Energy Aware Task Scheduling in Data Centers*

Weicheng Huai, Zhuzhong Qian[†], Xin Li, Gangyi Luo, and Sanglu Lu
State Key Laboratory for Novel Software Technology
Department of Computer Science and Technology, Nanjing University, Nanjing 210023, P.R.China
huaiweicheng@dislab.nju.edu.cn, qzz@nju.edu.cn, lixin@dislab.nju.edu.cn
luogangyi@dislab.nju.edu.cn, sanglu@nju.edu.cn

Abstract

Nowadays energy consumption problem is a major issue for data centers. The energy consumption increases significantly along with its CPU frequency getting higher. With Dynamic Voltage and Frequency Scaling (DVFS) techniques, CPU could be set to a suitable working frequency during the running time according to the workload. On the other side, reducing frequency implies that more servers will be utilized to handle the given workload. It is a critical problem to make a tradeoff between the number of servers and the frequency of each server for current workload. In this paper, we investigate the task scheduling problem in a heterogeneous servers environment. To choose a suitable server among heterogeneous resources, the Benefit-driven Scheduling (BS) algorithm is designed to match the tasks to the best suitable type of server. This paper proved that the task scheduling problem based on DVFS, with the target of minimizing power consumption in a heterogeneous environment is NP-Hard. Then we proposed two heuristic algorithms based on different ideas. Power Best Fit (PBF) is based on a locally greedy manner, it always uses the least power consumption increment placement as its choice. Load Balancing (LB) uses a load balancing way to avoid over-consolidation. LB usually has a better performance than PBF, while PBF is easily turned into an online version. Compared with First Fit Decreasing (FFD) algorithm, the results show that PBF can get 12% to 13% power saving on average and LB are about 14% power saving, although PBF and LB use about 1.3 times number of servers.

Keywords: DVFS, cpufreq, Power Consumption, Request Scheduling, Cluster of Servers

1 Introduction

Nowadays, energy consumption is a major and costly problem in data centers, along with the rising of large-scale Internet services, such as Google search, Facebook, eBay and web 2.0 online video sites. Data centers consume a large amount of energy to maintain these web services, which is an important proportion of the overall cost. Information shows that [2], from 2005 to 2010, data center electricity consumption occupies 1.3% and 2% of total consumption in the world and US, respectively. And a recent report shows that [3], the energy consumption, that is consumed by servers and additional facilities used for cooling and failure-tolerance, is likely to surpass the cost on new servers, which means maintaining current service is even more costly than buying new servers for the providers. In a word, the energy wasting becomes one of the key challenges in modern large-scale data centers.

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, volume: 4, number: 2, pp. 18-38

*This paper is an extended version of the work originally presented at the seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS'2013), Asia University, Taichung, Taiwan, July 2013, titled "Towards Energy Efficient Data Centers: A DVFS-based Request Scheduling Perspective" [1]. This version differs from our previous paper in that it contains an extension of previous algorithms into a two-phase solutions, a frame which are able to deal with cluster of heterogeneous servers.

[†]Corresponding author: Tel: +86-25-89686292, Email: qzz@nju.edu.cn, Web: `http://cs.nju.edu.cn/dislab/qzz/qzz.html`

Generally speaking, the energy consumption in a data center consists of several parts, as follows: servers, storage, network and cooling system. Typically, the greatest power budget has been assigned to servers [2] and the mayor energy consumption device for a single server is the CPU. In data centers, plenty of jobs will be done by some “hot” CPU cores, which makes the cooling system consume more energy to maintain the regular running of data centers. Large-scale Internet services, unlike traditional computational tasks, whose request rates of clients vary along with time, which causes the provision of servers and devices more unpredictable. Many previous works show that, some servers in data centers usually keep a low average utilization, while others are utilized at a high level of frequency. The former servers are a waste of resources, while the latter servers are consuming a massive amount of energy. This phenomenon is caused by the lack of effective cooperative scheduling.

To reduce energy consumption and increase the utilization of servers in data centers, there are two ways. The first method is making consolidation of tasks. Previously, researchers aimed at using the least number of servers (First Fit Decreasing, FFD, which has an approximation of (11/9 OPT + 1) to get the least number of bins [4] in bin packing problem) to achieve power saving, while guarantee the given workload can be finished by the cluster. The second method is using a technique called Dynamic Voltage and Frequency Scaling (DVFS). The technique is a combination of Dynamic Voltage Scaling (DVS) technique and Dynamic Frequency Scaling (DFS) technique [5]. With DVFS technique, CPU of a single server could be set to a suitable working frequency during the running time according to the workload. On the other side, lowering voltage and frequency of CPUs will also result in computing performance degradation. In this paper, we combine these methods together to come up a new solution.

Nowadays cluster of heterogeneous servers respond to multiple requests together. Reducing frequency of servers implies that, more servers will be utilized to handle the given workload. These methods cause that, each server will be consolidated by requests as many as possible. This will make CPUs work at a high level of voltage and frequency, which causes a considerable amount of energy consumption for a single server. Consequently a tradeoff between the number of servers and the frequency of each server must be made for the current workload, while servers could maintain the regular running of the given workload to maintain clients’ need.

In this paper, we investigate the task scheduling problem in a heterogeneous servers environment, with the target of minimizing energy consumption of a cluster. One of this typical application is web application service deployed in a cluster. The cluster manager will use our solutions to allocate requests and applications into different types of servers. Our solutions consist of two phases. Firstly, according to the types of servers, we design a Benefit-driven Scheduling (BS) algorithm to classify the requests, such that each request category is associated with one type of server. Then we propose two heuristic algorithms to make the tradeoff between the number of servers and the frequency of each server. The algorithms are denoted as Power Best Fit (PBF) and LB(Load Balancing), to place the requests in a homogeneous server set. PBF is based on a locally greedy method, it always uses the least power consumption increment placement as its choice. LB uses a load balancing way to avoid over-consolidation. LB usually has a better performance than PBF, while PBF is easily turned into an online version. Finally, we compare our solutions with FFD algorithm and get a result that the least number of servers often can not get the most power saving. The main contributions of our paper can be summarized as follows:

1. We use a benefit-driven method to solve the requests classifications problem on heterogeneous servers.

2. We propose an task scheduling problem that dynamically places requests into heterogeneous servers based on DVFS and prove the problem is NP-Hard. Then we present two heuristic algorithms based on different ideas.

3. We make extensive simulations, and the results show that our approaches could get significant

power savings compared to the traditional method, which aims at using the least number of servers.

The rest of the paper is organized as follows. Section 2 reviews related work. Typical scenario, preliminaries, problem statement and proof of hardness are given in Section 3. The corresponding algorithms are proposed in Section 4. Section 5 presents the simulation results. Finally, we conclude the paper in Section 6.

2 Related Work

The energy consumption problem in data centers has received more and more attentions in recent years. There have been extensive works contributing to the problem in different views.

Dynamic Voltage and Frequency Scaling (DVFS) is implemented in both hardware and software ways. Since hardware method needs more specific jobs, we use the software way called cpufreq to build our experiment environment in this paper. cpufreq is a subsystem of Linux kernel 2.6.0, which can be used to dynamically scaling processors's frequency according to the current CPU's load. There are five common governors, which are Performance, Powersave, Userspace, Ondemand and Conservative governors[6]. Performance governor always keeps the CPU at the highest possible frequency point within a user-specified range. It is the least energy efficient way in these governors. Powersave governor is opposite to Performance governor, it always keeps the CPU at the lowest possible frequency within a user-specified range. It may reduce CPU performance when workload is high since Powersave governor keep the frequency at a low level. At the same time, it is the most energy efficient among governors. Userspace governor exports the available frequency information to the user level and permits user-space control of the CPU frequency. It allows you to set the level of frequency manually. All user-space dynamic CPU frequency governors use this governor as their proxy. We use this governor to run experiments in this paper. Ondemand governor was introduced into Linux kernel in 2.6.10. It is rooted in a basic idea of allocation of resources on demand. The controller will check the utilization of CPU and dynamically increase or decrease the level of frequency to maintain the utilization within a user-specified range (a relatively high range at most time). Conservative governor is a fork of the ondemand governor with a slightly different algorithm to decide on the target frequency since Linux kernel 2.6.12. It allows to scale the level of frequency in a gradual speed.

Researches based on DVFS techniques can be classified as interval-based, inter-task and intra-task [7]. Interval-based approaches are mainly used in a single server. It uses the historical data of CPU utilization to predicate the CPU utilization of the next interval and dynamic scale the CPU voltage and frequency according to the predicated CPU utilization [8]. Wierman et al. and Andrew et al. [9][10] give a theoretic certification that online algorithms can not be better than 2-competitive than its offline edition interval-based approaches. Inter-task and intra-task approaches are designed for data centers such as we do. Inter-task approaches concentrate on dividing various types of tasks into servers running in different speeds [11][12]. By contrast, intra-task approaches give a solution more fine-grained than inter-task within a single task as it splits the whole program into small pieces each could be computed in its own slots [13][14]. The grains concentrated on between the latter two manners differ. Intra-task will be more suitable for high performance computing.

The notion of power aware (proportional) computing is not a new conception since it has been used in many embedded systems and mobile computing for a long time. Since the total energy can be used in these scenarios is limited by the electric discharge device. Many researchers have carried out lots of work on the topic [15][16]. The power aware computing idea lies in the roots of DVFS technique and Varying on Varying off (VOVO) method. VOVO method is the traditional method preparing the number of servers to adjust a given workload in the scenario where servers are homogeneous. M. Elnozahy et al. [17] argue that the combination of DVFS and VOVO could get a better result than applying them alone.

In high performance computing (HPC) field, the solutions emphasized on tasks spilt or job allocation to maintain each segment's deadline. For example, S. K. Garg et al. [18] contribute to allocate jobs in distributed data center and take greenhouse gases emission and economic cost into consideration. Lawson et al. [19] designed a scheme to switch CPU into "sleep" mode according to CPU utilization to achieve power saving. There are some methods in [20] to allocate a task to a cloud server with minimum energy, which has mainly concentrated on tasks which has deadlines. In [21], a two-level scheduling method is introduced, which has taken visualization into consideration.

Recently web service deployed in power proportional clusters has also been attracted by many researchers. S. Srikantaiah et al. [22] deployed web services on heterogeneous clusters. A new server will be turned on when there is no machines could accommodate current requests and servers will be turned off when they are idle. Some researchers do their work on the objective of minimizing mean response time of web workload[23]. Andrew Krioukov et al. [24] design a power proportional cluster consists of different servers in various platforms, which are traditional servers, mobile platforms and so on. They use greed methods to place the requests. Tomoya Enokido et al. [25] design a laxity-based algorithm to select a server in clusters of servers. Researches based on queue theory have also used for discussing in [26]. This paper also mainly focuses on the web requests since web workload are getting more and more popular.

3 Problem Statement

In this section, we first present a typical scenario to explain the motivation. Then we give the preliminaries of our problem. Finally, we give a formulation of the problem and analyze its hardness.

3.1 Typical Scenario

There are multiple servers which are placed in units of racks in data centers. For a single server, the power consumption consists of two parts, a constant part and a variable part. The constant part indicates the basic energy consumption running a server and the variable part varies along with the CPU working frequency and voltage, whose value is scaled according to the current load.

One of the typical applications in data centers is web application. The scheduling of web applications is different to traditional jobs, which is limited in computing cells in order to generate computational results. We mainly concentrate on web service applications, which is deployed in a traditional three-tier architecture, similar to [27], [24], as shown in Figure 1. Requests are submitted to the front-end servers, which run cluster manager programs to manage the cluster. Our algorithms are running on the front-end server as a part of cluster manager. In the middle layer, cluster servers conduct computation to respond to the requests. The cluster servers communicate with the storage layer to get the data they need to compete the jobs. Thus, our solutions are different from them.

3.2 Preliminaries

DVFS could reduce power consumption of servers significantly by changing the frequency and voltage of CPU, using cpufreq or other technologies. Meanwhile the computing performance will be reduced since the working frequency of CPU core is decreased. The computing performance of servers can be decided not only by CPU core but also registers, memory and so on. It means that, the computing performance will not scale as we scale the frequency proportionally[7]. For example, we lower down the frequency of CPU to half, while the computing performance will be a little higher than half since other computing components, such as registers, do not have the similar half effect. The distance between proportional

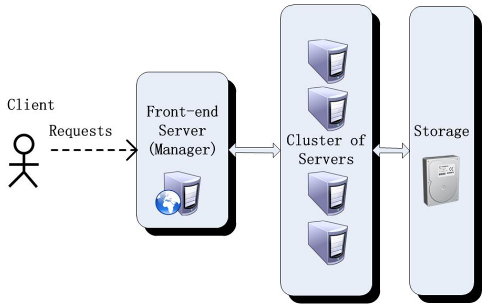


Figure 1: The architecture

value and realistic value will be different since the workloads and CPUs are different. Previously, researchers have used a more accurate equation between performance and frequency, which was proposed by Hsu et al. [28]:

$$T(f_i) = T(f_{max}) \times \left(\gamma^{cpu} \times \left(\frac{f_{max}}{f_i} - 1\right) + 1\right) \tag{1}$$

where $T(f_i)$ is the execution time of the application when the CPU frequency's value equals f_i and $T(f_{max})$ is the execution time of the application at the maximum CPU frequency, denoted as f_{max}. γ^{cpu} is the CPU boundness of the application. There are typical multimedia corresponding CPU-boundness for each benchmarks in the SPECfp95 benchmark suite in [28]. For instance, Etinski et al. [29] and Freeh et al. [30] use a fixed γ^{cpu} value to compute the power consumption of CPUs. Obviously the proportional metrics sets the $\gamma^{cpu} = 1$. In this paper, we make the assumption that the scaling is proportional for simplicity, similar to S. K. Garg et al. [18] since the influence is little or we can just replace the proportional method into more specific function in the algorithms as this is not the key point of the paper.

Generally, the power consumption model of a CPU using DVFS is given as:

$$Power = I_{leakage}V + \beta V^2 f + Power_{short} \tag{2}$$

where $Power$ is the power consumption of a CPU measured with $Watt$ (W), V is the supply voltage, $I_{leakage}$ is the leakage current and f is clock frequency of CPU, respectively [31]. The last term $Power_{short}$ means power dissipated when CPU is in voltage switching period and is generally negligible. As there is a relation between clock frequency and voltage which has generally used a linear function. Thus the power consumption can be given by:

$$Power = \alpha + \beta f^3 \tag{3}$$

The parameters α and β are constants, which are used to fit the relation and can be various among different servers. In simulations, we will use realistic real data to fit the curve and get the parameters in Section 5.1.

In a web service scenario, request rates vary along with time, which makes the computing capacity of this application changes. Our target is to deal with each request at a proper level of frequency on servers without violation of the guarantee of QoS. In other words, we should get as more energy saving as we can, which is to decrease the level of frequency, while preserving acceptable performance. We demonstrate an example shown in Figure 2, where S_1, S_2, S_3 are totally homogeneous servers and R_1, R_2, R_3 are also the same requests, which are Home browsing behaviors in Section 5.1.1 [32]. The only difference is that the level of frequency of each server differs from the another, which makes a different result in QoS (here we consider "delay", one of the most important parameters).

In order to get acceptable performance when we slow down the frequency, we must specify a baseline, which can be given by QoS generally, although QoS may also be influenced by other factors (mainly

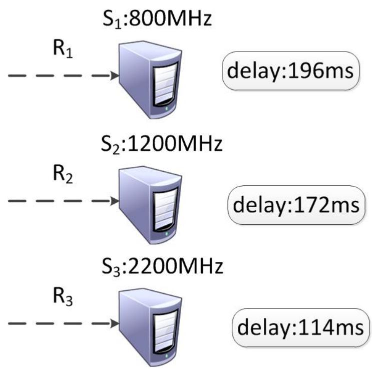


Figure 2: The delay of requests which are at different levels of frequencies in same servers, the delay data come from Section 5.1.1, which are the average delay of Home browsing behavior in [32].

network), which is out of the scope of this paper. In this paper, we make an assumption for simplicity: dynamically adjusting the frequency level according to the request rate can get nearly execution time. For example, the mean execution time of 10 requests at a frequency of 1GHz equals to the mean execution time of 20 requests at a frequency at 2GHz. In this paper, we do not take network factor into consideration, and the mean execution time of requests indicates the delay of requests. Thus the proportional adjustment of frequency according to request rate does little to delay, which is a major parameter of Qos. The experiment, which is used to validate the basic idea, is shown in Section 5.1.1. To get a better realistic situation, plenty of experiments on various services and applications should be done and get a more proper and specific model, which is beyond this paper's scope. We use one typical web service model, RUBiS[32], to support our views.

3.3 Formulation

The scheduling problem consists of two parts. Firstly, we should confirm the suitable type of server to handle the requests among heterogeneous servers, which is supported by some knowledge of simple experiments or historical data. This is very essential since different types of servers differ a lot in energy-efficiency, throughput, start time of server. A proper type of server will be chosen to handle the request after the first phase. Then, phase two is designed for scheduling requests into servers at a specific level of frequency in homogeneous environment.

Here are some assumptions we make:

1. The computing capacity is considered as the application's resource demand and do not take other resources, such as memory, into consideration as we model the application CPU-driven tasks, such as high levels of concurrent web requests.

2. The computing capacity is proportional to the frequency of CPU.

3. A proportional method of frequency to request rates can get nearly execution time of requests.

There is a simple demonstration to help readers understand better. When a new server is turned on, we think the performance of the server is not occupied and denote the server a full size. Then some requests are directed into the server, DVFS is used to scale the level of frequency of the server, which makes the requests achieve a QoS guarantee. The ratio of the current frequency and full frequency is considered as the occupied size of this request. The remaining size of server is the full size subtracts the

Table 1: Notations

Notations	Notes
V_i	The remaining size of the i^{th} server
v_j	The size of the j^{th} request
v_{kj}	The size of the j^{th} request when the type of v_j is given by k
$Benefit_{kj}$	The benefit of the placement of j^{th} request into k^{th} type of server
$Power_i$	The power consumption of the i^{th} server
$Power$	$Power = \sum_{i=1}^{n} Power_i$
$\mathbf{R} = [v_{kj}]_{t\times n}$	$v_{kj} = 1$, v_{kj} is put in k^{th} type of server; otherwise, $v_{kj} = 0$
$\mathbf{A} = [a_{ij}]_{m\times n}$	$a_{ij} = 1$, v_{kj} is put in i^{th} server; otherwise, $a_{ij} = 0$
m	number of current servers which are on

occupied size. For different types of servers and requests, the normalization of the size should be done before implementing the algorithms. The notations in formulation and solutions are shown in Table 1.

The first phase is a selection of a specific type of server for each request. In this paper, we use a benefit-driven method to make the choice among different servers, as shown in Algorithm 1. Different types of servers differ a lot in many aspects, including power model, peak throughput, energy efficiency and so on. It seems that, even at the same frequency settings different types of servers will get different results. Some types of traditional business servers may use as many as several times of power consumption of atom mobile servers. We can denote the benefit function $Benefit_{kj}$ of request v_j placed into the type k server as:

$$Benefit_{kj} = f(power\ model_k, peak\ throughput_k, energy\ efficiency_k, ...v_j) \tag{4}$$

Since different types differ a lot in these parameters. For example, a Nehalem Server (2x Intel Xeon X5550 Quad Core) can handle requests at a rate of $340 req/s$ with power 248 $Watt$, while Atom Mobile can just only deal with requests at $35 req/s$ with power $28 Watt$ [24]. Thus, applications which need a high level of computing resource, prefer to use traditional business servers, because traditional servers could handle effectively and have a better energy efficiency per req. However, small and agile requests are better if we make them be handled by Atom Mobile or Embedded BeagleBoard platforms. In this paper, we design an order, which indicates the order of priority of each request placed into each type of sever. Each request chooses the top priority type of server as the placement destination.

Phase 1: Heterogeneous Server Type Selection.

Given a set R of n requests with sizes v_j ($j \in J = \{1,2,...,n\}$), and t types of CPUs. The target is to redirect these requests into a proper type of server, which is the most "suitable" type according to the request. In other words, the target is to find a partition(packing) of R into R_1, R_2, ..., R_t. The objective is to maximize the value of $Benefit_j$.

$$\begin{aligned}
\textbf{max.}\ & Benefit_j = Max(Benefit_{kj}) \\
\textbf{s.t.}\ & v_j = \sum_{k=1}^{t} v_{kj} \\
& where\ \ v_{kj} = \begin{cases} v_j, & if\ the\ j^{th}\ request\ is\ placed\ in\ the\ k^{th}\ type\ of\ server; \\ 0, & otherwise. \end{cases}
\end{aligned} \tag{5}$$

$k \in K = \{1, 2, ..., t\}$.

After we make a suitable choice for each requests we move to the phase two. We give a problem statement about the second phase.

Phase 2: Task Scheduling in Homogeneous Servers

Given a set R_k (one of item in R_1, R_2, ..., R_t)of n_k (n_k is the proportion of n into the k^{th} type of servers) requests with sizes v_{kj} ($j \in J = \{1, 2, ..., n\}$), and an unlimited number of identical 1-dimensional servers (bins) having sizes V. Each server can use DVFS technology to accommodate the request according to the computing capacity in a self-adaption way. The target is to find a partition (packing) of R_k into P_1, P_2, ..., P_m. The objective is to minimize the total value of Power.

$$\begin{aligned}
\textbf{min.}\ & Power = \sum_{i=1}^{m} Power_i \\
\textbf{s.t.}\ & Power_i = \alpha + \beta \left(\sum_{j=1}^{n} a_{ij} v_{kj} \right)^3 \\
& V_i = \sum_{j=1}^{n} a_{ij} v_{kj} \le V \\
& where\ \ a_{ij} = \begin{cases} 1, & if\ the\ j^{th}\ request\ in\ k^{th}\ type\ of\ server\ is\ placed\ in\ the\ i^{th}\ server; \\ 0, & otherwise. \end{cases}
\end{aligned} \tag{6}$$

for all P_i, $i \in I = \{1, 2, ..., +\infty\}$.

Here α, β are stated in Section 3.2. In this formulation, k is a given value, we use v_j instead of v_{kj} for short in Algorithm 2 and Algorithm 3 of Section 4.

3.4 Hardness Analysis

It is easy to notice that the difference between the Task Scheduling in Homogeneous Servers and 1-dimensional Vector Bin Packing Problem [1] lies in the objective function. The former aims to achieve the least number of bins (servers in our settings) while the latter aims to get a minimal total power consumption of all servers. Then, we give the theorem and its proof.

Theorem 1. *The task scheduling problem in a heterogeneous servers environment is NP-hard.*

Proof. We prove this theorem via a special case, let all of the servers are homogeneous. Suppose the power consumption of each server (bin) in the Task Scheduling in Homogeneous Servers in Phase two is the same constant value c. That is, the power consumption of a single server equals a constant regardless of load on it. So the objective of the minimum power consumption equals the minimum number of servers (bins) logically. As we all know, the 1-dimensional Vector Bin Packing Problem is NP-hard [33], then the special case of our problem is also NP-hard. Hence, the task scheduling problem is NP-hard. □

4 The Task Scheduling Algorithms

In this section, we propose three algorithms to solve the placement of request into heterogeneous servers. According to our design, we partition the problem into two phases. In the first phase, we design a Benefit-driven Scheduling (BS) algorithm to assign requests into different types of servers. Then we propose two

[1]Given n items with sizes $v_1, v_2, ...v_n \in (0, 1]$, find a packing method in unit-sized bins to minimize the number of bins used.

heuristic algorithms to solve the problem of power consumption explosion in phase two. We denote them as Power Best Fit (PBF) and Load Balancing (LB). Power Best Fit (PBF) is based on a locally greedy method, it always uses the least power consumption increment placement as its choice. Load Balancing (LB) uses a load balancing way to avoid over-consolidation.

4.1 Benefit-driven Scheduling Algorithm

Algorithm 1 Benefit-driven Scheduling (BS) Algorithm

```
Input: $v_1, v_2, ...v_n, \alpha_1, \alpha_2, ...\alpha_t, \beta_1, \beta_2, ...\beta_t, where\ t\ is\ the\ amount\ of\ heterogeneous\ server\ types.$
Output: $\mathbf{R} = [v_{kj}]_{t \times n}$.
 1: for $j \leftarrow 1$ to $n$ do
 2:     for $k \leftarrow 1$ to $t$ do
 3:         Calculate benefit of $Benefit(v_{kj})$;
 4:     end for
 5:     $K_j \leftarrow argmax(Benefit(v_{kj}))$; /*chose the maximum one as the type*/
 6:     $v_{kj} \leftarrow 1$;
 7:     Redirect the $j^{th}$ request $v_j$ to $K_j^{th}$ type of server; /*request redirection*/
 8: end for
 9: for $k \leftarrow 1$ to $t$ do
10:     PBF($\{v_j | where\ K_j = k\}, V = \{V_1, ...V_i, ...V_m\}, \alpha_k, \beta_k$) or
        LB($\{v_j | where\ K_j = k\}, V = \{V_1, ...V_i, ...V_m\}, \alpha_k, \beta_k$);
11: end for
12: return R;
```

The Benefit-driven Scheduling (BS) algorithm is designed to solve the Heterogeneous Server Scheduling Problem. The algorithm will be executed as follows. For each request, we calculate the benefit function of v_{ij}, which denotes the i^{th} request is redirected to j^{th} type of servers (lines 1-4). We use the largest benefit as the choice of the current request v_i (lines 5-8). After we making the choice we use one of the two algorithms PBF and LB to deal with the Task Scheduling in Homogeneous Servers. We denote the requests redirected into one type by an array. The computing capacity of that type can also be denoted by another array. Based on these parameters, We propose two algorithms, PBF algorithm and LB algorithm according to two different policies although part of load balancing use a similar way of PBF to do the second phase optimization. PBF is firstly introduced, which is given in a greedy manner. LB is based on the basic idea of avoiding requests over-consolidation, which leads to “high frequency high power consumption” effect (lines 9-11). The algorithm return the choice matrix **R** (lines 12).

4.2 Power Best Fit Algorithm

In this part, we propose Power Best Fit (PBF).[2] It is based on a greedy manner to place the current requests into servers. PBF is shown in Algorithm 2. The algorithm executes as follows:

1. In the initialization phase, the algorithm sets the total power consumption *Power* and m as well as all the a_{ij} in **A** equals 0, where $i \in I = \{1, 2, ..., +\infty\}$, $j \in J = \{1, 2, ..., n\}$ (lines 1-2).

2. When a request v_j arrives, the algorithm scans the servers which are ON state.

[2]Since k is a given value as we have decided in Algorithm 1, we use v_j instead of v_{kj} for short in Algorithm 2 and Algorithm 3.

- If there is no server could contain the current request v_j, the algorithm turns on a new server to hold the current request v_j. The new server's sequence number equals $m+1$, which means current number of servers added one. We set the corresponding a_{mj} to 1 (lines 3-11).
- If there are some servers could contain the current request v_j, the algorithm uses greedy manners to make the choice. The locally optimal choice method use the least power consumption increment as the most power saving placement target(lines 12-25).

Algorithm 2 Power Best Fit (PBF) Algorithm

```
Input: $v = \{v_1, ...v_j, ...v_n\}, V = \{V_1, ...V_i, ...V_{+\infty}\}, \alpha, \beta$.
Output: $\mathbf{A} = [a_{ij}]_{m \times n}$.
1: $Power \leftarrow 0$; $m \leftarrow 0$;
2: $a_{ij} \leftarrow 0$, for all $i, j$ pairs; /*initialization*/
3: for $j \leftarrow 1$ to $v.size$ do
4:     $i \leftarrow 1$;
5:     while $v_j > V_i$ do
6:         $i \leftarrow i+1$; /*$i^{th}$ server can not contain the current $j^{th}$ request*/
7:     end while
8:     if $i == m$ then
9:         Consolidate the current $j^{th}$ request $v_j$ on the $m+1^{th}$ server; /*a new server*/
10:        $m \leftarrow m+1$;
11:        $a_{mj} \leftarrow 1$;
12:    else
13:        $Power_{Consolidation} \leftarrow min(Power^k_{Consolidation})$; /*chose a minimal power consumption increment*/
14:        $K \leftarrow argmin(Power^k_{Consolidation})$; /*record its sequence number*/
15:        $Power_{Addition} \leftarrow \alpha + \beta v_j^3$;
16:        if $Power_{Addition} < Power_{Consolidation}$ then
17:            Consolidate the current $j^{th}$ request $v_j$ on the $m+1^{th}$ server; /*a new server*/
18:            $m \leftarrow m+1$;
19:            $a_{mj} \leftarrow 1$;
20:        else
21:            Consolidate the current $j^{th}$ request $v_j$ on $K^{th}$ server;
22:            $a_{Kj} \leftarrow 1$;
23:        end if
24:    end if
25: end for
26: Get $Power$ and $m$ using $\mathbf{A}$ ($[a_{ij}]_{m \times n}$);
27: return $\mathbf{A}$;
```

$Power^k_{Consolidation}$ means the power increment when we consolidate the current request in the k^{th} server. We denote $Power_{Consolidation}$ as the minimum $Power^k_{Consolidation}$ which means the most power saving placement of current request v_j is in K^{th} server. $Power_{Addition}$ indicates the power increment when a new appending server is turned ON to hold the current one. Then a greedy choice should be done by comparing them. The algorithm choses the minimum value to decide whether to consolidate current request to K^{th} server or turn on a new server ($m+1^{th}$) to hold current request. In this manner, we always chose the locally minimum one then place the current request into the minimum increment of *Power* as the most "suitable" server.

$$Power_{Consolidation} = min(Power^{k}_{Consolidation}) \tag{7}$$

$$K = argmin(Power^{k}_{Consolidation}) \tag{8}$$

where

$$Power^{k}_{Consolidation} = \beta\,(V_k + v_j)^3 - \beta V_k^3 \tag{9}$$

and

$$Power_{Addition} = Power^{m+1}_{Consolidation} = \alpha + \beta\,(v_j)^3 \tag{10}$$

The power model used here has been stated in Section 3.2.

3. When all requests are handled, the algorithm calculates the consumed power for each server, the number of servers *m* and *Power*, then returns the placement matrix **A** (lines 26-27).

For *n* requests, each request need a comparison of $O(m)$ placement choices. Thus, the complexity of the PBF algorithm is $O(mn)$. It is more complex than a random choice of one server whose complexity of algorithm is $O(n)$, but it will get a better result. And in a more reality environment in data centers, the communication between front-end server and clusters is very usual that we can add the information our algorithm need into the Heartbeat packet. Thus, the complexity is not the main concern we should argue. However, PBF do the placement choice by locally optimal choice and it usually get a result that is not good enough in many situations.

4.3 Load Balancing Algorithm

Load Balancing (LB) algorithm proceeds from the balancing method. From the power model equation, we figure out the most "power frequency ratio" way to measure server's occupied capacity in a relatively high level but not a full level. Traditionally, the server uses the most power consumption per computing capacity in a very low frequency. And the ratio will get lower along with we use more computation capacity. However, the decrement of ratio will not keep forever. When the computing capacity are very high, the ratio will be also getting high since the cube increment of power consumption with frequency. We use a cut-off rule to judge upon the levels of computing capacity. The cut-off level works as follows: when the current capacity of the server does not exceed the cut-off level, we will put more requests into the server as many as possible and we never put more requests into the server when the capacity exceeds the cut-off level. LB is shown in Algorithm 3. The algorithm will execute as follows.

1. In the initialization phase, the algorithm runs the same initialization as Algorithm 2 (lines 1-2).

2. The algorithm investigates the power frequency ratio to get the most powerfrugal setting. As the ratio is not monotone and has a local minimum ratio value at a superior level. The local minimum ratio value is set to be the cut-off level of the algorithm (lines 3).

3. Using the cut-off level the algorithm divides the requests into two parts.

   - The algorithm places each request that are larger than cut-off level into a single server (lines 4-10).
   - Requests below the cut-off level can be estimated at a lower bound by simply dividing total requests by the maximum capacity of a single server (lines 11-14).
   - The algorithm places these requests in a decreasing order (lines 15-18).

Algorithm 3 Load Balancing (LB) Algorithm

```
Input: $v = \{v_1, ...v_j, ...v_n\}, V = \{V_1, ...V_i, ...V_{+\infty}\}, \alpha, \beta$
    $R = \{R_{f_{min}}, ...R_{f_l}, ...R_{f_{max}}\}$ $f_l$ are discrete values of frequencies between $f_{min}$ and $f_{max}, min \le l \le max$.
    $R_{f_l}$ are the ration between power consumption and frequency.
Output: $\mathbf{A} = [a_{ij}]_{m\times n}$.
1: $Power \leftarrow 0$; $m \leftarrow 0$;
2: $a_{ij} \leftarrow 0$, for all $i, j$ pairs;
3: $cut \leftarrow argmax(R_{f_l})$; /*the cut-off level*/
4: for $j \leftarrow 1$ to $v.size$ do
5:     $i \leftarrow 1$; $L \leftarrow 0$;
6:     if $v_j \ge cut$ then
7:         Consolidate the current $j^{th}$ request $v_j$ on $i^{th}$ server; /*the current $j^{th}$ request exceeds the cut-off
           level c, we put it into a server and not matter put other requests into this server*/
8:         $a_{ij} \leftarrow 1$;
9:         $i \leftarrow i + 1$;
10:        continue;
11:    else
12:        $D \leftarrow D + v_j$ ;
13:    end if
14: end for
15: for $j \leftarrow 1$ to $v.size$; do
16:     Prepare $\lceil \frac{D}{cut} \rceil$ servers; /*Get an estimation of the number of servers other requests need*/
17:     Consolidate $v_j$ whose $\sum_{i=1}^{n} a_{ij} == 0$ in a decreasing order;
18: end for
19: while $v_j$ which have not been placed do
20:     PBF($\bigcup v_j, V = \{V_1, ...V_i, ...V_m\}, \alpha, \beta$); /*Use PBF or other methods to deal with these small re-
        quests*/
21: end while
22: Get $Power$ and $m$ using $\mathbf{A}$ ($[a_{ij}]_{m\times n}$);
23: return $\mathbf{A}$;
```

- There are still some requests which are not placed in servers until now. For simplicity, the algorithm uses PBF to deal with these requests since these requests have little effect to the results. We can get a better result of this phase in the future by using more accurate balancing method (lines 19-21).

4. Finally, requests are put into the servers respectively. The algorithm calculates the power for each server, the number of servers m and *Power*, then returns the placement matrix **A** (lines 22-23).

For requests that exceed the cut-off level, the placement complexity is $O(n)$. For requests which are lower, the placement complexity is $O(mn)$. Above all, the complexity of the algorithm is also $O(n) + O(mn) = O(mn)$. We can also use FFD in the lower part of requests instead of PBF, thus the complexity of the algorithm will be nearly $O(n)$, which may lead to a small decrement in optimization effect. However, as we have stated in the Section 4.2 the decrement of complexity of algorithm is not necessary so that we chose a better performance one. LB deals with the placement by a load balancing way and avoids over consolidation efficiently since the most power wasting situation of the problem is the "high frequency high power consumption" effect.

5 Experiments Evaluation

In this section, we use simulation experiments to evaluate the performance of our algorithms. Firstly, we make some experiments to support the assumptions in the Section 3. Then we do the comparisons of several algorithms in phase two. The First Fit Decreasing (FFD) algorithm acts as the benchmark and we compare the Power Best Fit (PBF) algorithm and Load Balancing (LB) algorithm with the benchmark. We will focus on the power totally used each algorithm calculates and the number of servers each algorithm occupies.

5.1 Simulation Setup

5.1.1 RUBiS & cpufreq

RUBiS is an auction site prototype modeled after eBay.com, which is used to evaluate application design patterns and application servers performance scalability[32]. It is an auction site benchmark, which implements the core functionality of a typical auction site including selling, browsing, bidding and so on. A main characteristic of RUBiS is that the obvious partition of three kinds of user sessions, which are visitors, buyers, and sellers. For a visitor session, users need not register but are only allowed to browse. Buyer and seller sessions require registration. By contrast to the functionality provided during visitor sessions, buyers can bid on items and consult a summary of their current bids, rating and comments left by other users during a buyer session. During seller sessions, sellers put up an item for sale and they also have the functionality of buyers.

The RUBiS are implementing for several versions, each of them includes a server end and a client end. The server end is the processing of requests and the client end is an emulator of users behavior for various workload patterns and provides statistics. The benchmark defines 26 different interactions that can be performed from the client's Web browser, some of them are Browse, BrowseCategories, SearchItemsInCategory, ViewItem and so on. Each of them represents one of typical auction process. We use a combination of cpufreq and RUBiS to execute experiments, which aims at showing the effect of DVFS on web service requests. We build a RUBiS environment for the experiments, which includes a server end and a client end. Then we use cpufreq to change the frequency of the server end so that the server handle the client's requests in different levels of frequencies. The client end can record requests' average delay, which is used to show the influence of DVFS on CPU's performance (The network factor are totally the same).

The interface of cpufreq for each CPU will be under sysfs, typically at /sys/devices/system/cpu/cpuX/cpufreq (Ubuntu 12.04), where X ranges from 0 through n-1 (n is the total number of logical CPUs). The files in the directory store the settings of cpufreq, where all the frequency values are in kHz in these files. cpuinfo_max_freq and cpuinfo_ min_freq gives the maximum and minimum frequency supported by the CPU hardware, respectively. And cpuinfo_cur_freq denotes the current frequency of CPU. scaling_available_frequencies lists out all the available frequencies for the CPU, which are steps of frequency levels traditionally. scaling_available_governors lists out all the governors supported by the kernel, which are from the previous five governors. The administrator can also echo a particular available governor into scaling_governor in order to change the governor on a particular CPU. scaling_cur_freq returns the cached value of the current frequency from the cpufreq subsystem by contrast to hardware. scaling_max_freq and scaling_min_freq are user controlled upper and lower limits of frequencies, within which the governor will operate at any time. scaling_driver names the CPU-specific driver which is used to scale the CPU frequency. stats/ is used to store the statistics of CPU information, which are the proportions of each frequency occupies.

The way we use cpufreq is simple. We can use

```
cpufreq-info
```

to see the information of CPUs which includes type, hardware limit, current frequency, available frequency steps, current policy and cpufreq stats.

The dynamic scaling method is implemented by this:

```
cpufreq-set
options:
-f: denotes the frequency users want to set.
-d: denotes the min frequency CPU can achieve by cpufreq.
-u: denotes the max frequency CPU can achieve by cpufreq.
-g: denotes the governor users want to set.
```

Thus, we can use the cpufreq in a very simple way and we will do some experiments to validate our assumption in this paper. The hardware limits the minimum and maximum frequencies to be 800MHz and 2.2GHz, respectively. Thus, we use cpufreq to scale the level of frequencies, which are 800MHz, 1.2GHz, 1.6GHz and 2.2GHz in Table 2. The RUBiS client's end simulates several types of behaviors in traditional auction site browsing process, which are Home, Browse, BrowseCategories, SearchItemsIn-Category, BrowseRegions, BrowseCategoriesInRegion, SearchItemsInRegion and ViewItem.

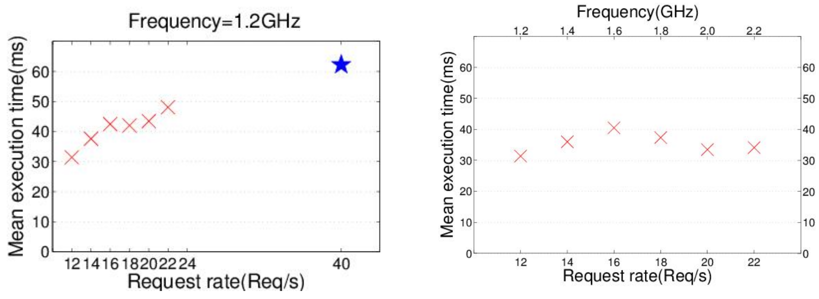
Table 2: Summary of different behaviors at different levels of frequencies

Behavior Names		Frequencies			
		800MHz	1.2GHz	1.6GHz	2.2GHz
Home	Count of requests	2876	2863	2848	2914
	Avg Time (average time of delay, ms)	196	172	118	114
Browse	Count	2962	2933	2921	2993
	Avg Time (ms)	5	6	8	12
BrowseCategories	Count	1974	1987	2019	2053
	Avg Time (ms)	12	15	15	12
SearchItemsInCategory	Count	3725	3768	3838	3881
	Avg Time (ms)	82	85	86	85
BrowseRegions	Count	872	843	806	836
	Avg Time (ms)	13	14	13	14
BrowseCategoriesInRegion	Count	857	831	791	821
	Avg Time (ms)	11	15	16	10
SearchItemsInRegion	Count	1617	1579	1489	1536
	Avg Time (ms)	38	41	39	40
ViewItem	Count	3220	3152	3351	3265
	Avg Time (ms)	129	140	136	131
Total	Count	**18116**	**17968**	**18073**	**18315**
	Avg Time (ms)	**78**	**77**	**69**	**67**

We can see from Table 2: different types of behaviors differ in Avg Time (ms) and the Avg Time decreases as the level of frequency grows. All of the delay includes the network factor such that we can view the difference of the Avg Time as the execution time of CPUs since the amount is large, which makes the network fluctuation effect negligible. Thus, we see the effect of DVFS on computing performance. We can use the data to measure the parameter γ^{cpu} in equation 3.2. As we can see the

γ^{cpu} of RUBiS is about 0.08 to 0.1 although some data may have some deviations, which is a similar to the tomcatv in SPECfp95 of [28]. The bigger γ^{cpu} of application is, the more CPU-instensive the application is.

In data centers, more complex requests than RUBiS are happening in every single day. We design another CPU-instensive experiments whose γ^{cpu} will be close to 1. We explore the quality of requests at different frequencies in Figure 3. In order to eliminate the effect of network delay factor, the request service time here is the execution time, when the server gets the result locally, which is not the request service time traditionally. As shown in Figure 3(a), the mean execution time of requests will increase when request rates increase and CPU is fixed at a specific frequency. When the request rates are big enough (such as point when request rates=40, the blue point in Figure 3(a)), the mean execution time of requests dramatically increases. Figure 3(b) gives the result that dynamic scaling the frequency and voltage the mean execution time of requests will keep steady (The maximum and minimum are both between 30ms and 40ms).



(a) The mean delay (execution time) of requests at frequency=1.2GHz (b) The mean delay (execution time) of requests at different frequencies

Figure 3: The effect of using DVFS on delay

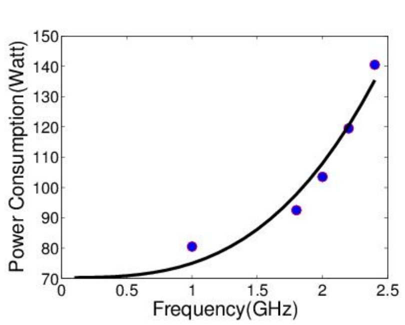
5.1.2 Power Consumption Evaluation

Table 3: Summary of CPUs from different types with their power consumption of different frequencies

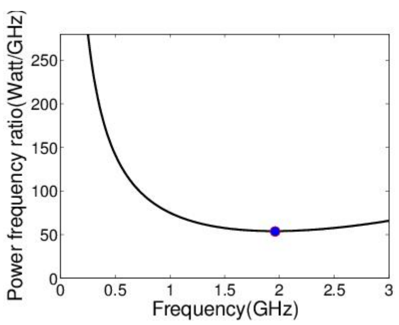
Type	Frequency (GHz)	Power (Watt) [1]
Transmeta	0.333, 0.400, 0.533, 0.677, 0.733	9, 9.5, 10.5, 12, 12.5
Blue	0.8, 1.8, 2.0, 2.2	74.5, 93.5, 105.5, 120.5
Silver	1.0, 1.8, 2.0, 2.2, 2.4	80.5, 92.5, 103.5, 119.5, 140.5
Green	1.0, 1.8, 2.0	77, 108, 131

[1] The Power data here is the value when the CPU is in busy state.

Many researches have used a cubic relation to model power with frequency [34][35]. We have stated in the assumption that we will ignore the power consumption of other parts like [36][17]. There are several types of CPUs listed in Table 3, which are Transmeta, Blue, Silver and Green. In order to implement LB algorithm, we get a power frequency ratio function shown in Figure 4(b) of Silver. Intuitively, the smaller the ratio is, the more power saving we can get. The data used come from the Silver fitting curve of power in Figure 4(a) since it is nearest to the most widely used machines in data



(a) The fitting curve of power consumption and frequency of Silver

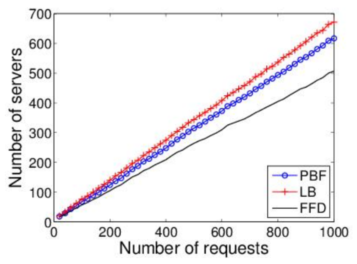


(b) The ration between power consumption and frequency of Silver

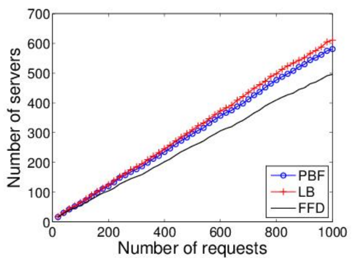
Figure 4: Silver

centers. As we can see in the figure, it is the most power wasting which means we should consolidate small requests when the frequency is very low. The most saving setting of frequency in the scenario is about 1.95GHz (The blue point in Figure 4)(b). When the frequency exceeds that point, the energy efficiency becomes bad. We will use the blue point as the cut-off level of LB algorithm.

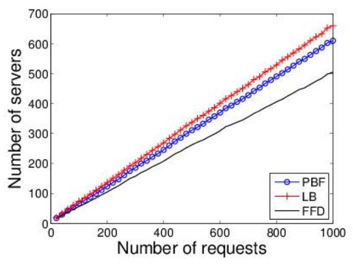
5.2 Experimental Results



(a) number of servers when using Silver. Sample data sets' size is every 20 from 0 to 1000.



(b) number of servers when using Transmeta. Sample data sets' size is every 20 from 0 to 1000.

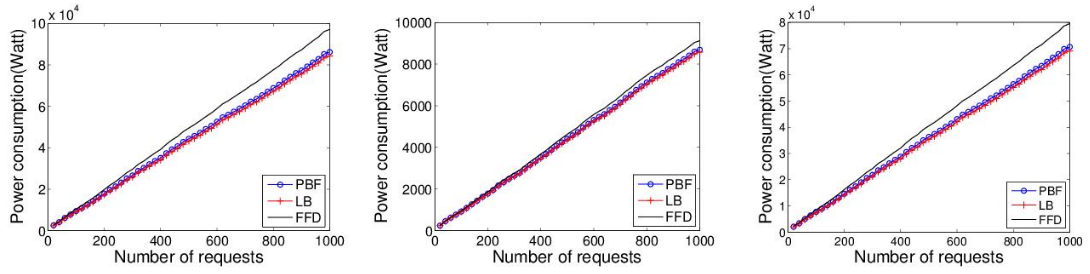


(c) number of servers when using a hybrid scheme of Silver and Transmeta. Sample data sets' size is every 20 from 0 to 1000.

Figure 5: (a) shows the number of servers using Silver and (b) shows the number of servers using Transmeta. (c) is a hybrid of these two types.

FFD processes the items which will do the placement in a decreasing order. For each item, it attempts to place the item in the first bin that can accommodate the item. If no bins is found, it opens a new bin and puts the item within the new bin. The algorithm gets a (11/9 OPT + 1) approximation [4]. We take the FFD result to the baseline of our algorithms and compare the results of our algorithms to FFD. We get the conclusion that the minimum number FFD gets is not always the optimal result of power consumption in the intuitive thought. Figure 5 and Figure 6 shows the results of number of servers (bins) and power consumption when the using some randomized requests' demand with 2 different parameters in Figure 4. Silver is a traditional mid-performance server which is the most common server in data center and Transmeta is a low-performance server that has a frequency of less than 1GHz.

In Figure 5, we use the number of servers as the comparison objective among three algorithms. We can see that FFD get the least number of servers, LB get the most number of servers, and PBF is between



(a) power consumption of servers when using Silver. Sample data sets' size is every 20 from 0 to 1000.

(b) power consumption of servers when using Transmeta. Sample data sets' size is every 20 from 0 to 1000.

(c) power consumption of servers when using a hybrid scheme of Silver and Transmeta. Sample data sets' size is every 20 from 0 to 1000.

Figure 6: (a) shows the power consumption of servers using Silver and (b) shows the power consumption of servers using Transmeta. (c) is a hybrid of these two types.

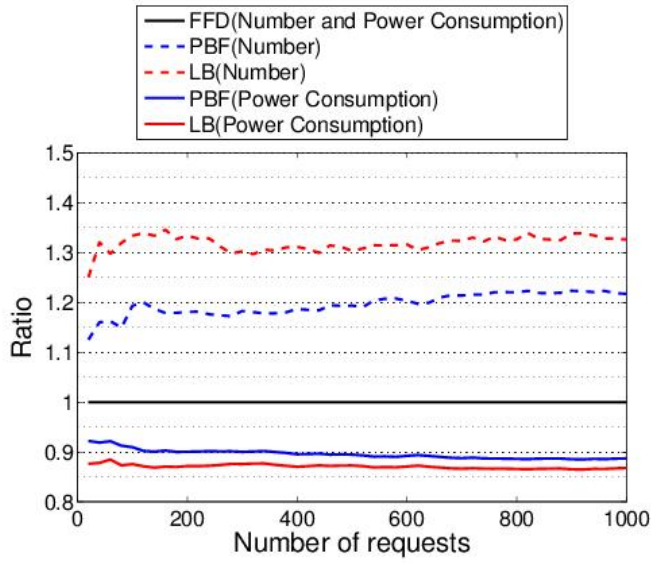


Figure 7: Comparisons between our algorithms and FFD when using Silver.

them. In Figure 6, we use the total power consumption of servers as the comparison objective among three algorithms. FFD use the most power consumption, PBF can get power saving and LB get the most power saving. LB gets a better result than PBF since PBF can not deal with the over-consolidation. A new request will consolidate on the current server although the current server exceeds the cut-off level defined in LB. Thus, it makes the power frequency ratio increase and may do harm to the power saving. We can figure out that the distance between our algorithms and FFD of the former is more obvious as the variable part which leads to "high frequency high power consumption" effect is greater than the latter.

As Sliver is a very common server in data centers, we give a figure which shows our simulation results' comparisons using Silver. Figure 7 shows the ratio of number the ratio of power consumption. We use the FFD algorithm as the benchmark. The size of requests get larger the performance better. The Number in the Figure 7 means the number of homogeneous servers, which is Silver. The blue line indicates the results of algorithm PBF and the red ones represent that of LB. PBF use 1.2 times of number of servers and get a power saving of 12% to 13 % in comparison with FFD when tends to be stable. 1.3 times of number of servers will be turned on but the power saving is about 14% of LB algorithm in comparison with FFD. Thus, we actually get a power saving although we use more number of servers, which is opposite to traditional opinion.

6 Conclusion

In this paper, we develop a two-phase methods with the target of minimizing the energy consumption of task (requests) scheduling in data centers and use heuristic algorithms to solve it due to its hardness. In the first phase, BS algorithm, which is based on benefit-driven method, makes the choice for server type. Then in the second phase, we make a tradeoff between the number of servers and frequency of each server in a homogeneous environment. We prove that dynamic resource allocation based on DVFS with the target of minimizing energy consumption in a heterogeneous environment is NP-Hard. And we propose two heuristic algorithms Power Best Fit (PBF) and Load Balancing (LB) based on different basic ideas to solve this problem. We also compare our algorithms with the well-known First Fit Decreasing (FFD) algorithm, and the simulation results show that we can get power saving although we use more number of servers than FFD in our settings due to the significant increment of power consumption when the CPU frequency becomes higher, which is a different conclusion to general idea.

Acknowledgment

This work is partially supported by the National Natural Science Foundation of China under Grant No. 61073028, 61021062, 61202113; the National Basic Research Program of China (973) under Grant No. 2009CB320705; Jiangsu Natural Science Foundation under Grant No. BK2011510.

References

[1] W. Huai, Z. Qian, X. Li, and S. Lu, "Towards Energy Efficient Data Centers: A DVFS-based Request Scheduling Perspective," in *Proc. of the 7th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS'13), Taichung, Taiwan (accepted)*. IEEE, July 2013.

[2] J. Koomey, "Growth in data center electricity use 2005 to 2010," Oakland, CA: Analytics Press, August 2011, http://www.analyticspress.com/datacenters.html.

[3] C. D. Patel and P. Ranganathan, "Enterprise power and cooling," ASPLOS Tutorial, October 2006.

[4] M. Yue, "A simple proof of the inequality FFD (L)$\leq$ 11/9 opt (L)+ 1,$\forall$ L for the FFD bin-packing algorithm," *Acta mathematicae applicatae sinica*, vol. 7, no. 4, pp. 321–331, 1991.

[5] "Voltage and frequency scaling," Wikipedia, http://en.wikipedia.org/wiki/Voltage_and_frequency_scaling.

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

[7] G. C. Buttazzo, "Scalable applications for energy-aware processors," in *Proc. of the 2nd International Conference on Embedded Software (EMSOFT'02), Grenoble, France, LNCS*, vol. 2491. Springer-Verlag, October 2002, pp. 153–165.

[8] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," in *Proc. of the 1st USENIX conference on Operating Systems Design and Implementation (OSDI'94), Monterey, California, USA*. USENIX Association, November 1994.

[9] A. Wierman, L. L. Andrew, and A. Tang, "Power-aware speed scaling in processor sharing systems," in *Proc. of the 28th IEEE Conference on Computer Communications (INFOCOM'09), Rio de Janeiro, Brazil*. IEEE, April 2009, pp. 2007–2015.

[10] L. L. Andrew, M. Lin, and A. Wierman, "Optimality, fairness, and robustness in speed scaling designs," vol. 38, no. 1, pp. 37–48, 2010.

[11] A. Weissel and F. Bellosa, "Process cruise control: event-driven clock scaling for dynamic power management," in *Proc. of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'02), Greenoble, France*. ACM, October 2002, pp. 238–246.

[12] K. Flautner, S. Reinhardt, and T. Mudge, "Automatic performance setting for dynamic voltage scaling," *Wireless Networks*, vol. 8, no. 5, pp. 507–520, 2002.

[13] S. Lee and T. Sakurai, "Run-time voltage hopping for low-power real-time systems," in *Proc. of the 37th Annual Design Automation Conference (DAC'00), Los Angeles, California, USA*. ACM, June 2000, pp. 806–809.

[14] J. R. Lorch and A. J. Smith, "Improving dynamic voltage scaling algorithms with pace," *ACM SIGMETRICS Performance Evaluation Review*, vol. 29, no. 1, pp. 50–61, 2001.

[15] F. Gruian, "Hard real-time scheduling for low-energy using stochastic data and DVS processors," in *Proc. of the 2001 International Symposium on Low Power Electronics and Design (ISLPED'01), Huntington Beach, California, USA*. ACM, August 2001, pp. 46–51.

[16] J. Pouwelse, K. Langendoen, and H. Sips, "Dynamic voltage scaling on a low-power microprocessor," in *Proc. of the 7th Annual International Conference on Mobile Computing and Networking (MobiCom'01), Rome, Italy*. ACM, July 2001, pp. 251–259.

[17] E. M. Elnozahy, M. Kistler, and R. Rajamony, "Energy-efficient server clusters," in *Proc. of the 2nd International Workshop on Power-Aware Computer Systems (PACS'02), Cambridge, Massachusetts, USA, LNCS*, vol. 2325. Springer-Verlag, February 2003, pp. 179–197.

[18] S. K. Garg, C. S. Yeo, A. Anandasivam, and R. Buyya, "Environment-conscious scheduling of hpc applications on distributed cloud-oriented data centers," *Journal of Parallel and Distributed Computing*, vol. 71, no. 6, pp. 732–749, 2011.

[19] B. Lawson and E. Smirni, "Power-aware resource allocation in high-end systems via online simulation," in *Proc. of the 19th Annual International Conference on Supercomputing (ICS'05), Cambridge, Massachusetts, USA*. ACM, June 2005, pp. 229–238.

[20] L. M. Zhang, K. Li, and Y.-Q. Zhang, "Green task scheduling algorithms with speeds optimization on heterogeneous cloud servers," in *Proc. of the 2010 IEEE/ACM International Conference on Green Computing and Communications (GreenCom'10) & International Conference on Cyber, Physical and Social Computing (CPSCom'10), Hangzhou, China*. IEEE, December 2010, pp. 76–80.

[21] Y. Fang, F. Wang, and J. Ge, "A task scheduling algorithm based on load balancing in cloud computing," in *Proc. of the 7th International Workshop on Web Information Systems and Mining (WISM'10), Sanya, China, LNCS*, vol. 6318. Springer-Verlag, October 2010, pp. 271–277.

[22] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Proc. of the 2008 Conference on Power Aware Computing and Systems (HotPower'08), San Diego, California, USA*. USENIX Association, December 2008.

[23] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," vol. 35, no. 5, pp. 103–116, 2001.

[24] A. Krioukov, P. Mohan, S. Alspaugh, L. Keys, D. Culler, and R. Katz, "Napsac: Design and implementation of a power-proportional web cluster," *ACM SIGCOMM computer communication review*, vol. 41, no. 1, pp. 102–108, 2011.

[25] T. Enokido, A. Aikebaier, and M. Takizawa, "Computation and transmission rate based algorithm for reducing the total power consumption," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, vol. 2, no. 2, pp. 1–18, 2011.

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

[27] S. Craß, T. Dönz, G. Joskowicz, E. Kühn, and A. Marek, "Securing a space-based service architecture with coordination-driven access control," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, vol. 4, no. 1, pp. 76–97, 2013.

[28] C.-H. Hsu and U. Kremer, "The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction," *ACM SIGPLAN Notices*, vol. 38, no. 5, pp. 38–48, 2003.

[29] M. Etinski, J. Corbalan, J. Labarta, M. Valero, and A. Veidenbaum, "Power-aware load balancing of large scale MPI applications," in *Proc. of IEEE International Symposium on Parallel & Distributed Processing (IPDPS'09), Rome, Italy*. IEEE, 2009, pp. 1–8.

[30] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal, "Analyzing the energy-time trade-off in high-performance computing applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 835–848, 2007.

[31] T. D. Burd and R. W. Brodersen, "Energy efficient CMOS microprocessor design," in *Proc. of the 28th Annual Hawaii International Conference on System Sciences (HICSS'95), Kihei, Maui, Hawaii, USA*, vol. 1. IEEE, January 1995, pp. 288–297.

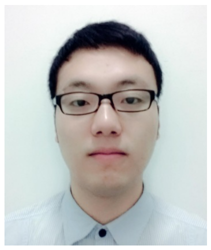
[32] "RUBiS," http://rubis.ow2.org/.

[33] M. R. Garey and D. S. Johnson, *Computers and intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[34] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing server energy and operational costs in hosting centers," vol. 33, no. 1, pp. 303–314, 2005.

[35] L. Wang and Y. Lu, "Efficient power management of heterogeneous soft real-time clusters," in *Proc. of Real-Time Systems Symposium (RTSS'08), Barcelona, Spain*. IEEE, November-December 2008, pp. 323–332.

[36] P. Bohrer, E. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony, "The case for power management in web servers," in *Power Aware Computing, Series in Computer Science*, R. Graybill and R. Melhem, Eds. Springer-Verlag, 2002, pp. 261–289.



Weicheng Huai was born in 1990. He obtained B.S. degree in Software Engineering from Jilin University, China in 2011. He is currently a M.S. candidate in the Department of Computer Science and Technology, Nanjing University, China. His research areas include Cloud Computing, Data Centers, Virtualization and VM Placement problems.

Zhuzhong Qian received his Ph.D. degree in computer science from Nanjing University (NJU) in 2007. He is currently an associate professor in the department of computer science and technology, Nanjing University. He is also a research fellow of the State Key Laboratory for Novel Software Technology. His current research interests include Distributed System and Cloud Computing.

Xin Li received his B.S. degree in Computer Science from Nanjing University in 2008. He is a Ph.D. candidate in the Department of Computer Science and Technology, Nanjing University, China. His research interests include resource scheduling in Cloud Computing, Service-Oriented computing, and Pervasive Computing.

Gangyi Luo was born in 1989. He studied in NanTong University during 2007 - 2011 and majored in Computer Science. Now he is a M.S. candidate in Nanjing Univeristy. His research interests focus on Cloud Computing, especially in VM consolidation problems. He has participated in building an experimental IAAS in his lab and took part in a project of constructing a high-performance web server.

Sanglu Lu received her B.S., M.S. and Ph.D. degrees from Nanjing University in 1992, 1995, and 1997, respectively, all in computer science. She is currently a professor in the Department of Computer Science & Technology and the State Key Laboratory for Novel Software Technology. Her research interests include distributed computing, wireless networks and pervasive computing.