Power-Efficient Workload Distribution for Virtualized Server Clusters

Leping Wang, Ying Lu
Department of Computer Science and Engineering
University of Nebraska - Lincoln
Lincoln, NE 68588
{lwang, ylu}@cse.unl.edu

Abstract
With growing cost of electricity, the power management of server clusters has become an important problem. Most existing research, however, either do not apply to virtualized environments or do not focus on power-efficient workload distribution. To fill in this research gap, we propose a workload distribution algorithm for virtualized server clusters to reduce their power consumptions and provide quality of service (QoS). Built upon optimization, queuing theory and control theory techniques, our approach achieves the design goal, where QoS is provided to a larger number of requests with a smaller amount of power consumption.

1 Introduction
According to the U.S. Environmental Protection Agency (EPA) 2007 Report to Congress, data centers in the United States incur a total energy cost of approximately $4.5 billion in 2006 and in the absence of intervention, this figure is expected to double in year 2012. As the rising cost of energy makes it the single most important factor in data center operating costs, energy minimization becomes one of the critical challenges in data centers. To avoid the projected energy expenditure, we must take measures such as to aggressively consolidate servers and enable power management at data center level.

To address the aforementioned challenge, many cluster-level power management mechanisms [5, 10, 14, 20, 2] have been proposed. Most of them, however, are only applicable to homogenous systems. Clusters are almost invariably heterogeneous in terms of performance, capacity, and energy consumption of their hardware components [10]. The heterogeneity also comes from servers with different types of services [13]. Therefore, we must explicitly consider cluster heterogeneity when developing power management mechanisms. A few researchers [10, 20], including ourselves [22], have investigated this problem, where we have demonstrated that it is important to control workload distribution in heterogeneous clusters and different workload distributions result in very different energy consumptions.

Existing studies have one common limitation. They do not consider virtualized environments. More and more modern clusters are, however, built on virtualization technology [15], where multiple services called virtual machines (i.e., VMs) are consolidated and placed together on a physical machine, leading to a smaller-sized and more energy-efficient cluster. In a virtualized cluster, besides heterogeneous hardware, we also face workload heterogeneity, where different servers often host different VMs, serving different workloads.

This paper focuses on the problem of power-efficient workload distribution for virtualized server clusters. There are many new challenges. First, unlike a homogenous cluster, where it is optimal to distribute workload evenly among active servers, identifying the optimal load distribution for a virtualized server cluster is a non-trivial task. Second, traditional strategy on dynamic voltage scaling (DVS) cannot be directly applied in a virtualized environment. Because VMs hosted on a physical server share the same CPU, whose state change will affect all their performance and may violate their quality of service (QoS) if unattended.

This paper proposes a workload distribution algorithm for virtualized server clusters to reduce their power consumptions and provide QoS. We make two main contributions. First, the proposed algorithm is based on simple but effective mathematical models, resulting in low software customization costs when deployed to new platforms. Second, integrating optimization, queuing theory and control theory techniques, we develop a novel scheme for power and performance management of virtualized clusters. Simulation results show the advantages of our algorithm.

The remainder of this paper is organized as follows. Sections 2 and 3 respectively present the models and state the problem. We discuss the algorithms in Sections 4 and 5, and evaluate their performance in Section 6. The related work is illustrated in Section 7 and Section 8 concludes the paper.

2 Models
In this section we present our models and state assumptions related to these models.

2.1 System Architecture
As illustrated by Figure 1, a cluster consists of a front-end server, connected to N back-end servers. We assume a typical cluster environment in which the front-end server does not participate in the request processing and is mainly responsible for accepting and distributing requests to back-end servers. A virtualized computing environment is assumed, where there are M online services hosted in virtual machines (VMs) of the cluster. Virtualization allows a single back-end server to be shared among multiple performance-isolated VMs. The placement of the M types of VMs on the N physical servers is given by an
The capacity model relates the CPU operating frequency by switching between two adjacent supported discrete frequencies. If a processor only supports discrete frequencies, we follow an approach similar to that proposed in [24] to approximate the desired continuous frequency setting by switching between two adjacent supported discrete frequency values. The capacity model relates the CPU operating frequency to the server’s throughput and the power model describes the relation between the CPU frequency and the power consumption. While our approach could be generalized to different capacity and power models, in this paper we assume and use the following specific models to illustrate our method.

2.2 Capacity Model

We assume that the cluster provides CPU-bound services. This is normal for many web servers where much of the data are kept in memory [3, 20, 25]. Therefore, to measure the capacity of a back-end server, its CPU throughput is used as the metric, which is assumed to be proportional to the CPU operating frequency. That is, the $i$th server’s throughput, denoted as $\mu_i$, is expressed as $\mu_i = \alpha_if_i$, where $\alpha_i$ is the CPU performance coefficient. Different servers may have different values for $\alpha_i$. With the same CPU frequency setting, the higher the $\alpha_i$ the more powerful the server is. For a particular VM on server $i$, its throughput depends on its allocated CPU time and its requirement for CPU. That is:

$$\mu_{ij} = \frac{\alpha_if_{ij}}{e_j} \quad (1)$$

where $f_{ij}$ denotes the virtual frequency of the $j$th VM on the $i$th server and we have $f_i = \max(\sum_{j=1}^{M} x_{ij}, f_{i_{\text{min}}})$. $e_j$ is the CPU demand factor of the $j$th service. The higher $e_j$, the more CPU time is required to process a $j$th-type request.

2.3 Power Model

The power consumption $P_i$ of a server consists of a constant part and a variable part. Similar to previous work [7, 5, 11], we approximate $P_i$ by the following function:

$$P_i = c_i + \beta_if_i^p \quad (2)$$

where $c_i$ denotes the constant power consumption of the server. It is assumed to include the base power consumption of the CPU and the power consumption of all other components. In addition, the CPU also consumes a power $\beta_if_i^p$ that is varied with the CPU operating frequency $f_i$. In the remainder of this paper, we use $p = 3$ to illustrate our approach.

Hence, in the cluster the power consumption of all back-end servers can be expressed as follows:

$$J = \sum_{i=1}^{N}(c_i + \beta_if_i^3) \quad (3)$$

Here, for the purpose of differentiation, $J$ is used to denote the cluster’s power consumption while $P$ denotes a server’s power consumption. Following the aforementioned models, each physical server is specified with five parameters: $f_{i_{\text{min}}}, f_{i_{\text{max}}}, \alpha_i, c_i$, and $\beta_i$. To obtain these parameters, only a little performance profiling is required. As a result, an algorithm designed based on these models has very low customization costs when deployed to new platforms.

3 Power Management Problem

Given a cluster of $N$ heterogeneous back-end servers, each specified with parameters $f_{i_{\text{min}}}, f_{i_{\text{max}}}, \alpha_i, c_i$, and $\beta_i$, the objective is to minimize the power consumed by the cluster while satisfying the following QoS requirement: $R_{ij} \approx \hat{R}_j$, where $R_{ij}$ and $\hat{R}_j$ respectively stand for the average and desired response time of the $j$th type of VM. The average response time $\hat{R}_j$ is determined by the $j$th VM’s capacity and workload. We use $\mu_{ij} = \frac{\alpha_if_{ij}}{e_j}$ to denote the VM’s capacity and $\lambda_{ij}$, the VM’s average request rate, to represent the workload. Thus, $R_{ij}$ is a function of these two parameters, i.e., $R_{ij} = g(\mu_{ij}, \lambda_{ij})$. To enforce $R_{ij} \approx \hat{R}_j$, we must control $\mu_{ij} = \frac{\alpha_if_{ij}}{e_j}$ and $\lambda_{ij}$ properly. As a result, the power management problem is formed as follows:

minimize

$$J = \sum_{i=1}^{N}(c_i + \beta_if_i^3) \quad (4)$$

subject to:

$$\sum_{i=1}^{N} x_{ij} \lambda_{ij} = \lambda_j, \quad j = 1, 2, \cdots, M$$

$$g(\mu_{ij}, \lambda_{ij}) \approx \hat{R}_j, \quad i = 1, 2, \cdots, M \quad (5)$$

$$f_i = \max(\sum_{j=1}^{M} x_{ij}, f_{i_{\text{min}}}) \quad i = 1, 2, \cdots, N$$

$$f_i \leq f_{i_{\text{max}}} \quad i = 1, 2, \cdots, N$$

where $\lambda_j$ is the average request rate for service $j$. The first set of constraints guarantees that all requests should be processed by the corresponding virtual machines and the second set ensures the QoS requirement.

For the power management, the front-end server decides the workload distribution $\lambda_{ij}$ for all VMs. On the back-end, each physical server dynamically adjusts its CPU operating frequency
4 Workload Distribution Algorithms

In this section, we first present our power-efficient (PE-based) workload distribution. Then, for comparison, two baseline (LB and Capacity-based) algorithms are described.

4.1 PE-based Workload Distribution

In the previous section, the power management is formed as an optimization problem. To analytically solve the problem and get the optimal $\lambda_{ij}$, the workload distribution for each VM, however, is not that easy because the optimization problem has a nonlinear objective function. Therefore, what we will do next is to find a linear function to approximate and substitute the original objective function.

When processing the same workload, different servers consume different amounts of power. Even for one server, when it operates under different frequencies, its power efficiencies are different. We name this feature as Server Efficiency, which is related to not only the static physical parameters but also the operating status of the server. It describes how server performance changes as power consumption varies. In this paper, we use the derivative of power consumption with respect to performance to represent the inverse of the Server Efficiency. Server $i$’s power consumption is $P_i = c_i + \beta_i f_i^3 = c_i + \beta_i (\frac{\lambda_{ij}}{R_{ij}})^3$ and performance can be represented by its throughput $\mu_i$. Thus, server $i$’s Inverse Efficiency $E_i$ is expressed as follows:

$$E_i = \frac{dP_i}{d\mu_i} = 3\frac{\beta_i}{\alpha_i} f_i^2$$  \hspace{1cm} (10)

If we can keep all $E_i$ (i.e., $\sqrt{E_i}$) as low as possible, the cluster’s power consumption increases the slowest as the workload grows. Based on this idea, we propose an optimization problem with a linear objective function to approximate the one formed in Section 3. The new optimization problem is as follows:

minimize

$$J = \sum_{i=1}^{N} (c_i + \beta_i f_i^3)$$  \hspace{1cm} (8)

subject to:

\[
\begin{align*}
\sum_{i=1}^{N} x_{ij} \lambda_{ij} &= \lambda_j, \\
fi &= \max\left(\sum_{j=1}^{M} x_{ij} \frac{c_i}{\alpha_i} (\lambda_{ij} + \frac{1}{R_{ij}}), f_{\lambda_{\text{min}}}, f_{\lambda_{\text{max}}}ight), & i = 1, 2, \ldots, N
\end{align*}
\]  \hspace{1cm} (9)

To achieve the optimal power consumption and guarantee the average response time, the key therefore lies in the front-end, i.e., the workload distribution, which we discuss in detail next.

4.2 LB-based Workload Distribution

The first baseline algorithm balances the workload among all physical servers and equally utilizes them. It tries to keep all physical servers run at similarly low CPU frequencies with respect to their maximum levels. We model this requirement as follows:

minimize

$$\nu$$  \hspace{1cm} (11)

subject to:

\[
\begin{align*}
\sqrt{3\beta_i f_i} - \nu &\le 0, & i = 1, 2, \ldots, N \\
\sum_{i=1}^{N} x_{ij} \lambda_{ij} &= \lambda_j, & j = 1, 2, \ldots, M \\
fi &= \max\left(\sum_{j=1}^{M} x_{ij} \frac{c_i}{\alpha_i} (\lambda_{ij} + \frac{1}{R_{ij}}), f_{\lambda_{\text{min}}}, f_{\lambda_{\text{max}}}ight), & i = 1, 2, \ldots, N
\end{align*}
\]  \hspace{1cm} (12)

After solving this standard linear programming problem, we obtain the desired workload distribution $\lambda_{ij}$ for each VM. We call this method as Power-Efficient Workload Distribution, PE-based for short.

For the comparison purpose, we will next present two baseline workload distribution algorithms.

4.3 Capacity-based Workload Distribution

When distributing workload, baseline algorithm II considers VM’s capacity, where a VM hosted in a more powerful physical server gets more requests. We consider the throughput $\mu_i$ as the $i^{th}$ server’s capacity. The workload will be distributed to VMs in proportion to the capacities of their physical servers. Therefore, the workload distribution $\lambda_{ij}$ can be expressed as follows:
\[
\lambda_{ij} = \frac{x_{ij} \mu_i}{\sum_{i=1}^{N} x_{ij} k_i} \lambda_j
\]  

We name this method as \textit{Capacity-based Workload Distribution}.

4.4 DVS and CPU Resource Allocation

Previous sections present three workload distribution algorithms. No matter which of these algorithms is adopted by the front-end server, back-end servers always use the same dynamic voltage scaling (DVS) mechanism. Based on M/M/1 queuing model, a back-end server’s CPU frequency \( f_i \) should be set at \( \sum_{j=1}^{M} x_{ij} f_{ij} \), where \( f_{ij} = \frac{\alpha_j}{\lambda_j + \frac{1}{R_j}} \). This approximation, however, may introduce modeling inaccuracy. To overcome this inaccuracy, we use feedback control to adjust the frequency.

Applying control theory, we design a feedback control loop for each virtual machine \( VM_{ij} \). In the control loop, the desired response time \( R_j \) is the set point and the measured response time \( R_{ij} \) is the controlled variable. Their difference \( d_{ij}[k] = R_{ij}[k] - \hat{R}_j \) is computed at each sampling period \( k \) and passed to a PI controller. Based on this input, the controller determines the virtual frequency adjustment \( \Delta f_{ij} \) for each VM. That is, we combine the feedback control output with the queuing theoretic prediction: the desired CPU resource allocation becomes \( f'_{ij}[k] = f_{ij} + \Delta f_{ij} \), where \( f_{ij} = \frac{1}{\lambda_j + \frac{1}{R_j}} \) is the queuing theory based prediction. Therefore, the server’s CPU frequency setting is:

\[
f_i[k] = \begin{cases} 
    f_{\text{min}}, & \text{if } \sum_{j=1}^{M} x_{ij} f'_{ij}[k] < f_{\text{min}} \\
    f_{\text{max}}, & \text{if } \sum_{j=1}^{M} x_{ij} f'_{ij}[k] > f_{\text{max}} \\
    \sum_{j=1}^{M} x_{ij} f'_{ij}[k], & \text{otherwise}
\end{cases}
\]

and it is shared by VMs with the following weights:

\[
w_{ij}[k] = \frac{x_{ij} f'_{ij}[k]}{\sum_{m=1}^{M} x_{im} f'_{im}[k]}, \quad j = 1, \ldots, M
\]

i.e., the actual amount of CPU resource allocated to a VM at the \( k \)th sampling period is \( f_{ij}[k] = w_{ij}[k] f_i[k] \).

5 Admission Control Algorithms

In the previous section, three workload distribution methods are described. These methods can meet the QoS requirement only if the cluster is not overloaded. To ensure the QoS, we, therefore, also need to design admission control algorithms to avoid overloading the cluster. Since PE and LB-based methods follow the same workload constraints (i.e., \( \sum_{i=1}^{N} x_{ij} \lambda_{ij} = \lambda_j \), \( f_i = \max(\sum_{j=1}^{M} x_{ij} \frac{\alpha_j}{\lambda_j + \frac{1}{R_j}}, f_{\text{min}}) \), and \( f_i \leq f_{\text{max}} \)), their corresponding admission control algorithms are the same.

In section 5.1, we present an admission control algorithm that is applicable to these two methods. The admission control algorithm for the capacity-based workload distribution is described in section 5.2.

5.1 PE and LB-based Admission Control

Unlike a single-service cluster, whose admission control algorithm simply rejects extra requests to keep the demand equal to the cluster capacity, the algorithm for a virtualized multiple-service cluster is much more complicated. In some cases, overloads are caused not by the inadequate cluster capacity but by the insufficient placement of some VM services. Feasibility analysis is thus needed to identify overloaded services.

Feasibility Analysis. Given a group of services \( G_k \), we compute their total workload demand \( d_k \) and compare it with the maximum physical server capacity \( C_k \) that can be used by these services. If the capacity is smaller than the demand (i.e., \( \hat{C}_k < d_k \)), we find an overloaded group of services. The detailed procedure is as follows.

Assume there are \( m \) services in \( G_k \) and their VMs are hosted on \( n \) physical servers. Without loss of generality, they are assumed to be the first \( m \) services and the first \( n \) servers. Thus, the maximum physical server capacity that can be used by \( G_k \) is:

\[
\hat{C}_k = \sum_{i=1}^{n} \alpha_i f_{i,\text{max}}
\]

The total workload demand of \( G_k \)'s \( m \) services is:

\[
d_k = \sum_{j=1}^{m} e_j \lambda_j
\]

Their difference \( \hat{E}_k = \max(d_k - \hat{C}_k, 0) \) indicates how overloaded \( G_k \) is. It denotes the amount of workload that needs to be rejected for this group of services. Next, we explain how our algorithm divides this \( \hat{E}_k \) amount of workload rejection among the \( m \) services.

Here, we introduce a new concept called the capacity quota of a service. We assume that the placement of VMs implies capacity quotas allocated to services. A service with more VM instances is assumed to have a larger capacity quota. The capacity quota \( q_j \) is calculated as follows:

\[
q_j = \sum_{i=1}^{n} x_{ij} \frac{\mu_i}{s_i}
\]

where \( s_i \) denotes the number of different VMs hosted on server \( i \). In the above, we have assumed that the server capacity quota \( \mu_i = \alpha_i f_{i,\text{max}} \) is shared fairly among hosted services.

When we need to reject \( \hat{E}_k \) amount of workload, \( q_j \) will be used to calculate the share of rejection for each service. The more extra workload a service has, i.e., the larger \( (e_j \lambda_j - q_j) \), the more rejection it has. However, simply rejecting the extra workload of all services does not work, because in most cases not all services exceed their quotas. To avoid high reject ratio and low system utilization, we instead divide the \( \hat{E}_k \) amount of workload rejection among overloaded services as follows:

\[
\delta_j = \begin{cases} 
0, & \text{if } q_j \geq e_j \lambda_j \\
\frac{E_k (e_j \lambda_j - q_j)}{\sum_{j \in \{i | q_j < e_j \lambda_j\}} (e_j \lambda_j - q_j)}, & \text{if } q_j < e_j \lambda_j
\end{cases}
\]
where \( j = 1, 2, \ldots, m \).

If a straightforward approach were followed, we need to carry out the aforementioned analysis and control for every possible service group. For a cluster serving \( M \) services, that means \( 2^M \) repetitions of the above procedure. This large time complexity is not acceptable. Therefore, our admission control algorithm instead follows a reactive and iterative approach, which repeats the procedure only if the solver repetitively fails to find a feasible solution for the optimization problem (i.e., Equations (11) and (12) or Equations (13) and (14)). That is, when the optimization solver fails for the first time. We start the analysis and control procedure for the first group \( G_1 \) of services (i.e., the group that includes all \( M \) services). If cutting the workload for \( G_1 \) resolves the overload situation and makes the optimization problem solvable, we are done with the admission control. Otherwise, the procedure is repeated for the next largest group that remains to be tested. Based on our experience, the overload is often eliminated after only 2 or 3 iterations.

### 5.2 Capacity-based Admission Control

The previous subsection describes PE and LB-based admission control algorithm. In this subsection, we present the admission control strategy for the capacity-based workload distribution.

According to the capacity-based algorithm (see Section 4.3), the total workload demand distributed to server \( i \) is:

\[
\mu'_{ji} = \sum_{j=1}^{M} e_j \lambda_{ij} = \sum_{j=1}^{M} e_j \frac{x_{ij} \mu_i}{N} \lambda_j
\]

(20)

while server \( i \)'s total capacity is:

\[
\mu_i = \alpha_i f_{\omega_{\text{max}}}
\]

(21)

Their difference \( \hat{E}_i = \max(\mu'_{ji} - \mu_i, 0) \) is the amount of extra workload that needs to be rejected for server \( i \). To achieve that, the reject ratio of service \( j \) should be:

\[
t_j = \max\{\frac{x_{ij} \hat{E}_i}{\mu'_{ji}}|1 \leq i \leq N\}
\]

(22)

that is, only \((1 - t_j)\lambda_j\) amount of requests should be admitted for service \( j \).

### 6 Performance Evaluation

This section evaluates the performance of the proposed algorithms.

#### Virtualized Cluster Configuration

A discrete simulator has been developed to simulate heterogeneous virtualized clusters that are compliant to models presented in Section 2. We simulate the following clusters:

- **Cluster1.** First, we simulate a cluster that consists of 4 back-end servers. They are all single processor machines: server 1 has an AMD Athlon 64 3000+ 1.8GHz CPU; server 2 has an AMD Athlon 64 X2 4800+ 2.4GHz CPU; server 3 has an Intel Pentium 4 630 3.0GHz CPU and server 4 has an Intel Pentium D 950 3.4GHz CPU. To derive server parameters, experimental data from [20, 1, 8]

<table>
<thead>
<tr>
<th>Server</th>
<th>( f_{\omega_{\text{max}}} )</th>
<th>( c_i )</th>
<th>( \beta_i )</th>
<th>( \alpha_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.8</td>
<td>44</td>
<td>2.915</td>
<td>495.00</td>
</tr>
<tr>
<td>2</td>
<td>2.4</td>
<td>53</td>
<td>4.485</td>
<td>548.75</td>
</tr>
<tr>
<td>3</td>
<td>3.0</td>
<td>70</td>
<td>2.370</td>
<td>287.00</td>
</tr>
<tr>
<td>4</td>
<td>3.4</td>
<td>68</td>
<td>3.206</td>
<td>309.12</td>
</tr>
</tbody>
</table>

Table 1: Parameters of 4 Types of Server

are referred. Table 1 lists the estimated server parameters. In addition, we assume that the processors only support discrete frequencies, i.e., a processor’s frequency can only be set to one of ten discrete levels in the range \([f_{\omega_{\text{min}}}, f_{\omega_{\text{max}}}]\), where \(f_{\omega_{\text{min}}} = 25\% f_{\omega_{\text{max}}}\). This cluster is assumed to provide 4 different services.

- **Cluster2.** Second, we simulate a cluster that has 16 back-end servers of 4 different types, whose parameters are the same as those listed in Table 1. The processors only support discrete frequencies, i.e., a processor’s frequency can only be set to one of ten discrete levels in the range \([f_{\omega_{\text{min}}}, f_{\omega_{\text{max}}}]\), where \(f_{\omega_{\text{min}}} = 25\% f_{\omega_{\text{max}}}\). This cluster is assumed to provide 16 different services.

#### Virtual Machine Placement

Another key configuration is on the placement of VMs. A VM placement can be defined in terms of the total number of VMs, their types and their distribution among physical servers. We assume no two VMs in a physical server are the same. Thus, a physical server can host up to 4 VMs in Cluster1 and up to 16 VMs in Cluster2. It is expected that algorithms could perform differently under different VM placements. Thus, to fairly compare algorithms, we often evaluate their performance under various different placements. In particular, Section 6.3 investigates the effects of VM placement on algorithm performance.

#### Workload Generation

A request is specified by a tuple \((A_i, E_i)\), where \(A_i\) is the arrival time and \(E_i\) is the execution time on the default server, i.e., server 1, when it is operating at its maximum frequency. 4 and 16 request streams are generated for Cluster1 and Cluster2 respectively.

To generate requests for service \( j \), we assume their inter-arrival time follows a series of exponential distribution with a time varied mean \(\frac{1}{\alpha_i f_{\omega_{\text{max}}}^j}\). In Sections 6.1 and 6.2, we simulate cases where a cluster is not overloaded. As illustrated in Figure 2, the total workload of Cluster1 (i.e., \(\lambda_{\text{cluster}}[k] = \sum_{j=1}^{4} 4\lambda_j[k]\)) changes in the range \([10\%, 90\%]\) of the cluster capacity. Similar workload patterns are generated for Cluster2. In Sections 6.3 and 6.4, we also generate overloaded workloads, where the total workload sometimes reaches 2.1 times of the cluster capacity.

Request execution time \(E_i\) is assumed to follow exponential distribution, whose average is \(\frac{\mu_i}{\alpha_i f_{\omega_{\text{max}}}^j}\) for service \( j \).

Table 2 illustrates the parameters of the 4 services in Cluster1, including CPU demand factor \(e_j\) and average response time target \(R_j\).
Each simulation lasts 3000 seconds and periodically, i.e., every 30 seconds, the system measures the current workloads and predicts the average request rates $\lambda_j[k]$ for the next period. We adopt a method proposed in [9] for the workload prediction. Based on the predicted rates $\hat{\lambda}_j[k]$, the corresponding workload distribution (i.e., $\hat{\lambda}_{ij}[k]$) are derived by the algorithm. According to $\hat{\lambda}_{ij}[k]$, the back-end server DVS mechanism decides the server’s frequency setting $f_i[k]$. Since a CPU only supports discrete frequencies, we approximate the desired continuous frequency $f_i[k]$ by switching the CPU frequency between two adjacent discrete values, e.g., to approximate 2.65GHz frequency, during the 30-second sampling period, the CPU frequency is first set at 2.4GHz for 11.25 seconds and then at 2.8GHz for 18.75 seconds.

To evaluate algorithm performance, we measure two metrics: average response time and power consumption.

### 6.1 Non-Overloaded Cluster1

This section evaluates the performance of workload distribution algorithms in Cluster1. We choose a virtual machine placement where there are 2 VMs on each physical server and each service has 2 corresponding VM instances. As mentioned, moderate workload is generated, which does not overload the cluster. Figure 3 presents the cluster power consumption, which demonstrates that our PE-based workload distribution algorithm consumes the least power in all sampling periods. We illustrate the resultant average response times in Figures 4, 5 and 6. From the data, we can see that all three algorithms ensure QoS, successfully keeping average response times around their targets: 0.5, 0.9, 1.4 and 2 seconds.

### 6.2 Non-Overloaded Cluster2

In this section, we evaluate algorithm performance in Cluster2, where we choose a VM placement that includes a total of 192 VMs. Experimental data are similar to those shown in Section 6.1. Due to the space limitation, we only illustrate the power consumption data in Figure 7. Again, the PE-based algorithm leads to the smallest power consumption.

### 6.3 Effects of Virtual Machine Placement

It is expected that algorithms could perform differently under different VM placements. Therefore, in this section, we investigate the effects of VM placement on algorithm performance.
Figure 6: Capacity-based Non-Overloaded Cluster1: Average Response Time

<table>
<thead>
<tr>
<th>Alg</th>
<th>Power (Watt)</th>
<th>Avg</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE-based</td>
<td>1100</td>
<td>4.41</td>
<td></td>
</tr>
<tr>
<td>Capacity-based</td>
<td>1201</td>
<td>6.28</td>
<td></td>
</tr>
<tr>
<td>LB-based</td>
<td>1144</td>
<td>3.98</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7: Non-Overloaded Cluster2: Power Consumption

Figure 8: Feasibility of Algorithms under Different VM Place-ments

Next, we evaluate how VM placement affects the power consumption. This experiment runs on Cluster2. For each algorithm, we simulate the same workload under 4 different VM placement groups. Placement groups are distinguished by the total number of VMs, where placements with the same total number of VMs fall into one group. We choose 4 groups whose placements have 128, 176, 192 and 240 VMs respectively. From each group, 10 placements are randomly picked. We test how an algorithm performs under these placements. For each sampling period, we calculate the resultant power consumption averaged among each placement group. Figures 9, 10 and 11 respectively show the results for PE, LB and capacity-based algorithms. We can see that with the PE or LB-based algorithm, the resultant power consumption is insensitive to the VM placements. Despite that different placement groups have different numbers of VMs, their average power consumptions are pretty much the same. This is not the case for the capacity-based algorithm, where the difference of power consumption among the 4 groups are obviously much larger. This is because the capacity-based algorithm employs a workload distribution strategy that is not adaptive to the placement, while for PE and LB-based algorithms, they always strive to find the most power-efficient or balanced workload distribution based on the current placement.

A point worthy of notice: these figures again provide strong evidence that the PE-based workload distribution saves power significantly.

6.4 Admission Control Performance

In this section, we evaluate the corresponding admission control algorithms. When a cluster is overloaded, the admission control module starts to work. It rejects some requests to ensure QoS for the remaining requests. We thus use an experiment to evaluate the resultant request reject ratio of the three algorithms.

This experiment runs on Cluster1 with a fixed VM placement. We simulate each algorithm with 4 different workload levels.
For a workload, the ratio of its peak volume vs. the total cluster capacity is used to represent its level. We test the following 4 workload levels: 1.2, 1.5, 1.8 and 2.1. The resultant request reject ratios are shown in Table 3. Since PE and LB-based algorithms share the same admission control module, they result in the same reject ratio. Compared with the capacity-based algorithm, they always reject less requests.

Figures 12, 13 and 14 respectively present the average response time of the three algorithms subject to the highest workload level 2.1. Since admission control modules have rejected extra requests, the average response times of admitted requests are successfully kept around their targets.

In Figures 15, 16, 17 and 18, we show the power consumption with the 4 workload levels respectively. Our PE-based algorithm still performs the best in most sampling periods. When request rejection happens, by rejecting the largest number of requests, the capacity-based algorithm sometimes leads to the least power consumption. Table 4 summarizes the power consumption data. From Tables 3 and 4, we conclude that the PE-based algorithm always serves the largest number of requests with nearly the least amount of power consumption.

7 Related Work

Power management of server clusters has become an important problem. To save power in a cluster, there are three common strategies [4, 5, 7, 14, 18, 19]. First, we can consolidate services to a fewer number of servers and then turn off idle machines. Second, hardware components can be put in low-power states, for example, the operating frequency of a CPU can be lowered by dynamic voltage scaling (DVS). Furthermore, in a heterogeneous cluster, a proper workload distribution [10, 20, 22] leads to significant power saving.

The aforementioned research results, however, do not apply to virtualized environments. As the virtualization technology being widely adopted, the power management of virtualized clusters becomes an important problem. Wang et al. [23] focus their investigation on how to control CPU frequency and share...
it among different VMs to satisfy power and performance constraints. Nathuji et al. [17] studies power-budgeting methods to ensure that the total power consumption of a virtualized cluster does not exceed the specified budget. In papers [13, 12], the resource provisioning problem is investigated, where methods are proposed to match VMs workload demand with the cluster capacity. These methods could be leveraged to guide the power-efficient server consolidation. Previous research [6] has

<table>
<thead>
<tr>
<th>Alg</th>
<th>Power (Watt)</th>
<th>Avg</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE-based</td>
<td>399.2</td>
<td>2.64</td>
<td></td>
</tr>
<tr>
<td>Capacity-based</td>
<td>399.8</td>
<td>3.67</td>
<td></td>
</tr>
<tr>
<td>LB-based</td>
<td>416.4</td>
<td>2.52</td>
<td></td>
</tr>
</tbody>
</table>

![Figure 13: LB-based Overloaded Cluster1: Average Response Time](image)

![Figure 14: Capacity-based Overloaded Cluster1: Average Response Time](image)

![Figure 15: Power Consumption - Workload Level 1.2](image)

![Figure 16: Power Consumption - Workload Level 1.5](image)

![Figure 17: Power Consumption - Workload Level 1.8](image)

![Figure 18: Power Consumption - Workload Level 2.1](image)
also studied how to properly place VMs on a cluster’s physical servers to improve energy efficiency. These work’s contributions are complementary to ours.

The most closely related work is by Mukherjee et al. [16], where they investigate thermal-aware job scheduling in virtualized heterogeneous data centers. Their assumed workload and system models are, however, different from ours. For instance, they do not assume workload heterogeneity, where a request can only be served by some but not all physical servers.

8 Conclusion
This paper presents a novel power-efficient workload distribution algorithm for virtualized server clusters. The algorithm has two advanced features. First, based on simple but effective mathematical models, the algorithm leads to low software customization costs when deployed to new platforms. Second, the algorithm is developed upon a solid theoretical foundation, where we integrate optimization, queuing theory and control theory techniques. By experiments, we compare the proposed algorithm with two baseline algorithms. We study how these algorithms perform in different clusters, VM placements and workloads. Simulation results have demonstrated the strong advantages of our algorithm.

References