5-24-2008

# Efficient Power Management of Heterogeneous Soft Real-Time Clusters

Leping Wang
*University of Nebraska-Lincoln*, lwang@cse.unl.edu

Ying Lu
*University of Nebraska-Lincoln*, ying@unl.edu

University of Nebraska–Lincoln, Computer Science and Engineering
Technical Report TR-UNL-CSE-2008-0004
Issued May 24, 2008

# Efficient Power Management of Heterogeneous Soft Real-Time 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. However, most previous researchers only address the challenge in homogeneous environments. Considering the increasing popularity of heterogeneous systems, this paper proposes an efficient algorithm for power management of heterogeneous soft real-time clusters. It is built on simple but effective mathematical models. When deployed to a new platform, the software incurs low configuration cost because no extensive performance measurements and profiling are required. To strive for efficiency, a threshold-based approach is adopted. In this paper, we systematically study this approach and its design decisions.*

## 1 Introduction

Clusters of commodity-class PCs are widely used. When designing such a system, traditionally researchers have focused on maximizing performance. Recently, with a better understanding of the overall cost of computing [1], researchers have started to pay more attention to optimizing performance per unit of cost. According to [1], the total cost of ownership (TCO) includes the cost of cluster hardware, software, operations and power. As a result of recent advances in chip manufacturing technology, the performance per hardware dollar keeps going up. However, the performance per watt has remained roughly flat over time. If this trend continues, the power-related costs will soon exceed the hardware cost and become a significant fraction of the total cost of ownership.

To reduce power and hence improve the performance per watt, cluster power management mechanisms [4, 5, 8, 11, 14, 15, 16, 17] have been proposed. Most of them, however, are only applicable to homogenous systems. It remains a difficult problem to manage power for heterogeneous clusters. Two new challenges have to be addressed. First, according to load and server characteristics, a power management mechanism must decide not only how many but also which cluster servers should be turned on; second, unlike a homogenous cluster, where it is optimal to evenly distribute load among active servers, identifying the optimal load distribution for a heterogeneous cluster is a non-trivial task.

A few researchers [11, 17] have investigated mechanisms to address the aforementioned challenges. However, their mechanisms all require extensive performance measurements ("at most few hours for each machine" [17]) or time-consuming optimization processes. These high customization costs are prohibitive, especially if the processes need to be executed repetitively. Composed of a large number of machines, a cluster is very dynamic, where servers can fail, be removed from or added to it frequently. To achieve high availability in such an environment, a mechanism that is easy to be modified upon changes is essential. This paper proposes an efficient algorithm for power management (PM) of heterogeneous soft real-time clusters. We make two contributions. First, the algorithm is based on simple but effective mathematical models, which reduces customization costs of PM components to new platforms. Second, the developed online mechanisms are threshold-based. According to an offline analysis, thresholds are generated that divide the workload into several ranges. For each range, the power management decisions are made offline. Dynamically, the PM component just measures and predicts the cluster workload, decides its range, and follows the corresponding decisions. In this paper, we systematically investigate this low-cost efficient power management approach. Simulation results show that our algorithm not only incurs low overhead but also leads to near optimal power consumptions.

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

## 2 Related Work

Power management of server clusters [4, 5, 8, 14, 15, 16] has become an important problem. The authors of [3, 19] were the first to point out that cluster-based servers could benefit significantly from dynamic voltage scaling (DVS). Besides server DVS, dynamic resource provisioning (server power on/off) mechanisms were investigated in [8, 14] to conserve power in clusters.

The aforementioned research has all focused on homogeneous systems. However, clusters are almost invariably heterogeneous in term of their performance, capacity and power consumption [11]. Survey [2] discusses the recent work on power management for server systems. It lists power management of heterogeneous clusters as one of the major challenges.

The most closely related work is that of [11, 17]. Authors of [11] consider request distribution to optimize both power and throughput in heterogeneous server clusters. Their mechanism takes the characteristics of different nodes and request types into

account. In [17], energy efficient real-time heterogeneous clusters are investigated. Both papers note that in heterogeneous clusters it is difficult to properly order servers with respect to power efficiency and it may not be optimal to turn on the smallest number of machines to satisfy the current load.

However, both approaches depend on time-consuming optimizations to find the best cluster configuration for every possible load. Even though the optimizations are executed offline, they need to be repeated every time upon new installations, server failures, cluster upgrades or changes. Extensive performance measurements and long optimization processes [11, 17] lead to high customization costs. To avoid these prohibitive costs, we propose in this paper a simple power management algorithm for heterogeneous soft real-time clusters. The algorithm is based on mathematical models that require minimum performance profiling. Instead of solving the optimization problem for every possible load, our algorithm derives thresholds, divides load into several ranges and determines the best cluster configuration formula for each workload range, leading to a time-efficient optimization process. Furthermore, our algorithm incurs low overhead and achieves close-to-optimal power consumptions.

# 3 Models

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

## 3.1 System Model

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. The main role of the front-end server is to accept requests and distribute them to back-end servers. In addition, we deploy the power-management mechanism on the front-end server to enforce a server power on/off policy. Figure 1 shows a web server cluster example that fits our system model.

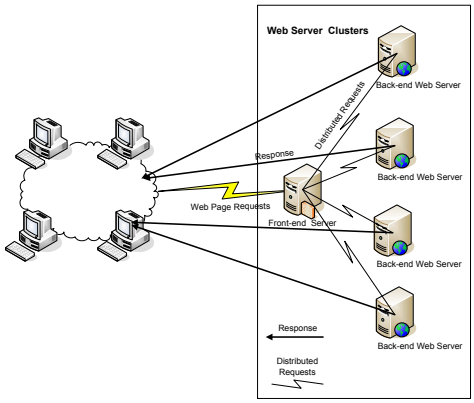


**Figure 1. System Model**

In a heterogeneous cluster, different back-end servers could have different computational capacities and power efficiencies. In the following, we provide their models. We assume processors on the back-end servers support dynamic voltage scaling and their operating frequencies could be continuously adjusted in the $(0, f_{i\_max}]$ range [1]. 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.

## 3.2 Capacity Model

We assume that the cluster provides CPU-bounded services, as typical web servers do today [3]. 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 $\mu_i = \alpha_i f_i$, 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.

## 3.3 Power Model

The power consumption $P_i$ of a server consists of a constant part and a variable part. Similar to previous work [8, 5, 12], we approximate $P_i$ by the following function:

$$P_i = x_i(c_i + \beta_i f_i^p) \tag{1}$$

where $x_i$ denotes the server's on/off state:

$$x_i = \begin{cases} 0 & \text{the } i^{th} \text{ server is off} \\ 1 & \text{the } i^{th} \text{ server is on} \end{cases} \tag{2}$$

When a server is off, it consumes no power; when it is on, it consumes $c_i + \beta_i f_i^p$ amount of power. In this model, $c_i$ 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 $\beta_i f_i^p$ that is varied with the CPU operating frequency $f_i$. In the remaining 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:

$$J = \sum_{i=1}^{N} x_i[c_i + \beta_i f_i^3] \tag{3}$$

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 server is specified with four parameters: $f_{i\_max}$, $\alpha_i$, $c_i$, and $\beta_i$. To obtain these parameters, only a little performance profiling is required.

# 4 Power Management Problem

Given a cluster of $N$ heterogeneous back-end servers, each specified with parameters $f_{i\_max}$, $\alpha_i$, $c_i$, and $\beta_i$, the objective is to minimize the power consumed by the cluster while satisfying the following QoS requirement: $R_i \approx \hat{R}$, where $R_i$ stands for the average response time of requests processed by the $i^{th}$ back-end server and $\hat{R}$ stands for the desired response time. The average

[1] In Section 6.3, we also evaluate the algorithm's performance on servers with only discrete frequency settings.

response time $R_i$ is determined by the back-end server's capacity and workload. We use $\mu_i = \alpha_i f_i$ to denote the server's capacity and $\lambda_i$, the server's average request rate, to represent the workload. Thus, $R_i$ is a function of these two parameters, i.e., $R_i = g(\mu_i, \lambda_i)$. To enforce $R_i \approx \hat{R}$, we must control $\mu_i = \alpha_i f_i$ and $\lambda_i$ properly. As a result, the power management problem is formed as follows:

**minimize**

$$J = \sum_{i=1}^{N} x_i[c_i + \beta_i f_i^3] \tag{4}$$

**subject to:**

$$\begin{cases} \sum_{i=1}^{N} x_i \lambda_i = \lambda_{cluster} \\ x_i(1 - x_i) = 0, & i = 1, 2, \cdots, N \\ g(\alpha_i f_i, \lambda_i) \approx \hat{R}, & i = 1, 2, \cdots, N \end{cases} \tag{5}$$

where $\lambda_{cluster}$ is the current average request rate of the cluster. We assume the cluster is not overloaded, that is, the average response time requirement $\forall i,\ g(\alpha_i f_i, \lambda_i) \approx \hat{R}$ is feasible for the cluster with a $\lambda_{cluster}$ request rate [2]. The first optimization constraint guarantees that each request is processed by an active back-end server while the second constraint says a server is either in an on or an off state.

For the power management, the front-end component decides the server's on/off state ($x_i$) and the workload distribution among the active servers ($\lambda_i$). On the back-end, each active node adjusts its CPU operating frequency $f_i$ in the $(0, f_{i\_max}]$ range to ensure the response time requirement, where a combined feedback control with queuing theoretic prediction approach, similar to that in [18], is adopted.

According to the $M/M/1$ queuing model, function $R_i = g(\mu_i, \lambda_i)$ is approximated as follows:

$$R_i = \frac{1}{\mu_i - \lambda_i} = \frac{1}{\alpha_i f_i - \lambda_i} \tag{6}$$

To guarantee $R_i \approx \hat{R}$, we approximate the proper $f_i$ to be:

$$f_i = \frac{\lambda_i}{\alpha_i} + \frac{1}{\alpha_i \hat{R}} \tag{7}$$

when $0 < \lambda_i \leq \alpha_i f_{i\_max} - \frac{1}{\hat{R}}$. This approximation, however, may introduce modeling inaccuracy. To overcome this inaccuracy, we combine feedback control with queuing-theoretic prediction for the dynamic voltage scaling (DVS). Nevertheless, in Section 6.4, experimental data shows that the queuing model estimate (Equation (7)) is very close to the real $f_i$ setting of the combined approach. This close approximation justifies the adoption of the queuing estimated $f_i$ in the problem formulation. The power management problem becomes:

**minimize**

$$J = \sum_{i=1}^{N} x_i[c_i + \beta_i \times (\frac{\lambda_i}{\alpha_i} + \frac{1}{\alpha_i \hat{R}})^3] \tag{8}$$

**subject to:**

$$\begin{cases} \sum_{i=1}^{N} x_i \lambda_i = \lambda_{cluster} \\ x_i(1 - x_i) = 0, & i = 1, 2, ..., N \\ 0 \leq \lambda_i \leq \alpha_i f_{i\_max} - \frac{1}{\hat{R}}, & i = 1, 2, ..., N \end{cases} \tag{9}$$

[2] A simple admission control mechanism could enforce this constraint.

As shown above, the optimal solution is determined by two variables: individual server's on/off state $x_i$ and workload distribution $\lambda_i$. To achieve the optimal power consumption and to guarantee the average response time, the key therefore lies in the front-end, i.e., the power on/off and workload distribution strategies. We present these strategies in the next section.

# 5 Algorithms

When we design the power management strategies, one major focus is on their efficiencies. For a given workload $\lambda_{cluster}$, the front-end power management needs to decide 1) how many and which back-end servers should be turned on and 2) how much workload should be distributed to each server. Since $\lambda_{cluster}$ changes from time to time, these decisions have to be reevaluated and modified regularly. Thus, the decision process has to be very efficient.

The mechanism we propose is built on a sophisticated but low-cost offline analysis. It provides an efficient threshold-based online strategy. Assuming $\hat{\lambda}_{cluster}$ is the maximum workload that can be handled by the cluster without violating the average response time requirement. The offline analysis generates thresholds $\Lambda_1, \Lambda_2, \cdots, \Lambda_N$ and divides (0, $\hat{\lambda}_{cluster}$] into $(0, \Lambda_1], (\Lambda_1, \Lambda_2], \cdots, (\Lambda_k, \Lambda_{k+1}], \cdots, (\Lambda_{N-1}, \Lambda_N]$ ranges (where $\Lambda_N = \hat{\lambda}_{cluster}$). For each range, the power on/off and workload distribution decisions are made offline. Dynamically the system just measures and predicts the workload $\lambda_{cluster}$, decides the range $\lambda_{cluster}$ falls into, and follows the corresponding power management decisions. Next, we present the details of our algorithm.

## 5.1 Optimization Heuristic Framework

In Section 4, the power management is formed as an optimization problem (Equations (8) and (9)). Instead of solving it for all possible workload $\lambda_{cluster}$ in the (0, $\hat{\lambda}_{cluster}$] range, we propose a heuristic to simplify the problem. It is constructed with the following framework:

- The heuristic first orders the heterogeneous back-end servers. It gives a sequence, called *ordered server list*, for activating machines. To shut down machines, the reverse order is followed.
- Second, the optimal thresholds $\Lambda_k, k \in \{1, 2, 3, \cdots N\}$ for turning on and off servers are identified: if $\lambda_{cluster}$ is in the $(\Lambda_{k-1}, \Lambda_k]$ range, it is optimal to turn on the first $k$ servers of the *ordered server list*. This also means if $\lambda_{cluster}$ changes between adjacent ranges, such as from $(\Lambda_{k-1}, \Lambda_k]$ to $(\Lambda_k, \Lambda_{k+1}]$, the heuristic requires on/off state change for just *one* machine. Considering the high overhead of turning on/off servers (e.g., several minutes), this approach is superior in that it minimizes the server on/off state changes.
- Third, the optimal workload distribution problem is solved for $N$ scenarios where $\lambda_{cluster} \in (\Lambda_{k-1}, \Lambda_k], k = 1, 2, \cdots, N$. When $\lambda_{cluster} \in (\Lambda_{k-1}, \Lambda_k]$, it is optimal to turn on the first $k$ servers of the *ordered server list*, i.e., $x_i = 1, i = 1, 2, \cdots k$ and $x_i = 0, i = k+1, k+2, \cdots, N$. With values of $x_i$ fixed, the optimization problem (Equations (8) and (9)) becomes:

  **minimize**

$$J_k = \sum_{i=1}^{k} [c_i + \beta_i \times (\frac{\lambda_i}{\alpha_i} + \frac{1}{\alpha_i \hat{R}})^3] \quad (10)$$

**subject to:**

$$\begin{cases} \sum_{i=1}^{k} \lambda_i = \lambda_{cluster} \\ 0 \le \lambda_i \le \alpha_i f_{i\_max} - \frac{1}{\hat{R}}, \quad i = 1, 2, ..., k \end{cases} \quad (11)$$

The analysis is simplified to solving the above optimization problem for $k = 1, 2, \cdots, N$.

**Time Efficiency Analysis.** If we consider solving the optimization problem (Equations (10) and (11)) as the basic operation, the time efficiency of the proposed heuristic is $\Theta(N)$, while the time efficiency to obtain the optimal power management solution (i.e., solving Equations (8) and (9)) for all integer points in the (0, $\hat{\lambda}_{cluster}$] range is $\Theta(\lceil \hat{\lambda}_{cluster} \rceil 2^N)$.

In the next three subsections, we discuss the decisions on *ordered server list*, *server activation thresholds* and *workload distribution* respectively. For each decision, several strategies are investigated.

## 5.2 Ordered Server List

Our algorithm follows a specific order to turn on and off machines. To optimize the power consumption, this order must be based on the server's power efficiency, which is defined as the amount of power consumed per unit of workload (i.e., $\frac{P_i(\lambda)}{\lambda}$). Servers with better power efficiencies are listed first.

According to the power model and the dynamic voltage scaling mechanism adopted by back-end servers (Sections 3 & 4), the power consumption $P_i(\lambda)$ of a server includes a constant part $c_i$ and a variable part $\beta_i \times (\frac{\lambda}{\alpha_i} + \frac{1}{\alpha_i \hat{R}})^3$ (see Equation (8)). Given any two servers $i$ and $j$, if $c_i \le c_j$ and $\frac{\beta_i}{\alpha_i^3} \le \frac{\beta_j}{\alpha_j^3}$, server $i$ has a better power efficiency than server $j$. However, if $c_i < c_j$ and $\frac{\beta_i}{\alpha_i^3} > \frac{\beta_j}{\alpha_j^3}$, the power efficiency order of the two is not fixed. When the server workload $\lambda$ is small, $P_i(\lambda)$ is less than $P_j(\lambda)$ and server $i$ has a better power efficiency; while as $\lambda$ increases, $P_i(\lambda)$ gets larger than $P_j(\lambda)$ and server $j$'s power efficiency becomes better. In the proposed method, to trade for online algorithm's efficiency and minimum server on/off operations, the *ordered server list* is determined offline and is not subject to dynamic changes. Therefore, even if the servers' power efficiency order is not fixed, their activation order is nevertheless determined statically. Next we present our method and list several alternatives for generating the activation order.

- *T*ypical *P*ower based policy (TP). We assume the typical workload for a server is $\lambda_i'$. In our heuristic, servers are ordered by their power consumption efficiency under the typical workload, i.e., $\frac{P_i(\lambda_i')}{\lambda_i'}$. A server with smaller $\frac{P_i(\lambda_i')}{\lambda_i'}$, i.e., smaller $\frac{c_i + \beta_i \times (\frac{\lambda_i'}{\alpha_i} + \frac{1}{\alpha_i \hat{R}})^3}{\lambda_i'}$, is listed earlier in the *ordered server list*. A power management mechanism usually turns on a server when needed or when it leads to a reduced power consumption (see Section 5.3). As a result, an active server usually works under a high workload. Thus we choose a workload that requires $80\%$ capacity of a server as its typical workload $\lambda_i'$. This way the *ordered server list* is created by comparing $\frac{P_i(\lambda_i')}{\lambda_i'}$ and is solely based on the server's static parameters $\alpha_i$, $c_i$, and $\beta_i$.
- *A*ctivate *A*ll policy (AA). This activation policy always turns on all back-end servers. Therefore in this case the power on/off mechanism is not needed. Neither is the *ordered server list*.
- *RAN*dom policy (RAN). This policy generates a random *ordered server list* for server activation.
- *S*tatic *P*ower based policy (SP). This policy orders machines by their static power consumption. A server with a smaller static power consumption $c_i$ is listed earlier in the *ordered server list*.
- *P*seudo *D*ynamic *P*ower based policy (PDP). This policy orders machines by the dynamic power consumption parameter $\beta_i$. A server with a smaller $\beta_i$ is listed earlier in the *ordered server list*. According to the definition of power efficiency $\frac{P_i(\lambda)}{\lambda}$, its dynamic part is $\frac{\frac{\beta_i}{\alpha_i^3} \times (\lambda + \frac{1}{\hat{R}})^3}{\lambda}$. As we can see, the dynamic power efficiency is not solely determined by $\beta_i$. This policy is therefore called *pseudo* dynamic power based policy.

## 5.3 Server Activation Thresholds

In the previous section we introduced the *ordered server list* that specifies which servers to choose when we need to turn on or off machines. This section presents our threshold-based strategy to decide the optimal number of active servers.

The goal is two-fold. First, an adequate number of servers should be turned on to guarantee the response time requirement. Second, the number of active servers should be optimal with respect to the consumed power.

To meet the response time requirement, the number of active servers should increase monotonically with the workload $\lambda_{cluster}$. The heavier the workload, the greater the number of active servers required. It suggests that we turn on more servers only when the current capacity becomes inadequate to process the workload. Accordingly $N$ capacity thresholds $\Lambda_{c1}, \Lambda_{c2}, \cdots, \Lambda_{cN}$ are developed and each $\Lambda_{ck}$ corresponds to the maximum workload that can be processed by the first $k$ servers. According to Equation (6), when a server is operating at its maximum frequency $f_{i\_max}$, it can process at most $\lambda_{i\_max}$ amount of workload and meet the response time requirement:

$$\lambda_{i\_max} = \alpha_i f_{i\_max} - \frac{1}{\hat{R}} \quad (12)$$

Thus, we have:

$$\Lambda_{ck} = \sum_{i=1}^{k} \lambda_{i\_max} = \sum_{i=1}^{k} \alpha_i f_{i\_max} - \frac{k}{\hat{R}} \quad (13)$$

When the current workload exceeds this threshold $\Lambda_{ck}$, at least $k+1$ servers of the *ordered server list* have to be activated.

However, the above thresholds may not be optimal with respect to the power consumption. The power consumed by a server is composed of two parts: the static part $c_i$ and the dynamic part $\beta_i f_i^3$. When adding an active server, the cluster's static power consumption increases but its dynamic power consumption may actually decrease. The reason is that with more

active servers to share the workload, the workload distributed to each server decreases; consequently, the CPU operating frequency $f_i$ required for each server may get smaller, which could lead to a reduced dynamic power consumption of the cluster.

To derive the optimal-power threshold, scenarios when activating $k+1$ servers is better than activating $k$ servers are identified. In such scenarios, $k$ servers are adequate to handle the workload. But if we activate $k+1$ servers, the system consumes less power. We assume that the optimal power consumption using the first $k$ servers to handle $\lambda_{cluster}$ workload, where $\lambda_{cluster} \in (0, \Lambda_{ck}]$, is $\hat{J}_k(\lambda_{cluster})$ (see Section 5.4 for $\hat{J}_k(\lambda_{cluster})$'s derivation). It is a monotonically increasing function of $\lambda_{cluster}$. We analyze the following equation:

$$\hat{J}_k(\lambda_{cluster}) = \hat{J}_{k+1}(\lambda_{cluster}) \tag{14}$$

According to characteristics of functions $\hat{J}_k(\lambda_{cluster})$ and $\hat{J}_{k+1}(\lambda_{cluster})$ (see Section 5.4), there is at most one solution for Equation (14). If such a solution $\lambda'_{cluster}$ is found, then activating $k+1$ servers is more power efficient than activating $k$ servers when $\lambda_{cluster} > \lambda'_{cluster}$. The proof is as follows. 1) $\hat{J}_k(\lambda_{cluster})$ is less than $\hat{J}_{k+1}(\lambda_{cluster})$ for small $\lambda_{cluster}$; 2) functions $\hat{J}_k(\lambda_{cluster})$ and $\hat{J}_{k+1}(\lambda_{cluster})$ increase monotonically with $\lambda_{cluster}$; and 3) if and only if $\lambda_{cluster} = \lambda'_{cluster}$ activating $k$ or $k+1$ servers consumes the same amount of power. Therefore, once $\lambda_{cluster}$ exceeds $\lambda'_{cluster}$, $\hat{J}_{k+1}(\lambda_{cluster})$ becomes less than $\hat{J}_k(\lambda_{cluster})$, i.e., it becomes more power efficient to activate $k+1$ servers.

Therefore, when there is a solution $\lambda'_{cluster} \in (0, \Lambda_{ck}]$ for Equation (14), we find the optimal-power threshold $\Lambda_{pk} = \lambda'_{cluster}$ where activating $k+1$ servers is more power efficient than activating $k$ servers when $\lambda_{cluster}$ exceeds this threshold; otherwise, we assign $\Lambda_{pk} = -1$. After analyzing Equation (14) for $k = 1, 2, \cdots, N-1$, we obtain another series of thresholds: optimal-power thresholds $\Lambda_{p1}, \Lambda_{p2}, ..., \Lambda_{p(N-1)}$.

By combining capacity and optimal-power thresholds, we get the server activation thresholds $\Lambda_k, k = 1, 2, \cdots, N$:

$$\Lambda_k = \begin{cases} \Lambda_{ck} & \text{for } \Lambda_{pk} = -1 \text{ or } k = N \\ \Lambda_{pk} & \text{for } \Lambda_{pk} \neq -1 \end{cases}$$

We use the symbol $CP$ to denote the above *C*apacity-*P*ower-based strategy. For comparison, a baseline *CA*pacity-only strategy, denoted as $CA$, is also investigated, for which $\Lambda_k = \Lambda_{ck}$. In the *A*ctivate *A*ll policy (AA), no server activation thresholds are needed.

### 5.4 Workload Distribution

Last two sections solved the problem of deciding how many and which back-end servers should be activated for a given workload. This section proposes a strategy to optimally distribute the workload among active servers.

According to Section 5.1, if the first $k$ servers of the *ordered server list* are activated, the optimization problem becomes:

**minimize**

$$J_k = \sum_{i=1}^{k} [c_i + \beta_i \times (\frac{\lambda_i}{\alpha_i} + \frac{1}{\alpha_i \hat{R}})^3] \tag{15}$$

**subject to:**

$$\begin{cases} \sum_{i=1}^{k} \lambda_i = \lambda_{cluster} \\ 0 \leq \lambda_i \leq \alpha_i f_{i\_max} - \frac{1}{\hat{R}}, \quad i = 1, 2, ..., k \end{cases} \tag{16}$$

The analysis is to find optimal solutions for all $J_k, k = 1, 2, \cdots, N$.

To solve the optimization for $J_k$, we first assume that all $k$ back-end servers are running below their maximum capacities, i.e, $0 \leq \lambda_i < \alpha_i f_{i\_max} - \frac{1}{\hat{R}}, i = 1, 2, ..., k$. Since the second constraint of the problem is satisfied, the optimization becomes:

**minimize**

$$J_k = \sum_{i=1}^{k} [c_i + \beta_i \times (\frac{\lambda_i}{\alpha_i} + \frac{1}{\alpha_i \hat{R}})^3] \tag{17}$$

**subject to:**

$$\sum_{i=1}^{k} \lambda_i = \lambda_{cluster} \tag{18}$$

According to *Lagrange's Theorem* [7], the first-order necessary condition for $J_k$'s optimal solution is:

$$\begin{aligned} \exists \delta,\ J_k(\lambda_i, \delta) = \sum_{i=1}^{k} [c_i + \beta_i \times (\tfrac{\lambda_i}{\alpha_i} + \tfrac{1}{\alpha_i \hat{R}})^3] \\ + \delta(\sum_{i=1}^{k} \lambda_i - \lambda_{cluster}) \end{aligned} \tag{19}$$

and its first-order derivatives satisfy

$$\begin{cases} \frac{\partial J_k(\lambda_i, \delta)}{\partial \lambda_i} = 0, \quad i = 1, ...k \\ \frac{\partial J_k(\lambda_i, \delta)}{\partial \delta} = 0 \end{cases} \tag{20}$$

Solving the above condition, we obtain the optimal workload distribution $\lambda_i, i = 1, ..., k$ as:

$$\lambda_i = \frac{\alpha_i (\lambda_{cluster} + \frac{k}{\hat{R}})}{\sum_{j=1}^{k} \alpha_j \sqrt{\frac{\alpha_j}{\beta_j}}} \sqrt{\frac{\alpha_i}{\beta_i}} - \frac{1}{\hat{R}} \tag{21}$$

The corresponding power consumption is:

$$\hat{J}_k = \sum_{i=1}^{k} c_i + \frac{(\lambda_{cluster} + \frac{k}{\hat{R}})^3}{(\sum_{j=1}^{k} \alpha_j \sqrt{\frac{\alpha_j}{\beta_j}})^2} \tag{22}$$

The above solution is optimal when all $k$ back-end servers are running below their maximum capacities. That is, when $\lambda_i$ (Equation (21)) satisfies the constraint that $0 \leq \lambda_i < \alpha_i f_{i\_max} - \frac{1}{\hat{R}}, i = 1, 2, ..., k$. Thus, the above condition holds true only for light workloads. As $\lambda_{cluster}$ increases, servers start to be saturated one after another. That is, a server's shared workload $\lambda_i$ reaches its maximum level $\alpha_i f_{i\_max} - \frac{1}{\hat{R}}$ where we have:

$$\begin{aligned} \lambda_i &= \frac{\alpha_i (\lambda_{cluster} + \frac{k}{\hat{R}})}{\sum_{j=1}^{k} \alpha_j \sqrt{\frac{\alpha_j}{\beta_j}}} \sqrt{\frac{\alpha_i}{\beta_i}} - \frac{1}{\hat{R}} \\ &= \alpha_i f_{i\_max} - \frac{1}{\hat{R}} \end{aligned} \tag{23}$$

Solving Equation (23) for system workload $\lambda_{cluster}$, we get:

$$\lambda_{cluster} = f_{i\_max}\sqrt{\frac{\beta_i}{\alpha_i}}\sum_{j=1}^{k}\alpha_j\sqrt{\frac{\alpha_j}{\beta_j}} - \frac{k}{\hat{R}} \quad (24)$$

This result seems to indicate that among the $k$ active servers, the one with a smaller value of $f_{i\_max}\sqrt{\frac{\beta_i}{\alpha_i}}$ reaches its full capacity earlier as $\lambda_{cluster}$ increases. We therefore order the $k$ servers by their $f_{i\_max}\sqrt{\frac{\beta_i}{\alpha_i}}$ values and generate the *saturated order list*. When a server gets saturated, its shared workload should not be increased any more. Otherwise its response time $R_i$ will violate the requirement. As a result, after the first server's saturation, i.e., the saturation of the first server on the *saturated order list*, we have the server's shared workload as $\lambda_1 = \alpha_1 f_{1\_max} - \frac{1}{\hat{R}}$ and the system workload as:

$$\lambda_{cluster} = f_{1\_max}\sqrt{\frac{\beta_1}{\alpha_1}}\sum_{j=1}^{k}\alpha_j\sqrt{\frac{\alpha_j}{\beta_j}} - \frac{k}{\hat{R}} \quad (25)$$

The workload distribution problem becomes:

**minimize**

$$\begin{aligned} J_k &= \sum_{i=2}^{k}[c_i + \beta_i \times (\frac{\lambda_i}{\alpha_i} + \frac{1}{\alpha_i\hat{R}})^3] \\ &+ (c_1 + \beta_1 f_{1\_max}^3) \end{aligned} \quad (26)$$

**subject to:**

$$\sum_{i=2}^{k}\lambda_i = \lambda_{cluster} - (\alpha_1 f_{1\_max} - \frac{1}{\hat{R}}) \quad (27)$$

Here, servers are indexed following their *saturated order list*. Similar to Equations (17) and (18), we solve the above problem by applying *Larange's Theorem* and get the following optimal solution for $\lambda_i, i = 2, 3, \cdots, k$:

$$\lambda_i = \frac{\alpha_i(\lambda_{cluster} - \alpha_1 f_{1\_max} + \frac{k}{\hat{R}})}{\sum_{j=2}^{k}\alpha_j\sqrt{\frac{\alpha_j}{\beta_j}}}\sqrt{\frac{\alpha_i}{\beta_i}} - \frac{1}{\hat{R}} \quad (28)$$

The corresponding power consumption is:

$$\hat{J}_k = \sum_{i=1}^{k} c_i + \frac{(\lambda_{cluster} - \alpha_1 f_{1\_max} + \frac{k}{\hat{R}})^3}{(\sum_{j=2}^{k}\alpha_j\sqrt{\frac{\alpha_j}{\beta_j}})^2} + \beta_1 f_{1\_max}^3 \quad (29)$$

Again, we let $\lambda_i$ (Equation (28)) be equal to the maximum workload $\alpha_i f_{i\_max} - \frac{1}{\hat{R}}$ and solve for $\lambda_{cluster}$. We get:

$$\lambda_{cluster} = f_{i\_max}\sqrt{\frac{\beta_i}{\alpha_i}}\sum_{j=2}^{k}\alpha_j\sqrt{\frac{\alpha_j}{\beta_j}} + \alpha_1 f_{1\_max} - \frac{k}{\hat{R}} \quad (30)$$

This result verifies our hypothesis that servers saturate following the *saturated order list* — the smaller the value of $f_{i\_max}\sqrt{\frac{\beta_i}{\alpha_i}}$, the earlier the server is saturated. The system workload that starts to saturate the first two servers is:

$$\lambda_{cluster} = f_{2\_max}\sqrt{\frac{\beta_2}{\alpha_2}}\sum_{j=2}^{k}\alpha_j\sqrt{\frac{\alpha_j}{\beta_j}} + \alpha_1 f_{1\_max} - \frac{k}{\hat{R}} \quad (31)$$

We define $\lambda_k^m$ as:

$$\lambda_k^m = f_{m\_max}\sqrt{\frac{\beta_m}{\alpha_m}}\sum_{j=2}^{k}\alpha_j\sqrt{\frac{\alpha_j}{\beta_j}} + \sum_{i=1}^{m-1}\alpha_i f_{i\_max} - \frac{k}{\hat{R}} \quad (32)$$

In general, when $\lambda_{cluster} \in [\lambda_k^m, \lambda_k^{m+1})$, $m$ of the k active servers are saturated. That is, $\lambda_i = \alpha_i f_{i\_max} - \frac{1}{\hat{R}}, i = 1, 2, \cdots, m$. The optimization problem becomes:

**minimize**

$$\begin{aligned} J_k &= \sum_{i=m+1}^{k}[c_i + \beta_i \times (\frac{1}{\alpha_i\hat{R}} + \frac{\lambda_i}{\alpha_i})^3] \\ &+ \sum_{i=1}^{m}(c_i + \beta_i f_{i\_max}^3) \end{aligned} \quad (33)$$

**subject to:**

$$\sum_{i=m+1}^{k}\lambda_i = \lambda_{cluster} - \sum_{j=1}^{m}(\alpha_j f_{j\_max} - \frac{1}{\hat{R}}) \quad (34)$$

and the optimal solution is :

$$\begin{aligned} \lambda_i &= \frac{\alpha_i(\lambda_{cluster} - \sum_{j=1}^{m}\alpha_j f_{j\_max} + \frac{k}{\hat{R}})}{\sum_{j=m+1}^{k}\alpha_j\sqrt{\frac{\alpha_j}{\beta_j}}}\sqrt{\frac{\alpha_i}{\beta_i}} - \frac{1}{\hat{R}} \\ &\text{for } i = m+1, m+2, \cdots, k \end{aligned} \quad (35)$$

$$\begin{aligned} \hat{J}_k &= \sum_{i=1}^{k} c_i + \frac{(\lambda_{cluster} - \sum_{j=1}^{m}\alpha_j f_{j\_max} + \frac{k}{\hat{R}})^3}{(\sum_{j=m+1}^{k}\alpha_j\sqrt{\frac{\alpha_j}{\beta_j}})^2} \\ &+ \sum_{i=1}^{m}\beta_i f_{i\_max}^3 \end{aligned} \quad (36)$$

**Baseline Algorithms.** We denote our algorithm proposed above as $OP$, the *OP*timal workload distribution. For comparison, the following three baseline algorithms are investigated:

- *RAN*dom (uniform) workload distribution (RAN). In this strategy, every incoming request is distributed to a randomly picked active server.
- *CA*pacity based workload distribution (CA). This strategy distributes the workload among active servers in proportion to their processing capacities, i.e. $\alpha_i f_{i\_max}$.
- *O*ne-by-*O*ne *S*aturation policy (OOS). In this policy, requests are distributed to active servers following a default order. For every incoming request, we pick the first active server that is not saturated to process it.

### 5.5 Algorithm Nomenclature

The previous three subsections have respectively presented different strategies for deriving the *ordered server list*, *server activation thresholds* and *workload distribution*. By following the proposed framework (Section 5.1), we could generate many different algorithms by combining different strategies for the three

| $Server$ | $f_{i\_max}$ | $c_i$ | $\beta_i$ | $\alpha_i$ |
|---|---|---|---|---|
| 1 | 1.8 | 44 | 2.915 | 495.00 |
| 2 | 2.4 | 53 | 4.485 | 548.75 |
| 3 | 3.0 | 70 | 2.370 | 287.00 |
| 4 | 3.4 | 68 | 3.206 | 309.12 |

**Table 1. Parameters of a 4-Server Cluster**

modules, for instance, TP-CP-OP, AA-AA-CA and SP-CA-CA. The nomenclature of the algorithms includes three parts corresponding to the three design decisions. The first part denotes the adopted strategy for deciding the *ordered server list*: TP, AA, RAN, SP or PDP. The second part represents the choice for deriving *server activation thresholds*: CP, CA or AA. In the third portion of the name, OP, RAN, CA or OOS denotes the *workload distribution* strategy. However, not all combinations are feasible. For instance, CP can only be combined with OP and AA is combined with AA.

## 6 Performance Evaluation

In previous section, we proposed various threshold-based strategies for the power management of heterogeneous soft real-time clusters. In this section, we experimentally compare their performance relative to each other and to the optimal solution of the power management problem (Equations (8) and (9)).

**Cluster Configuration.** We use a discrete simulator to simulate heterogeneous clusters that are compliant to the system model presented in Section 3:

- First, we simulate a small 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 [17, 6, 9] are referred. Table 1 lists the estimated parameters.

  We simulate two cases: a) a server's frequency can be continuously adjusted in the $(0, f_{i\_max}]$ range; b) a server's frequency can only be set to discrete values in the $[f_{i\_min}, f_{i\_max}]$ range.

- Second, we simulate a large cluster that has $128$ back-end servers of $8$ different types. They are all single processor type of machines whose parameters are as shown in Table 2.

**Workload Generation.** A request is specified by a tuple $(A_i, E_i)$, where $A_i$ is its arrival time and $E_i$ is its execution time on a default server when it is operating at its maximum frequency. To generate requests, we assume that the inter-arrival time follows a series of exponential distributions with a time-varied mean of $\frac{1}{\lambda_{cluster}(t)}$. As shown in Figure 2, we simulate a workload $\lambda_{cluster}(t)$ that gradually increases from requiring $20\%$ to $90\%$ of the cluster capacity. Request execution time $E_i$ is assumed to follow a gamma distribution with a specified mean of $\frac{1}{\mu'}$, where $\mu'$ is the default server's maximum processing rate. The request execution time varies on different servers

| $Server$ | $f_{i\_max}$ | $c_i$ | $\beta_i$ | $\alpha_i$ |
|---|---|---|---|---|
| Type1 | 1.8 | 65 | 7.5 | 222.22 |
| Type2 | 1.8 | 75 | 5 | 250.00 |
| Type3 | 2.4 | 60 | 60 | 229.17 |
| Type4 | 2.4 | 75 | 5.2 | 250.00 |
| Type5 | 3.0 | 90 | 4.5 | 250.00 |
| Type6 | 3.0 | 105 | 6.5 | 266.67 |
| Type7 | 3.2 | 90 | 4.0 | 237.50 |
| Type8 | 3.2 | 105 | 4.4 | 253.13 |

**Table 2. Parameters of a 128-Server Cluster**

and is assumed to be reciprocally proportional to a server's capacity. Assuming small requests, their desired average response time $\hat{R}$ is set at $1$ second.

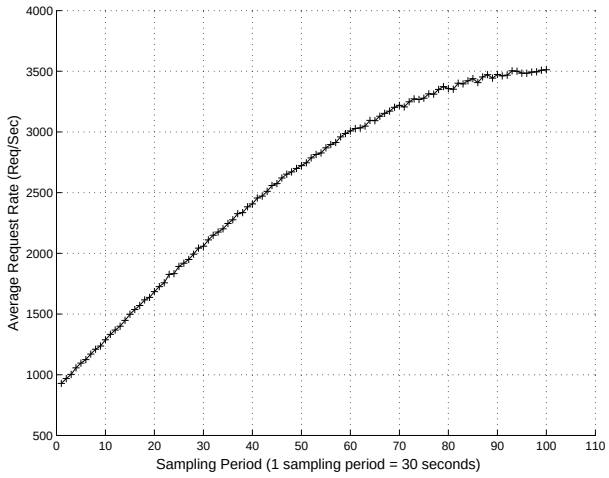


**Figure 2. Average Request Rate**

By offline analysis, a threshold-based algorithm derives the *ordered server list*, *server activation thresholds* and *workload distribution formulas* for a cluster based on the server parameters. Once these three modules are deployed on the head node, the cluster is able to handle different levels of workload. To evaluate an algorithm's performance, we use two metrics: the average response time and the consumed power. For all figures in this paper, we demonstrate the algorithm's performance with the time-varied workload $\lambda_{cluster}(t)$ as shown in Figure 2. Each simulation lasts $3000$ seconds. Periodically, i.e., every $30$ seconds, the system measures the current workload and predicts the average request rate $\lambda_{cluster}(t)$ for the next period. We adopt a method proposed in [10] for the workload prediction. Based on the range the predicted $\lambda_{cluster}(t)$ falls into, the corresponding power management decisions on server on/off ($x_i$) and workload distribution ($\lambda_i$) are followed. According to $\lambda_i$, the back-end server DVS mechanism decides the server's frequency setting $f_i$. In this paper, we use curves to show the average response time, while for clarity, both curves and bar figures are used to illustrate the power consumption.

We evaluate the effects of major design choices and compare the proposed algorithms in Sections 6.1 and 6.2. Section 6.3 compares the threshold-based algorithms with the optimal power management solution. In Section 6.4, we experimentally evaluate the feedback control mechanism's impact on the back-end server DVS.

### 6.1 Effects of Ordered Server List

We first evaluate an algorithm's performance with respect to different policies in deciding the *ordered server list*. Our heuristic: *T*ypical *P*ower based policy (TP) and baseline strategies: *A*ctivate *A*ll policy (AA), *RAN*dom policy (RAN), *S*tatic *P*ower based policy (SP) and *P*seudo *D*ynamic *P*ower based policy (PDP) are compared. We evaluate the following algorithms: TP-CA-CA, AA-AA-CA, RAN-CA-CA, SP-CA-CA and PDP-CA-CA. Except for AA-AA-CA, which activates all servers, the other algorithms only differ in the *ordered server list* but have the same capacity based (CA) strategies for deciding *server activation thresholds* and *workload distribution*. Figures 3 and 5 show the simulation results.

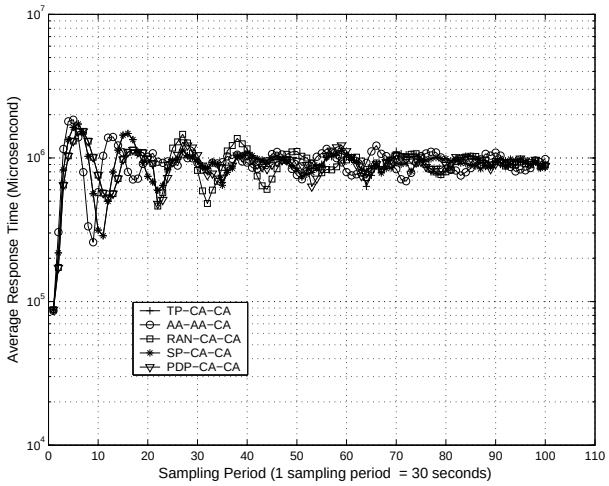


**Figure 3. Effects of Ordered Server List: Time**

Since algorithms adopt capacity based (CA) strategies for deciding *server activation thresholds* and *workload distribution*, we can see from Figure 3 they all achieve the response time goal and keep the average response time around 1 second. One interesting observation is that the *A*ctivate *A*ll policy (AA) does not decrease the response time. The reason is on a back-end server, the local DVS mechanism always sets the CPU frequency at the minimum level that satisfies the time requirement. Therefore, even though AA policy turns on all back-end servers, it does not lead to reduced response times.

Figure 5 shows the power consumption with the increasing cluster workload. Algorithm TP-CA-CA, built on our *T*ypical *P*ower based policy (TP), always consumes the least power. It performs especially well at a low/medium cluster request rate when a good power management mechanism is needed the most. As workload increases, all back-end servers have to be activated and the algorithms begin to have similar performance. From this experiment, we demonstrate that the *server activation order* has

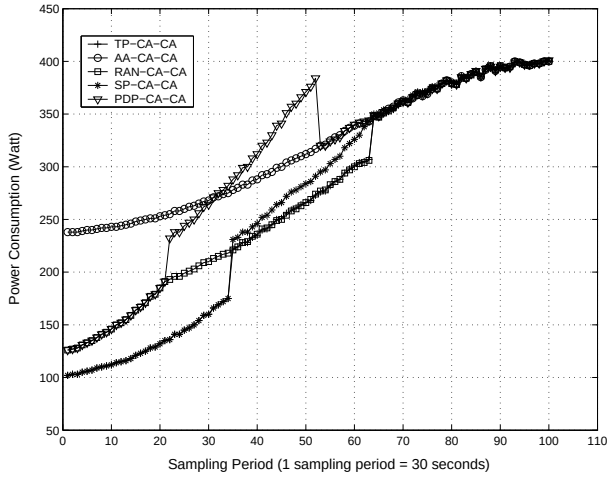


**Figure 4. Effects of Ordered Server List: Power**

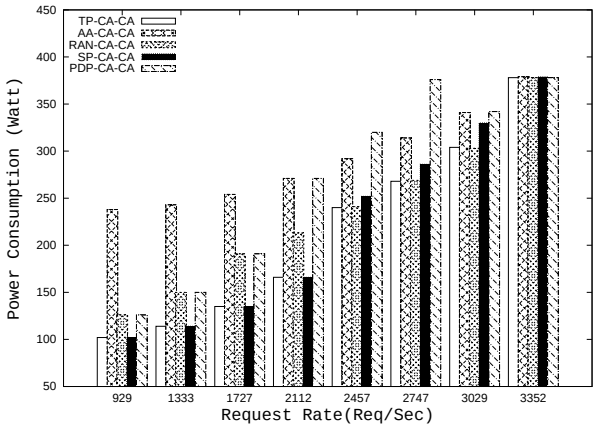


**Figure 5. Effects of Ordered Server List: Power**

a big impact on the power efficiency. When adopting a bad order, such as that by *RAN*dom policy (RAN) or *P*seudo *D*ynamic *P*ower based policy (PDP), a high level of power is consumed. Occasionally, the *P*seudo *D*ynamic *P*ower based policy (PDP-CA-CA) performs even worse than the *A*ctivate *A*ll policy (AA-AA-CA). It shows under such scenarios activating more servers consumes less power.

### 6.2 Effects of Activation Thresholds and Workload Distribution

In this subsection, to evaluate polices that decide *server activation thresholds* and *workload distribution* we simulate the following algorithms: RAN-CP-OP that is based on our heuristic and RAN-CA-OOS, RAN-CA-CA and RAN-CA-RAN baseline algorithms. For RAN-CP-OP, the last two modules are combined together since optimal-power thresholds depend on the optimal workload distribution. Therefore we evaluate the two polices together. For these algorithms, a common *RAN*domly generated *ordered server list* is used.

Figures 6 and 8 show the simulation results. From Figure 6, we can see that algorithm RAN-CA-RAN fails to provide response time guarantee: under several workload conditions, the average response time goes above the 1 second target. The reason is for a heterogeneous cluster, this *RAN*dom (uniform)

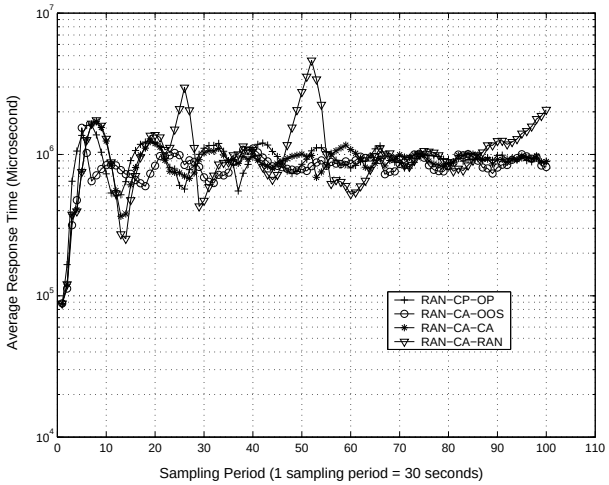


**Figure 6. Effects of Activation Thresholds and Workload Distribution: Time**

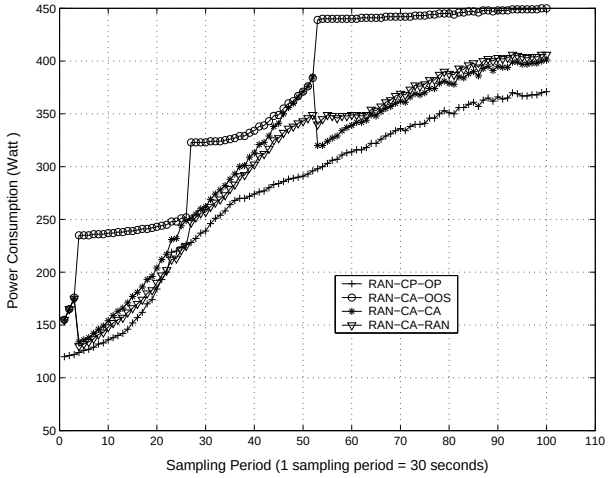


**Figure 7. Effects of Activation Thresholds and Workload Distribution: Power**

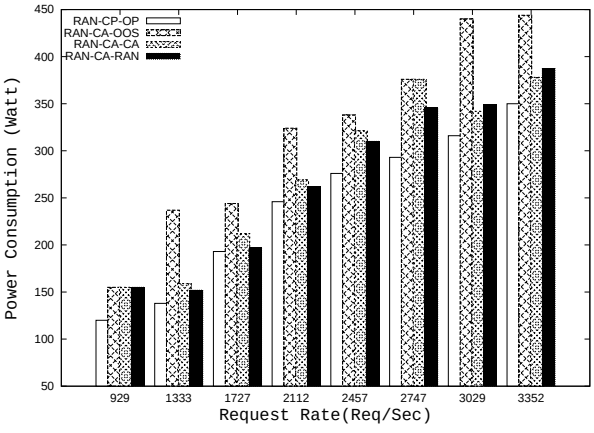


**Figure 8. Effects of Activation Thresholds and Workload Distribution: Power**

workload distribution does not prevent a server from being overloaded. Even though the *CA*pacity-based server activation policy has ensured that the cluster capacity is adequate to handle the workload, the bad workload distribution still causes the QoS violation. Since all other algorithms consider a server's capacity for workload distribution, they meet the time requirement.

Figure 8 illustrates the power consumption results. Under all scenarios, the algorithm based on our heuristic, RAN-CP-OP, always consumes the least power. In addition, unlike other three algorithms, RAN-CP-OP's power consumption increases monotonically and smoothly with the workload. The main reasons behind these results are as follows.

**More Servers but Less Power.** As discussed in Section 5.3, more servers do not always consume more power. Our *C*apacity-*P*ower-based strategy (CP) takes this factor into account. For example, while $\lambda_{cluster}(t) = 929$ req/sec, the baseline *CA*pacity-only based algorithms activate one server and when $\lambda_{cluster}(t) = 2747$ req/sec, they activate three servers. In contrast, our algorithm RAN-CP-OP turns on two and four servers respectively under these two scenarios. It leads to much less power consumptions. When $\lambda_{cluster}(t)$ increases to $2800$ req/sec, RAN-CA-CA algorithm turns on the forth server. The result is that, with four servers its power consumption for a heavier workload (say $3029$ req/sec) is *less* than that of three servers for a lighter workload (say $2747$ req/sec).

**Optimal Workload Distribution.** Our heuristic forms and solves the workload distribution as an optimization problem. The simulation results demonstrate that the resultant distribution is indeed optimal. In Figure 8, When $\lambda_{cluster}(t)$ is greater than $2800$ req/sec, four algorithms all activate the same number of servers. But our algorithm RAN-CP-OP still consumes the least power due to its optimal distribution of the workload. Unlike RAN-CP-OP, algorithm RAN-CA-OOS experiences a sudden change of the consumed power whenever a new server is activated. For this *O*ne-by-*O*ne *S*aturation strategy (OOS) on workload distribution, after adding an active server, its static power consumption increases but its dynamic power consumption does not decrease because it does not reduce the workload distributed to the other servers. Thus, their dynamic power consumption does not decrease. As we observe, this strategy leads to the highest power consumptions.

### 6.3 Evaluation of Integrated Algorithms

This subsection evaluates the following integrated algorithms: our heuristic TP-CP-OP and AA-AA-CA, SP-CA-CA and PDP-CA-CA baseline algorithms. When choosing baseline algorithms for comparison, we exclude the "deficient" algorithms, i.e., those based on RAN and OOS workload distribution policies. In addition, we compare these algorithms with the optimal power management solution: OPT-SOLN. To obtain the optimal solution, we solve the power management problem, i.e., Equations (8) and (9), for all integer points $\lambda_{cluster}$ in the (0, $\hat{\lambda}_{cluster}$] range. The optimal server on/off ($x_i$) and workload distribution ($\lambda_i$) is recorded for every $\lambda_{cluster}$. Dynamically, based on the predicted $\lambda_{cluster}(t)$, the corresponding optimal configuration is followed.

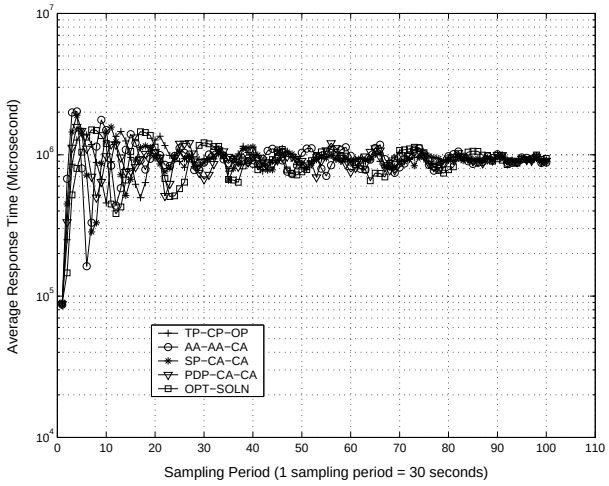


**Figure 9. Integrated Algorithms: Time**

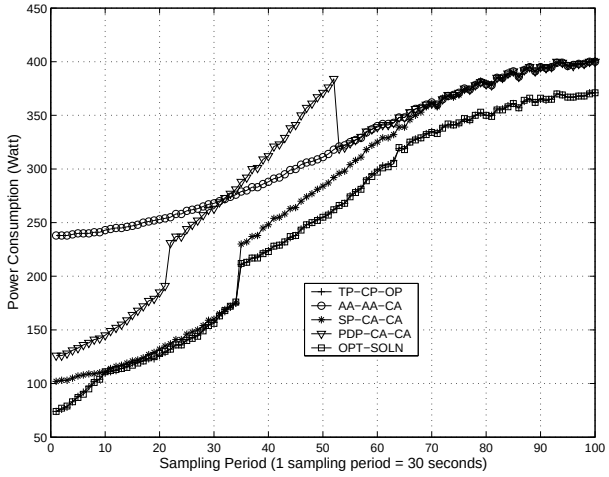


**Figure 10. Integrated Algorithms: Power**

Figures 9 and 11 respectively show the average response time and the power consumption. As expected, our algorithm TP-CP-OP performs better or as good as the baseline algorithms under all scenarios. Compared to the results of OPT-SOLN, our heuristic TP-CP-OP leads to only a negligible, $0.09\%$, more power consumption. In addition, for the simulated workload, the OPT-SOLN algorithm switches on/off back-end servers for a total of $12$ times, while our algorithm TP-CP-OP only turns on the $4$ servers at their individual appropriate moments following *ordered server list*. Although our current simulator does not simulate the server on/off overhead, in real clusters it usually takes several minutes and consumes some extra power to turn on/off a machine. Following the threshold-based approach, our algorithm minimizes the server on/off overhead, which will lead to better QoS performance and smaller power consumptions. As an interesting future work, we plan to compare our algorithm TP-CP-OP with the "optimal" algorithm OPT-SOLN to see which algorithm will perform better in real cluster environments.

**Effects of Discrete Frequencies.** So far we have assumed that the CPU frequency could be tuned continuously. However,

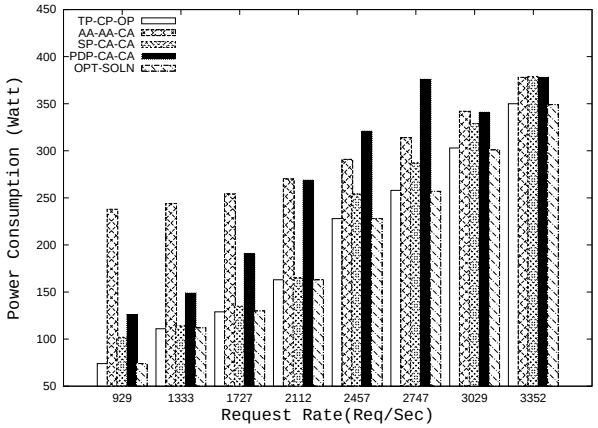


**Figure 11. Integrated Algorithms: Power**

current commercial processors only support DVS with a limited number of frequencies. For example, Intel Pentium M $1.6GHz$ CPU supports 6 voltages from $0.956V$ to $1.484V$, thus leading to 6 different frequencies.

Next we, therefore, evaluate our algorithm's performance on servers with discrete frequency settings. We simulate the same 4-server cluster but assume a server's frequency can only be set to 10 discrete values in the $[f_{i\_min}, f_{i\_max}]$ range, where $f_{i\_min}$ is assumed to be $37.5\%$ of $f_{i\_max}$. To satisfy the response time requirement and to save power, out of the 10 levels, the back-end server DVS chooses the smallest adequate frequency. We again combine feedback control with queuing-theoretic prediction for the DVS. A discrete feedback control approach similar to that in [13] is adopted.

The simulation results are showed in Figures 12 and 14. We can see in Figure 12 that due to the constraint of discrete frequencies, the resultant response time has a larger fluctuation around the target. Comparing Figure 14 with Figure 11, similar power consumptions are achieved and the power consumption ranking of the algorithms does not change and our algorithm still consumes the least power.

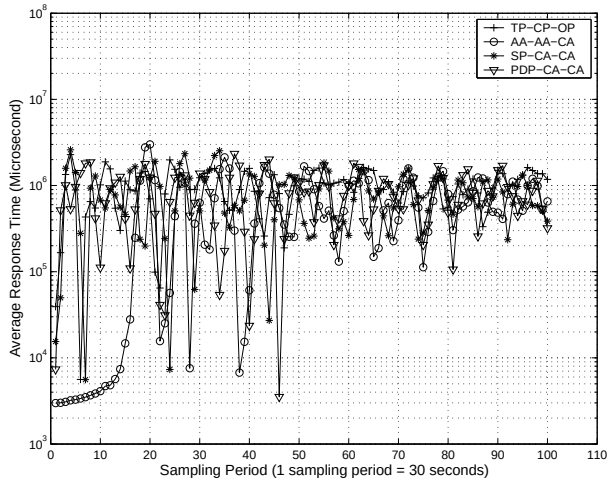


**Figure 12. Effects of Discrete Frequencies: Time**

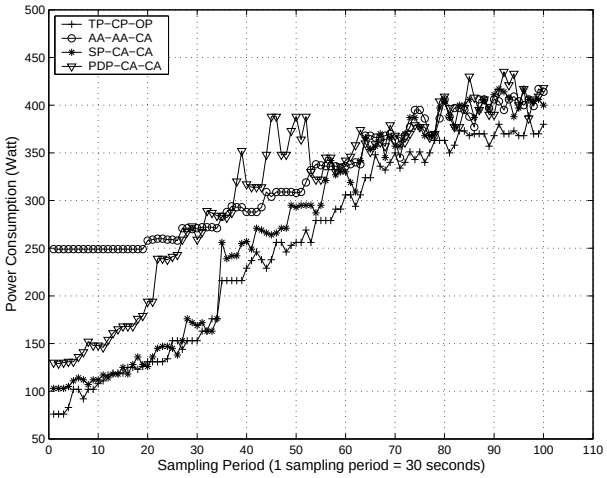


**Figure 13. Effects of Discrete Frequencies: Power**

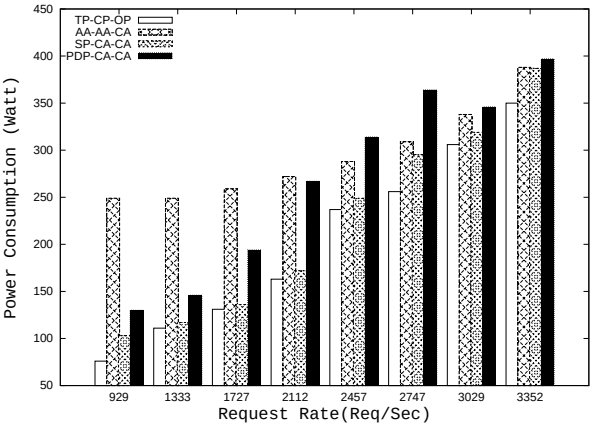


**Figure 14. Effects of Discrete Frequencies: Power**

## 6.4 Effects of Feedback Control

As described in Section 4, to overcome the inaccuracy of $M/M/1$ queuing model, we apply a combined feedback control with queuing-theoretic prediction mechanism for back-end server DVS. This section evaluates the feedback control mechanism's impact. We compare the combined DVS mechanism with a queuing prediction only DVS mechanism where no feedback control is applied.

Figure 15 shows the average response time when the feedback control is not applied. As we can see, due to the modeling inaccuracy, the resultant response time is not close to the 1 second target. In contrast, when the feedback control is combined with the queuing-theoretic prediction, the average response time, as shown in Figure 9, is kept around the target. These results demonstrate that the feedback control mechanism is effective in regulating the response time.

On the other hand, when comparing the power consumption of DVS mechanisms with and without feedback control, the differences are negligible. For illustration, Figure 16 presents the power consumption curves of TP-CP-OP algorithm with and without feedback DVS control. On average, the feedback control mechanism only reduces the frequency by $0.000925$GHz and the power by $0.66$Watts.

The aforementioned results show that the average response time is sensitive to the operating frequency changes. A small frequency change can lead to a large difference in response time. As a result, although the feedback control mechanism is effective in regulating the response time, it only slightly modifies the queuing estimated frequency $f_i$ and leads to a little bit better power consumptions.

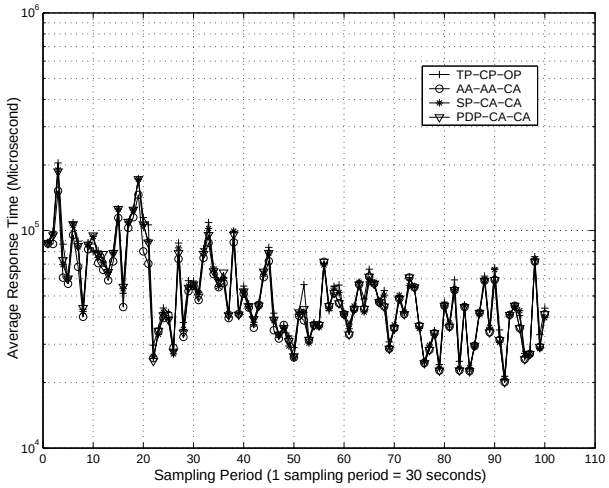


**Figure 15. Effects of Feedback Control: Time**

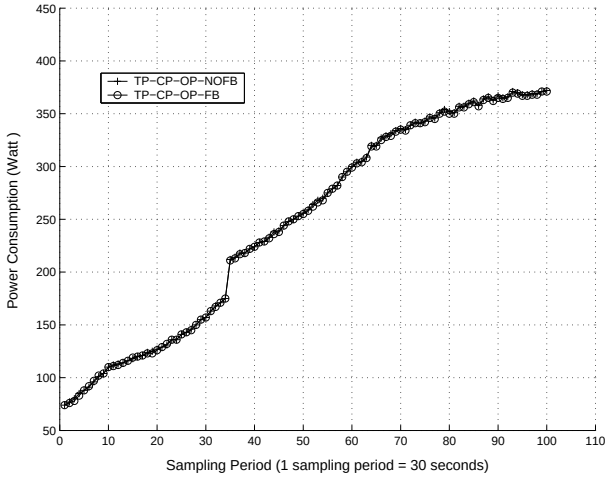


**Figure 16. Effects of Feedback Control: Power**

## 6.5 Performance on Large Cluster

In practice, large cluster typically runs with hundreds of back-end nodes. Therefore, in this subsection, we evaluate the algorithm's performance on a large cluster with 8 types of 128 nodes (see Table 2 for their parameters). Similar to Section 6.3, we compare three baseline algorithms: AA-AA-CA, SP-CA-CA and PDP-CA-CA with our heuristic: TP-CP-OP. Figures 17, 18

and 19 show the simulation results. As we can see, before the system workload reaches around 47444 req/sec, AA-AA-CA consumes much more power than algorithms with power on/off mechanisms. But as workload increases, AA-AA-CA outperforms SP-CA-CA and PDP-CA-CA algorithms. This result again proves that more servers do not always consume more power. Our algorithm TP-CP-OP considers both static and dynamic power efficiencies. Its mechanisms on power on/off and workload distribution strive to achieve an optimal power consumption. As a result, it always performs the best in all workload conditions.

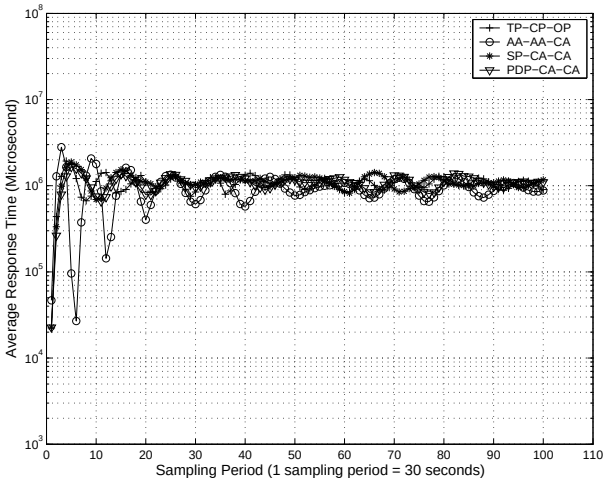


**Figure 17. Performance on a Large Cluster: Time**

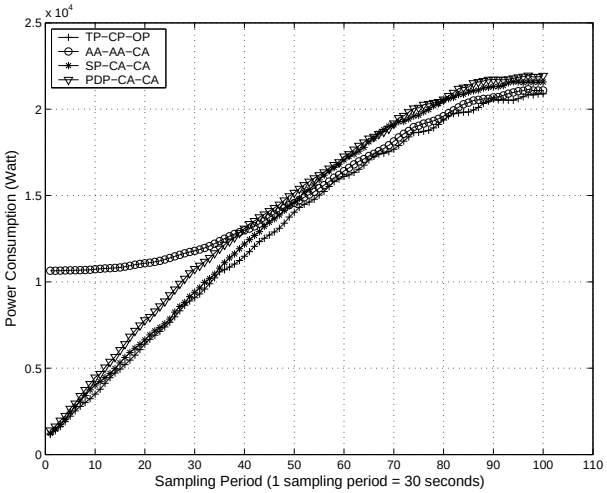


**Figure 18. Performance on a Large Cluster: Power**

## 7 Conclusion

In this paper, we present a new threshold-based approach for efficient power management of heterogeneous soft real-time clusters. Following this approach, a power management algorithm needs to make three important design decisions on *ordered server list*, *server activation thresholds* and *workload distribution*. We systematically study this approach and the impact of these design decisions. A new algorithm denoted as TP-CP-OP is proposed. When deciding the *server activation order*, the algorithm considers both static and dynamic power efficiencies. Its *server activation thresholds* and *workload distribution* are explicitly designed to achieve optimal power consumption. By simulation, we clearly demonstrate the algorithm's advantages in power consumption: it incurs low overhead and leads to near-optimal power consumption.

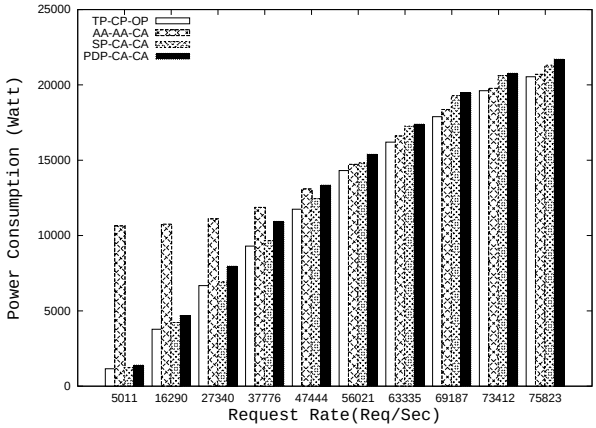


**Figure 19. Performance on a Large Cluster: Power**

## References


[1] L. A. Barroso. The price of performance. *Queue*, 3(7):48–53, 2005.

[2] R. Bianchini and R. Rajamony. Power and energy management for server systems. *Computer*, 37(11):68–74, 2004.

[3] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony. The case for power management in web servers. pages 261–289, 2002.

[4] J. Chase and R. Doyle. Balance of power: Energy management for server clusters. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS'01)*, May 2001.

[5] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing server energy and operational costs in hosting centers. In *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 303–314, New York, NY, USA, 2005. ACM Press.

[6] M. Chin. Desktop cpu power survey. *Silentpcreview.com*, April 2006.

[7] E. Chong and S. H. Żak. *An Introduction to Optimization*. Wiley.

[8] M. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. In *Proceedings of the Second Workshop on Power Aware Computing Systems*, February 2002.

[9] T. Hardware. Cpu performance charts. *Tom's Hardware*, 2006.

[10] J. P. Hayes. Self-optimization in computer systems via on-line control: Application to power management. In *ICAC'04: Proceedings of the First International Conference on Autonomic Computing (ICAC'04)*, pages 54–61, Washington, DC, USA, 2004. IEEE Computer Society.

[11] T. Heath, B. Diniz, E. V. Carrera, W. M. Jr., and R. Bianchini. Energy conservation in heterogeneous server clusters. In *PPoPP*

*'05: Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 186–195, New York, NY, USA, 2005. ACM Press.

[12] J. Heo, D. Henriksson, X. Liu, and T. Abdelzaher. Integrating adaptive components: An emerging challenge in performance-adaptive systems and a server farm case-study. In *28th IEEE International Real-Time Systems Symposium*, pages 227–238, Tuscon, AZ,, December 2007.

[13] T. Horvath. Dynamic voltage scaling in multitier web servers with end-to-end delay control. *IEEE Trans. Comput.*, 56(4):444–458, 2007. Member-Tarek Abdelzaher and Senior Member-Kevin Skadron and Member-Xue Liu.

[14] L.Mastroleon, N.Bambos, C.Kozyrakis, and D.Economou. Autonomic power management schemes for internet servers and data centers. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*,, 2005.

[15] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *Proceedings of the International Workshop on Compilers and Operating Systems for Low Power*, May 2001.

[16] K. Rajamani and C. Lefurgy. On evaluating request-distribution schemes for saving energy in server clusters. In *ISPASS '03: Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 111–122, Washington, DC, USA, 2003. IEEE Computer Society.

[17] C. Rusu, A. Ferreira, C. Scordino, A. Watson, R. Melhem, and D. Mosse. Energy-efficient real-time heterogeneous server clusters. In *Proceedings of the Twelfth Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, April 2006.

[18] L. Sha, X. Liu, Y. Lu, and T. Abdelzaher. Queueing model based network server performance control. In *IEEE Real-Time Systems Symposium*, Austin, TX, December 2002.

[19] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu. Power-aware QoS management in web servers. In *RTSS '03: Proceedings of the 24th IEEE International Real-Time Systems Symposium*, page 63, Washington, DC, USA, 2003. IEEE Computer Society.