An Efficient Threshold-Based Power Management Mechanism for Heterogeneous Soft Real-Time Clusters

Leping Wang and Ying Lu

Abstract—With growing cost of electricity, the power management (PM) 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 PM 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.

Index Terms—Dynamic power management (PM), dynamic voltage scaling, heterogeneous clusters, soft real-time.

I. 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 (PM) mechanisms [6], [12], [14], [17], [24], [27] 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 PM 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 nontrivial task.

A few researchers [12], [17] have investigated mechanisms to address the aforementioned challenges. However, their mechanisms require either an extensive performance measurement (“at most few hours for each machine” [17]) or a time-consuming optimization procedure [12]. These high customization costs are prohibitive, especially if these procedures 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 PM of heterogeneous soft real-time clusters. Here, a soft real-time cluster is defined as a cluster that has an upper-bound constraint on the average response time but can nevertheless tolerate late responsiveness to some degree. In this paper, 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 PM 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 PM approach. Simulation results show that our algorithm incurs low overhead and leads to optimal power consumption.

The remainder of this paper is organized as follows. The related work is illustrated in Section II. Sections III and IV, respectively, present the models and state the problem. We discuss the algorithms in Section V and evaluate their performance in Section VI. Section VII concludes this paper.

II. RELATED WORK

Power management of server clusters [2], [6], [15], [16], [28] has become an important problem. The authors of [3], [5] 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 [9], [14] to conserve power in clusters.

The aforementioned research has all focused on homogenous systems. However, clusters are almost invariably...
heterogeneous in term of their performance, capacity and power consumption [12]. Survey [4] discusses the recent work on PM for server systems. It lists PM of homogeneous clusters as one of the major challenges.

Heath et al. [12] considered proper cluster configuration and request distribution to optimize power and throughput in heterogeneous server clusters. Their mechanism takes characteristics of different nodes and request types into account. However, this approach depends on a time-consuming optimization procedure. Besides, it is not designed for real-time clusters.

Some researchers [17], [24], [26], [27] have investigated the PM problem for heterogeneous real-time systems. Among them, the most closely related work is by Rusu et al. [17], where they investigated energy efficient real-time heterogeneous clusters. Like Heath et al. [12], the authors of that paper [17] note that in heterogeneous clusters it may not be optimal in terms of power consumption to turn on just the smallest number of machines to satisfy the current load. They assume that servers in a heterogeneous cluster can be easily ordered with respect to their power efficiencies and thus servers are powered on/off following the power efficiency order to optimize power and provide real-time guarantees. However, our work shows that the proper order for switching on/off servers is not always obvious. In this paper, therefore, we systematically study different server ordering schemes (see Section V-B for details).

The approach by Rusu et al. [17] also requires an extensive performance measurement, which needs to be carried out upon new installations, cluster upgrades or changes. Extensive performance measurements [17] and long optimization procedures [12] lead to high customization costs. To avoid these prohibitive costs, we propose in this paper a simple PM 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 procedure. Furthermore, our algorithm incurs low overhead and achieves optimal power consumption. We have published our preliminary work in a conference paper [22]. This paper significantly extends that work [22]—we tailor our PM algorithm and make it also applicable to clusters, where servers have significant switch on/off overheads and their CPUs only support discrete frequencies. Our research focuses on clusters that are composed of single-processor servers which provide CPU-bounded services. How to control SMP, SMT architectures, or I/O intensive clusters is out of the scope of this paper.

III. MODELS

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

A. 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 PM mechanism on the front-end server to enforce a server power on/off policy. Fig. 1 shows a web server cluster example that fits our system model.

In a heterogeneous cluster, different back-end servers could have different computational capacities and power efficiencies. In the following, we describe their models. We assume processors on the back-end servers support dynamic voltage scaling and their operating frequencies could be continuously adjusted in the range \((0, f_{i,\text{max}}]\). If a processor only supports discrete frequencies, we follow an approach similar to that proposed in [21] 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.

B. Capacity Model

We assume that the cluster provides CPU-bounded services. This is normal for many web servers where much of the data are kept in memory [5], [17], [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 \(\mu_i = \alpha_i f_i\), 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.

C. Power Model

The power consumption \(P_i\) of a server consists of a constant part and a variable part. Similar to previous work [6], [9], [13], we approximate \(P_i\) by the following function:

\[
P_i = x_i (\alpha_i + \beta_i f_i^p) \quad (1)
\]
where \( x_i = 1/0 \) denotes the server’s on/off state. 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:

\[
J = \sum_{i=1}^{N} x_i \left[ c_i + \beta_i f_i^3 \right]. \tag{2}
\]

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: \( f_{i,\text{max}}, \alpha_i, c_i, \) and \( \beta_i \). To obtain these parameters, only a little performance profiling is required.

**IV. POWER MANAGEMENT PROBLEM**

Given a cluster of \( N \) heterogeneous back-end servers, each specified with parameters \( f_{i,\text{max}}, \alpha_i, c_i, \) and \( \beta_i \), the objective is to minimize the power consumption, while guaranteeing the soft real-time property, i.e., satisfying \( R_i \leq \hat{R} \) most of the time, where \( R_i \) stands for the average response time of requests processed by the \( i \)th back-end server and \( \hat{R} \) the response time upper-bound. Since shorter response time means higher power consumption, to ensure the soft real-time property while minimizing power consumption, it is adequate to satisfy the following requirement: \( R_i \approx \hat{R} \). The average 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 PM problem is formed as follows:

\[
\text{minimize} \quad J = \sum_{i=1}^{N} x_i \left[ c_i + \beta_i f_i^3 \right] \tag{3}
\]

subject to:

\[
\left\{ \begin{array}{l}
\sum_{i=1}^{N} x_i \lambda_i = \lambda_{\text{cluster}} \\
x_i(1 - x_i) = 0, \quad i = 1, 2, \ldots, N \\
g(\alpha_i f_i, \lambda_i) \approx \hat{R}, \quad i = 1, 2, \ldots, N
\end{array} \right. \tag{4}
\]

where \( \lambda_{\text{cluster}} \) 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 \( \lambda_{\text{cluster}} \) request rate.\(^1\)

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 off state.

For the PM, 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, an approach that combines feedback control with queueing theoretic prediction, similar to that proposed in [18], is adopted to decide the CPU frequency setting to ensure the response time requirement. Accordingly, each active server adjusts its CPU operating frequency \( f_i \) in the range \( \big[0, f_{i,\text{max}}\big] \). In the case where CPU frequency can only be set to some discrete values, to approximate the desired continuous frequency setting, we switch the CPU frequency at appropriate time moments to two adjacent supported discrete values.

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

\[
R_i = \frac{1}{\mu_i - \lambda_i} = \frac{1}{\alpha_i f_i - \lambda_i}. \tag{5}
\]

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

\[
f_i = \frac{\lambda_i}{\alpha_i} + \frac{1}{\alpha_i \hat{R}} \tag{6}
\]

when \( 0 < \lambda_i \leq \alpha_i f_{i,\text{max}} - (1/\hat{R}) \). This approximation, however, may introduce modeling inaccuracy. To overcome this inaccuracy, we combine feedback control with queueing-theoretic prediction for the dynamic voltage scaling (DVS).

According to (4) and (5), the offline analysis generates thresholds \( \lambda_i \) for servers, so we can decide if the average response time is feasible. The on-line feedback manager uses these thresholds to control the CPU frequency in the range \( \big[0, f_{i,\text{max}}\big] \) to guarantee the soft real-time property.

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.

**V. ALGORITHMS**

When we design the PM strategies, one major focus is on their efficiencies. For a given workload \( \lambda_{\text{cluster}} \), the front-end PM 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 \( \lambda_{\text{cluster}} \) 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 \( \lambda_{\text{cluster}} \) is the maximum workload that can be handled by the cluster without violating the average response time requirement. The offline analysis generates thresholds \( \lambda_1, \lambda_2, \ldots, \lambda_N \) and divides \( (0, \lambda_{\text{cluster}}) \) into ranges \( (0, \lambda_1], (\lambda_1, \lambda_2], \ldots, (\lambda_{N-1}, \lambda_N], (\lambda_N, \lambda_{\text{cluster}}] \).
For each range, the power on/off and workload distribution decisions are made offline. Dynamically, the system just measures and predicts the workload $\lambda_{\text{cluster}}$, decides the range $\lambda_{\text{cluster}}$ falls into, and follows the corresponding PM decisions. Next, we present the details of our algorithm.

A. Optimization Heuristic Framework

In Section IV, the PM is formed as an optimization problem [(7) and (8)]. Instead of solving it for all possible workload $\lambda_{\text{cluster}}$ in the range $[0, \hat{\lambda}_{\text{cluster}}]$, 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 $\lambda_{\text{cluster}}$ is in the range $(\Lambda_{k-1}, \Lambda_k]$, it is optimal to turn on the first $k$ servers of the ordered server list. This also means if $\lambda_{\text{cluster}}$ 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., tens of seconds), 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 $\lambda_{\text{cluster}} \in (\Lambda_{k-1}, \Lambda_k]$, $k = 1, 2, \cdots, N$. When $\lambda_{\text{cluster}} \in (\Lambda_{k-1}, \Lambda_k]$, 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 [(7) and (8)] becomes

\[
\text{minimize } J_k = \sum_{i=1}^{k} \left[ \frac{c_i + \beta_i \times \left( \frac{\lambda_i}{\alpha_i} + \frac{1}{\alpha_i R} \right)^3}{\alpha_i \lambda_i} \right] \quad (9)
\]

subject to:

\[
\left\{ \sum_{i=1}^{k} \lambda_i = \lambda_{\text{cluster}} \right\} \quad (10)
\]

The analysis is simplified to solving the above optimization problem for $k = 1, 2, \cdots N$. In contrast, to obtain the optimal PM solution [i.e., solving (7) and (8)] by an exhaustive search of all possible server on/off scenarios] for every integer point in the range $(0, \hat{\lambda}_{\text{cluster}}]$, requires solving $\left[ \hat{\lambda}_{\text{cluster}} \right]^{2N}$ number of problem instances in the form of (9) and (10).

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.

B. 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., $P_i(\lambda)/\lambda$). 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 III and IV), the power consumption $P_i(\lambda)$ of a server includes a constant part $c_i$ and a variable part $\beta_i \times ((\lambda/\alpha_i) + (1/\alpha_i R))^3$ [see (7)]. Given any two servers $i$ and $j$, if $c_i \leq c_j$ and $(\beta_i / \alpha_i)^3 \leq (\beta_j / \alpha_j)^3$, server $i$ has a better power efficiency than server $j$. However, if $c_i > c_j$ and $(\beta_i / \alpha_i)^3 > (\beta_j / \alpha_j)^3$, 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.

- Typical Power-based policy (TP). We assume the typical workload for a server is $X_i$. In our heuristic, servers are ordered by their power consumption efficiency under the typical workload, i.e., $P_i(X_i)/X_i$. A server with smaller $P_i(X_i)/X_i$ i.e., smaller $(c_i + \beta_i \times ((X_i/\alpha_i) + (1/\alpha_i R))^3)/X_i$, is listed earlier in the ordered server list. A PM mechanism usually turns on a server when needed or when it leads to a reduced power consumption (see Section V-C). 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 $X_i$. This way the ordered server list is created by comparing $P_i(X_i)/X_i$ and is solely based on the server’s static parameters $c_i$, $\beta_i$, and $\alpha_i$.

- Activate All 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.

- RANdom policy (RAN). This policy generates a random ordered server list for server activation.

- Static Power-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.

- Pseudo Dynamic Power-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 $P_i(X_i)/X_i$, its dynamic part is $((\beta_i / \alpha_i)^3 \times (X_i + (1/\alpha_i R))^3)/X_i$. 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.

C. 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 twofold. 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.

Following our mechanism, to meet the response time requirement, the number of active servers should increase monotonically with the workload $\lambda_{\text{cluster}}$. 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 $\Lambda_{c1}, \Lambda_{c2}, \ldots, \Lambda_{cN}$ are developed and each $\Lambda_{ck}$ corresponds to the maximum workload that can be processed by the first $k$ servers. According to (5), when a server is operating at its maximum frequency $f_{i,\text{max}}$, it can process at most $\lambda_{i,\text{max}}$ amount of workload and meet the response time requirement

$$\lambda_{i,\text{max}} = \frac{c_i f_{i,\text{max}}}{R_i}.$$  
Thus, we have

$$\Lambda_{ck} = \sum_{i=1}^{k} \lambda_{i,\text{max}} = \sum_{i=1}^{k} \frac{c_i f_{i,\text{max}}}{R_i}.$$  

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, servers are adequate to handle the workload. However, 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 $\lambda_{\text{cluster}}$ workload, where $\lambda_{\text{cluster}} \in (0, \Lambda_{ck})$, is $J_k(\lambda_{\text{cluster}})$ (see Section V-D for $J_k(\lambda_{\text{cluster}})$’s derivation). It is a monotonically increasing function of $\lambda_{\text{cluster}}$. We analyze the following equation:

$$J_k(\lambda_{\text{cluster}}) = J_{k+1}(\lambda_{\text{cluster}}),$$  

According to characteristics of functions $J_k(\lambda_{\text{cluster}})$ and $J_{k+1}(\lambda_{\text{cluster}})$ (see Section V-D), there is at most one solution for (13). If such a solution $\lambda'_{\text{cluster}}$ is found, then activating $k + 1$ servers is more power efficient than activating $k$ servers when $\lambda_{\text{cluster}} > \lambda'_{\text{cluster}}$. Here is a sketch of the proof: 1) $J_k(\lambda_{\text{cluster}})$ is less than $J_{k+1}(\lambda_{\text{cluster}})$ for small $\lambda_{\text{cluster}}$: 2) functions $J_k(\lambda_{\text{cluster}})$ and $J_{k+1}(\lambda_{\text{cluster}})$ increase monotonically with $\lambda_{\text{cluster}}$: and 3) if and only if $\lambda_{\text{cluster}} = \lambda'_{\text{cluster}}$ activating $k$ or $k + 1$ servers consumes the same amount of power. Therefore, once $\lambda_{\text{cluster}} > \lambda'_{\text{cluster}}$, $J_k(\lambda_{\text{cluster}})$ becomes smaller than $J_{k+1}(\lambda_{\text{cluster}})$, i.e., it becomes more power efficient to activate $k + 1$ servers.

To solve the optimization for $J_k$, we first assume that all $k$ back-end servers are running below their maximum capacities, i.e., $0 \leq \lambda_i < \alpha_i f_{i,\text{max}} - (1/R_i), i = 1, \ldots, k$. Since the second constraint of the problem is satisfied, the optimization becomes

$$\text{minimize}$$  
$$J_k = \sum_{i=1}^{k} \left[ c_i + \beta_i \times \left( \frac{\lambda_i}{\alpha_i} + \frac{1}{\alpha_i R_i} \right)^3 \right],$$  

subject to:

$$\sum_{i=1}^{k} \lambda_i = \lambda_{\text{cluster}},$$  

$$0 \leq \lambda_i \leq \alpha_i f_{i,\text{max}} - \frac{1}{R_i}, \quad i = 1, \ldots, k.$$  

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

According to Section V-A, if the first $k$ servers of the ordered server list are activated, the optimization problem becomes

$$\text{minimize}$$  
$$J_k = \sum_{i=1}^{k} \left[ c_i + \beta_i \times \left( \frac{\lambda_i}{\alpha_i} + \frac{1}{\alpha_i R_i} \right)^3 \right],$$  

subject to:

$$\sum_{i=1}^{k} \lambda_i = \lambda_{\text{cluster}},$$  

According to Lagrange’s Theorem [8], the first-order necessary condition for $J_k$’s optimal solution is:

$$\exists \delta, \quad J_k(\lambda_i, \delta) = \sum_{i=1}^{k} \left[ c_i + \beta_i \times \left( \frac{\lambda_i}{\alpha_i} + \frac{1}{\alpha_i R_i} \right)^3 \right] + \delta \left( \sum_{i=1}^{k} \lambda_i - \lambda_{\text{cluster}} \right)$$  

and its first-order derivatives satisfy

$$\left\{ \begin{array}{ll}
\frac{\partial J_k(\lambda_i, \delta)}{\partial \lambda_i} = 0, & i = 1, \ldots, k \\
\frac{\partial J_k(\lambda_i, \delta)}{\partial \delta} = 0.
\end{array} \right.$$
Solving the above condition, we obtain the optimal workload distribution \( \lambda_i, i = 1, \ldots, k \) as

\[
\lambda_i = \frac{\alpha_i \left( \lambda_{\text{cluster}} + \frac{k}{R} \right) \sqrt{\frac{\alpha_i}{\beta_i}} - 1}{R}. \quad (20)
\]

The corresponding power consumption is

\[
k = \sum_{i=1}^{k} c_i + \left( \lambda_{\text{cluster}} + \frac{k}{R} \right)^3 \sqrt{\frac{\alpha_i}{\beta_i}}. \quad (21)
\]

The above solution is optimal when all \( k \) back-end servers are running below their maximum capacities. That is, when \( \lambda_i \) (20) satisfies the constraint that \( 0 < \lambda_i < \alpha_i f_{1,\text{max}} - (1/R), \) \( i = 1, 2, \ldots, k. \) Thus, the above condition holds true only for light cluster workloads. As \( \lambda_{\text{cluster}} \) increases, servers start to be saturated one after another. That is, a server’s shared workload \( \lambda_i \) reaches its maximum level \( \alpha_i f_{1,\text{max}} - (1/R) \) where we have

\[
\lambda_i = \frac{\alpha_i \left( \lambda_{\text{cluster}} + \frac{k}{R} \right) \sqrt{\frac{\alpha_i}{\beta_i}} - 1}{R} = \alpha_i f_{1,\text{max}} - \frac{1}{R}. \quad (22)
\]

Solving (22) for system workload \( \lambda_{\text{cluster}}, \) we get

\[
\lambda_{\text{cluster}} = f_{1,\text{max}} \sqrt{\frac{k}{\alpha_i}} \sum_{j=1}^{k} \alpha_j \sqrt{\frac{\alpha_j}{\beta_j}} - \frac{k}{R}. \quad (23)
\]

This result seems to indicate that among the \( k \) active servers, the one with a smaller value of \( f_{1,\text{max}} \sqrt{\beta_j/\alpha_i} \) reaches its full capacity earlier as \( \lambda_{\text{cluster}} \) increases. We therefore order the \( k \) servers by their \( f_{1,\text{max}} \sqrt{\beta_j/\alpha_i} \) 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 \( R_k \) 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 \( \lambda_1 = \alpha_1 f_{1,\text{max}} - (1/R) \) and the system workload as

\[
\lambda_{\text{cluster}} = f_{1,\text{max}} \sqrt{\frac{k}{\alpha_1}} \sum_{j=1}^{k} \alpha_j \sqrt{\frac{\alpha_j}{\beta_j}} - \frac{k}{R}. \quad (24)
\]

The workload distribution problem becomes

\[
\begin{align*}
\text{minimize} & \quad J_k = \sum_{i=2}^{k} c_i + \beta_i \times \left( \frac{\lambda_i}{\alpha_i} + \frac{1}{\alpha_i R} + \epsilon \right)^3 \\
\text{subject to:} & \quad \sum_{i=2}^{k} \lambda_i = \lambda_{\text{cluster}} - \left( \alpha_1 f_{1,\text{max}} - \frac{1}{R} \right). \quad (25)
\end{align*}
\]

Here, servers are indexed following their saturated order list. Similar to (16) and (17), we solve the above problem by applying Larange’s Theorem and get the following optimal solution for \( \lambda_i, i = 2, 3, \ldots, k: \)

\[
\lambda_i = \frac{\alpha_i \left( \lambda_{\text{cluster}} + \alpha_i f_{1,\text{max}} + \frac{k}{R} \right) \sqrt{\frac{\alpha_i}{\beta_i}} - 1}{R}. \quad (27)
\]

The corresponding power consumption is

\[
J_k = \sum_{i=2}^{k} c_i + \beta_i \times \left( \frac{\lambda_i}{\alpha_i} + \frac{1}{\alpha_i R} + \epsilon \right)^3 + \beta_i f_{1,\text{max}}^3. \quad (28)
\]

Again, we let \( \lambda_i \) (27) be equal to the maximum workload \( \alpha_i f_{1,\text{max}} - (1/R) \) and solve for \( \lambda_{\text{cluster}}. \) We get

\[
\lambda_{\text{cluster}} = f_{1,\text{max}} \sqrt{\frac{k}{\alpha_i}} \sum_{j=2}^{k} \alpha_j \sqrt{\frac{\alpha_j}{\beta_j}} + \alpha_1 f_{1,\text{max}} - \frac{k}{R}. \quad (29)
\]

This result verifies our hypothesis that servers saturate following the saturated order list—the smaller the value of \( f_{1,\text{max}} \sqrt{\beta_j/\alpha_i} \) of the earlier server is saturated. The system workload that starts to saturate the first two servers is

\[
\lambda_{\text{cluster}} = f_{2,\text{max}} \sqrt{\frac{k}{\alpha_i}} \sum_{j=2}^{k} \alpha_j \sqrt{\frac{\alpha_j}{\beta_j}} + m \alpha_1 f_{1,\text{max}} - \frac{k}{R}. \quad (30)
\]

We define \( \lambda_{\text{m}}^k \) as

\[
\lambda_{\text{m}}^k = f_{m,\text{max}} \sqrt{\frac{k}{\alpha_i}} \sum_{j=m+1}^{k} \alpha_j \sqrt{\frac{\alpha_j}{\beta_j}} + \sum_{i=1}^{m-1} \alpha_i f_{1,\text{max}} - \frac{k}{R}. \quad (31)
\]

In general, when \( \lambda_{\text{cluster}} \in [\lambda_{\text{m}}, \lambda_{\text{m+1}}), m \) of the \( k \) active servers are saturated. That is, \( \lambda_i = \alpha_i f_{1,\text{max}} - (1/R, i = 1, 2, \ldots, m). \) The optimization problem becomes

\[
\begin{align*}
\text{minimize} & \quad J_k = \sum_{i=m+1}^{k} c_i + \beta_i \times \left( \frac{\lambda_i}{\alpha_i} + \frac{1}{\alpha_i R} \right)^3 \\
\text{subject to:} & \quad \sum_{i=m+1}^{k} \lambda_i = \lambda_{\text{cluster}} - \sum_{j=1}^{m} \left( \alpha_j f_{j,\text{max}} - \frac{1}{R} \right). \quad (32)
\end{align*}
\]

and the optimal solution is

\[
\begin{align*}
\lambda_i & = \frac{\alpha_i \left( \lambda_{\text{cluster}} - \sum_{j=m+1}^{m} \alpha_j f_{j,\text{max}} + k \right) \sqrt{\frac{\alpha_i}{\beta_i}} - 1}{R} \quad \text{for } i = m+1, m+2, \ldots, k \quad (34)
\end{align*}
\]
The above solution shows how to optimally distribute workload among active servers when they are in steady on states. However, since it takes some time (e.g., tens of seconds) to switch on a server and start software processes on it, there is a short server switch-on transient stage. During this transient interval, the to-be-active server cannot process any request yet, thus instead, the workload is distributed to active servers in proportion to their processing capacities. This temporary workload distribution method balances the load and avoids overloading the most power-efficient server during the transient stage. Once the transition is complete and the server is active and ready to process requests, the algorithm again begins to optimally distribute workload based on the aforementioned optimal solution.

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

- RANdom (uniform) workload distribution (RAN). In this strategy, every incoming request is distributed to a randomly picked active server.
- CApacity-based workload distribution (CA). This strategy distributes the workload among active servers in proportion to their processing capacities, i.e., \( \alpha_i f_i \max \).
- One-by-One Saturation policy (OOS). This policy distributes requests following a default order. For each incoming request, we pick the first active server that is not saturated to process it.

E. 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 V-A), we could generate many different algorithms by combining different strategies for the three 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.

VI. PERFORMANCE EVALUATION

In the previous section, we proposed various threshold-based strategies for the PM of heterogeneous soft real-time clusters. In this section, we experimentally compare their performance relative to each other, to an existing approach [17], and to the optimal solution.

A discrete simulator has been developed to simulate a range of heterogeneous clusters that are compliant to models presented in Section III. The server on/off switch overloads are also simulated. There are two types of switch overloads: time overhead and power overhead. It takes some time, assumed to be 10 s, to turn on/off a server, during which interval no service can be provided by the server. To simulate the power overhead, we assume in the switch on/off interval, a server \( S_i \) consumes power at the maximum level, i.e., \( P_i = c_i + \beta_i f_i \max \).

Cluster Configuration: The following clusters are simulated:

- **Cluster1.** First, we simulate a small cluster that consists of four back-end servers. They are all single processor machines: server 1 has an AMD Athlon 64 3000+ 1.8 GHz CPU; server 2 has an AMD Athlon 64 × 2 4800+ 2.4 GHz CPU; server 3 has an Intel Pentium 4 630 3.0 GHz CPU and server 4 has an Intel Pentium D 950 3.4 GHz CPU. To derive server parameters, experimental data from [7], [10], and [17] are referred. Table I lists the estimated 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 \([f_i \min, f_i \max]\), where \( f_i \min = 25\% f_i \max \).

- **Cluster2.** Second, we simulate a large cluster that has 128 back-end servers of eight different types. Due to the space limitation, this paper only reports simulation results on Cluster 1. Please refer to [19] for similar simulation results that have been obtained for the other case.

Workload Generation: A request is specified by a tuple \((A_j, E_j)\), where \( A_j \) is its arrival time and \( E_j \) is its execution time on a default server when it is operating at its maximum frequency.

- To generate requests for Workload1, we assume that the interarrival time follows a series of exponential distributions with a time-varied mean of \( 1/\lambda \) (\( \lambda \)). As shown in Fig. 2(a), we simulate a workload \( \lambda (t) \) that gradually increases from requiring 20% to 90% of the cluster capacity. Request execution time \( E_j \) is assumed to follow a gamma distribution with a specified mean of \( 1/\mu \), where \( \mu = 801 \) req/s is the default server’s maximum processing rate of this workload. The request execution time varies on different servers and is assumed to be reciprocally proportional to a server’s capacity. Assuming small requests, their desired average response time \( R_j^* \) is set at 1 s.

- Workload2 is generated according to empirical distributions based on a server log file [20]. From the log file, we extract request arrival time and requested file size information. The log file records all requests that arrived in a day. To expedite the simulation, we replay these requests faster than real-time, i.e., we have proportionally reduced request interarrival times. The average request execution time is

\[
\lambda = \sum_{i=1}^{k} c_i + \left( \frac{\lambda_{\text{cluster}} - \sum_{j=1}^{m} \alpha_j f_j \max + \frac{k}{R}}{\left( \sum_{j=m+1}^{k} \alpha_j \sqrt{\frac{\beta_j}{P_j^3}} \right)^2} \right)^3 + \sum_{i=1}^{m} \beta_i f_i^3 \max
\]
assumed to be $1/\mu_2$, where $\mu_2 = 541$ req/s is the default server’s maximum processing rate of this workload. In addition, we assume request execution time $E_i$ grows linearly with requested file size. To simulate same application accesses, we choose requests with modest execution time variances, i.e., those 95% requests that access files of the majority types (e.g., html, jpg, gif, javascript, and flash files). Fig. 2(b) shows the generated request rate $\lambda_{\text{cluster}}(t)$.

By offline analysis of cluster server parameters, a threshold-based algorithm derives the ordered server list, server activation thresholds and workload distribution formulas. Once these three modules are deployed on the head node, the cluster is able to handle different levels of workload. Each simulation lasts 3000 s and periodically, i.e., every 30 s, the system measures the current workload and predicts the average request rate $\lambda_{\text{cluster}}(t)$ for the next period. We adopt a method proposed in [11] for the workload prediction. Based on the range the predicted $\lambda_{\text{cluster}}(t)$ falls into, the corresponding PM 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 $f_i$. Since a CPU only supports discrete frequencies, we approximate the desired continuous frequency $f_i$ by switching the CPU frequency between two adjacent discrete values, e.g., to approximate 2.65 GHz frequency, during the 30-s sampling period, the CPU frequency is first set at 2.4 GHz for 11.25 s and then at 2.8 GHz for 18.75 s. For OPT-SOLN algorithm, to make its solution closer to the optimal, it is assumed to know the true $\lambda_{\text{cluster}}(t)$ accurately. To evaluate algorithm performance, we measure two metrics: average response time and power consumption. Curves are used to show the average response time, while for clarity, we use bar figures to illustrate the power consumption.

The first group of simulations (Sections VI-A–VI-C) simulate Cluster1 with the synthetic Workload1. We evaluate the effects of major design choices and the corresponding algorithms in Sections VI-A and VI-B. Section VI-C compares threshold-based algorithms with an existing approach [17] and with the optimal solution. In Sections VI-D and VI-E, we simulate Cluster1 with the empirical Workload2 and experimentally evaluate the feedback control mechanism’s impact on the back-end server DVS.

A. 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: Typical Power-based policy (TP) and baseline strategies: Activate All policy (AA), Static Power-based policy (SP) and Pseudo Dynamic Power-based policy (PDP) are compared. We evaluate the following algorithms: TP-CA-CA, AA-AA-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. Figs. 3 and 4 show the simulation results.

Since algorithms adopt capacity-based (CA) strategies for deciding server activation thresholds and workload distribution, we can see from Fig. 3 they all achieve the response time goal and keep the average response time around 1 s. One interesting observation is that the Activate All 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

![Fig. 2. Average request rate. (a) Workload1. (b) Workload2.](image)
the minimum levels that satisfy the time requirement. Therefore, as long as the desired frequency levels are equal or above the CPU’s minimum frequency $f_{\text{min}}$, even though AA policy turns on all back-end servers, it does not lead to reduced response times. The simulation results also demonstrate that our approach in approximating a continuous frequency by switching CPU between its two adjacent supported discrete frequencies works as expected.

In Fig. 4, we use a table to list the average power consumption and its standard deviation achieved by different algorithms over the 100 sampling periods and bars to show the sampled power consumptions in different sampling periods as cluster workload changes. Algorithm TP-CA-CA, built on our Typical Power-based policy (TP), always consumes the least power. It performs especially well at a low/medium cluster request rate when a good PM 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 a big impact on the power efficiency. When adopting a bad order, such as that by the Pseudo Dynamic Power-based policy (PDP), a high level of power is consumed. Occasionally, i.e., when 2747 req/s, the Pseudo Dynamic Power-based policy (PDP-CA-CA) performs even worse than the Activate All policy (AA-AA-CA). It shows that under such scenarios activating more servers consumes less power.

### B. Effects of Activation Thresholds and Workload Distribution

In this subsection, to evaluate policies 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 RANdomly generated ordered server list is used.

Figs. 5 and 6 show the simulation results. From Fig. 5, we can see that algorithm RAN-CA-RAN fails to provide response time guarantee: in multiple sampling periods, the average response time significantly exceeds the 1 s target. The reason is for a heterogeneous cluster, this RANdom (uniform) workload distribution does not prevent a server from being overloaded. Even though the CApacity-based server activation policy has ensured that the cluster capacity is adequate to handle the workload, the bad workload distribution still causes the big violation of response time requirement. Since all other algorithms consider a server’s capacity for workload distribution, they meet the time requirement.

Fig. 6 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 V-C, more servers do not always consume more power. Our Capacity-Power-based strategy (CP) takes this factor into account. For example, when $\lambda_{\text{cluster}}(t) = 929$ req/s, the baseline CPacity-only-based algorithms activate one server and when $\lambda_{\text{cluster}}(t) = 2747$ req/s, they activate three servers. In contrast, our algorithm RAN-CP-OP turns on two and four servers,

---

### Table 1: Power Consumption

<table>
<thead>
<tr>
<th>Alg</th>
<th>Power (Watt)</th>
<th>Avg</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP-CA-CA</td>
<td>264</td>
<td>0.33</td>
<td></td>
</tr>
<tr>
<td>AA-AA-CA</td>
<td>319</td>
<td>0.11</td>
<td></td>
</tr>
<tr>
<td>SP-CA-CA</td>
<td>269</td>
<td>0.43</td>
<td></td>
</tr>
<tr>
<td>PDP-CA-CA</td>
<td>306</td>
<td>0.39</td>
<td></td>
</tr>
</tbody>
</table>

---

**Fig. 4.** Effects of ordered server list: Power.

**Fig. 5.** Effects of activation thresholds and workload distribution: Time.
respectively, under these two scenarios. It leads to much less power consumptions. When \(\lambda_{\text{cluster}}(t)\) increases to 2800 req/s, 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/s) is less than that of three servers for a lighter workload (say 2747 req/s). **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 Fig. 6, When \(\lambda_{\text{cluster}}(t)\) is greater than 2800 req/s, four algorithms all activate the same number of servers. However, 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 One-by-One Saturation 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 consumptions do not decrease. As we observe, this strategy leads to the highest power consumptions.

C. Evaluation of Integrated Algorithms

This subsection evaluates the following integrated algorithms: AA-AA-CA and SP-CA-CA. When choosing baseline algorithms for comparison, we exclude the "deficient" algorithms, i.e., those based on PDP server activation strategy, RAN or OOS workload distribution policies. In addition, we compare these threshold-based algorithms with the optimal PM solution: OPT-SOLN. To obtain the optimal solution, we solve the PM problem, i.e., (7) and (8), for all integer points \(\lambda_{\text{cluster}}\) in the range \((0, \hat{\lambda}_{\text{cluster}}]\). The optimal server on/off \((x_i)\) and workload distribution \(\lambda_p\) is recorded for every possible \(\lambda_{\text{cluster}}\). Dynamically, based on the true \(\lambda_{\text{cluster}}(t)\), the corresponding optimal configuration is followed. We also implement an existing algorithm proposed by Rusu et al. [17]. For that algorithm, since the authors simply assume servers can be easily ordered with respect to their power efficiencies, they only provide a very short discussion on the server activation order. Therefore, to compare that algorithm with our TP-CP-OP algorithm, we focus on the other two algorithmic decisions on: server activation thresholds and workload distribution, while adopting the same TP ordered server list for both algorithms. We denote that algorithm as EE-RT-HSC, which is the acronym of the paper’s title [17].

Figs. 7 and 8, respectively, show the average response time and the power consumption. As expected, our algorithm TP-CP-OP performs better or as good as baseline algorithms under all scenarios. On average, TP-CP-OP consumes 28.1% and 8% less power than AA-AA-CA and SP-CA-CA, respectively. Surprisingly, compared to the results of OPT-SOLN, our heuristic TP-CP-OP leads to an average of 2% less power consumption. For the simulated workload, OPT-SOLN algorithm switches on/off back-end servers for a total of 11 times, while our algorithm TP-CP-OP only turns on three additional servers at their individual appropriate moments following the ordered server list. During some sampling periods, OPT-SOLN algorithm may cause two server on/off switches, for instance, turning off a low-capacity server while turning on a high-capacity server at the same sampling period. In Fig. 8, we observe when \(\lambda_{\text{cluster}}(t) = 2457\) req/s, OPT-SOLN algorithm leads to a quite high-power consumption, which is caused by two server
switches in that sampling period. Following the threshold-based approach, our algorithm minimizes the server on/off overhead, resulting in a smaller power consumption. In comparison with EE-RT-HSC algorithm, on average, our TP-CP-OP algorithm consumes 3.2% less power. By analyzing the experimental data, we notice that these two algorithms switch on/off back-end servers for the same number times. However, two methods adopted by EE-RT-HSC lower the algorithm’s power efficiency. First, to avoid overloading the cluster so that a tighter quality-of-service (QoS) (i.e., 95% deadline met) can be achieved, the algorithm activates new servers in advance based on the possible maximum load increase during a monitoring period (which is set at 5 s). This strategy leads to servers being turned on earlier, resulting in higher power consumptions. Second, in order to void frequent server switches, EE-RT-HSC algorithm adds several transition states so that servers are not turned on/off immediately to improve power efficiency. This method, however, does not necessarily save power and in many cases, on the contrary, it leads to more power consumptions.

When implementing EE-RT-HSC, we notice that it is difficult to apply the algorithm to save power for web clusters that have fluctuating workloads. Web traffic is known to be self-similar [23] and thus has significant variability over a wide range of time scales. This huge variance makes it hard for the algorithm to estimate the possible maximum load increase per monitoring period. For instance, if we apply it to handle the empirical workload (i.e., Workload2 that we have generated based on a web log file), due to the large value of the max_load_increase, EE-RT-HSC algorithm will almost turn on all servers at the beginning of the simulation. As a result, it will consume a quite high level of power. Because of this limitation, we choose not to simulate EE-RT-HSC in the next two subsections where the cluster executes the empirical Workload2.

D. Empirical Workload Simulation

To evaluate algorithms in a more realistic setting, for the second group of simulations (Sections VI-D and VI-E), we simulate the cluster with the empirical Workload2, which is generated based on a web log file [20]. Figs. 9 and 10 show the simulation results for the same integrated algorithms that have been analyzed in the previous subsection. From Fig. 9, we can see, although the workload does not match the assumed M/M/1 queueing model, TP-CP-OP and SP-CA-CA algorithms can still satisfy the response time requirement due to the effective feedback control of DVS. From 30 to 60 sampling periods, OPT-SOLN turns on/off servers for a total of 28 server on/off switches, while our heuristic

<table>
<thead>
<tr>
<th>Alg</th>
<th>Power (Watt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP-CP-OP</td>
<td>249.03</td>
</tr>
<tr>
<td>AA-AA-CA</td>
<td>319.04</td>
</tr>
<tr>
<td>SP-CA-CA</td>
<td>269.03</td>
</tr>
<tr>
<td>EE-RT-HSC</td>
<td>257.03</td>
</tr>
<tr>
<td>OPT-SOLN</td>
<td>254.03</td>
</tr>
</tbody>
</table>

Fig. 8. Integrated algorithms: Power.

Fig. 9. Simulation result of web log-based workload: Time.

Authorized licensed use limited to: University of Nebraska - Lincoln. Downloaded on August 14,2010 at 19:47:36 UTC from IEEE Xplore. Restrictions apply.
TP-CP-OP only causes 12 switches. There are nine sampling periods when OPT-SOLN algorithm causes two server switches and ten sampling periods when it has one switch. During the period when $\lambda_{\text{chaser}}(t) = 524 \text{ req/s}$, our algorithm TP-CP-OP does not cause any server switch, while OPT-SOLN leads to one switch. That is why during that period OPT-SOLN algorithm consumes more power than TP-CP-OP.

E. Effects of Feedback Control

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

Fig. 11(a) shows the resultant average response times when the feedback control is not applied. As we can see, due to the modeling inaccuracy, the response times are no longer around 1 s. In contrast, when the feedback control is combined with the queueing-theoretic prediction, the average response times, as shown in Fig. 9, are kept close to the target. These results demonstrate that the feedback control mechanism is effective in regulating the response time. On the other hand, when comparing power consumptions of DVS mechanisms with and without feedback control, the differences are negligible. For illustration, the curves in Fig. 11(b) show the power consumption of TP-CP-OP algorithm with and without DVS feedback control. On average, the difference in power consumption is only 1.82 Watt and server frequencies differ by 0.0294 GHz. As we can see, the response time is very sensitive to frequency changes. A small frequency change can lead to a large variation of response time. Consequently, to effectively regulate the response time, the feedback control mechanism only needs to slightly modify the queueing estimated frequency $f_i$ and thus leads to a very small difference in power consumption. These experimental results show that the queueing model-based estimate of $f_i$ is very close to the real frequency setting, which justifies the adoption of queueing estimated $f_i$ in the optimization problem formulation (see Section IV).

VII. CONCLUSION

This paper presents a threshold-based method for efficient PM of heterogeneous soft real-time clusters. Following this approach, a PM algorithm makes 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 demonstrate the algorithm’s advantages in power consumption: it incurs low overhead and leads to optimal power consumption.

REFERENCES

Leping Wang received the B.S. and M.S. degrees from Nankai University, Tianjin, China, in 2002 and 2006, respectively. He is currently working towards the Ph.D. degree at the Department of Computer Science and Engineering, University of Nebraska-Lincoln. His research interests include grid computing, real-time scheduling, and power management in cluster computing system.

Ying Lu received the B.S. degree in computer science in 1996 from the Southwest Jiaotong University, Chengdu, China, and the M.S. and Ph.D. degrees in computer science in 2001 and 2005, respectively, from the University of Virginia, Charlottesville. Since August 2005, she has been at the University of Nebraska-Lincoln, where she is an Assistant Professor in the Department of Computer Science and Engineering. Her research interests include real-time systems, autonomic computing, cluster and grid computing, and web systems. She has done significant work on feedback control of computing systems.