Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 4386362, 15 pages
http://dx.doi.org/10.1155/2016/4386362

Research Article

A Two-Tier Energy-Aware Resource Management for Virtualized Cloud Computing System

Wei Huang,[1] Zhen Wang,[2] Mianxiong Dong,[3] and Zhuzhong Qian[2]

[1]*School of Computer Engineering, Nanjing Institute of Technology, Nanjing 211167, China*
[2]*State Key Lab. for Novel Software Technology, Nanjing University, Nanjing 210023, China*
[3]*Department of Information and Electronic Engineering, Muroran Institute of Technology, Muroran 050-8585, Japan*

Correspondence should be addressed to Zhuzhong Qian; qzz@nju.edu.cn

Received 22 February 2016; Accepted 1 September 2016

Academic Editor: Tomàs Margalef

The economic costs caused by electric power take the most significant part in total cost of data center; thus energy conservation is an important issue in cloud computing system. One well-known technique to reduce the energy consumption is the consolidation of Virtual Machines (VMs). However, it may lose some performance points on energy saving and the Quality of Service (QoS) for dynamic workloads. Fortunately, Dynamic Frequency and Voltage Scaling (DVFS) is an efficient technique to save energy in dynamic environment. In this paper, combined with the DVFS technology, we propose a cooperative two-tier energy-aware management method including local DVFS control and global VM deployment. The DVFS controller adjusts the frequencies of homogenous processors in each server at run-time based on the practical energy prediction. On the other hand, Global Scheduler assigns VMs onto the designate servers based on the cooperation with the local DVFS controller. The final evaluation results demonstrate the effectiveness of our two-tier method in energy saving.

1. Introduction

Cloud computing provides elastic computing resources on a pay-as-you-go basis for most conceivable forms of applications but it also causes huge amounts of electric energy consumption. Almost 0.5% of world's total power usage is consumed by the servers in data centers [1]. Among them, processors (CPUs) account for the most significant part of power and have the most dynamical power that can be adjusted, while other components can only be completely or partially turned off [2]. Owing to these reasons, reducing energy consumption of processors using the dynamic nature of CPUs' power has become a hot research topic in cloud computing system.

To service more users for more income, service providers prefer to share cluster resources among users. In cloud environments, the virtualization technique is widely adopted to allow users to share the physical resources. Making the working servers for Virtual Machines (VMs) as less as possible and letting the idle servers be in a low-power mode will improve the utilization of resources and reduce energy consumption, which is known as VM consolidation. In each server, by applying Dynamic Voltage and Frequency Scaling (DVFS), which enables dynamic adjustment of execution frequency on demand, more energy can be saved. The dynamic power consumption of CPU is proportional to the frequency and to the square of voltage. Scaling down the execution frequency will reduce the power while it may also reduce the performance and increase the execution time, which may instead cause more energy consumption (energy is equal to the line integral of power P to time t, $E = \int_0^t P\,dt$). On the other hand, real-time tasks in the cloud computing system usually have requirements on execution speed; the extension of execution time may violate QoS requirements. Thus, it is nontrivial to reduce energy consumption by scaling the execution frequencies of tasks [3].

VM consideration could improve the resource indeed and many previous works [4–6] achieve significant result on energy saving in virtualized cloud system. However, most of them do not take the advantage of DVFS strategy. Some others only apply the DVFS after allocation while not considering

the influence of DVFS before allocation. However, if taking impacts of DVFS technique on energy into consideration before allocation, much more energy can be reduced. But it is not trivial to minimize the total energy consumption by VM allocation algorithm and DVFS strategies in this way. Several problems need to be solved: (1) how to define Quality of Service (QoS) requirements; (2) which servers can load arrival VMs with requirements; (3) which server brings minimum energy consumption by DVFS for arrival VMs and ensures QoS requirements.

In this paper, we propose a cooperative two-tier energy-aware management by taking the DVFS into consideration, which offload real-time tasks to VMs on clusters and scaling frequencies. On the local tier, we propose a novel way to find the best combinations of frequencies of different CPUs that consume the least energy based on the practical energy prediction. We take both the frequency-power and utilization-power relationship into consideration when forecasting energy consumption. On the global tier, by cooperating with local DVFS controller, the Global Scheduler assigns a VM to its favourite processor in a cluster that brings minimum energy change. The frequency to execute the arrival workload has been decided before allocation instead of after allocation. The framework proposed in this paper takes both the energy consumption and performance into consideration and achieves good tradeoff between energy and consumption. The status of each hosts is controlled by the global master by regular communication, so the master can control the energy consumption and performance. Meanwhile, the computing can be done in parallel in each candidate for allocation to improve the effectiveness of computation. In summary, the main contributions of this paper are as follows:

(i) We propose the multiprocessor power model, which is shown to be close to the real power consumption of a server according to the evaluation of the model. The multiprocessor power model helps us to precisely estimate the energy consumption.

(ii) We transform the energy minimization problem of frequency scaling to a node searching problem in directed graphs. We also prove that the optimal state which consumes the least energy can be found from an initial state in which all processors' frequencies are maximum.

(iii) We provide a novel scheduling algorithm, which can work in parallel and efficiently cooperates with local DVFS controller, for the problem of energy-aware scheduling. The experiments justify the effectiveness of our strategy on energy saving.

The rest of this paper is organized as follows. Section 2 introduces some related works. Section 3 introduces the framework of our solution and Section 4 introduces the task model and the analysis of the request of a VM. In Section 5, we introduce the energy prediction method and energy minimizing algorithm of local DVFS controller. The global VM allocation algorithm is presented in Section 6. In Section 7, we evaluate our solution through some experiments. Finally, we conclude this paper in Section 8.

2. Related Work

Reducing energy consumption has already been a critical issue of data center in recent years. Many works study the energy saving strategies in virtualized environment. Kusic et al. [7] defined a dynamic resource management as a sequential optimization in virtualized environment. The sequential optimization whose objective is maximizing the profit of provider is solved using Limited Lookahead Control (LLC) by minimizing both energy cost and SLA. But the framework captures the behavior of each application by simulation-based learning and the complexity of the model makes the approach not suitable for large scale data center.

In [8], the authors have developed dynamic resource provisioning and allocation problem with virtualized technique for energy-efficient cloud computing. They propose self-manage and energy-aware mechanisms to allocate the Virtual Machines (VMs) and migrate VMs according to CPU utilizations and energy consumption. The placing problem of allocation which can be seen as a bin packing problem is solved by Modification Best Fit Decreasing (MBFD). For the migration problem, three policies are proposed to choose VMs to migrate in order to reduce energy consumption.

Cardosa et al. [9] have presented a novel approach for power-efficient VM placement for the heterogeneous data centers by leveraging min-max and share features of the VMs based on the DVFS and soft scaling technique. The power consumption and utilization obtained from the running time of a VM are optimized by being set a priori. However, their approach does not strictly support SLAs and the information of applications' priorities is needed. Cao and Dong [10] propose an energy-aware heuristic framework for VM consolidation which can obtain a better tradeoff between energy saving and performance. A SLA violation decision algorithm is proposed to determine hosts' status for SLA violation. Based on the hosts' status, the minimum power and maximum utilization policy for VM migration are used to achieve the energy saving.

Reference [11] maximizes the utilization at virtual machine level in the environment of container. The objective of the paper is to dynamically set the sizes of virtual machines in order to improve the utilization of VMs, which saves overall energy consumption. Experiments show that their method can achieve 7.55% of energy consumption compared to scenarios where the virtual machine sizes are fixed. Reference [12] proposes a VM allocation algorithm to reduce energy consumption and SLA violation, which uses the historical record of VMs' usage.

Some other works mainly focus on the DVFS strategy to decrease processors' power consumption in hosts. Some of them periodically adjust the frequency according to the performance of server. Reference [13] monitors the utilization of processors periodically and the frequency is decreased very carefully when there are observable impacts on execution time of tasks. Hsu and Feng [14] proposed a β-adaption algorithm that periodically evaluates the performance and automatically adapts the frequency and voltage at run-time. Reference [15] also developed the periodic DVFS controller for multicore processor without using any performance model.

However, the length of period has a great impact on the performance of algorithms, it should be evaluated very carefully.

Scaling the frequency according to the types of workloads is another efficient way to carry out DVFS control. They achieve the goal of energy saving with a little or limited performance loss by decreasing the frequency during the communication, data access, memory access, or idle phases. Lim et al. [16] proposed a run-time scheduler that applies DVFS control during the communication phases which is identified by intercepting the MPI calls. In [17], the authors presented a novel algorithm that utilizes the opportunities in execution of hybrid MPI/OpenMP application to scale the frequency and reduce energy consumption. Tan et al. implement the DVFS scheduling strategy for data intensive application in [18] and achieved the energy saving. Their strategy adaptively sets the suitable frequency according to the percentage of CPU-bound time in the total execution time of workloads and is implemented in source code level.

The DVFS is able to reduce the energy consumption, but it is limited on a single server. A lot of work developed the DVFS-based task scheduling among servers because the distribution of workloads influences the overall energy. References [19, 20] propose similar energy-aware strategies that schedule a set of tasks onto physical machine. They adjust supply voltage by utilizing slack time of noncritical jobs. Reference [19] also discussed the tradeoff between energy consumption and scheduling length. Khan and Ahmad [21] studied the problem of task allocation in grid and they utilized the cooperative game theory to minimize the energy consumption and makespan of tasks for DVFS-based clusters. Similar to [21], Mezmaz et al. studied the problem for the dependent precedence-constrained parallel applications [22]. Different to these works, we study the independent real-time services with deadline constraints in multiprocessor system.

References [23–27] researched energy-efficient task scheduling for real-time system. Luo and Jha studied the scheduling of periodic tasks in heterogeneous system and gave a power-efficient solution [27]. In [24], authors proposed an energy-aware task partitioning algorithm with polynomial time complexity for DVFS-based heterogeneous system. Awan and Petters proposed an energy-aware partitioning of tasks method which consists of two phases and they use a realistic power model to estimate power consumption [23]. Our task allocating algorithm cooperates with the local DVFS controller to predict the energy consumption in different situations; the influence of frequency scaling to energy consumption is taken into account before allocation for saving more energy.

3. Overview

Our framework can accept and analyze the arrival workloads and package them by Virtual Machines (VMs) and allocate them to the suitable server to reduce energy consumption. We first describe the architecture of our solution in Figure 1 and subsequently introduce the real-time analysis in this section [3]. In our solution, the *Global Scheduler* assigns a task to a VM to execute it and guarantees its QoS requirement. This VM will be allocated to a host which can offload it without causing any violation of QoS requirement and brings minimum energy consumption. Our objective is to find the allocation method for VMs and frequencies scaling method for tasks to reduce the energy consumption.

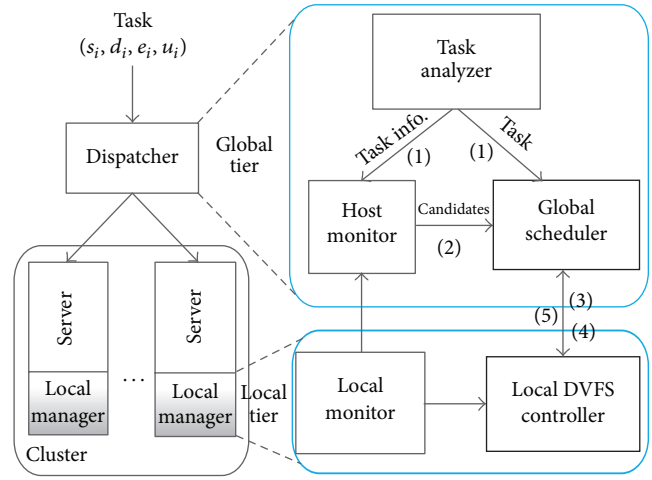


Figure 1: System architecture of our solution to the energy-aware resource management.

Definition 1 (host model). Let $\text{host}_j = (U_j, F_j)$ be denoted as resources of jth host, where U_j and F_j are vectors that record the utilizations and frequencies of each processor.

The *Task Analyzer* in Dispatcher receives and analyzes the information of incoming task and sends it to other components when necessary. The *Host Monitor* is an assistant component which connects to each server and gathers the basic information of servers. The *Local Monitor* monitors the resources of a server and sends the basic information to *Host Monitor* when necessary. The basic information of servers is recorded in the *Host Model* (Definition 1). We mainly focus on the resource of processor, so we only record the states of processors in the *Host Model*. The main work mechanism of our solution to schedule a new task request task_n is described as follows:

(1) When task_n comes, the *Task Analyzer* analyzes the basic information of task_n and sends it to the *Host Monitor* and *Global Scheduler* (Section 6).

(2) When the *Host Monitor* receives the information of task_n, it selects a set of candidates who can load task_n according to the basic information of servers and sends the set to the *Global Scheduler*. In the large datacenter, the number of candidates can be carefully selected to improve the effectiveness of allocation.

(3) When the *Global Scheduler* receives the candidates and the basic information of task_n, it sends the task information to the servers who are in the candidate set.

(4) When a candidate receives the task information, the *local DVFS controller* (Section 5) will run to estimate the minimum energy change if task_n is allocated to

one of its VM according to the monitored information. Then the controller returns the result to the *Global Scheduler*.

(5) When the *Global Scheduler* receives responses from all the candidates, it allocates task_n to the best server using our allocation algorithm. There may be some network error in communications like packet error or loss or high network delay. We can set some threshold for the *Global Scheduler*, for example, time threshold for response time or retry times. When response time or retry times of a candidate are larger than the thresholds, the *Global Scheduler* can discard this candidate.

This is the simple architecture for energy-aware task scheduling and some project implemented details or optimizations are not discussed in this paper. We mainly focus on the energy-aware scheduling for tasks and provide a solution to this problem. Some problems like single point of failure and network error are also important for the distributed cloud system. We consider that these problems have the maturing solutions in today's cloud system and these aspects may not be a problem to our solution.

4. Task Model

The request of service in the cloud computing system usually has deadline constraints which is the major aspect of Service Level Agreements (SLAs). We explore energy saving method for the cluster that accepts request for tasks. We define the *task model* (Definition 2) to describe the request for a task. The task model records some important information that users provide. s_i and d_i describe the requirements of tasks and e_i and u_i describe the execution characters of tasks.

Definition 2 (task model). Let $\text{task}_i = (s_i, d_i, e_i, u_i)$ describe ith task, where s_i, d_i, e_i, u_i represent the start time, relative deadline, predicted execution time, and the average utilization, respectively.

For the isolation, scalability, and stability of system, tasks are usually run in the VMs independently in the cloud computing system. We can regard each task as a VM, so the allocation of the tasks is equivalent to the allocation of VMs in some degree. In our model, we assign a task to a VM to run and the VM will be allocated to appropriate host. When a task finishes, the VM loading this task will be shut off or turned into sleep. The living time for a VM to run a task is equal to the execution time of this task. Therefore, the living time for the VM should not exceed the deadline of the tasks. Let VM_i represent the virtual machine load task_i.

We designed the *Task Analyzer* to accept and analyze the incoming request of tasks. It sends the basic information of tasks to other components after preprocessing. The living time of a task (i.e., VM) usually includes computing time and CPU idle time. The CPU idle time may consist of communication, memory, or disk access. The real-time analysis we designed is to distinguish the computing time and CPU idle time. The average utilization of a VM can reflect the computation and CPU idle time in some degree. Let $T_c(f)$ and T_i represent computing time at frequency f and idle time of a VM, respectively. We estimate the computing time $T_c(f_{\max}) = e_i \cdot u_i$ and idle time $T_i = e_i \cdot (1 - u_i)$ for ith task. The *Task Analyzer* calculates $T_c(f_{\max})$ and T_i and sends these information to other modules.

The computing time has a tight relation to the CPU frequency which shows a linear extension to the reduction in frequency [28, 29], while the idle time of a task will barely change due to frequency scaling. Therefore, the living time of the VM of ith task frequency f can be expressed as

$$T^i(f) = T_c(f_{\max}) \frac{f_{\max}}{f} + T_i. \tag{1}$$

When the *local DVFS controller* predicts the energy consumption in different frequency, the living time of VMs can be calculated by (1) according to the task information provided by Task Analyzer. Although the running time can be predicted under different frequency, the energy prediction and DVFS controller are not a easy task. We will introduce details of our method to solve them in next sections.

5. Local DVFS Controller

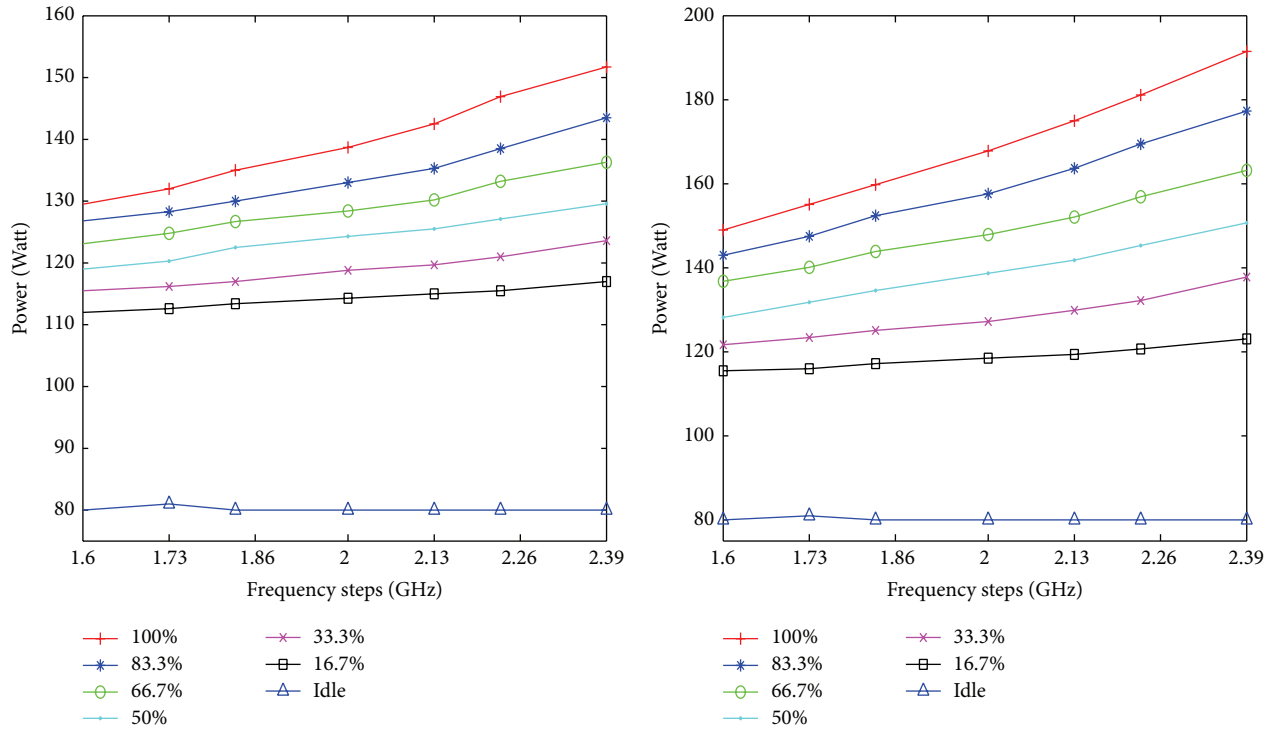
The *DVFS Controller* plays an important role in our framework and it has two main functions. On the one hand, it predicts the energy consumption of a multiprocessor server according to the processors' utilizations, frequencies, and the living time of VMs. On the other hand, based on the energy prediction, it runs the *k-Phase energy Prediction (kPP)* algorithm to find the best frequencies combinations that bring minimum energy consumption.

5.1. Energy Prediction for Multiprocessor Servers. The electric energy consumption is the integral of the active power with respect to time. Therefore, the power prediction of server is crucial. Previous works like [30–34] provided serval methods to estimate the power of a server. However, they only focused on the frequency-power or utilization-power relationship and the detailed power prediction for multiprocessor platform is also ignored. In this paper, we provide a practical power prediction for multiprocessor servers based on the frequency-power and utilization-power relationship. We utilize the fact that the homogenous processors will consume the same power when they are under the same condition to predict the power consumption.

The power consumption of a server consists of two parts: static and dynamic power consumption. The static parts include the power consumption of main board, hard disk, fan, and so forth. CPU accounts for the largest part of dynamic power. According to the previous studies, the dynamic power consumption of CPU is proportional to the frequency and to the square of voltage [34], which can be express as

$$P_{\text{dynamic}} \simeq A \times C \times V^2 \times f, \tag{2}$$

where A is the percentage of active gates, C is total capacitance, V is supply voltage, and f is the operating frequency. According to [31], the voltage has a linear relationship to



(a) Only one processor is busy

(b) Two processors are busy

FIGURE 2: Real power consumption of Dell R710 whose configurations are introduced in Table 1 and the legend represents the different utilizations.

frequency, so the dynamic power of a processor can be reduced as a function of frequency: $P_{\text{dynamic}} = \alpha \times f^3$, where α is a proportional coefficient. Processors also have static power when they are active. Let P_s represent the static power of a server and P_{CPU_s} represent the static power of processor. The power of a host in which all homogenous processors work in the same frequency f with full utilization can be expressed as follows:

$$P(f) = P_s + N_c\left(P_{\text{CPU}_s} + \alpha f^3\right), \tag{3}$$

where N_c is the number of CPUs. We want to eliminate the static power of processors, which is not easy to measure. For a given host, we can easily measure its maximum power which is $P_{\max} = P_s + N_c(P_{\text{CPU}_s} + \alpha f_{\max}^3)$. Therefore, we can estimate the power consumption of a host in which all processors work in the same frequency f:

$$P(f) = P_{\max} - \alpha N_c\left(f_{\max}^3 - f^3\right). \tag{4}$$

The power consumption is also related to utilizations. Figure 2(a) shows the power consumption with only one processor running and Figure 2(b) shows the power of two processors that work in same utilization and frequency. As we can see, the power with different utilization under same frequency is different. The power and the utilization present a linear relationship which is with one voice to [30, 32, 33]. Therefore, the power consumption of one homogenous CPU with frequency f and utilization u can be denoted as

$$P_{\text{CPU}}(u, f) = \frac{1}{N_c}\left[P_{\max} - P_s - \alpha N_c\left(f_{\max}^3 - f^3\right)\right]u. \tag{5}$$

Finally, the power of prediction of a homogenous multiprocessor server can be expressed as

$$\begin{aligned} P_{\text{host}} &= P_s + \sum_{c=1}^{N_c} P_{\text{CPU}}^c \\ &= P_s \\ &\quad + \frac{1}{N_c}\sum_{c=1}^{N_c}\left[P_{\max} - P_s - \alpha N_c\left(f_{\max}^3 - F_{j,c}^3\right)\right]U_{j,c}. \end{aligned} \tag{6}$$

We can view the power of jth host as a function of utilizations and frequencies, which is presented as $P_{\text{host}}(F_j, U_j)$, where U_j and F_j are defined in *Host Model*.

Definition 3 (Frequency Scaling Unit). A time interval $[t_1, t_2)$ is a Frequency Scaling Unit (FSU) if (1) $\forall t \in [t_1, t_2)$, $NT(t) = NT(t_1)$ and (2) $NT(\cdot)$ changes at t_1 and t_2, where $NT(t)$ represents the number of VMs at time t in a server.

The energy consumption depends on both the execution time and the power. We define the concept of *Frequency Scaling Unit* (FSU, Definition 3) to estimate the living time

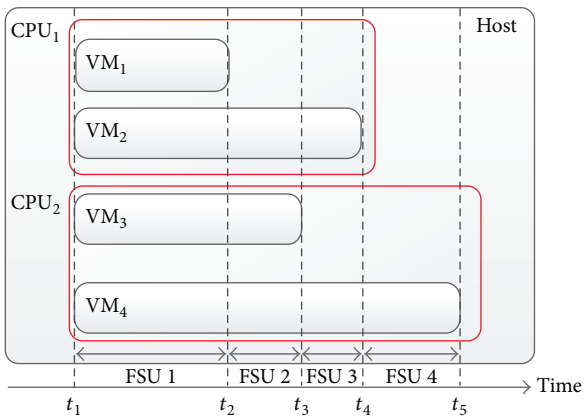


FIGURE 3: An example of FSU.

of VMs. The FSU represents a period of time that the number of VMs does not change. Once the number of VMs changes, that is, a VM coming or leaving, it enters the next FSU. An example of FSU is shown in Figure 3, which includes four FSUs. Assuming a VM is stopped at time t_2 and next VM is ended at time t_3, then $T = t_3 - t_2$ is an FSU. If a VM is allocated to the server at t_1 and the VM finished at t_2, $T = t_2 - t_1$ is said to the *first FSU* from current time. It is obvious that the number of VMs in the host is equal to the number of FSUs if all VMs finish at the different time, and we suppose that VMs are ended at the different time in a host in the rest of this paper.

If we set consistent frequencies for all processors in an FSU, the power state in this FSU is relatively stable because the workloads in this FSU are fixed. We know the length of this FSU, so the energy consumption in an FSU can be predicted conveniently and precisely by the following equation: $E = P \times T$, where P and T represent power and time, respectively. Based on the definition of FSU, power function, and related notations in Notations, the energy consumption of jth host to finish all the VMs can be predicted as follows:

$$E_j = \sum_{k=1}^{N_{j,p}} P_{j,k}\left(U_{j,k}, F_{j,k}\right) T_{j,k}, \tag{7}$$

where the power of host $P_{j,k}$ can be calculated by (6) in different situations. The length of FSU can also be estimated under different frequencies by (1).

5.2. kPP Algorithm for Energy Minimization. According to the analysis of energy prediction, if we set consistent frequencies in an FSU, then we can predict the energy consumption in an FSU conveniently. If we set FSUs with different frequencies, the living time of VMs and power state of server will be different, which brings different energy consumption. There is an optimal solution that consumes minimum energy when all VMs end in this server. We want to find the frequencies combinations for all FSUs that bring minimum energy on the promise of ensuring the requirements of VMs. Based on the energy prediction, the energy minimization problem of frequency scaling in homogenous multiprocessor platforms can be formalized as follows:

$$\begin{aligned} &\min_{F_{j,k} \in F_j} \quad \sum_{k=1}^{N_{j,p}} P_{j,k}\left(U_{j,k}, F_{j,k}\right) T_{j,k} \\ &\text{s.t.} \quad \forall j \in H,\ \mathrm{VM}_i \in J_j,\ s_i + \sum_{m=1}^{N_p^i} t_{j,m}^i < d_i. \end{aligned} \tag{8}$$

For clearly describing the problem, we define $(F_{j,1}, F_{j,2}, \ldots, F_{j,|J_j|})$ as a *state* of possible frequencies combinations for FSU 1 to $|J_j|$ using the notations in Notations. If there is only one frequencies combination that is different between two states, we say they are neighbors. For example, if there are two states $s_1 : (F_{j,1}, F_{j,2}, \ldots, F_{j,|J_j|})$ and $s_2 : (F_{j,1}, F_{j,2}, \ldots, F'_{j,|J_j|})$ and the frequencies combinations of FSU $|J_j|$ in s_1 and s_2 are different while others are the same, then s_1 and s_2 are neighbors. In addition, we define $E(s)$ as the cost function of total energy consumption of s according to (7) if we scale the frequencies like s in each FSU. Let a node present a state and an edge (u, v) between two nodes presents neighborhood between u and v. The minimization problem is to find the "optimal" node that brings minimum energy without any violation of SLAs from the initial node in the graph.

Lemma 4. *Let the initial node represent the state in which all processors' frequencies are highest in all FSUs. If the initial state ensures SLAs for all VMs, there is a path from initial node to the optimal node with minimum energy consumption without any violation of SLAs.*

Proof. Let $s_m = (F_{j,1}, F_{j,2}, \ldots, F_{j,|J_j|})$ represent the optimal state with minimum energy consumption without any violation of SLAs. If all processors' frequencies are highest in all FSU of s_m, the initial state is the optimal state. Otherwise, we select the FSU r in which the frequencies are not highest for all CPUs. If we scale the frequencies to highest in r, the new state s_n will also ensure the SLAs for all VMs because processors are working at higher frequencies which leads to shorter execution time. s_n is one of the neighbors of the optimal state, which means that s_n can also move to s_m. Repeating the process above, we can find a path from s_m to initial state s_i, which represents that there is a path from s_i to s_m. □

5.2.1. k-Phase Energy Prediction (kPP) Algorithm. By energy prediction of k FSUs, the best frequencies combinations can be found moving from the initial state according to Lemma 4. There are $|F_j|$ possible frequencies combinations in each FSU, so there may be $|F_j|^{|J_j|}$ possible states of all FSU with different energy consumption. Let FL_j represent the frequency levels of jth host; we have $|F_j| = |FL_j|^{|C_j|}$. We want to find the optimal solution with minimum energy consumption in these $|F_j|^{|J_j|}$ possible states while still ensuring SLAs. However, the searching space may be $|FL_j|^{|C_j||J_j|}$ which is too huge if there are many VMs. Therefore, we provide two heuristic algorithms to search the "optimal" solutions which are based on simulated annealing (SA) [35] and variable depth search (VDS) [36], respectively.

Input:
The state of host$_j$;
Output:
The possible state s;
(1) J_j = host$_j$.getVM(), F_j = host$_j$.getFreqSpace()
(2) set s_0 be the state that the frequency is max for each CPU in all FSU
(3) $s = s_0$, $e = e_{\max} = E(s_0)$, $t = 0$, $k = |J_j|$
(4) **while** $t < t_{\max}$ or s doesn't change in l rounds **do**
(5) $st = s$, r = random(k)
(6) $F_{j,r}$ = random($F_j - s$.get(r)) /*Select a neighbor*/
(7) st.set(r, $F_{j,r}$) /*Change frequencies combination of FSU r to $F_{j,r}$*/
(8) **for** $i = 1$ to k **do**
(9) **if** VM$_i$ violates SLAs according to st **then**
(10) go to (17)
(11) **end if**
(12) **end for**
(13) $et = E(st)$
(14) **if** $et \le e_{\max}$ or random() < exp($-(et - e)t/pT$) **then**
(15) $s = st$, $e = et$ /*Change states*/
(16) **end if**
(17) $t = t + 1$
(18) **end while**
(19) **return** s

Algorithm 1: SA based kPP algorithm.

(1) Simulated Annealing Based Heuristic Algorithm. By comparing energy consumption of a random neighbor, we can find a better state that brings less energy. If we repeat the process many times, we may find the optimal state. Let s_0 represent the initial state in Lemma 4. In fact, since the simulated annealing (SA) algorithm has been proved to converge to the optimum with probability 1, it can be expected that our algorithm will output nice results by enough iterations. If we know the frequency steps and tasks' information, the living time of VMs is determined. Therefore, we can estimate the total energy of the situation of state s_0 (line 3) using the energy cost function $E(\cdot)$. The algorithm runs $t_{\max}$ iterations to find the state where less energy is consumed compared to the initial state. In each iteration, the algorithm randomly selects an FSU r to change the frequencies of processors and generates a new state st. This step takes $O(1)$ time. If the random neighbor st violates SLAs for any one of VMs, the state is discarded and our algorithm enters into the next iteration. This step takes $O(N_{j,p})$ time, where $N_{j,p}$ is the number of tasks. Otherwise, if predicted energy et is less than e, st is selected as compared state for next iteration due to less energy consumption. The energy prediction takes $O(N_{j,p} \cdot N_c)$ time according to (7), where N_c is the number of processors. Besides, the algorithm also changes the state from s to st with the probability $\exp(-(et - e)/pT)$ suggested by Metropolis et al. [37] to give the possible to find optimal solution. The details of the simulated annealing based kPP algorithm are presented in Algorithm 1. Obviously, the time complexity of SA-based kPP algorithm is $O(N_{j,p} \cdot N_c \cdot t_{\max})$.

(2) Variable Depth Search Based Heuristic Algorithm. The VDS-based kPP algorithm selects the state that brings minimum energy in a subset of neighbors and compares it to the current state. If the selected state consumes less energy on the promise of ensuring the SLAs of VMs, we will change the state to it. The initialized state of VDS-based algorithm is the same as the initialization of SA-based algorithm. The algorithm selects a subset of neighbors whose frequencies combination of FSU r are different (lines 5-6). The frequencies combination of FSU r with minimum energy will be selected (line 7) and generates a new state. The energy prediction takes $O(N_{j,p} \cdot N_c)$ time, so the selection of state with minimum energy takes $O(|X| \cdot N_{j,p} \cdot N_c)$ time, where $|X|$ is the size of subset. The algorithm checks the violations of SLAs of new state (lines 9–13). The process repeats for $t_{\max}$ times or until the state s does not change in l iterations. Therefore, the time complexity of this algorithm is $O(|X| \cdot N_{j,p} \cdot N_c \cdot t_{\max})$. The effectiveness of the variable depth search is proved in [36]. The details of VDS-based kPP algorithm are shown in Algorithm 2.

The *local DVFS controller* runs the kPP algorithm when the *Global Scheduler* asks it to predict the minimum energy consumption and return it to *Global Scheduler*. This is one of the opportunities to run kPP algorithm. When the workload changes, the power state will change. In addition, the execution time of VMs may have some errors which may lead to the error of energy prediction. Therefore, we apply the frequencies scaling when a VM finishes and scale the frequency for first FSU, which means that the algorithm only scales the CPUs' frequencies just for the first FSU while predicting the frequencies combinations for k FSUs. As the example shown in Figure 3, if the VM$_4$ comes at the time t_1 and is allocated to the host, this host applies the kPP algorithm at that time and sets the CPUs' frequencies like FSU 1 of the result. When

Input:
The state of host$_j$;
Output:
The possible state s;
(1) J_j = host$_j$.getVM(), F_j = host$_j$.getFreqSpace()
(2) set s_0 be the state that the frequency is max for each CPU in all FSU
(3) $s = s_0$, $t = 0$, $k = |J_j|$
(4) **while** $t < t_{\max}$ or s doesn't change in l rounds **do**
(5) $st = s$, r = random(k)
(6) randomly select a subset $X \subseteq F_j$
(7) x = argmin($E(st.\text{set}(r, x))$), for all $x \in X$ /*Select the state form subset with minimum energy consumption*/
(8) st.set(r, x) /*Change frequencies combination of FSU r to x*/
(9) **for** $i = 1$ to k **do**
(10) **if** VM$_i$ violates SLAs according to st **then**
(11) go to (15)
(12) **end if**
(13) **end for**
(14) $s = st$ /*Change states*/
(15) $t = t + 1$
(16) **end while**
(17) **return** s

Algorithm 2: VDS-based kPP algorithm.

VM$_1$ finishes at the t_2, the kPP algorithm also runs to obtain the "optimal" state s and scales frequencies according to the result. Due to the specialities of kPP algorithm at running time, the iteration should be completed in a short time so that the local stage can scale the frequencies in time.

6. Global Scheduler

Different allocations of a new VM may affect the overall energy consumption, because the new VM executed on different servers will bring different energy consumption. We want to find the appropriate scheduling to minimize the energy consumption to finish all the VMs. We can obtain the different energy consumption with different allocation if we ask each host to predict the minimum energy consumption. Using the results of different allocations, we can select a better allocating scheme to reduce the energy consumption. Our goal is to minimize the overall energy cost of the whole cluster for finishing all VMs including the new VM VM$_n$. To solve the energy minimization problem of VM scheduling, we first formalize the problem. Let decision parameter $a^i_{j,c}$ be 1 if the ith VM is allocated to cth CPU of jth host, otherwise 0. The energy-efficient VM allocation problem can be expressed as

$$\begin{aligned}
\min \quad & \sum_{j\in H}\left\{\min_{F_{j,k}\in F_j}\sum_{k=1}^{N_{j,p}} P_{j,k}\left(U_{j,k}, F_{j,k}\right)T_{j,k}\right\}\\
\text{s.t.} \quad & \sum_{j\in H, c\in C_j} a^n_{j,c} \le 1;\\
& J_j = J_j \cup \{\text{VM}_n\}, \quad \text{if } a^n_{j,c} = 1;\\
& \forall j \in H,\ \text{VM}_i \in J_j,\ s_i + \sum_{m=1}^{N^i_p} t^i_{j,m} < d_i;\\
& \forall k \in N_{j,p},\ j \in H,\ \sum_{c\in C_j} U^c_{j,k} < UT_j.
\end{aligned} \tag{9}$$

The energy minimization problem of VM scheduling is to find the server which brings minimum energy of whole cluster if VM$_n$ is allocated to it. The minimum energy consumption of each server can be predicted by kPP algorithm, represented by EMIN$_j$ for jth host. If an incoming VM is allocated to jth host, the value of EMIN$_j$ changes while the minimum energy consumption of other hosts does not change. When VM$_n$ is allocated to the yth host, the energy consumption becomes EMIN$'_y$ + $\sum_{j\in H-\{y\}}$ EMIN$_j$, where EMIN$'_y$ is the minimum energy cost if VM$_n$ is allocated to yth host. We have

$$\begin{aligned}
E_{\min} &= \text{EMIN}'_y + \sum_{j\in H-\{y\}} \text{EMIN}_j\\
&= \Delta\text{EMIN}_y + \text{EMIN}_y + \sum_{j\in H-\{y\}} \text{EMIN}_j\\
&= \Delta\text{EMIN}_y + \sum_{j\in H} \text{EMIN}_j.
\end{aligned} \tag{10}$$

So we can select the host that brings minimum energy change ΔEMIN$_y$ to run the incoming VM. We call the scheduling algorithm *Minimum energy Change* (MC), shown in Algorithm 3. The *Global Scheduler* sends the information of a VM after analyzing to a subset of host (line 1) and each host returns the predicted minimum energy change on it. Therefore, we can run the energy-efficient algorithm in parallel to

Input:
A new VM VM_n;
Output:
Designate host and processor for loading VM_n;
(1) *Host Monitor* selects a subset of active hosts that can load the VM
(2) notify the information of VM_n to all candidates
(3) each host estimates the minimum energy change $\Delta EMIN_j$ if VM is allocated to processor c of $host_j$
(4) $host_j = argmin_{j \in H}(\Delta EMIN_j)$
(5) **return** $host_j$

Algorithm 3: Minimum energy change allocation.

obtain the minimum energy consumption for each host when a VM arrives. After the *local DVFS controller* predicts the minimum energy change, it also records the best processor to hold this VM. When this VM is really allocated to it, the VM will be scheduled onto this best processor. Once deciding the host, the selected host will start a VM to run the VM working under the selected frequencies. It is obvious that the time complexity is $O(|H|+L+T)$, where L and T represent the time complexity of local predicting algorithm and communication time, respectively.

The number of candidate hosts will affect the total cost of a cluster, we evaluate the influence of kPP strategy on the total energy cost. Assume the total VM number is N_t; the size of subset for candidate hosts is N_h and the average run-time of local DVFS algorithm is 0.5 seconds. Let the mean power consumed by a VM be $\overline{P}$ Watt and the average length of VMs be $\bar{t}$ seconds. The kPP algorithm runs when the MC algorithm asks candidate hosts to estimate energy consumption; the energy consumption of kPP algorithm in this part is $N_h \cdot \overline{P} \cdot N_t \cdot 0.5$. The kPP algorithm also runs when a VM is allocated and finished, so the energy for this part is $2 \cdot N_t \cdot \overline{P} \cdot 0.5$, and the total energy produced by all VMs is $N_t \cdot \overline{P} \cdot \bar{t}$. Therefore, the *energy consumption ratio* (ECR) of kPP algorithm compared to the total energy cost is

$$\mathrm{ECR} = \frac{0.5 \cdot N_h + 1}{\bar{t}}. \tag{11}$$

In a large scale data center, the mean VM length can be acquired according to the historical data, and we can carefully select the size of candidate host estimating the energy change of offloading a new VM to increase the energy consumption of kPP algorithm as less as possible.

7. Experimental Evaluation

7.1. Evaluating Power Prediction. The energy prediction of server depends on the accuracy of power prediction under different utilizations and frequencies. We have evaluated the multiprocessor power prediction method by comparing the real power consumption to the estimation of power model in different status for a specific host. The real experimental environment is shown in Figure 4. The details of server R710 used in our paper are shown in Table 1. We explore the real power consumption R710 and use the first seven steps when

Table 1: Details of servers.

Name	Dell PowerEdge R710	Dell PowerEdge R720
CPUs	Two Intel Xeon processors E5645 @2.4 GHz	Two Intel Xeon processors E5-2620 @2.1 GHz
Frequency steps (GHz)	1.60, 1.73, 1.86, 2.00, 2.13, 2.26, 2.39, 2.40	1.20, 1.30, 1.40, 1.50, 1.60, 1.70, 1.80, 1.90, 2.00, 2.10
Memory	24 G 1333 Mhz DDR3	64 G ECC DDR3
Disk	Two 10 k SAS, 250 GB	One 10 k SAS, 300 GB
Operation system	CentOS 6.5	CentOS 6.5

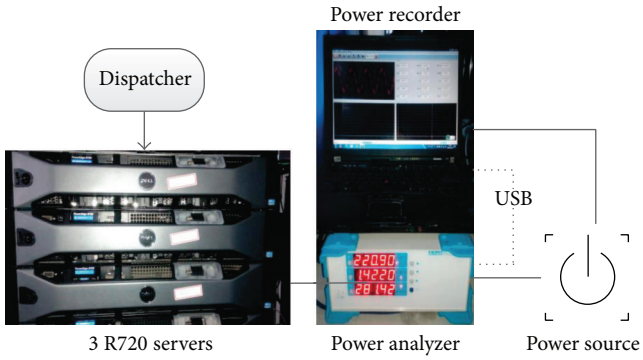


Figure 4: Real system architecture.

we evaluate the power model because the last frequency is very close to the frequency 2.39 GHz. The power consumption of the host when both processors are fully utilized at frequency level 2.39 GHz is 192 Watt and the static power when the system is not idle is 110 Watt. The proportional coefficient $\alpha = 2.33135$ is obtained and calibrated by offline experiments.

For evaluating the multiprocessor power prediction, we randomly select some frequencies and utilizations of two processors and use power model to estimate the power consumption. At the same time, we measure the real power consumption of R710 server with the same frequencies and utilizations of processors and results are shown in Table 2. We use the Aitek AWE 2101 power analyzer to measure power. Table 2 shows that the estimated power is very close to the real

Table 2: Power comparison.

Frequencies (GHz)		Utilizations		Power (Watt)		Error
CPU_1	CPU_2	CPU_1	CPU_2	Estimation	Real	
1.60	2.00	61%	87%	145.33	147.01	−1.14%
1.73	2.13	95%	84%	156.45	157.81	−0.86%
1.86	1.73	19%	14%	117.50	117.03	+0.40%
2.00	1.60	59%	9%	127.96	128.32	−0.28%
2.13	2.13	61%	84%	155.67	155.68	+0.00%
2.13	1.86	58%	57%	141.94	142.68	−0.52%
2.26	2.00	58%	30%	139.10	137.89	+0.87%
2.26	2.39	98%	81%	178.21	181.69	+1.91%
2.39	1.60	92%	18%	150.87	149.79	+0.72%
2.39	2.13	12%	59%	133.48	133.15	+0.25%

Table 3: Run-time versus SA iterations.

SA iterations	2 CPUs with 12 VMs		4 CPUs with 24 VMs	
	RT (ms)	EC (J)	RT (ms)	EC (J)
0	0	2466027	0	3210277
10	1	2397905	1	3195957
100	3	2385065	7	3117021
1000	20	2323755	50	3004518
10000	170	2301687	486	2952581
100000	1676	2293194	4895	2923625
1000000	16645	2289410	46070	2851156

Table 4: Run-time versus VDS iterations.

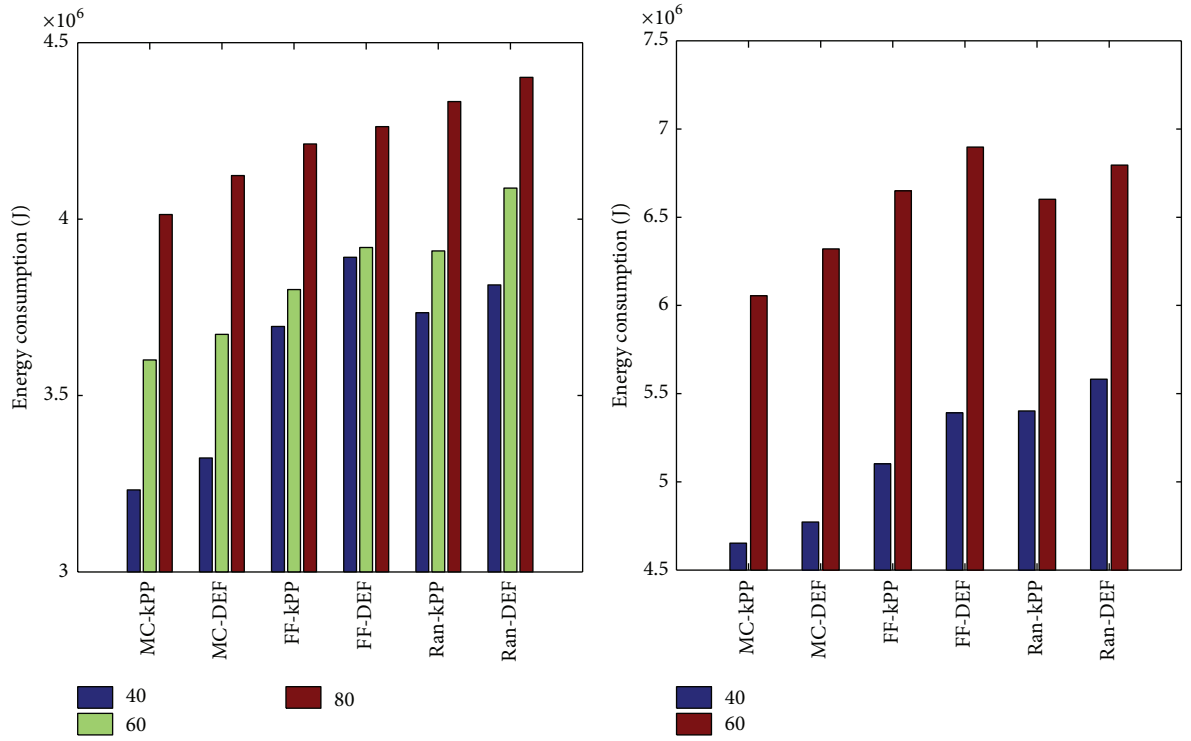
VDS iterations	2 CPUs with 12 VMs		4 CPUs with 24 VMs	
	RT (ms)	EC (J)	RT (ms)	EC (J)
0	0	2466027	0	3210277
10	5	2379716	12	3080983
100	38	2302226	118	2926700
1000	361	2293796	1145	2859979
10000	3519	2289368	11682	2855693
100000	35245	2289438	115659	2843434
1000000	349465	2289410	1140554	2848026

power consumption of the server with the same utilizations and frequencies of different processors.

7.2. Convergence Speed. In this subsection, we compare the convergence speeds of the two algorithms and present them in Tables 3 and 4. The reported run-times and iteration times are for running the two algorithms of a synthetic 2-processor and 4-processor with 7 frequency levels machine where 12 and 24 VMs are executed in parallel. The RT and EC in Tables 3 and 4 represent running time and energy consumption, respectively. We can draw three obvious conclusions: (1) the SA-based algorithm iterates significantly faster than the VDS-based algorithm which means that more iterations can be executed during the same period; (2) the VDS-based algorithm outperforms the SA-based algorithm while it leads to the same iteration times if the host is equipped with serval processors; (3) when the processor number and frequency levels are relatively small, both the two algorithms converge rapidly and obtain the close results. And the VDS-based algorithm perform better than SA-based algorithm when the number of processors is small. This conclusion suggests that the service provider may prefer the SA-based algorithm if they persist in finding the best frequency configurations.

Notice that, in this experiment, we use a very extreme setup where a 4-processor host is enforced to run as much as 24 VMs at the same time, which means a processor must be responsible for 6 VMs on average. In fact, in the real datacenters, it can be expected that the average VM number on a single processor is far less than 6. Thus our algorithm can run efficiently enough to serve for our online VM scheduling algorithm and obtain an accepted result within 1 second which is close to results of more iterations.

7.3. Experiments in Real Environment. Our real experimental environment has three servers and a controller on a virtual machine. The power is measured by Aitek Power Analyzer AWE2101. Each R720 server whose details are shown in Table 1 runs the kPP algorithm to predict energy and control processors' speed. We combine the kPP algorithm with the random (Ran) and first-fit (FF) VM scheduling. The *random* scheme allocates the coming task to the processor randomly from the subset of processors which can offload the new task without causing any violations of QoS requirements. The *first-fit* scheme gives each processor an index and allocates the coming task to the processor with smallest index who can offload the new task without causing any violations of QoS requirements. Meanwhile, the proposed global assignment (MC) is also combined with the default DVFS controller Ondemand [38] (DEF) in Linux. The two-tier energy-aware resource management proposed in this paper is represented by *MC-kPP*. We compare these six strategies to evaluate the performance of our solution on energy savings. For each VM, its execution time is generated uniformly at random between a *minimum* and *maximum* living time represented by $ET_{\min}$ and $ET_{\max}$, respectively. The deadline of a VM is set from 1 to 1.5 times longer to its execution time randomly. Moreover,



(a) $ET_{\min} = 600, ET_{\max} = 3600$ (b) $ET_{\min} = 600, ET_{\max} = 7200$

Figure 5: Energy consumption of real system. The legend in (a) and (b) means that the number of VMs needs to be allocated.

their utilizations requirements follow the normal distribution with $\mu = 0.75$ and $\sigma = 1$. In addition, the arriving times of VMs follow a Poisson distribution with different average rates.

In these experiments, the iteration times are 10000 for SA-based kPP and 1000 for VDS-based kPP and the number of neighbors in VDS-based kPP algorithm is 20. The results of SA-based and VDS-based algorithms are very close, so we show the results of VDS-based kPP algorithm in Figure 5 whose legend represents different numbers of VMs. Meanwhile, the size of candidates in MC algorithm is equal to the number of servers. As we can see in Figure 5, the energy savings of our solution can reach from 8% to 17% in the real environment with 3 servers.

7.4. Simulation Results. Due to the inaccessibility of a large scale datacenter, we conduct the simulations to evaluate MC-kPP solution in a larger cluster. We model Dell R710 servers to service the dynamically arriving VMs. Meanwhile, the attributes of generated VMs are the same as the attributes introduced in Section 7.3.

As we can see in Figure 6(a), the local kPP algorithm can reduce energy consumption of a specific server compared to Ondemand strategy. In addition, the influences of global scheduling algorithm are greater than the influences of local DVFS controller on energy savings when different scheduling algorithms are applied. The lengths of VMs in Figure 6(b) are generated uniformly and randomly between 600 and 7200 seconds. The legend in Figure 6(b) represents the arriving ratio of VMs in one minute. The results show an increasing tendency of the energy saving ratio with the increments of VM numbers and the best result can reach about 28%. In Figure 6(c), we investigate the influence of lengths of VMs; VMs are generated in different lengths which are shown in the legend. As shown in Figure 6(c), MC-kPP can also save more energy when the VMs become more. At the same time, MC-kPP performs better when the average execution time of VMs becomes longer, because the influence of local kPP algorithm itself becomes smaller and the effectiveness of frequencies scaling becomes more obvious. In addition, the size of subset in MC algorithm is also investigated in Figure 6(d); the energy consumption of kPP algorithm in local machine is below 0.5% of total energy consumption when the average execution time is long. With the increment of subset size, the performance of MC-kPP is improved because a better server can be found in a larger scale. We also evaluate the effectiveness of MC-kPP in different scales of data centers ranging from 50 to 5000 servers with different features of arriving VMs. According to the results of Figure 7, the MC-kPP outperforms other strategies in different scales of datacenters, which can reach about 25% energy savings. With the increasement of host numbers and VM numbers, MC-kPP performs stably in different scenarios.

8. Conclusions

In this paper, we propose a cooperative two-tier energy-efficient strategy to manage the VM allocations and adapt

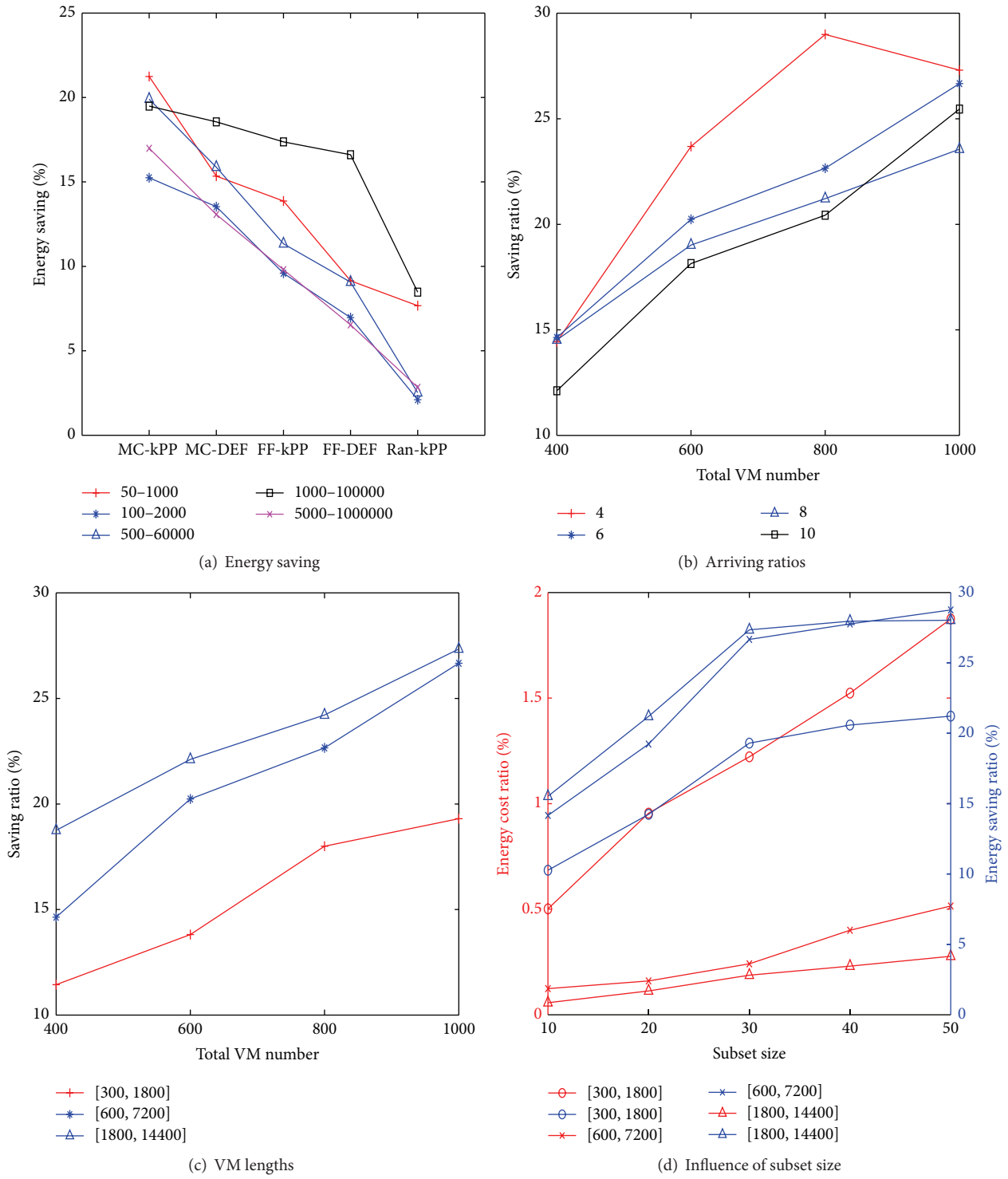


FIGURE 6: Performance evaluation on different aspects. The legend in (a) represents the "server numbers-total VM number." The legend in (b) represents the "VM request number arriving in a minute." The legend in (c) and (d) represents the "(minimum living time, maximum living time)."

frequencies scaling for saving energy. A frequency scaling algorithm is proposed based on the practical power and energy prediction. The Global Scheduler collaborates with local DVFS controller to assign VMs and save overall energy. In addition, two heuristic algorithms are provided for searching the optimal solutions which predict minimum energy consumption. The time complexities of both the algorithms are acceptable with satisfactory results according to the

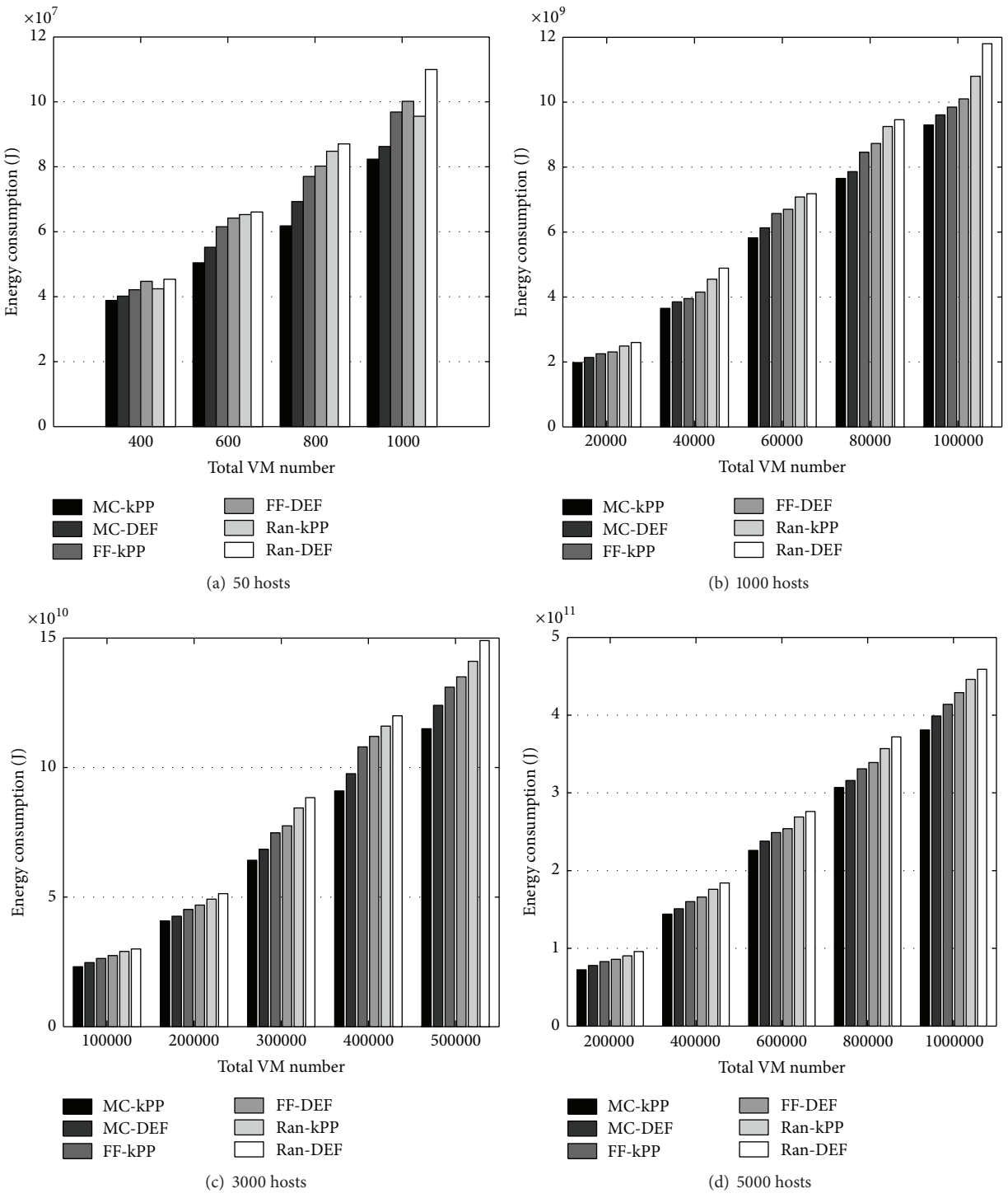


Figure 7: Energy consumption of different scale of datacenters with different number of VMs.

experiments. Finally, the real experiment results justify the effectiveness of MC-kPP.

Notations

J_j: The set of jobs (VMs) in jth host
C_j: CPU set of jth host
F_j: All possible combinations of frequencies for CPUs on jth host
$F_{j,k}$: Combinations of frequencies for CPUs on jth host in kth
$U_{j,k}$: Utilizations of CPUs of jth host in kth FSU
$U_{j,k}^c$: Utilization of cth CPU of jth host in kth FSU
$T_{j,k}$: Time length of kth FSU of jth host

$N_{j,p}$: The number of FSUs from the current time of jth host

$P_{j,k}(\cdot)$: The power function of jth host in kth FSU.

Disclosure

This is a substantially extended version of the paper presented at ICA3PP 2015 [3].

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China under Grants nos. 61472181, 61100197, and 61202113; Jiangsu College Natural Science Foundation under Grant no. 14KJB520016; Jiangsu Natural Science Foundation under Grant no. BK20151392; and JSPS KAKENHI Grant no. 16K00117. And this work is also partially supported by Collaborative Innovation Center of Novel Software Technology and Industrialization.

References

[1] W. Forrest, "How to cut data centre carbon emissions?" 2008.

[2] A. Beloglazov, R. Buyya, Y. C. Lee et al., "A taxonomy and survey of energy-efficient data centers and cloud computing systems," *Advances in Computers*, vol. 82, no. 2, pp. 47–111, 2011.

[3] W. Huang, J. Shi, Z. Wang, and Z. Qian, "BiTEM: a two-tier energy efficient resource management framework for real-time tasks in clusters," in *Algorithms and Architectures for Parallel Processing*, G. Wang, A. Zomaya, G. M. Perez, and K. Li, Eds., vol. 9529 of *Lecture Notes in Computer Science*, pp. 494–508, Springer, Berlin, Germany, 2015.

[4] A. Verma, P. Ahuja, and A. Neogi, "pMapper: power and migration cost aware application placement in virtualized systems," in *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware (Middleware '08)*, pp. 243–264, Springer, Leuven, Belgium, December 2008.

[5] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No power struggles: coordinated multi-level power management for the data center," *ACM SIGARCH Computer Architecture News*, vol. 36, pp. 48–59, 2008.

[6] B. Guenter, N. Jain, and C. Williams, "Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning," in *Proceedings of the 30th IEEE International Conference on Computer Communications (INFOCOM '11)*, pp. 1332–1340, Shanghai, China, April 2011.

[7] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," *Cluster Computing*, vol. 12, no. 1, pp. 1–15, 2009.

[8] R. Buyya, A. Beloglazov, and J. Abawajy, "Energy-efficient management of data center resources for cloud computing: a vision, architectural elements, and open challenges," https://arxiv.org/abs/1006.0308.

[9] M. Cardosa, M. R. Korupolu, and A. Singh, "Shares and utilities based power consolidation in virtualized server environments," in *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM '09)*, pp. 327–334, June 2009.

[10] Z. Cao and S. Dong, "An energy-aware heuristic framework for virtual machine consolidation in Cloud computing," *The Journal of Supercomputing*, vol. 69, no. 1, pp. 429–451, 2014.

[11] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "Efficient virtual machine sizing for hosting containers as a service (SERVICES 2015)," in *Proceedings of the IEEE World Congress on Services (SERVICES '15)*, pp. 31–38, New York, NY, USA, June-July 2015.

[12] Z. Zhou, Z. Hu, and K. Li, "Virtual machine placement algorithm for both energy-awareness and sla violation reduction in cloud data centers," *Scientific Programming*, vol. 2016, Article ID 5612039, 11 pages, 2016.

[13] G. Semeraro, D. H. Albonesi, S. G. Dropsho, G. Magklis, S. Dwarkadas, and M. L. Scott, "Dynamic frequency and voltage control for a multiple clock domain microarchitecture," in *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '02)*, pp. 356–367, IEEE, November 2002.

[14] C.-H. Hsu and W.-C. Feng, "A power-aware run-time system for high-performance computing," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC '05)*, IEEE Computer Society, Seattle, Wash, USA, November 2005.

[15] J. P. Halimi, B. Pradelle, A. Guermouche et al., "Reactive DVFS control for multicore processors," in *Proceedings of the IEEE Green Computing and Communications (GreenCom '13)*, pp. 102–109, August 2013.

[16] M. Lim, V. W. Freeh, and D. K. Lowenthal, "Adaptive, transparent frequency and voltage scaling of communication phases in mpi programs," in *Proceedings of the ACM/IEEE Conference on Supercomputing (SC '06)*, p. 14, Tampa, Fla, USA, November 2006.

[17] D. Li, B. R. de Supinski, M. Schulz, D. S. Nikolopoulos, and K. W. Cameron, "Strategies for energy-efficient resource management of hybrid programming models," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 144–157, 2013.

[18] L. Tan, Z. Chen, Z. Zong, D. Li, and R. Ge, "A2E: adaptively aggressive energy efficient DVFS scheduling for data intensive applications," in *Proceedings of the IEEE 32nd International Performance Computing and Communications Conference (IPCCC '13)*, pp. 1–10, IEEE, San Diego, Calif, USA, December 2013.

[19] L. Wang, G. Von Laszewski, J. Dayal, and F. Wang, "Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS," in *Proceedings of the 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid '10)*, pp. 368–377, IEEE, Melbourne, Australia, May 2010.

[20] H. Kimura, M. Sato, Y. Hotta, T. Boku, and D. Takahashi, "Emprical study on reducing energy of parallel programs using slack reclamation by DVFS in a power-scalable high performance cluster," in *Proceedings of the IEEE International Conference on Cluster Computing (Cluster '06)*, pp. 1–10, September 2006.

[21] S. U. Khan and I. Ahmad, "A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 3, pp. 346–360, 2009.

[22] M. Mezmaz, N. Melab, Y. Kessaci et al., "A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud

computing systems," *Journal of Parallel and Distributed Computing*, vol. 71, no. 11, pp. 1497–1508, 2011.

[23] M. A. Awan and S. M. Petters, "Energy-aware partitioning of tasks onto a heterogeneous multi-core platform," in *Proceedings of the IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS '13)*, pp. 205–214, Philadelphia, Pa, USA, April 2013.

[24] J.-J. Chen, A. Schranzhofer, and L. Thiele, "Energy minimization for periodic real-time tasks on heterogeneous processing units," in *Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS '09)*, pp. 1–12, IEEE, Rome, Italy, May 2009.

[25] H.-R. Hsu, J.-J. Chen, and T.-W. Kuo, "Multiprocessor synthesis for periodic hard real-time tasks under a given energy constraint," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '06)*, pp. 1061–1066, European Design and Automation Association, Munich, Germany, March 2006.

[26] W. Y. Lee, "Energy-saving DVFS scheduling of multiple periodic real-time tasks on multi-core processors," in *Proceedings of the 13th IEEE/ACM Symposium on Distributed Simulation and Real-Time Applications (DS-RT '09)*, pp. 216–223, IEEE Computer Society, October 2009.

[27] J. Luo and N. K. Jha, "Power-efficient scheduling for heterogeneous distributed real-time embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 6, pp. 1161–1170, 2007.

[28] M. Etinski, J. Corbalan, J. Labarta, and M. Valero, "Understanding the future of energy-performance trade-off via DVFS in HPC environments," *Journal of Parallel and Distributed Computing*, vol. 72, no. 4, pp. 579–590, 2012.

[29] L. Tan, Z. Chen, Z. Zong, R. Ge, and D. Li, "A2E: adaptively aggressive energy efficient DVFS scheduling for data intensive applications," in *Proceedings of the IEEE 32nd International Performance Computing and Communications Conference (IPCCC '13)*, pp. 1–10, San Diego, Calif, USA, December 2013.

[30] M. A. Blackburn, *Five Ways to Reduce Data Center Server Power Consumption*, Green Grid, 2008.

[31] E. N. (Mootaz) Elnozahy, M. Kistler, and R. Rajamony, "Energy-efficient server clusters," in *Power-Aware Computer Systems*, B. Falsafi and T. N. Vijaykumar, Eds., vol. 2325 of *Lecture Notes in Computer Science*, pp. 179–197, Springer, Berlin, Germany, 2003.

[32] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA '07)*, pp. 13–23, ACM, June 2007.

[33] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, "Optimal power allocation in server farms," in *Proceedings of the ACM 11th International Joint Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '09)*, vol. 37, pp. 157–168, Seattle, Wash, USA, June 2009.

[34] T. Mudge, "Power: a first-class architectural design constraint," *Computer*, vol. 34, no. 4, pp. 52–58, 2001.

[35] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[36] J. Hromkovič, *Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*, Springer Science & Business Media, 2013.

[37] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[38] V. Pallipadi and A. Starikovskiy, "The ondemand governor," in *Proceedings of the Linux Symposium*, vol. 2, pp. 215–230, Ottawa, Canada, 2006.