu = (nc, ns, P_{static}, F, P)$, where nc and ns are the number of cores and the number of operating states of the processor, P_{static} is the static power of cpu, F and P are the sets of operating frequencies and dynamic power respectively. We assume that the nc cores of PM pm are homogeneous, so $F = \{f_1, f_2, \ldots, f_{ns}\}$, $P = \{p_1, p_2, \ldots, p_{ns}\}$, where f_j and p_j are the frequency and dynamic power of processor cpu_i's jth operating status. Without loss of generality, we suppose $f_1 < f_2 < \cdots < f_{ns}$ and $p_1 < p_2 < \cdots < p_{ns}$, and the minimal and maximal of the operating frequency and dynamic power are $f_{\min} = f_1, f_{\max} = f_{ns}$ and $p_{\min} = p_1, p_{\max} = p_{ns}$. And if the core sleeps, its frequency and dynamic power are set to zero.

The energy of the cloud consisting of n heterogeneous PMs when processing given tasks is consumed by all active PMs. The active PM will be powered down when the VMs allocated to it are finished, and the energy of each active PM can be computed according to Eq. (1). Therefore, the total energy consumption of the cloud for processing given tasks is

$$E = \sum_{i=1}^{n} y(i) \left((P_{i,s} + P_{i,static}) \max_{j=1}^{nc_i}\{t_{i,j}\} + \frac{1}{nc_i} \sum_{j=1}^{nc_i} x(i,j)\, P^{f(i,j)}_{i,dynamic} t_{i,j} \right) \qquad (2)$$

where $y(i) = 1$ if pm_i is active, otherwise $y(i) = 0$ $P_{i,static}$ is the static power of pm_i, $P^{f(i,j)}_{i,dynamic}$ and $t_{i,j}$ are the dynamic power and processing time of the jth core of pm_i's processor, $P_{i,s}$ is the power consumption of all components except processor of pm_i. The meaning of $x(i,j)$ in Eq. (2) is similar to that of $x(i)$ in Eq. (1), $x(i,j) = 1$ if the jth core of pm_i is active, otherwise $x(i,j) = 0$. For any given active PM pm_i, $f(i,j) \in F_i$ and $P^{f(i,j)}_{dynamic} \in P_i$, while no energy will be consumed by the inactive PMs.

3.3. Virtual machine model

We focus on computation-intensive VMs which arrive continuously to be processed on the cloud in this paper. The set of VMs is $VM = \{vm_1, vm_2, \ldots, vm_m\}$ and each virtual machine vm_i is assumed to be processed by a core at any time, i.e. vm_i cannot run on two or more cores at the same time. But VMs can be migrated from one core to another during the process. The execution time and power consumption of migrations of the virtual machine are ignored in this paper.

Each VM is defined as $vm_i = (w, at, d, st)$, where w is the computation of vm_i in terms of Million Instructions, at, st and d are the arriving time, actual starting time and the deadline of vm_i. The operating frequency can also be transformed by the unit of MIPS, hence the execution time can be computed by dividing the operating frequency of the core by the computation of the VM. When a virtual machine vm_i arrives, its w, at and d are pre-given, st is the time when vm_i is allocated successfully to a physical machine. We can see that vm_i should be finished between at and d, otherwise it will be marked as failed VM. The failed VMs will not be processed further to reduce the resource wastes.

Definition 1. The required resource rr of each VM is the minimal computation resource required to maintain that it can be finished successfully before its deadline. Required resource of a VM is the ratio of its computation uncompleted to the time period from now to the deadline.

The required resource of vm_i can be computed as

$$vm_i.rr = \frac{vm_i.w - vm_i.wf}{vm_i.d - t} \qquad (3)$$

where $vm_i.wf$ is the computation finished and t is the current time. Obviously, a VM vm_i cannot be assigned to the PM pm_j, if $vm_i.rr > pm_j.f_{\max}$. According to Eq. (3), the initial required resource of vm_i is $vm_i.rr = (vm_i.w - vm_i.wf)/(vm_i.d - vm_i.at)$. If vm_i cannot be allocated once arriving, it will be reallocated at time $vm_i.at + 1$ and its required resource should be updated $vm_i.rr' = (vm_i.w - vm_i.wf)/(vm_i.d - vm_i.at - 1)$. The required resource of vm_i will become larger if it still cannot be allocated at time $vm_i.at + 1$ or later.

If the required resource of vm_i is larger than the maximum frequency of the processors in cloud, vm_i will be marked as failed VM because it cannot be completed before the deadline. If vm_i is allocated to a physical machine pm at time t ($vm_i.at \le t < vm_i.d$), at least $vm_i.rr$ computation resource on pm will be used to run vm_i unless it is completed or migrated to other PMs. Therefore, vm_i can be completed successfully if it is allocated to a PM before its deadline.

The energy consumption of a PM for processing a VM is the energy consumed by the PM from the starting time to the deadline of the VM. As a PM may process many VMs simultaneously, and the VMs may be migrated to a PM to another, the energy consumption of the cloud to process given VMs is the energy consumed by all the PMs from the earliest arriving time to the latest finishing time of the VMs.

Since the power meters work periodically and scaling the power of the processors frequently may cause the overhead power consumption unnegligible, the scheduling can be divided into some equivalent schedule periods (each period is a second in this paper). Therefore, VMs will be scheduled at the beginning of each period and the energy of each physical machine will be the sum of energy consumed in all periods. As mentioned before, the energy and performance penalty of physical machines' state transition and task migration are ignored in this paper.

4. Energy efficient scheduling of virtual machines

4.1. Preliminaries

We describe the details of our scheduling algorithm based on aforementioned models. The cloud consists of many heterogeneous PMs, each of which supports DVFS and has different computation resources and power consumption. To schedule virtual machines efficiently in the cloud, we must solve the following two issues at first.

(1) Which PM and core in the cloud should be allocated to a VM?
(2) Which frequency does the selected core operate on to minimize total energy consumption?

As the heterogeneity of the PMs, each PM provides various computation capacities and power consumption. We give the notion of performance–power ratio to weight the physical machines for electing proper PMs to run the VMs.

Definition 2. Performance–power ratio, *ppr*, for a physical machine *pm* is the ratio of its computation capacity to the peak power, that is $pm.ppr = (pm.nc^{*}\, pm.f_{\max})/(pm.p_s + pm.p_{cpu})$, where $pm.p_{cpu}$ and $pm.p_s$ are the peak power of *pm*'s processor and that of other components.

Intuitively, the PM with higher performance–power ratio can provide more computation resource than that of lower *ppr* with certain power budget, and it consequently takes priority over the other ones when allocating VMs.

Given a VM vm with computation w and a PM *pm* with *nc* independent cores. We suppose vm is assigned to core c_j of *pm*, and c_j operates on frequency $f(j)$. Then the energy consumption of *pm* is $E = \left(P_s + P_{static} + P_{dynamic}^{f(j)}\right) t$. From Section 3.1, we can get

$$E = (\beta + \gamma) \frac{w}{\varphi f_{\max}} P_{dynamic}^{\max} + \frac{1}{nc} \frac{\varphi^{h-1}}{f_{\max}} w P_{dynamic}^{\max} \tag{4}$$

where $f(j) = \varphi f_{\max}$, $(0 < \varphi \le 1)$ is the operating frequency of core c_j, and $t = w/f(j)$ is the execution time of c_j.

The variable in Eq. (4) is φ, and we compute the derivative of Eq. (4) and set it to be 0, then we get the optimal value $\varphi^{*} = \sqrt[h]{nc\,(\beta+\gamma)/(h-1)}$, and the optimal frequency $f(j)$ can be obtained.

If q VMs $(vm_1, vm_2, \ldots, vm_q)$ are allocated to a PM, there are nc^{q} ways to select active cores, and much more ways to run considering the operating frequencies of each active core. We can see from Eq. (1) that the difference of each core's execution time should be minimized to reduce energy waste caused by the sleep cores. Meanwhile, the VMs should be first assigned to the inactive cores of the PM rather than active ones to reduce the energy overhead since P_s and P_{static} are stable for any active PM.

There are two novel ways to reduce energy consumption, minimizing the difference of execution time of all active cores, and minimizing the imbalance of the workload of active cores. In the former way, states of active cores should be changed frequently which will lead to much energy and time overhead. And sometimes the difference of execution time cannot be reduced because of the discrete frequency of the processor and deadline constraint of the VMs. In this paper we choose the latter method, minimizing the imbalance of workload of active cores, in which all active cores operate on the same frequency to further minimize the difference of their execution time.

Suppose $W = vm_1.w + vm_2.w + \cdots + vm_q.w$, k cores are used to process the q virtual machines, and each core has the same computation ideally, then the execution time of each active core is $W/(kf)$ and the energy consumption of the PM will be

$$E(k, f) = (\beta + \gamma) P_{dynamic}^{\max} \frac{W}{k \varphi f_{\max}} + \frac{W}{nc} \frac{\varphi^{h-1}}{f_{\max}} P_{dynamic}^{\max}. \tag{5}$$

Obviously, Eq. (5) is a monotonically decreasing function on variable k. Therefore, the more cores used for a processor, the less energy is consumed by the PM, because the power of other components and static power of the processor are shared by more cores. This is consistent with the aforementioned analysis. Computing the derivative of Eq. (5) about variable φ and setting it to be 0, we can get the optimal frequency for the active cores

$$\varphi^{*} = \sqrt[h]{nc\,(\beta+\gamma)/(k(h-1))}. \tag{6}$$

Specifically, $\varphi^{*} = \sqrt[h]{(\beta+\gamma)/(h-1)}$ when all cores are used.

Since the operating frequency of the processor must be in interval $[f_{\min}, f_{\max}]$, that is $f_{\min}/f_{\max} \le \varphi \le 1$. Then the theoretical optimal frequency of the active cores should be

$$f^{*} = \begin{cases} f_{\max}, & \varphi^{*} \ge 1 \\ f_{\min}, & \varphi^{*} \le f_{\min}/f_{\max} \\ \varphi^{*} f_{\max}, & \text{otherwise.} \end{cases} \tag{7}$$

Unfortunately, the operating frequency for the processor is discrete, $f \in \{f_1, f_2, \ldots, f_{ns}\}$, thus the theoretical optimal frequency f^{*} will not always be the operating status. Finally we get the practical optimal frequency that active cores should operate on

$$f_{opt} = \begin{cases} f_i, & f^{*} = f_i,\ 1 \le i \le ns \\ \arg\min\{E(f_i), E(f_{i+1})\}, & f_i < f^{*} < f_{i+1},\ 1 \le i < ns \end{cases} \tag{8}$$

where $f_i \in pm.F$, $f_{i+1} \in pm.F$, $E(f_i)$ and $E(f_{i+1})$ are the energy consumption of *pm* when it operates on frequency f_i and f_{i+1} and can be computed using Eqs. (4) or (5). In the rest of this paper, f^{*} is referred to as theoretical optimal frequency while f_{opt} is called practical optimal frequency or optimal frequency for short.

In practice, the workload of each active core may not balance as VMs arrive continuously and any VM cannot run on more than one active core at the same time. Therefore, we should try best to balance the workload of active cores of the PM at each schedule period. If not all cores of a PM are active, the idle cores should be first assigned to the VMs, otherwise the active core with minimal workload is chosen to run the VMs. Any active core of the PM should operate on the optimal frequency computed by Eq. (8) if the VMs can be completed before corresponding deadlines. Performance–power ratio cannot be used as the weight of heterogeneous PMs, because it has no relationship with the optimal frequency. The valid weight must reveal the performance–power ratio of the PMs operating on optimal frequency.

Definition 3. Optimal performance–power ratio, *oppr*, for a physical machine *pm* is the ratio of the computation resource to the power of *pm* when all cores are operating on optimal frequency. It can be computed as

$$pm.oppr = \frac{pm.nc^{*}\, f_{opt}}{pm.P_s + pm.P_{static} + pm.P_{dynamic}^{f_{opt}}} \tag{9}$$

where f_{opt} is computed by Eqs. (7)–(8) using $\varphi^{*} = \sqrt[h]{(\beta+\gamma)/(h-1)}$, that is all cores of *pm* are active.

In some previous researches, only the dynamic power of processor is considered, that is $E = P_{dynamic}t$. A well-known conclusion that processor running on lower frequency consumes less energy can be deduced in that case. This conclusion violates the result of Eqs. (6)–(8), because the dynamic and static power of processor and the power of other components are taken into account in this paper.

Therefore, three conclusions can be obtained from the aforementioned analysis, which is the basis of our scheduling algorithm described in Sections 4.2–4.5.

(1) The PM with higher optimal performance–power ratio is allocated prior to VMs.
(2) VMs are prior placed on the idle cores of active PMs.
(3) There exists optimal frequency for each PM to process the VMs to minimize the total energy consumption.

4.2. Scheduling algorithm

Since the VMs arrive continuously, they are scheduled once arriving. Fig. 1 shows the framework of scheduling at the tth period. The scheduler first picks the VMs that failed to allocate before t (vm_{l1} and vm_{l2}) and the ones that arrive currently (vm_{a1}, vm_{a2} and vm_{a3}), and places each of them to a proper PM, and further a core of the PM. The PMs are sorted with decreasing optimal performance–power ratio in advance ($pm_{a1}.oppr \geq pm_{a2}.oppr \geq pm_{s1}.oppr \geq pm_{s2}.oppr$), and the one with higher *oppr* is prior to be assigned to the virtual machines. If all the active physical machines (pm_{a1} and pm_{a2}) are not suitable for the virtual machine, some sleep ones (pm_{s1} and pm_{s2}) must be activated. The scheduling repeats until all VMs are finished or failed.

At each period, the scheduling can be divided into three phases: allocating VMs, updating VMs, and reconfiguring the cloud. In the first phase, we place each VM waiting for being scheduled in this period onto the proper PM and core. Then an optimal frequency should be set for each active core and update the information of both VMs and PMs in the second phase. Finally the cloud should be reconfigured to consolidate the computation resource to minimize the energy consumption. DVFS technology is adopted to set optimal frequency of each physical machine. The details of these three phases are shown in Sections 4.3–4.5. It is noticed that all the PMs should be sorted with decreasing *oppr* before the scheduling. This is supported by the three conclusions in Section 4.1.

Algorithm 1. EEVS

Input: set of physical machines *PM*, set of virtual machines *VM*
Output: schedule of *VM*, energy consumption, processing time

```
1 sort $PM$ with decreasing $oppr$;
2 foreach period $t$ do
    // VMAllocation, shown in Algorithm 2
3       allocate each VM in $VM(t)$;
    // VMProcess, shown in Algorithm 3
4       set frequency of each active PM;
5       update the information of active PMs and running VMs;
    // Reconfiguration, shown in Algorithm 4
6       reconfigure the cloud;
7       if all VMs are finished or failed then
8       return $Schedule$, $E_{total}$, $T_{total}$;
```

Our energy efficient VM scheduling algorithm for the cloud, EEVS, is shown in Algorithm 1. The idea of EEVS is as follows. The set of physical machines is sorted with optimal performance–power ratio decreasing in advance, thus VMs will be prior allocated to the PM with higher *oppr*. In each period t, VMs arriving at t and left in the periods before t will be placed onto proper cores of the PMs. When all VMs are allocated, we set optimal frequency of each active core and update the information of all running VMs and active PMs because the extra computation resource of the cores are also used to process VMs and some VMs may be finished. Reconfiguration is needed to consolidate the computation resources of the PMs to save energy. The process of period t+1 will continue if any VM is under processing or new VM arrives, otherwise, the scheduling is terminated. The descriptions of the variables used in Algorithm 1–4 are shown in Table 1. Transforming the operating states (i.e. frequencies) of each core is achieved by the DVFS-supported processors of all physical machines.

The total energy consumption of the cloud is the accumulation of the energy of all PMs at each period, so we get $E_{total} = \sum_{t=0}^{T} E_t = \sum_{t=0}^{T}\sum_{i=0}^{n} E_{i,t} = \sum_{t=0}^{T}\sum_{i=0}^{n}(E_{i,t}(cpu) + E_{i,t}(s))$, where E_t and $E_{i,t}$ are the energy consumptions of the cluster and the ith PM at the tth period, $E_{i,t}(cpu)$ and $E_{i,t}(s)$ are the energy consumed by the processor and other components of the ith PM at the tth period.

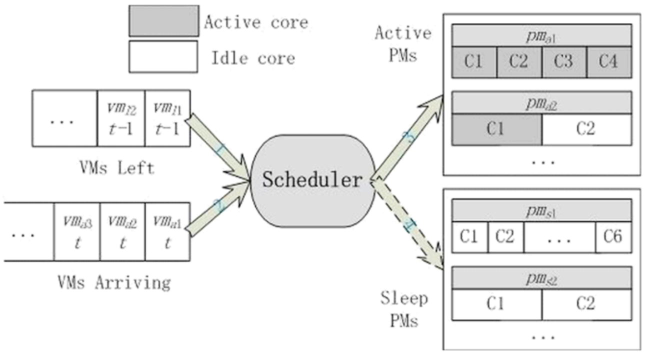


Fig. 1. The framework of the scheduling.

Table 1
Variables and their meanings used in Algorithm 1–4.

Variable	Meaning
$VM(t)$	The set of VMs need to be allocated at period t
E_{total}	Total energy of the cluster for processing all VMs
T_{total}	Execution time of the cluster for processing all VMs
AllocationList	The set of allocations of all VMs, in form of (vm, pm, $core$)
Schedule	The schedule of the VMs, in form of (vm, pm, $core$, t)
PMIoppr	The set of active PMs with increasing *oppr*
$core.w$	Sum of required resource of VMs allocated to *core* of PM
$pm.w$	Sum of required resource of VMs allocated to PM *pm*
$core.f$	The optimal frequency of *core* for processing $core.w$

4.3. VM allocation

The phase of VM allocation, as shown in Algorithm 2, is to find a proper physical machine and core for each VM in $VM(t)$. According to the conclusions in Section 4.1, VMs are allocated prior to the PMs with higher optimal performance–power ratio and their idle cores.

Since the PMs are sorted with decreasing *oppr*, we traverse the set *PM* and the first PM *pm* which has enough resource is just the proper one for a given VM. If there exist one or more idle cores on *pm*, one of the idle cores will be assigned to the VM, as shown in steps 3–4 of Algorithm 2. Otherwise, a core with enough computation resource on *pm* will be assigned to the VM, as shown in steps 6–8. When a VM is allocated successfully, its starting time is set to be current time t and its required resource should be added to the workload of the corresponding core and PM, as shown in steps 9–11.

Algorithm 2. VMAllocation

```
1 foreach $vm$ in $VM(t)$ do
2     foreach $pm$ in $PM$ do
3         if $pm$ has idle cores then
4             an idle $core$ is allocate for $vm$;
5         else
6             foreach $core$ of $pm$ do
7                 if $core$ has enough resource for $vm$ then
8                     $core$ is assigned to $vm$;
9         $AllocationList$.add$(vm, pm, core)$;
10         $vm.st = t$;
11         $core.w += vm.rr$;
12     if $vm$ is not allocated successfully then
13         update $vm.rr$;
14         insert $vm$ into the head of $VM(t + 1)$;
15 return $AllocationList$;
```

As to the VMs that failed to be allocated at period t, they will be left for scheduling in next period $t + 1$. And these VMs should be first scheduled in the $(t + 1)$th period to avoid that the required resource becomes too larger to exceed the maximum frequency of the PMs, as shown in steps 12–14.

4.4. VM processing

After all VMs are allocated to the proper cores on the PMs or fail to be allocated, we should set optimal frequency for each active core to run the VMs energy efficiently. This phase includes two sub-phases, as shown in Algorithm 3, setting optimal frequency for active cores (steps 15–22) and updating information of active PMs and running VMs (steps 5–13).

For each active PM, the number of active cores is used to compute the theoretical optimal frequency according to Eqs. (4) and (5). And the practical optimal frequency f_{opt} can be obtained using Eq. (6). If the required computation resource of the VMs on the core is larger than f_{opt}, the optimal frequency of this core is set to be the minimal frequency that is no less than the required computation resources, i.e. $core.f = \min\{f_i | f_i > core.w \,\&\, f_i \in pm.F\}$. Otherwise, the optimal frequency of $core$ will be f_{opt}.

Algorithm 3. VMProcess

```
1 foreach $pm$ in $PM$ do
2     foreach $core$ in $pm$ do
3         if $core$ is active then
4             **SetOptFrequency**$(core)$;
5 foreach $(vm, pm, core)$ in $AllocationList$ do
6     if $vm$ is completed in this period then
7         $core.w -= vm.rr$;
8         $pm.w -= rm.rr$;
9         if $core.w = 0$ then
10             translate core into sleep state;
11         if $pm.w = 0$ then
12             translate $pm$ into sleep state;
13         $AllocationList$.delete$(vm, pm, core)$;
14 return $AllocationList$;
**SetOptFrequency**$(pm, core)$
15 foreach $pm$ in $PM$ do
16     count active cores of $pm$;
17     compute $f_{opt}$ for $pm$ using Eqs. (4)–(6);
18     foreach active $core$ in $pm$ do
19         if $core.w \geq f_{opt}$ then
20             $core.f = \min\{f_i | f_i > \text{core.w} \,\&\, f_i \in pm.F\}$;
21         else $core.f = f_{opt}$;
22 return $core.f$;
```

Then all active cores operate on their optimal frequencies in any period, and each VM vm occupies the core for a sub-period of $vm.rr/core.w$ of the period. All the VMs on each active core should be updated after the period, including their computation finished and required resource. If some VMs are completed successfully, they must be removed from $AllocationList$. The core and PM will be transformed into sleep state if there are no VMs on it, as shown in steps 6–13.

4.5. Cluster reconfiguration

In the phase of cluster reconfiguration, the VMs will be migrated to some PMs with higher optimal performance–power ratio to save energy, as shown in Algorithm 4. The VMs on the physical machine with lowest $oppr$ will be migrated first.

Algorithm 4. Reconfiguration

```
1 foreach $pm$ in $PMIoppr$ do
2     foreach $core$ in $pm$ do
3         for each $vm$ on $core$ do
4             $(pm', core') =$ **MigrateVM**$(vm, pm)$;
5             if $pm \neq pm'$ then
6                 update: $core.w -= vm.rr$, $pm.w -= vm.rr$;
7                 update: $core'.w += vm.rr$, $pm'.w += vm.rr$;
8                 $AllocationList$.delete$(vm, pm, core)$;
9                 $AllocationList$.add$(vm, pm', core')$;
10 return $AllocationList$;
**MigrateVM**$(vm, pm)$
11 foreach $pm'$ with $ppr$ higher than $pm$ in $PM$ do
12     foreach $core'$ in $pm$ do
13         if $core'$ has enough resource for $vm$ then
14             return $(pm', core')$;
15 return $(vm, pm)$;
```

We traverse $PMIoppr$ sequentially to check whether the VMs on each PM pm can be migrated to another PM pm' with higher $oppr$. If a core of pm' has enough resource for a VM vm on pm, then vm can be migrated to the core $core'$ on pm'. Then the sum of required resource of pm, pm', $core$ and $core'$ should be updated, meanwhile the $AllocationList$ should be also updated, as shown in steps 6–9 in Algorithm 4.

5. Performance evaluation

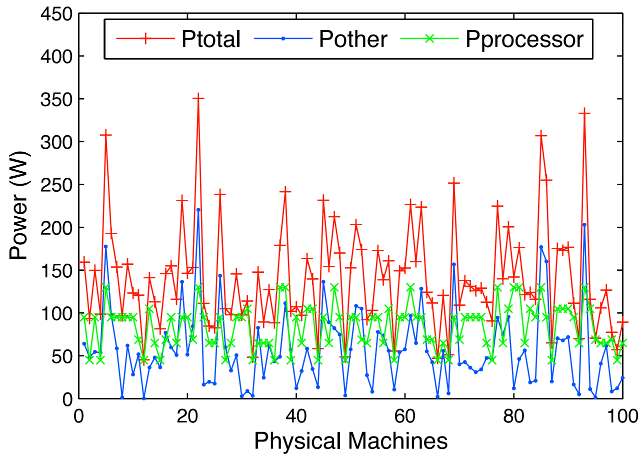
5.1. Simulation setup

We use simulation test to evaluate our proposed algorithms. In our simulated cloud environment, the processors of the PMs are shown in Table 2. We use four PC processors and four server processors to show the heterogeneity of the PMs. The idle power and peak power are the power of the PM running at status of 100% load and idle, the information are referred from tom's hardware website [28] and SPEC result website [29]. Other information is got from Intel and AMD website, CPU power is the max power of the processor. The power of components except processor of each PM $P_s = PeakPower - CPUPower$, and the static and max dynamic power of the processor are $P_{static} = IdlePower - P_s$, $P_{dynamic} = CPUPower - P_{static}$. Then parameters β and γ can be easily computed. We generate PMs using the processor shown in Table 2, P_s of each PM is randomly simulated with $\gamma \in (0, 2)$. The dynamic power of each processor is assumed to be a cubic function of frequency, i.e. $P^f_{dynamic} = (f/f_{\max})^3 P^{\max}_{dynamic}$.

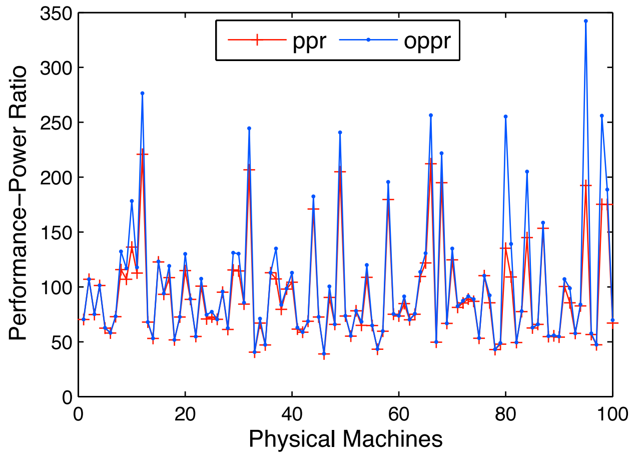
Each schedule period is assumed to be 1 s, the tasks arrive at the beginning of each period. All the tasks arrive between 0 and 100 s,

Table 2
The processors of PMs in the cloud.

PM	Processor	Number of cores	Max frequency (GHz)	CPU power (W)	Idle power (W)	Peak power (W)
1	AMD Athlon II X2 250	2	3.0	65	56	101
2	Intel Core i5-2300	4	2.8	95	45	112
3	Intel Core i7-970	6	3.2	130	79	198
4	AMD Athlon II X6 1055T	6	2.8	95	79	164
5	Intel Xeon X3220	4	2.4	105	79.8	132
6	Intel Xeon E3110	2	3.0	65	75.2	117
7	Intel Xeon E3-1265Lv3	4	2.5	45	19	58.5
8	Intel Xeon E3-1240V2	4	3.4	69	14.1	72.5



(a) Power consumption of 100 PMs.



(b) Performance–power ratio of 100 PMs.

Fig. 2. Power and performance–power ratio of 100 simulated PMs.

and their durations differ from 1 to 100 s. Since only computation-intensive workload is considered and the computation of each VM is in terms of Million Instructions in this paper, each task is the accumulation of float with different sizes. The accumulation can be transformed into instructions easily and is the basis of many scientific computations. In some extent the size of the accumulation can be viewed as the computation of each task, and the deadline is the bearable waiting time. A VM will be created for each task, the VM has more computation resource than the required resource of the task. Each VM cannot run on more than one core at the same time, but it can be migrated from one core to another. The VM that cannot be completed before deadline is marked as failed VM.

Fig. 2 shows the information of 100 simulated PMs. Fig. 2(a) reveals the power consumption of the PMs while Fig. 2(b) gives the performance–power ratio and optimal performance–power ratio of each PM. The legends 'Ptotal', 'Pcpu', and 'Ps' are the total power, the power of processor and power of other components of each PM respectively. It can be seen from Fig. 2(a) that the PMs with the same power of processor typically have different total power. We can also see in Fig. 2(b) that optimal performance–power ratio is larger than the performance–power ratio in most cases, and the optimal performance–power ratio of each PM has uncertain relationship with the performance–power ratio.

Fig. 3 reflects the information of 100 simulated VMs. Fig. 3(a) shows the arriving time and deadline of each VM, and the VMs are ordered with increasing arriving time. The required resource of each VM is shown in Fig. 3(b).

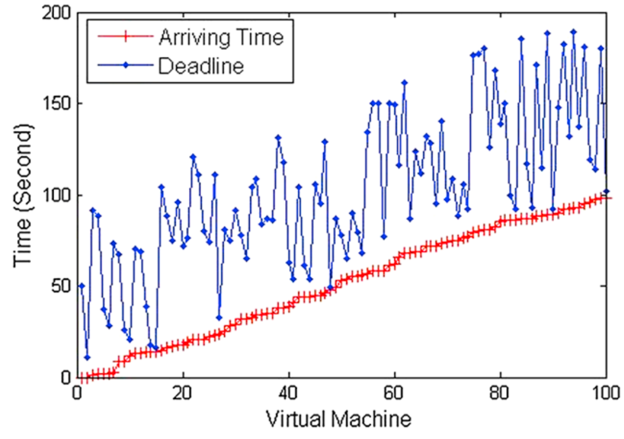
We implement our proposed algorithm EEVS, a naïve algorithm without sorting the physical machines named as EEVS-N, the modification of algorithm proposed in [6] named as Homogeneous, and the MBFD algorithm proposed in [20]. The only difference between EEVS and EEVS-N is that the latter does not sort the set of PMs, so EEVS-N randomly chooses a PM to run the VM. Scheduling in [6] is modified to suit for heterogeneous PMs here. And random selection for VM migration is used in MBFD. We compare the total energy consumption, processing time, failure of virtual machines and the number of active physical machines of the four algorithms. We run 100 to 5000 virtual machines on the cloud with the number of heterogeneous physical machines from 100 to 500.

5.2. Simulation results

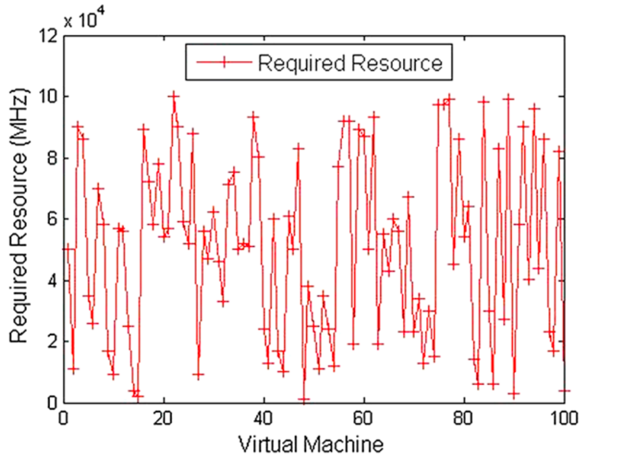
5.2.1. Energy consumption

Fig. 4 shows the total energy consumption of 100 PMs for light (100–1000 VMs) and heavy (1000–5000 VMs) load. From Fig. 4(a) we can see that the energy consumption of all algorithms are nearly proportional to the number of PMs, but the slope of EEVS is the least, that of EEVS-N is the largest. Since more PMs must be activated for processing more VMs, and EEVS chooses the PMs with higher optimal performance–power ratio, EEVS costs the least energy for certain VMs. For example the energy consumption of EEVS, Homogeneous and MBFD are 22 430.1, 24 614.1 and 19 874.9 J for the case of 100 VMs, while they consume 298 598, 351 612, 342 668 J for processing 800 VMs. EEVS consumes 14.8% less energy than MBFD for the case of 800 VMs. It can be seen from Fig. 4(b) that the proportionalities of all algorithms are destroyed, and the energy consumption of Homogeneous is the least when the number of VMs is no less than 3000. The main reason is that they cannot process all the VMs for heavy load, in other words, only part of the VMs are completed on time. And Homogeneous completed the least VMs, the details are shown in Fig. 8.

Fig. 5 reveals the energy consumption of 500 PMs for light and heavy load. Since 500 PMs can complete 5000 VMs on time for all the algorithms, the energy consumption of EEVS, EEVS-N, Homogeneous and MBFD are linear to the number of VMs. EEVS has the least slope, Homogeneous has the largest slope, and the slope of MBFD is also larger than that of EEVS-N and Homogeneous. For

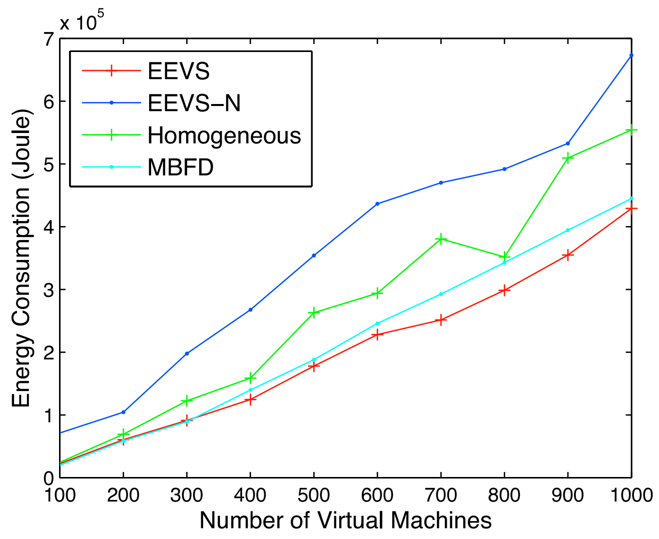


(a) Arriving time and deadline of 100 VMs.

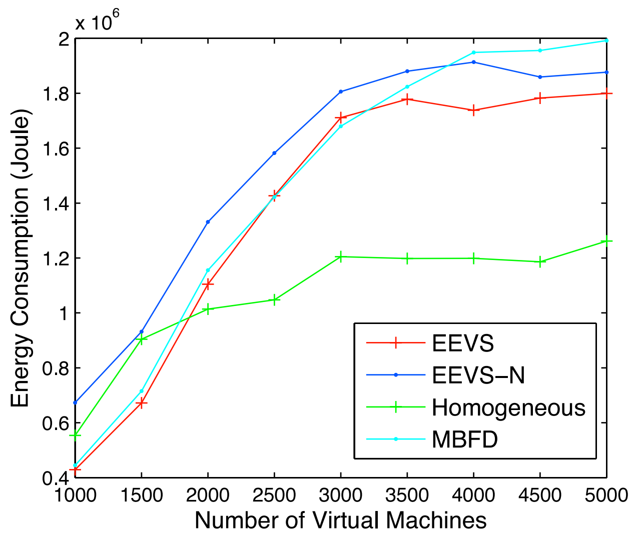


(b) Required resource of 100 VMs.

Fig. 3. Information of 100 simulated VMs.

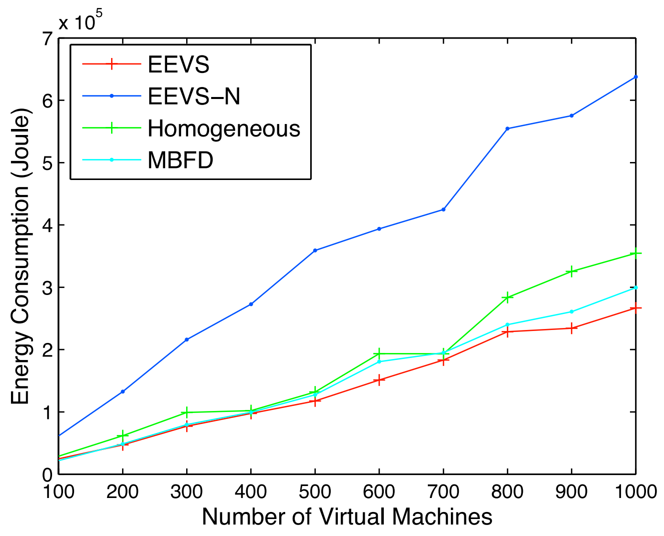


(a) 100 physical machines for light load.

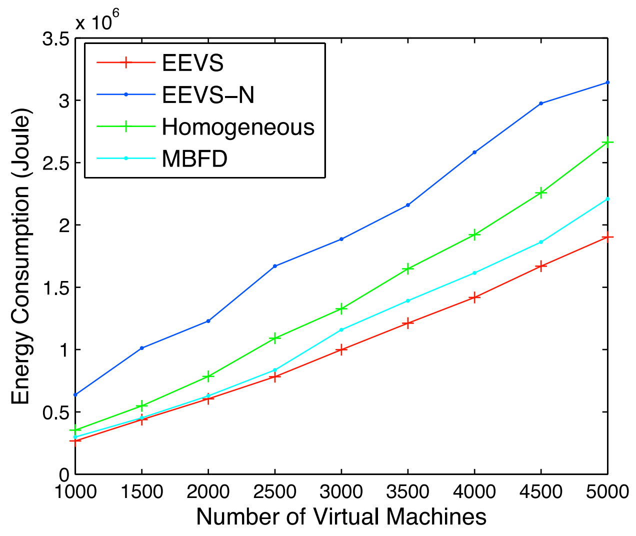


(b) 100 physical machines for heavy load.

Fig. 4. Total energy consumption of the cloud with 100 PMs.



(a) 500 physical machines for light load.



(b) 500 physical machines for heavy load.

Fig. 5. Total energy consumption of the cloud with 500 PMs.

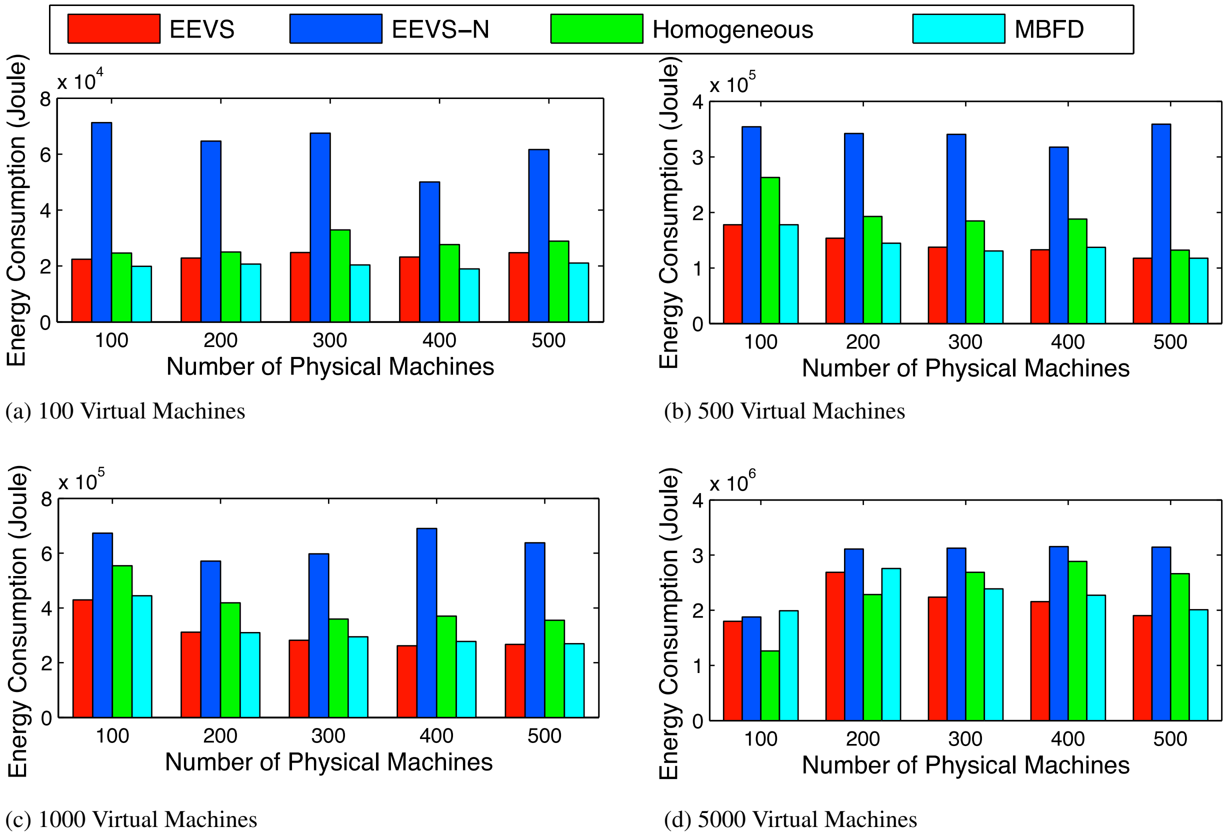


Fig. 6. Energy consumption of the cloud with different number of PMs when processing 100, 500, 1000 and 5000 VMs.

example, the energy consumptions of EEVS, EEVS-N, Homogeneous and MBFD for the cases of 100, 1000 and 5000 VMs are (24 758.5 J, 61 634.2 J, 28 922.1 J, 22 046.2 J), (266 717 J, 637 659 J, 354 657 J, 299 515 J) and (1903 450 J, 3143 500 J, 2663 800 J, 2208 540 J) respectively. EEVS consumes 24.8% less energy than Homogeneous, and 11% less than MBFD for processing 1000 VMs. The reason that MBFD has the largest slope of the algorithms except for EEVS is that MBFD views the processor as a whole, and the computation resource can be highly used while the others view core as the scheduling unit thus some computation resource on each core may be 'fragments' and cannot be used.

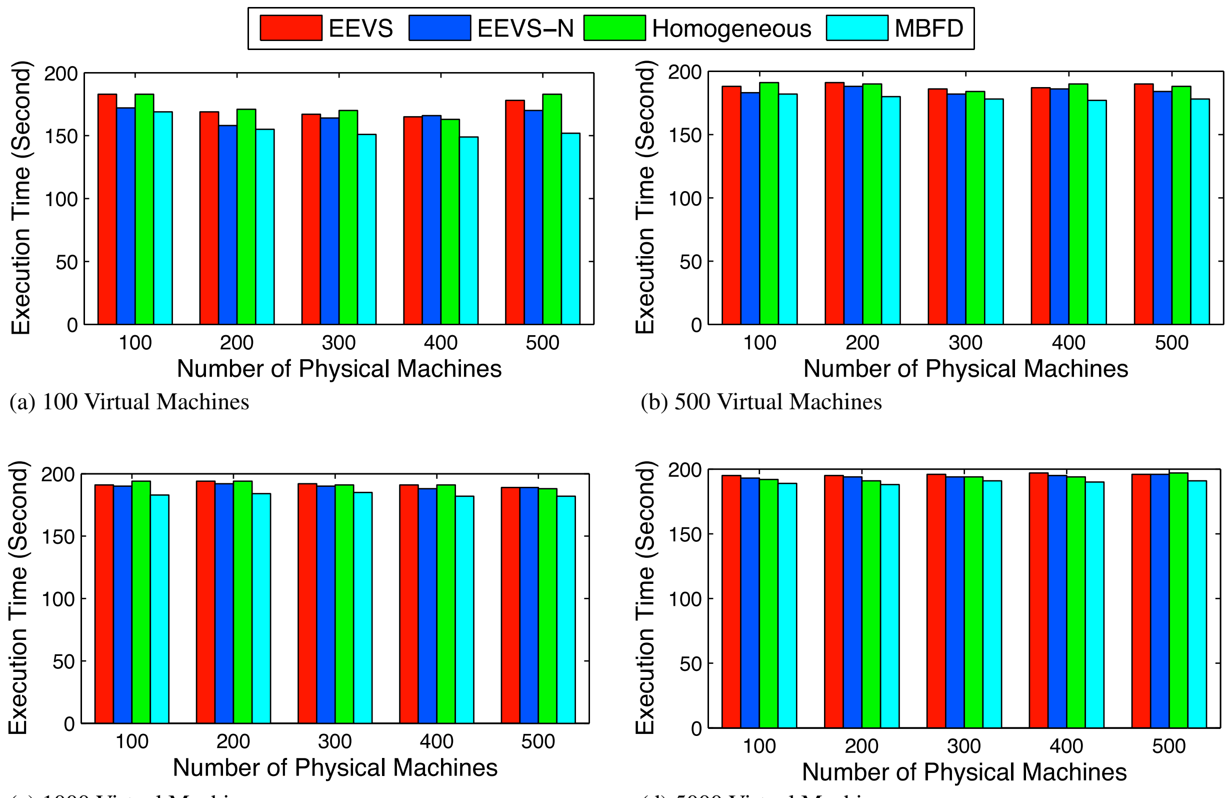
Fig. 6 reflects the energy consumption of the cloud with different number of PMs (from 100 to 500 PMs) for running certain VMs (100, 500, 1000 and 5000 VMs). It is obvious that the energy of EEVS is the least for all the cases except for 100 PMs and 5000 VMs, while EEVS-N consumes the most energy for all the cases except for 300 PMs and 5000 VMs. The energy of EEVS, Homogeneous and MBFD decreases with more physical machines in most cases when processing certain VMs and all of them can be finished on time. While EEVS-N consumes nearly the same energy for processing certain VMs using different number of PMs when all VMs can be completed successfully, because EEVS-N randomly chooses the physical machines to execute the VMs. For example, the energy consumptions of EEVS, Homogeneous and MBFD in Fig. 6(c) are (429 050 J, 311 664 J, 281 639 J, 261 592 J, 266 717 J), (554 127 J, 418 483 J, 359 566 J, 370 210 J, 354 657 J), and (444 722 J, 309 475 J, 294 379 J, 287 208 J, 299 515 J) respectively, while that of EEVS-N are (673 079 J, 571 225 J, 597 566 J, 690 297 J, 637 659 J). It can also be seen from Fig. 6 that the MBFD gets more profit than Homogeneous in terms of energy consumption when the multiple of the number of VMs and PMs becomes larger, because MBFD utilizes the PMs more efficiently.

5.2.2. Execution time

The execution time of the cloud for process given tasks is the time period from the earliest arriving time to the latest finishing time of the tasks. Fig. 7 depicts the execution time of the cloud with different number of PMs (from 100 to 500 PMs) for running certain VMs (100, 500, 1000 and 5000 VMs). A few conclusions can be obtained form Fig. 7. First of all, all algorithms spend more time to process more VMs for certain PMs. For example, the execution time of EEVS, EEVS-N, Homogeneous and MBFD for 100 PMs in Fig. 7(a)–(d) are (183 s, 172 s, 183 s, 169 s), (188 s, 183 s, 191 s, 182s), (191 s, 190 s, 149 s, 183 s) and (195 s, 193 s, 192 s, 189 s) respectively. The second conclusion is that MBFD spends the least time in all the cases, because all active PMs operate on the maximum frequency in MBFD, while EEVS and EEVS-N use the optimal frequency and Homogeneous uses the minimum frequency which should be no less than the required resource of the VMs. Finally we can see that the execution time of each algorithm for certain VMs is nearly the same when all VMs can be completed on time. The execution time is mainly determined by the VM which arrives latest and the one with latest deadline if there is enough computation resource. For example, the execution time for EEVS, EEVS-N, Homogeneous and MBFD in Fig. 7(c) are (191 s, 194 s, 192 s, 191 s, 189 s), (190 s, 192 s, 190 s, 188 s, 189 s), (194 s, 194 s, 191 s, 191 s, 188 s) and (183 s, 184 s, 185 s, 182 s, 182 s) respectively.

5.2.3. The number of failed VMs

The number of failed VMs for all algorithms is shown in Fig. 8. Since the cloud with 200 or more PMs can process all the VMs, we only show the case of 100 PMs. We can first see that Homogeneous cannot deal with the case of 2000 VMs, while the others failed in the case of 3000 VMs. Since Homogeneous operates on the minimal frequency which is no less than the required resource, some VMs may always wait for being allocated and finally cannot be



(a) 100 Virtual Machines (b) 500 Virtual Machines

(c) 1000 Virtual Machines (d) 5000 Virtual Machines

Fig. 7. Execution time of the cloud with different number of PMs when processing 100, 500, 1000 and 5000 VMs.

scheduled as the increasing required resource. Since MBFD always chooses the PM which may consume the least energy for each VM, some VMs with larger required resource may be failed to allocate as no PM can provide the enough computation resource for the VMs for the case of heavy load. Therefore MBFD leads to the most failed VMs. EEVS and EEVS-N lead to less failed VMs than Homogeneous and MBFD, because they consolidate the computation resources of the PMs after each period and PMs operate on the optimal frequency. The number of failed VMs of EEVS, EEVS-N, Homogeneous and MBFD for 4000, 4500 and 5000 VMs are (387, 432, 719, 1459), (775, 809, 896, 2073) and (731, 958, 1167, 2205) respectively. EEVS completes 10.1%, 3.9% and 15.4% more VMs successfully than Homogeneous for processing 4000, 45 000 and 5000 VMs.

The number of failed VMs is the criteria of performance. Each failed VM means that a task cannot be finished before deadline, and maybe the user gives up this cloud service. Hence reducing energy consumption should not lead to much performance penalty. Our EEVS algorithm satisfies this rule according to the results shown in this section.

5.2.4. *Active PMs*

Fig. 9 shows the number of active PMs when scheduling 500 and 5000 VMs on the cloud of 100 PMs. We can see that EEVS uses the least PMs (13 PMs at most) while MBFD uses the most PMs (22 PMs at most) most of the times in Fig. 9(a). That is because EEVS fully utilizes the computation resources of each active PM, while MBFD greedily chooses PM consuming the least energy for each VM and thus more PMs should be activated. The number of active PMs of EEVS-N and Homogeneous are both between that of EEVS and MBFD, because they randomly choose the PMs. Since all VMs arrive before 100 s, more PMs should be activated to process the increasing VMs from the beginning of the scheduling until the latest VM arrives. After that no sleep PMs will be activated for all algorithms after 100 s for the case of light load. In Fig. 9(b), MBFD

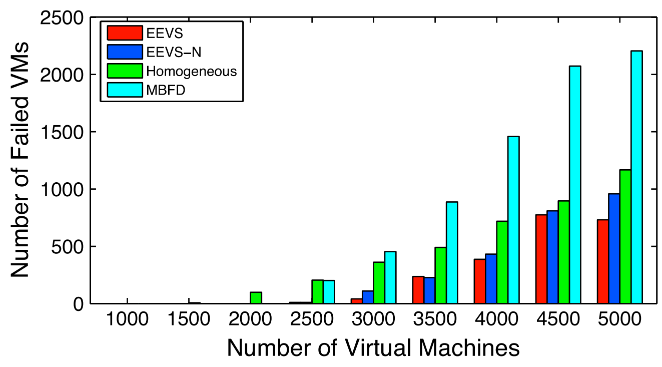


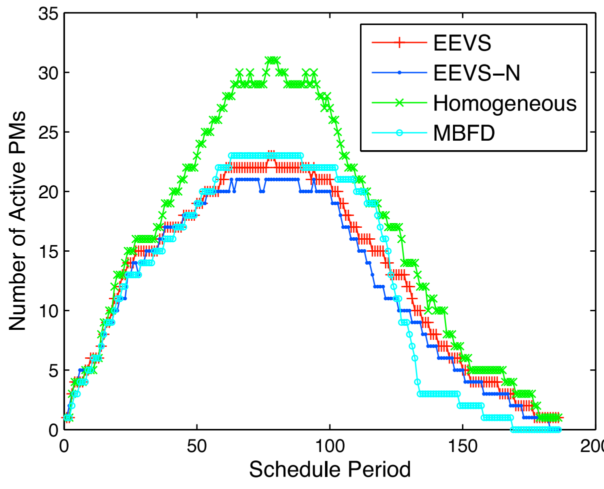
Fig. 8. Number of failed VMs on the cloud with 100 PMs.

activates more PMs than others most of the times. The results of EEVS and EEVS-N are nearly the same, because the heavy load makes full use of all PMs. Homogeneous uses less active PMs than the others after all PMs are activated, because the first failed VMs in Homogeneous appear earlier than that of EEVS, EEVS-N and MBFD.

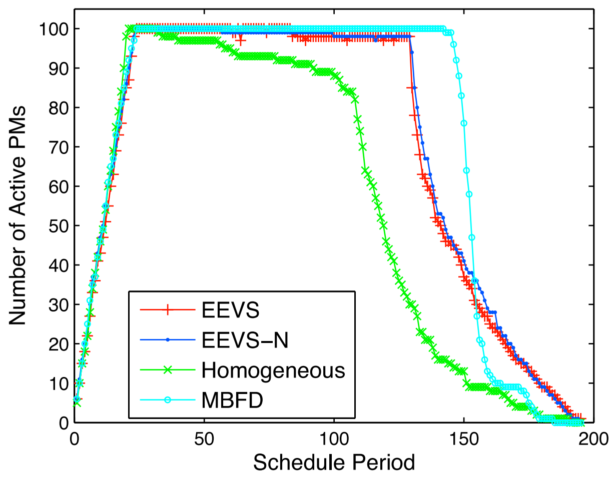
6. Conclusion and future work

How to reduce energy consumption of data centers has been an open challenge for both academe and industries. Cloud computing has been used in more and more fields, and virtualization is typically adopted in cloud computing nowadays. We can find that the main challenges for energy efficient scheduling of VMs in cloud computing are the heterogeneity of the PMs, the total power consumption of each PM and the adoption of some energy saving technologies for hardware such as DVFS.

To overcome these issues, we propose EEVS, an energy efficient scheduling algorithm of virtual machines to reduce the total energy consumed by the cloud, which also supports DVFS well. From



(a) 100 physical machines and 500 virtual machines.



(b) 100 physical machines and 5000 virtual machines.

Fig. 9. Number of active PMs when processing 500 and 5000 VMs on the cloud with 100 PMs.

the computation of total energy of a PM, we conduct that there is an optimal frequency for it to process certain VMs. Based on the optimal frequency we define the optimal performance–power ratio to weight the heterogeneities of the PMs. The PM with highest optimal performance–power ratio will be used to process the VMs first unless it does not have enough computation resources.

The process of EEVS is divided into some equivalent periods to simply the computation of total energy consumption of the PMs. In each period, we first find the VMs need to schedule and allocated them to the proper cores of the PMs, and then the optimal frequency of each active core can be computed according to the sum of the required resources of the VMs on it. Then the information of the VMs and PMs should be updated after each period especially when some VMs finished successfully. Finally the cloud should be reconfigured to consolidate the computation resources of the PMs to further reduce the energy consumption.

Though our EEVS consumes less energy and processes more VMs successfully than the existing methods in most cases, there are still some shortcomings. Two important assumptions are made in this paper, the performance and power penalties of status transitions of processor and VM migrations are ignored, and the PMs and workload are simulated though some information are checked. These assumptions do not work well in practical cloud environment. We will do a further study based on practical cluster and real workload in the next step to find more research issues.

Acknowledgments

This work is supported by National Natural Science Foundation of China (61373015), Research Fund for the Doctoral Program of High Education of China (No. 20103218110017), a project funded by the Priority Academic Program Development of Jiangsu Higher Education Institutions (PAPD), the Fundamental Research Funds for the Central Universities (Nos. NP2013307, NZ2013306).

References

[1] L.A. Barroso, The price of performance, Queue 3 (7) (2005) 48–53.
[2] US EPA ENERGY STAR Program, Report to congress on server and data center energy efficiency, Public law, 2007, pp. 109–431.
[3] W. Forrest, How to cut data centre carbon emissions? Website, December 2008. Available: http://www.computerweekly.com/Articles/2008/12/05/233748/how-tocut-data-centre-carbon-emissions.htm.
[4] L. Barroso, U. Holzle, The case for energy proportional computing, IEEE Comput. 40 (12) (2007) 33–37.
[5] A. Beloglazov, R. Buyya, Y.C. Lee, A. Zomaya, A taxonomy and survey of energy-efficient data centers and cloud computing systems, Adv. Comput. 82 (2011) 47–111.
[6] G. Laszewski, L. Wang, A.J. Younge, X. He, Power-aware scheduling of virtual machines in DVFS-enabled clusters, in: IEEE International Conference on Cluster Computing and Workshops, 2009, pp. 1–10.
[7] X. Fan, W.-D. Weber, L.A. Barroso, Power provisioning for a warehouse-sized computer, in: Proceedings of the 34th Annual International Symposium on Computer Architecture, 2007, pp. 13–23.
[8] J. Leverich, C. Kozyrakis, On the energy (in)efficiency of Hadoop clusters, Oper. Syst. Rev. 44 (1) (2010) 61–65.
[9] W. Lang, J.M. Patel, Energy management for MapReduce clusters, PVLDB 3 (1–2) (2010) 129–139.
[10] W. Lang, J.M. Patel, J.F. Naughton, On energy management, load balancing and replication, SIGMOD Rec. 38 (4) (2009) 35–42.
[11] Zhen Xiao, Weijia Song, Qi Chen, Dynamic resource allocation using virtual machines for cloud computing environment, IEEE Trans. Parallel Distrib. Syst. 24 (6) (2013) 1107–1117.
[12] S. Bazarbayev, M. Hiltunen, K. Josh, Content-based scheduling of virtual machines (VMs) in the cloud, in: International Conference on Distributed Computing Systems, 2013, pp. 93–101.
[13] I. Hwang, M. Pedram, Hierarchical virtual machine consolidation in a cloud computing system, in: IEEE Sixth International Conference on Cloud Computing, 2013, pp. 196–203.
[14] W. Vogels, Beyond server consolidation, Queue 6 (1) (2008) 20–26.
[15] X. Li, Z. Qian, S. Lua, J. Wu, Energy efficient virtual machine placement algorithm with balanced and improved resource utilization in a data center, Math. Comput. Modelling 58 (5–6) (2013) 1222–1235.
[16] G. Lovász, F. Niedermeier, H. Meer, Performance tradeoffs of energy-aware virtual machine consolidation, Cluster Comput. 16 (3) (2013) 481–496.
[17] J. Dong, X. Jin, H. Wang, Y. Li, P. Zhang, S. Cheng, Energy-saving virtual machine placement in cloud data centers, in: IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2013, pp. 618–624.
[18] N.Q. Hung, P.D. Nien, N.H. Nam, N.H. Tuong, N. Thoai, A genetic algorithm for power-aware virtual machine allocation in private cloud, in: International Conference on Information and Communication Technology, 2013, pp. 183–193.
[19] X. Liao, H. Jin, H. Liu, Towards a green cluster through dynamic remapping of virtual machines, Future Gener. Comput. Syst. 28 (2) (2012) 469–477.
[20] A. Beloglazov, J. Abawajy, R. Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing, Future Gener. Comput. Syst. 28 (5) (2012) 755–768.
[21] S. Esfandiarpoor, A. Pahlavan, M. Goudarzi, Virtual machine consolidation for data center energy improvement, CoRR, 2013. arXiv:1302.2227.
[22] N. Kim, J. Cho, E. Seo, Energy-credit scheduler: an energy-aware virtual machine scheduler for cloud systems, Future Gener. Comput. Syst. 32 (2014) 128–137.
[23] C.M. Kamga, G.S. Tran, L. Broto, Extended scheduler for efficient frequency scaling in virtualized systems, SIGOPS Oper. Syst. Rev. 46 (2) (2012) 28–35.
[24] H. Liu, C. Xu, H. Jin, J. Gong, X. Liao, Performance and energy modeling for live migration of virtual machines, Cluster Comput. 16 (2) (2013) 249–264.
[25] A. Strunk, W. Dargie, Does live migration of virtual machines cost energy, in: IEEE International Conference on Advanced Information Networking and Applications, 2013, pp. 514–521.
[26] S. Sotiriadis, N. Bessis, P. Gepner, N. Markatos, Analysis of requirements for virtual machine migration in dynamic clouds, in: IEEE International Symposium on Parallel and Distributed Computing, 2013, pp. 116–123.
[27] V. Medina, J.M. García, A survey of migration mechanisms of virtual machines, ACM Comput. Surv. 46 (3) (2014) 1–33.
[28] M. Masiero, A. Roos, December 23, 2012. http://www.tomshardware.com/reviews/cpu-performance-comparison,3370-17.html.
[29] SPEC, 12 November, 2014. http://www.spec.org/power_ssj2008/results/power_ssj2008.html.

Youwei Ding is a Ph.D. Candidate at the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. His research interests include energy efficient data management, cloud computing and data mining.

Liang Liu is an associated professor at the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. He receives his Ph.D. degree in 2012 form Nanjing University of Aeronautics and Astronautics. His research interests include wireless sensor network and data management in distributed environment.

Xiaolin Qin is a Professor at the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. His research interests include security database, temporal–spatial database, data management and security in distributed environment.

Taochun Wang is a Ph.D. Candidate at the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. He is also an associate professor at College of Mathematics and Computer Science, Anhui Normal University, China. His research interests include wireless sensor networks, privacy preservation and data management in distributed environment.