Summer 6-13-2014

POWER MANAGEMENT IN THE CLUSTER SYSTEM

Leping Wang
University of Nebraska-Lincoln, leping@huskers.unl.edu

POWER MANAGEMENT IN THE CLUSTER SYSTEM

by

Leping Wang

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Ying Lu

Lincoln, Nebraska

June, 2014

POWER MANAGEMENT IN THE CLUSTER SYSTEM

Leping Wang, M.S.

University of Nebraska, 2014

Adviser: Ying Lu

With growing cost of electricity, the power management of server clusters has become an important problem. However, most previous researchers have only addressed the challenge in traditional homogeneous environments. Considering the increasing popularity of heterogeneous and virtualized systems, this thesis develops a series of efficient algorithms respectively for power management of heterogeneous soft real-time clusters and a virtualized cluster system. 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. Built upon optimization, queuing theory and control theory techniques, our approach achieves the design goal, where QoS is provided to a larger number of requests with a smaller amount of power consumption. To strive for efficiency, a threshold based approach is adopted in the first part of the thesis. Then we systematically study this approach and its design decisions. To deploy our mechanisms on the virtualized clusters, we extend the work by developing a novel power-efficient workload distribution algorithm.

ACKNOWLEDGMENTS

It would not have been possible to write this master thesis without the help and support of the kind people around me, to only some of whom it is possible to give particular mention here.

Above all, I would like to express the deepest appreciation to my advisor Prof. Ying Lu for the continuous support of my graduate study and research, for his patience, motivation, enthusiasm, and immense knowledge. Her guidance helped me in all the time of research and writing of this thesis.

I would like to thank my committee members, Prof. Hong Jiang, Prof. Steve Goddard, Prof. David Swanson and Prof. Wei Qiao for serving as my committee members even at hardship. I also want to thank you for letting my defense be an enjoyable moment, and for your brilliant comments and suggestions, thanks to you.

Last but not the least, a special thanks to my family. Words cannot express how grateful I am to my parents, my parents-in-law and my wife for all of the sacrifices that you have made on me. Your great love for me was what sustained me thus far.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With growing demands for high performance computing, more and more giant data centers are being built. When designing such a system, traditionally researchers have focused on maximizing performance. Recently, with a better understanding of the overall cost of computing [2], researchers have started to pay more attention to optimizing performance per unit of cost. According to [2], 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. According to the U.S. Environmental Protection Agency (EPA) 2007 Report to Congress, data centers in the United States incur a total energy cost of approximately $4.5 billion in 2006 and in the absence of intervention, this figure is expected to double in year 2012. 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. As the rising cost of energy makes it the single most important factor in data center operating costs, energy minimization becomes one of the critical challenges in data centers.

To avoid the projected energy expenditure and hence improve the performance per watt, cluster power management mechanisms [8, 10, 19, 23, 31, 38, 41, 42, 4] have been proposed. Most of them, however, are only applicable to homogenous systems. Clusters are almost invariably heterogeneous in terms of performance, capacity, and energy consumption of their hardware components [23]. The heterogeneity also comes from servers with different types of services [29]. Therefore, we must explicitly consider cluster heterogeneity when developing power management mechanisms, in which two new challenges must 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 [23, 42] have investigated mechanisms to address the aforementioned challenges. However, their mechanisms all require extensive performance measurements ("at most few hours for each machine" [42]) 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. The first work in my thesis 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 work, 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 consumption.

Beside the platform heterogeneity, another trend of building an up-to-date server cluster is that virtualization technology is more commonly employed [33], where multiple services called virtual machines (i.e., VMs) are consolidated and placed together on a physical machine, leading to a smaller-sized and more energy-efficient cluster. In a virtualized cluster, besides heterogeneous hardware, we also face workload heterogeneity, where different servers often host different VMs, serving different workloads.

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

Chapter 2

Related Work

Power management of server clusters [8, 10, 19, 31, 38, 41] has become an important problem. The authors of [6, 44] 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 [19, 31] 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 [23]. Survey [5] discusses the recent work on power management for server systems. It lists power management of heterogeneous clusters as one of the major challenges.

The research most closely related to the first part of the thesis is that of [23, 42]. The authors of [23] 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 [42], 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.

Existing studies have another common limitation. They often do not consider virtualized environments. As the virtualization technology is being widely adopted, the power management of virtualized clusters becomes an important problem. Wang et al. [50] focus their investigation on how to control CPU frequency and share it among different VMs to satisfy power and performance constraints. Nathuji et al. [35] study power-budgeting methods to ensure that the total power consumption of a virtualized cluster does not exceed the specified budget. In papers [29, 26], the resource provisioning problem is investigated, where methods are proposed to match VMs workload demand with the cluster capacity. These methods could be leveraged to guide the power-efficient server consolidation. Previous research [17] has also studied how to properly place VMs on a cluster's physical servers to improve energy efficiency. These works' contributions are complementary to ours.

The most closely related research with the second part of our work is by Mukherjee et al. [34], where they investigate thermal-aware job scheduling in virtualized heterogeneous data centers. Their assumed workload and system models are, however, different from ours. For instance, they do not assume workload heterogeneity, where a request can only be served by some but not all physical servers.

Chapter 3

Efficient Power Management of Heterogeneous Soft Real-Time Clusters

In this chapter we introduce the first part of our Work, *Efficient Power Management of Heterogeneous Soft Real-Time Clusters*.

As mentioned in Chapter 2, the authors of [23, 42] studied a similar problem in heterogeneous systems. 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 server failure, cluster upgrades or changes. Extensive performance measurements [42] and a long optimization process [23, 42] lead to high customization costs. To avoid these prohibitive costs, we propose in this work a simple power management algorithm for heterogeneous 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 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.

3.1 Models

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

3.1.1 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 3.1 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 provide their models. We assume processors on the back-end servers support dynamic voltage scaling and their operating frequencies could be continuously adjusted in $(0, f_{i\_max}]$ 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

[1] In our report [47], we also evaluate the algorithm's performance on servers with only discrete frequency settings.

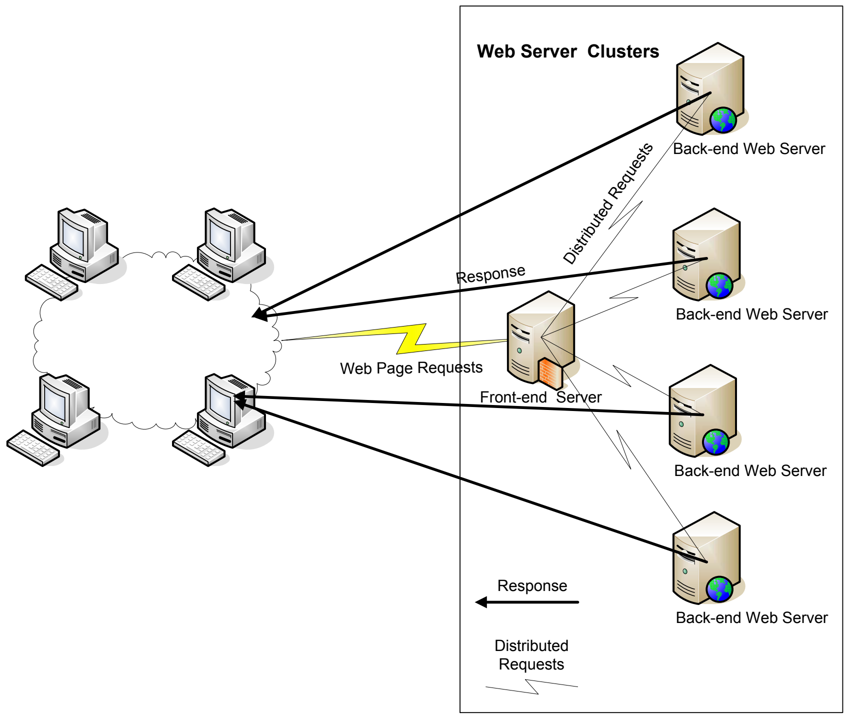


Figure 3.1: System Model

frequency and the power consumption. While our approach is general and could be applicable to different capacity and power models, in this paper we assume and use the following specific models to illustrate our method.

3.1.2 Capacity Model

We assume that the cluster provides CPU-bounded services, as typical web servers do today [6]. 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 μ_i, is expressed as $\mu_i = \alpha_i f_i$, where α_i is the CPU performance coefficient. Different servers may have different values for α_i. With the same CPU frequency setting, the higher the α_i the more powerful the server is.

3.1.3 Power Model

The power consumption P_i of a server consists of a constant part and a variable part. Similar to previous work [19, 10, 24], we approximate P_i by the following function:

$$P_i = x_i(c_i + \beta_i f_i^p) \tag{3.1}$$

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

$$x_i = \begin{cases} 0 & \text{the } i^{th} \text{ server is off} \\ 1 & \text{the } i^{th} \text{ server is on} \end{cases} \tag{3.2}$$

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:

$$J = \sum_{i=1}^{N} x_i[c_i + \beta_i f_i^3] \tag{3.3}$$

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}, α_i, c_i, and β_i. To obtain these parameters, only a little performance profiling is required.

3.2 Power Management Problem

Given a cluster of N heterogeneous back-end servers, each specified with parameters f_{i_max}, α_i, c_i, and β_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 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 λ_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 λ_i properly. As a result, the power management problem is formed as follows:

minimize

$$J = \sum_{i=1}^{N} x_i[c_i + \beta_i f_i^3] \tag{3.4}$$

subject to:

$$\begin{cases} \sum_{i=1}^{N} x_i \lambda_i = \lambda_{cluster} \\ x_i(1 - x_i) = 0, & i = 1, 2, \cdots, N \\ g(\alpha_i f_i, \lambda_i) \approx \hat{R}, & i = 1, 2, \cdots, N \end{cases} \tag{3.5}$$

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

[2] An admission control mechanism could be applied to enforce this constraint.

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 (λ_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 [43], is adopted.

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

$$R_i = \frac{1}{\mu_i - \lambda_i} = \frac{1}{\alpha_i f_i - \lambda_i} \tag{3.6}$$

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

$$f_i = \frac{\lambda_i}{\alpha_i} + \frac{1}{\alpha_i \hat{R}} \tag{3.7}$$

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, experimental data shows that the queuing model estimate (Equation (3.7)) 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

$$J = \sum_{i=1}^{N} x_i [c_i + \beta_i \times (\frac{\lambda_i}{\alpha_i} + \frac{1}{\alpha_i \hat{R}})^3] \tag{3.8}$$

subject to:

$$\begin{cases} \sum_{i=1}^{N} x_i\lambda_i = \lambda_{cluster} \\ x_i(1-x_i) = 0, & i = 1,2,...,N \\ 0 \le \lambda_i \le \alpha_i f_{i\_max} - \frac{1}{\hat{R}}, & i = 1,2,...,N \end{cases} \tag{3.9}$$

As shown above, the optimal solution is determined by two variables: individual server's on/off state x_i and workload distribution λ_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.

3.3 Algorithm

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 $(0, \Lambda_1], (\Lambda_1, \Lambda_2], \cdots, (\Lambda_k, \Lambda_{k+1}]$, $\cdots, (\Lambda_{N-1}, \hat{\lambda}_{cluster}]$ 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 $\lambda_{cluster}$, decides its range, and follows the corresponding power management decisions. Next, we present the details of our algorithm.

3.3.1 Optimization Heuristic Framework

In Section 3.2, the power management is formed as an optimization problem (Equation (3.8) and (3.9)). Instead of solving it for all possible workload $\lambda_{cluster}$ in the (0, $\hat{\lambda}_{cluster}$] range, we propose the following 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.
- 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 (Equation (3.8) and (3.9)) becomes:

  minimize

  $$J_k = \sum_{i=1}^{k} [c_i + \beta_i \times (\frac{\lambda_i}{\alpha_i} + \frac{1}{\alpha_i \hat{R}})^3] \tag{3.10}$$

  subject to:

$$\begin{cases} \sum_{i=1}^{k} \lambda_i = \lambda_{cluster} \\ 0 \le \lambda_i \le \alpha_i f_{i\_max} - \frac{1}{\hat{R}}, \quad i = 1, 2, ..., k \end{cases} \tag{3.11}$$

The analysis is simplified to solving the above optimization problem for $k = 1, 2, \cdots, N$. In contrast, to obtain the optimal power management solution (i.e., solving Equations (3.8) and (3.9) by an exhaustive search of all possible server on/off scenarios) for every integer point in the range (0, $\hat{\lambda}_{cluster}$] requires solving $\lceil \hat{\lambda}_{cluster} \rceil 2^N$ number of problem instances in the form of Equations (3.10) and (3.11).

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.

3.3.2 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 3.1 & 3.2), 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 (3.8)). 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 λ is small, $P_i(\lambda)$ is less than $P_j(\lambda)$ and server i has a better power efficiency; while as λ 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 λ'_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 3.3.3). 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 λ'_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 α_i, c_i, and β_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 β_i. A server with a smaller β_i is listed earlier in the *ordered server list*. According to the definition of power efficiency $\frac{P_i(\lambda_i)}{\lambda_i}$, its dynamic part is $\frac{\frac{\beta_i}{\alpha_i^3}\times(\lambda_i+\frac{1}{\hat{R}})^3}{\lambda_i}$. As we can see, the dynamic power efficiency is not solely determined by β_i. This policy is therefore called *pseudo* dynamic power based policy.

3.3.3 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.

Following our mechanism, 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 Λ_{ck} corresponds to the maximum workload that can be processed by the first k servers. According to Equation (3.6), when a server is operating at its maximum frequency f_{i_max}, it can process at most λ_{i_max} amount of workload and meet the response time requirement:

$$\lambda_{i\_max} = \alpha_i f_{i\_max} - \frac{1}{\hat{R}} \tag{3.12}$$

Thus, we have:

$$\Lambda_{ck} = \sum_{i=1}^{k} \lambda_{i\_max} = \sum_{i=1}^{k} \alpha_i f_{i\_max} - \frac{k}{\hat{R}} \tag{3.13}$$

When the current workload exceeds this threshold Λ_{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 3.3.4 for $\hat{J}_k(\lambda_{cluster})$'s derivation). It is a monotonically increasing function of $\lambda_{cluster}$. We analyze the following equation:

$$\hat{J}_k(\lambda_{cluster}) = \hat{J}_{k+1}(\lambda_{cluster}) \tag{3.14}$$

According to characteristics of functions $\hat{J}_k(\lambda_{cluster})$ and $\hat{J}_{k+1}(\lambda_{cluster})$ (see Section 3.3.4), there is at most one solution for Equation (3.14). 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 (3.14), 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 (3.14) 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 Capacity-Power-based strategy. For comparison, a baseline CApacity-only strategy, denoted as CA, is also investigated, for which $\Lambda_k = \Lambda_{ck}$. In the Activate All policy (AA), no server activation thresholds are needed.

3.3.4 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 3.3.1, if the first k servers of the *ordered server list* are activated, the optimization problem becomes:

minimize

$$J_k = \sum_{i=1}^{k} [c_i + \beta_i \times (\frac{\lambda_i}{\alpha_i} + \frac{1}{\alpha_i \hat{R}})^3] \tag{3.15}$$

subject to:

$$\begin{cases} \sum_{i=1}^{k} \lambda_i = \lambda_{cluster} \\ 0 \leq \lambda_i \leq \alpha_i f_{i\_max} - \frac{1}{\hat{R}}, \quad i = 1, 2, ..., k \end{cases} \tag{3.16}$$

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

$$J_k = \sum_{i=1}^{k} [c_i + \beta_i \times (\frac{\lambda_i}{\alpha_i} + \frac{1}{\alpha_i \hat{R}})^3] \tag{3.17}$$

subject to:

$$\sum_{i=1}^{k} \lambda_i = \lambda_{cluster} \tag{3.18}$$

According to *Lagrange's Theorem* [14], the first-order necessary condition for J_k's optimal solution is:

$$\begin{aligned} \exists \delta,\ J_k(\lambda_i, \delta) = \sum_{i=1}^{k} [c_i + \beta_i \times (\tfrac{\lambda_i}{\alpha_i} + \tfrac{1}{\alpha_i \hat{R}})^3] \\ + \delta(\sum_{i=1}^{k} \lambda_i - \lambda_{cluster}) \end{aligned} \tag{3.19}$$

and its first-order derivatives satisfy

$$\begin{cases} \frac{\partial J_k(\lambda_i, \delta)}{\partial \lambda_i} = 0, \quad i = 1, ...k \\ \frac{\partial J_k(\lambda_i, \delta)}{\partial \delta} = 0 \end{cases} \tag{3.20}$$

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

$$\lambda_i = \frac{\alpha_i(\lambda_{cluster} + \frac{k}{\hat{R}})}{\sum_{j=1}^{k} \alpha_j \sqrt{\frac{\alpha_j}{\beta_j}}} \sqrt{\frac{\alpha_i}{\beta_i}} - \frac{1}{\hat{R}} \tag{3.21}$$

The corresponding power consumption is:

$$\hat{J}_k = \sum_{i=1}^{k} c_i + \frac{(\lambda_{cluster} + \frac{k}{\hat{R}})^3}{(\sum_{j=1}^{k} \alpha_j \sqrt{\frac{\alpha_j}{\beta_j}})^2} \tag{3.22}$$

The above solution is optimal when all k back-end servers are running below their maximum capacities. That is, when λ_i (Equation (3.21)) 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 cluster workloads. As $\lambda_{cluster}$ increases, servers start to be saturated one after another. That is, a server's shared workload λ_i reaches its maximum level $\alpha_i f_{i\_max} - \frac{1}{\hat{R}}$ where we have:

$$\begin{aligned} \lambda_i &= \frac{\alpha_i(\lambda_{cluster} + \frac{k}{\hat{R}})}{\sum_{j=1}^{k} \alpha_j \sqrt{\frac{\alpha_j}{\beta_j}}} \sqrt{\frac{\alpha_i}{\beta_i}} - \frac{1}{\hat{R}} \\ &= \alpha_i f_{i\_max} - \frac{1}{\hat{R}} \end{aligned} \tag{3.23}$$

Solving Equation (3.23) for system workload $\lambda_{cluster}$, we get:

$$\lambda_{cluster} = f_{i\_max} \sqrt{\frac{\beta_i}{\alpha_i}} \sum_{j=1}^{k} \alpha_j \sqrt{\frac{\alpha_j}{\beta_j}} - \frac{k}{\hat{R}} \tag{3.24}$$

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:

$$\lambda_{cluster} = f_{1\_max}\sqrt{\frac{\beta_1}{\alpha_1}}\sum_{j=1}^{k}\alpha_j\sqrt{\frac{\alpha_j}{\beta_j}} - \frac{k}{\hat{R}} \tag{3.25}$$

The workload distribution problem becomes:

minimize

$$\begin{aligned} J_k &= \sum_{i=2}^{k}[c_i + \beta_i \times (\frac{\lambda_i}{\alpha_i} + \frac{1}{\alpha_i \hat{R}})^3] \\ &+ (c_1 + \beta_1 f_{1\_max}^3) \end{aligned} \tag{3.26}$$

subject to:

$$\sum_{i=2}^{k}\lambda_i = \lambda_{cluster} - (\alpha_1 f_{1\_max} - \frac{1}{\hat{R}}) \tag{3.27}$$

Here, servers are indexed following their *saturated order list*. Similar to Equations (3.17) and (3.18), we solve the above problem by applying *Larange's Theorem* and get the following optimal solution for $\lambda_i, i = 2, 3, \cdots, k$:

$$\lambda_i = \frac{\alpha_i(\lambda_{cluster} - \alpha_1 f_{1\_max} + \frac{k}{\hat{R}})}{\sum_{j=2}^{k}\alpha_j\sqrt{\frac{\alpha_j}{\beta_j}}}\sqrt{\frac{\alpha_i}{\beta_i}} - \frac{1}{\hat{R}} \tag{3.28}$$

The corresponding power consumption is:

$$\hat{J}_k = \sum_{i=1}^{k}c_i + \frac{(\lambda_{cluster} - \alpha_1 f_{1\_max} + \frac{k}{\hat{R}})^3}{(\sum_{j=2}^{k}\alpha_j\sqrt{\frac{\alpha_j}{\beta_j}})^2} + \beta_1 f_{1\_max}^3 \tag{3.29}$$

Again, we let λ_i (Equation (3.28)) be equal to the maximum workload $\alpha_i f_{i\_max} - \frac{1}{\hat{R}}$

and solve for $\lambda_{cluster}$. We get:

$$\lambda_{cluster} = f_{i\_max}\sqrt{\frac{\beta_i}{\alpha_i}}\sum_{j=2}^{k}\alpha_j\sqrt{\frac{\alpha_j}{\beta_j}} + \alpha_1 f_{1\_max} - \frac{k}{\hat{R}} \tag{3.30}$$

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:

$$\lambda_{cluster} = f_{2\_max}\sqrt{\frac{\beta_2}{\alpha_2}}\sum_{j=2}^{k}\alpha_j\sqrt{\frac{\alpha_j}{\beta_j}} + \alpha_1 f_{1\_max} - \frac{k}{\hat{R}} \tag{3.31}$$

We define λ_k^m as:

$$\lambda_k^m = f_{m\_max}\sqrt{\frac{\beta_m}{\alpha_m}}\sum_{j=2}^{k}\alpha_j\sqrt{\frac{\alpha_j}{\beta_j}} + \sum_{i=1}^{m-1}\alpha_i f_{i\_max} - \frac{k}{\hat{R}} \tag{3.32}$$

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

$$\begin{aligned} J_k &= \sum_{i=m+1}^{k}[c_i + \beta_i \times (\frac{1}{\alpha_i \hat{R}} + \frac{\lambda_i}{\alpha_i})^3] \\ &+ \sum_{i=1}^{m}(c_i + \beta_i f_{i\_max}^3) \end{aligned} \tag{3.33}$$

subject to:

$$\sum_{i=m+1}^{k}\lambda_i = \lambda_{cluster} - \sum_{j=1}^{m}(\alpha_j f_{j\_max} - \frac{1}{\hat{R}}) \tag{3.34}$$

and the optimal solution is :

$$\lambda_i = \frac{\alpha_i(\lambda_{cluster} - \sum_{j=1}^{m} \alpha_j f_{j\_max} + \frac{k}{\hat{R}})}{\sum_{j=m+1}^{k} \alpha_j \sqrt{\frac{\alpha_j}{\beta_j}}} \sqrt{\frac{\alpha_i}{\beta_i}} - \frac{1}{\hat{R}} \quad \text{for } i = m+1, m+2, \cdots, k \tag{3.35}$$

$$\hat{J}_k = \sum_{i=1}^{k} c_i + \frac{(\lambda_{cluster} - \sum_{j=1}^{m} \alpha_j f_{j\_max} + \frac{k}{\hat{R}})^3}{(\sum_{j=m+1}^{k} \alpha_j \sqrt{\frac{\alpha_j}{\beta_j}})^2} + \sum_{i=1}^{m} \beta_i f_{i\_max}^3 \tag{3.36}$$

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 *t*ransient 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 *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). 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.

3.3.5 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 3.3.1), 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.

3.4 Performance Evaluation

In the previous section, we proposed various threshold-based strategies for the power management of heterogeneous soft real-time clusters. In this section, we

Server	f_{i_max}	c_i	β_i	α_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 3.1: Parameters of a 4-Server Cluster

experimentally compare their performance relative to each other, to an existing approach [42], 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 3.1. The server on/off switch overheads are also simulated. There are two types of switch overheads: time overhead and power overhead. It takes some time, assumed to be 10 seconds, 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., $P_i = c_i + \beta_i f_{i\_max}^3$.

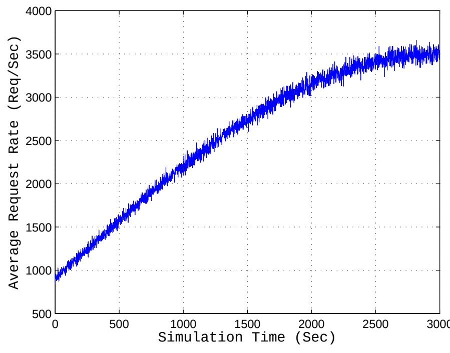
Cluster Configuration. The following clusters are simulated:

- Cluster1. 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 [42, 11, 21] are referred. Table 3.1 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 $f_{i\_min} = 25\% f_{i\_max}$.

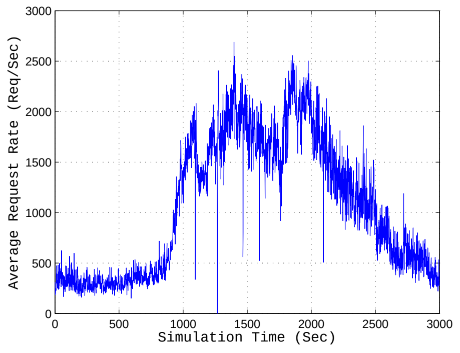
- **Cluster2**. Second, we simulate a large cluster that has 128 back-end servers of 8 different types.

In this thesis, we only report simulation results on Cluster1. Please refer to [47] for similar simulation results that have been obtained for the other case.

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.



(a) Workload1



(b) Workload2

Figure 3.2: Average Request Rate

- To generate requests for **Workload1**, 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 3.2a, 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 $\frac{1}{\mu_1'}$, where $\mu_1' = 891 req/sec$ 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 $\hat{R}$ is set at 1 second.

- **W**orkload2 is generated according to empirical distributions based on a server log file [1]. 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 assumed to be $\frac{1}{\mu'_2}$, where $\mu'_2 = 541 req/sec$ 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). Figure 3.2b shows the generated request rate $\lambda_{cluster}(t)$.

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 seconds and 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 [22] 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 (λ_i) are followed. According to λ_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 2.65GHz frequency, during the 30-second sampling period, the CPU frequency is first set at 2.4GHz for 11.25 seconds and then at 2.8GHz for 18.75 seconds. For OPT-SOLN algorithm, to make its solution closer to the optimal, it is assumed to know the true $\lambda_{cluster}(t)$ 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 3.4.1, 3.4.2 and 3.4.3) simulate Cluster1 with the synthetic Workload1. We evaluate the effects of major design choices and the corresponding algorithms in Sections 3.4.1 and 3.4.2. Section 3.4.3 compares threshold-based algorithms with an existing approach [42] and with the optimal solution. In Sections 3.4.4 and 3.4.5, we simulate Cluster1 with the empirical Workload2 and experimentally evaluate the feedback control mechanism's impact on the back-end server DVS.

3.4.1 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), *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, 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 3.3 and 3.4 show the simulation results.

Since algorithms adopt capacity based (CA) strategies for deciding *server activation thresholds* and *workload distribution*, we can see from Figure 3.3 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 levels that satisfy the time requirement. Therefore, as long as the desired frequency levels are equal or above the CPU's minimum frequency f_{i_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 Figure 3.4, we use a table to list the average power consumption 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 *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 activa-*

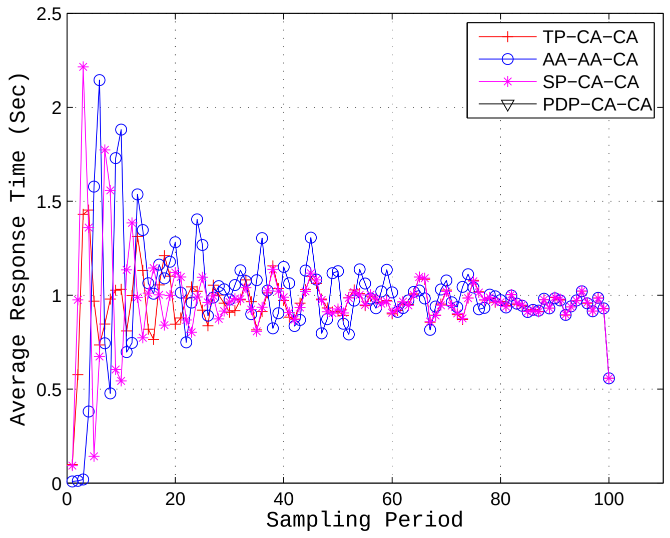


Figure 3.3: Effects of Ordered Server List: Time

tion order has a big impact on the power efficiency. When adopting a bad order, such as that by the *P*seudo *D*ynamic *P*ower based policy (PDP), a high level of power is consumed. Occasionally, i.e., when $\lambda_{cluster}(t)$ = 2457 or 2747 req/sec, 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 that under such scenarios activating more servers consumes less power.

Alg	Power (Watt)	
	Avg	Std
TP-CA-CA	264	0.33
AA-AA-CA	319	0.11
SP-CA-CA	269	0.43
PDP-CA-CA	306	0.39

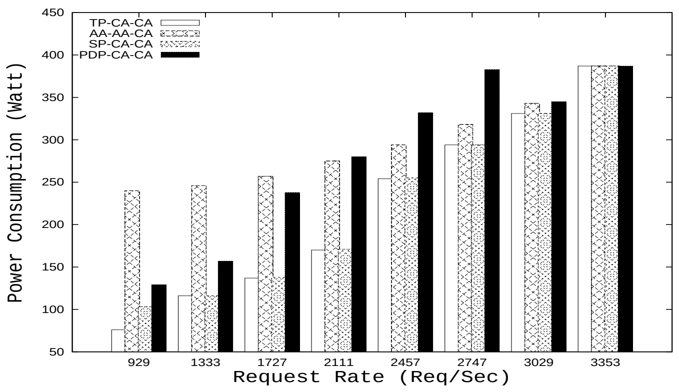


Figure 3.4: Effects of Ordered Server List: Power

3.4.2 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 3.5 and 3.6 show the simulation results. From Figure 3.5, 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 second target. The reason is for a heterogeneous cluster, this *RAN*dom (uniform) 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 3.6 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.

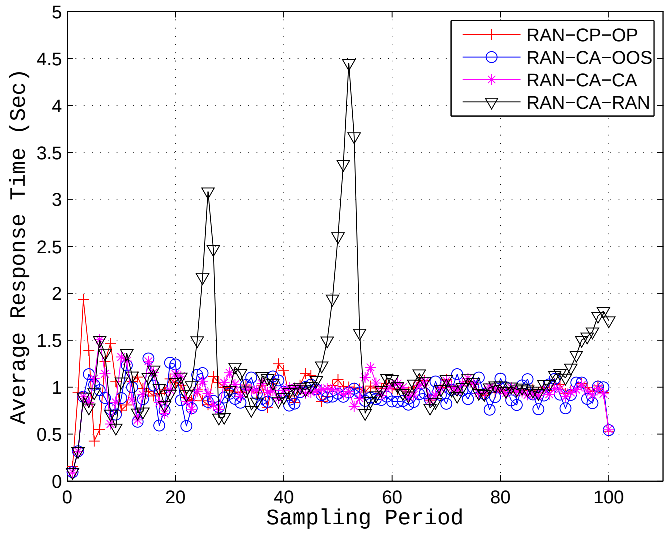


Figure 3.5: Effects of Activation Thresholds and Workload Distribution: Time

More Servers but Less Power. As discussed in Section 3.3.3, more servers do not always consume more power. Our Capacity-*P*ower-based strategy (CP) takes this factor into account. For example, when $\lambda_{cluster}(t) = 929$ 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 3.6, 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 consumptions do not decrease. As we observe, this strategy leads to the highest power consumptions.

Alg	Power (Watt)	
	Avg	Std
RAN-CP-OP	278	0.21
RAN-CA-OOS	364	0.85
RAN-CA-CA	309	0.47
RAN-CA-RAN	307	1.11

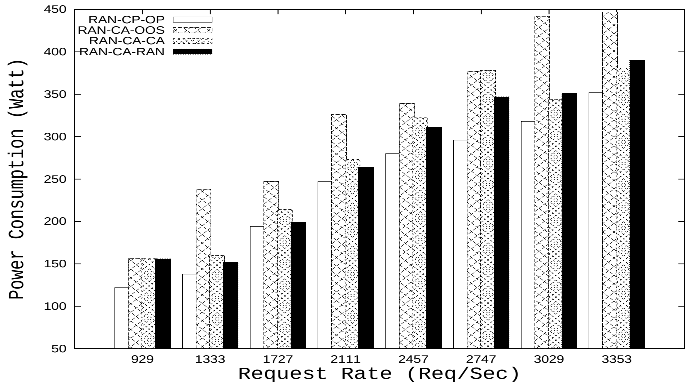


Figure 3.6: Effects of Activation Thresholds and Workload Distribution: Power

3.4.3 Evaluation of Integrated Algorithms

This subsection evaluates the following integrated algorithms: our heuristic TP-CP-OP and baseline 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 power management solution: OPT-SOLN. To obtain the optimal solution, we solve the power management problem, i.e., Equations (3.8) and (3.9), for all integer points $\lambda_{cluster}$ in the range $(0, \hat{\lambda}_{cluster}]$. The optimal server on/off (x_i) and workload distribution (λ_i) is recorded for every possible $\lambda_{cluster}$. Dynamically, based on the true $\lambda_{cluster}(t)$, the corresponding optimal configuration is followed. We also implement an existing algorithm proposed by Rusu et. al. [42]. 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.

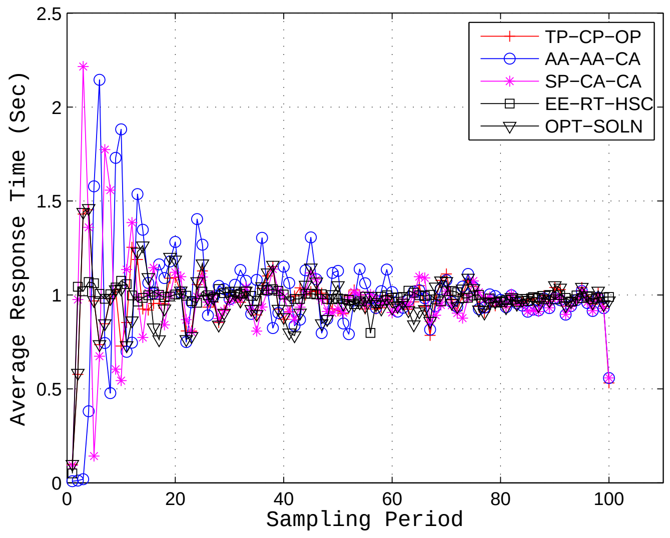


Figure 3.7: Integrated Algorithms: Time

We denote that algorithm as EE-RT-HSC, which is the acronym of the paper's title [42].

Alg	Power (Watt)	
	Avg	Std
TP-CP-OP	249	0.19
AA-AA-CA	319	0.11
SP-CA-CA	269	0.43
EE-RT-HSC	257	0.49
OPT-SOLN	254	1.38

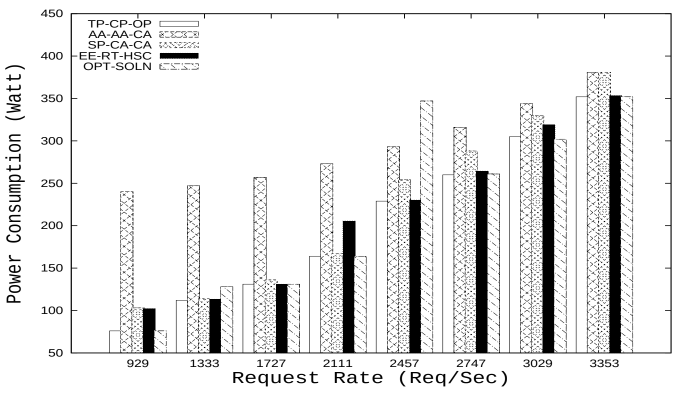


Figure 3.8: Integrated Algorithms: Power

Figures 3.7 and 3.8 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 3 additional servers at their individual appropriate moments following the *o*rdered 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 Figure 3.8 we observe when $\lambda_{cluster}(t) = 2457$ req/sec, 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 the same number of 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 QoS (i.e., 95% deadline met) can be achieved, the algorithm activates new servers in advance based on the possible *m*aximum load increase during a monitoring period (which is set at 5 seconds). This strategy leads to servers being turned on too early, resulting in more 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-HRS, 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 [16] and thus has significant variability over a wide range of time scales. This huge variance makes it hard for the algorithm to estimate the maximum possible 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 *m*ax_load_increase, EE-RT-HRS 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 deficiency, we choose not to simulate EE-RT-HRS in the next two subsections where the cluster executes the empirical Workload2.

3.4.4 Empirical Workload Simulation

To evaluate algorithms in a more realistic setting, for the second group of simulations (Sections 3.4.4 and 3.4.5), we simulate the cluster with the empirical Workload2, which is generated based on a web log file [1]. Figures 3.9 and 3.10 show the simulation results for the same integrated algorithms that have been analyzed in the previous subsection. From Figure 3.9, we can see, although the workload does not match the assumed M/M/1 queuing model, TP-CP-OP and SP-CA-CA algorithms can still satisfy the response time requirement due to the effective feedback control of DVS. OPT-SOLN violates the time requirement during [30, 60] sampling periods because of high server on/off overheads. In these 30 periods, OPT-SOLN turns on/off servers for a total of 15 times. AA-AA-CA algorithm produces very short response times at the beginning and the end of the simulation. During those periods, the workload is very low, where $\lambda_{cluster}(t) \approx$ 500 req/sec. However, AA-AA-CA algorithm always turns on all servers. With such low request rates, even when all processors are set at their minimum frequencies f_{i_min}, the total cluster capacity is still bigger than needed to satisfy the 1 second response time requirement. That explains why we observe lower response times at the beginning and the end of the simulation.

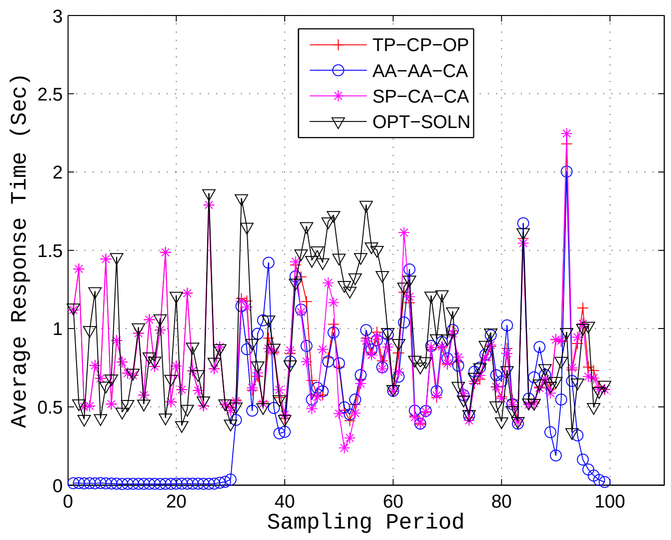


Figure 3.9: Simulation Result of Web Log Based Workload: Time

Alg	AvgPower (Watt)
TP-CP-OP	174
AA-AA-CA	283
SP-CA-CA	186
OPT-SOLN	181

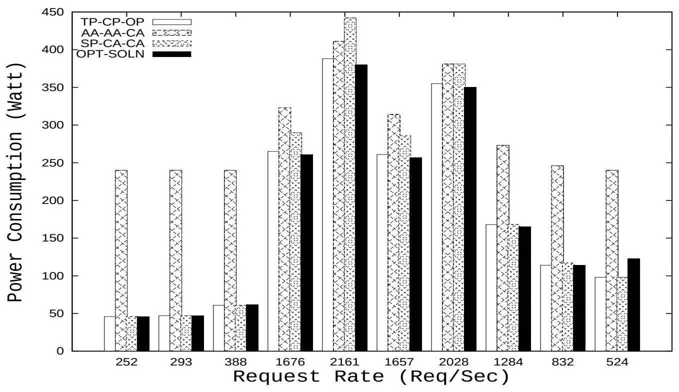


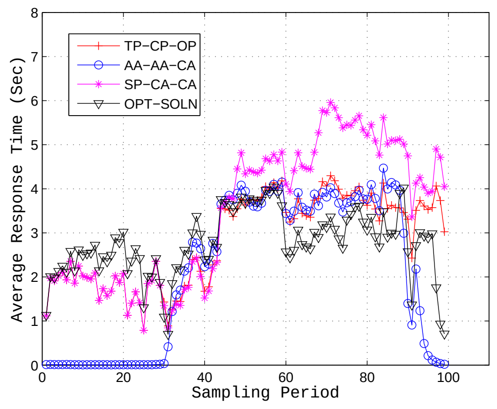
Figure 3.10: Simulation Result of Web Log Based Workload: Power

Figure 3.10 again demonstrates that our algorithm TP-CP-OP achieves the best power efficiency. It outperforms baseline algorithms to a large extent. Furthermore, on average, TP-CP-OP consumes 4% less power than OPT-SOLN because the latter leads to a total of 28 server on/off switches, while our heuristic TP-CP-OP only causes 12 switches. There are 9 sampling periods when OPT-SOLN algorithm causes 2 server switches and 10 sampling periods when it has 1 switch. During the period when $\lambda_{cluster}(t) = 524$ req/sec, our algorithm TP-CP-OP does not cause any server switch, while OPT-SOLN leads to 1 switch. That is why during that period OPT-SOLN algorithm consumes more power than TP-CP-OP.

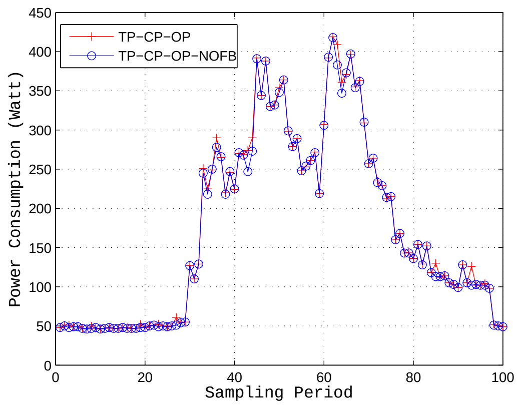
3.4.5 Effects of Feedback Control

As described in Section 3.2, to overcome the inaccuracy of the $M/M/1$ queuing model, we apply an approach that combines feedback control with queuing-theoretic prediction for back-end server DVS. This section evaluates the impact of the feedback control. We compare the combined mechanism with a queuing-theoretic prediction only mechanism where no feedback control of DVS is applied.

Figure 3.11a shows the resultant average response times when the feedback control is not applied. As we can see, due to the modeling inaccuracy, the re-



(a) Time



(b) Power

Figure 3.11: Effects of Feedback Control

sponse times are no longer around 1 second. In contrast, when the feedback control is combined with the queuing-theoretic prediction, the average response times, as shown in Figure 3.9, 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 Figure 3.11b 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 queuing estimated frequency f_i and thus leads to a very small difference in power consumption. These experimental results show that the queuing model based estimate of f_i is very close to the real frequency setting, which justifies the adoption of queuing estimated f_i in the optimization problem formulation (see Section 3.2).

Chapter 4

Power-Efficient Workload Distribution for Virtualized Sever Clusters

In this chapter, we will introduce the second part of our work. This research develops a workload distribution algorithm for virtualized server clusters to reduce their power consumptions and provide QoS. We extend our previous work presented in Chapter 3 and deploy the modified algorithm presented in Section 3.3 to a new platforms of virtualized server clusters. Simulation results show the advantages of our algorithm.

The remainder of this section is organized as follows. Sections 4.1 and 4.2 respectively present the models and state the problem. We discuss the algorithms in Sections 4.3 and 4.4, and evaluate their performance in Section 4.5.

4.1 Models

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

4.1.1 System Architecture

The system architecture is similar to the previous one illustrated by Figure 3.1 in Chapter 3. A cluster consists of a front-end server, connected to N back-end servers. We assume a typical cluster environment in which the front-end server does not participate in the request processing and is mainly responsible for accepting and distributing requests to back-end servers. A virtualized computing environment is assumed, where there are M online services hosted in virtual machines (VMs) of the cluster. Virtualization allows a single back-end server to be shared among multiple performance-isolated VMs. The placement of the M types of VMs on the N physical servers is given by an $N \times M$ matrix X, where $x_{ij} = 1$ means that the i^{th} physical server hosts a VM for the j^{th} service and $x_{ij} = 0$ indicates that the j^{th} VM is not placed on the i^{th} server.

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 f_i could be continuously adjusted in the range $[f_{i\_min}, f_{i\_max}]$. If a processor only supports discrete frequencies, we follow an approach similar to that proposed in [48] 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 thesis we assume and use the following specific models to illustrate our method.

4.1.2 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 [6, 42, 52]. 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 μ_i, is expressed as $\mu_i = \alpha_i f_i$, where α_i is the CPU performance coefficient. Different servers may have different values for α_i. With the same CPU frequency setting, the higher the α_i the more powerful the server is. For a particular VM on server i, its throughput depends on its allocated CPU time and its requirement for CPU. That is:

$$\mu_{ij} = \frac{\alpha_i f_{ij}}{e_j} \tag{4.1}$$

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

4.1.3 Power Model

The power consumption P_i of a server consists of a constant part and a variable part. Similar to previous work [19, 10, 24], we approximate P_i by the following function:

$$P_i = c_i + \beta_i f_i^p \tag{4.2}$$

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

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

$$J = \sum_{i=1}^{N} (c_i + \beta_i f_i^3) \tag{4.3}$$

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

4.2 Power Management Problem

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

minimize

$$J = \sum_{i=1}^{N} [c_i + \beta_i f_i^3] \tag{4.4}$$

subject to:

$$\begin{cases} \sum_{i=1}^{N} x_{ij}\lambda_{ij} = \lambda_j, & j = 1, 2, \cdots, M \\ g(\mu_{ij}, \lambda_{ij}) \approx \hat{R}_j, & i = 1, 2, \cdots, N, \\ & j = 1, 2, \cdots, M \\ f_i = max(\sum_{j=1}^{M} x_{ij} f_{ij}, f_{i\_min}) & i = 1, 2, \cdots, N \\ f_i \leq f_{i\_max}, & i = 1, 2, \cdots, N \end{cases} \tag{4.5}$$

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

For the power management, the front-end server decides the workload distribution λ_{ij} for all VMs. On the back-end, each physical server dynamically adjusts its CPU operating frequency f_i in the range $[f_{i\_min}, f_{i\_max}]$ and the CPU allocation f_{ij} to VMs. The back-end server ensures the average response time requirement by adopting an approach, similar to [43], that combines feedback control with queuing-theoretic prediction.

According to the $M/M/1$ queuing model, function $R_{ij} = g(\mu_{ij}, \lambda_{ij})$ is approximated as follows:

$$R_{ij} = \frac{1}{\mu_{ij} - \lambda_{ij}} = \frac{1}{\alpha_i f_{ij}/e_j - \lambda_{ij}} \tag{4.6}$$

To guarantee $R_{ij} \approx \hat{R}_j$, we approximate the proper f_{ij} to be:

$$f_{ij} = \frac{e_j}{\alpha_i}(\lambda_{ij} + \frac{1}{\hat{R}_j}) \tag{4.7}$$

Thus, the power management problem becomes:

minimize

$$J = \sum_{i=1}^{N}(c_i + \beta_i f_i^3) \tag{4.8}$$

subject to:

$$\begin{cases} \sum_{i=1}^{N} x_{ij}\lambda_{ij} = \lambda_j, & j = 1, 2, \cdots, M \\ f_i = max(\sum_{j=1}^{M} x_{ij}\frac{e_j}{\alpha_i}(\lambda_{ij} + \frac{1}{\hat{R}_j}), f_{i\_min}), & i = 1, 2, \cdots, N \\ f_i \le f_{i\_max}, & i = 1, 2, \cdots, N \end{cases} \tag{4.9}$$

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

4.3 Workload Distribution Algorithms

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

4.3.1 PE-based Workload Distribution

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

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

$$E_i = \frac{dP_i}{d\mu_i} = 3\frac{\beta_i}{\alpha_i} f_i^2 \tag{4.10}$$

If we can keep all E_i (i.e., $\sqrt{E_i}$) as low as possible, the cluster's power consumption increases the slowest as the workload grows. Based on this idea, we propose an optimization problem with a linear objective function to approximate the one formed in Section 4.2. The new optimization problem is as follows:

minimize

$$\nu \tag{4.11}$$

subject to:

$$\begin{cases} \sqrt{3\beta_i} f_i - \nu \leq 0, & i = 1, 2, \cdots, N \\ \sum_{i=1}^{N} x_{ij}\lambda_{ij} = \lambda_j, & j = 1, 2, \cdots, M \\ f_i = max(\sum_{j=1}^{M} x_{ij}\frac{e_j}{\alpha_i}(\lambda_{ij} + \frac{1}{\hat{R}_j}), f_{i\_min}), & i = 1, 2, \cdots, N \\ f_i \leq f_{i\_max}, & i = 1, 2, \cdots, N \end{cases} \tag{4.12}$$

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

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

4.3.2 LB-based Workload Distribution

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

minimize

$$\nu \tag{4.13}$$

subject to:

$$\begin{cases} \frac{f_i}{f_{i\_max}} - \nu \leq 0, & i = 1, 2, \cdots, N \\ \sum_{i=1}^{N} x_{ij}\lambda_{ij} = \lambda_j, & j = 1, 2, \cdots, M \\ f_i = max(\sum_{j=1}^{M} x_{ij}\frac{e_j}{\alpha_i}(\lambda_{ij} + \frac{1}{\hat{R}_j}), f_{i\_min}), & i = 1, 2, \cdots, N \\ f_i \leq f_{i\_max}, & i = 1, 2, \cdots, N \end{cases} \quad (4.14)$$

Solving this linear programming problem gives the workload distribution λ_{ij}. We call this method as *Load Balancing Workload Distribution*, LB-based for short.

4.3.3 Capacity-based Workload Distribution

When distributing workload, baseline algorithm II considers VM's capacity, where a VM hosted in a more powerful physical server gets more requests. We consider the throughput μ_i as the i^{th} server's capacity. The workload will be distributed to VMs in proportion to the capacities of their physical servers. Therefore, the workload distribution λ_{ij} can be expressed as follows:

$$\lambda_{ij} = \frac{x_{ij}\mu_i}{\sum_{i=1}^{N} x_{ij}\mu_i}\lambda_j \quad (4.15)$$

We name this method as *Capacity-based Workload Distribution*.

4.3.4 DVS and CPU Resource Allocation

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

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

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

and it is shared by VMs with the following weights:

$$w_{ij}[k] = \frac{x_{ij} f'_{ij}[k]}{\sum_{m=1}^{M} x_{im} f'_{im}[k]}, \quad j = 1, \cdots, M \tag{4.16}$$

i.e., the actual amount of CPU resource allocated to a VM at the k^{th} sampling period is $f^*_{ij}[k] = w_{ij}[k] f_i[k]$.

4.4 Admission Control Algorithms

In the previous section, three workload distribution methods are described. These methods can meet the QoS requirement only if the cluster is not overloaded. To ensure the QoS, we, therefore, also need to design admission control algorithms to avoid overloading the cluster. Since PE and LB-based methods follow the same workload constraints (i.e., $\sum_{i=1}^{N} x_{ij}\lambda_{ij} = \lambda_j$, $f_i = max(\sum_{j=1}^{M} x_{ij}\frac{e_j}{\alpha_i}(\lambda_{ij} + \frac{1}{\hat{R}_j}), f_{i\_min})$, and $f_i \leq f_{i\_max}$), their corresponding admission control algorithms are the same. In section 4.4.1, we present an admission control algorithm that is applicable to these two methods. The admission control algorithm for the capacity-based workload distribution is described in section 4.4.2.

4.4.1 PE and LB-based Admission Control

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

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

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

$$\hat{C}_k = \sum_{i=1}^{n} \alpha_i f_{i\_max} \tag{4.17}$$

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

$$d_k = \sum_{j=1}^{m} e_j \lambda_j \tag{4.18}$$

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

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

$$q_j = \sum_{i=1}^{n} x_{ij} \frac{\mu_i}{s_i} \tag{4.19}$$

where s_i denotes the number of different VMs hosted on server i. In the above, we have assumed that the server capacity quota $\mu_i = \alpha_i f_{i\_max}$ is shared fairly among hosted services. When we need to reject $\hat{E}_k$ amount of workload, q_j will be used to calculate the share of rejection for each service. The more extra workload a service has, i.e., the larger $(e_j\lambda_j - q_j)$ is, the more workload rejection it has. However, simply rejecting the extra workload of all services does not work, because in most cases not all services exceed their quotas. To avoid high reject ratio and low system utilization, we instead divide the $\hat{E}_k$ amount of workload rejection among overloaded services as follows:

$$\delta_j = \begin{cases} 0 & \text{if } q_j \geq e_j\lambda_j \\ \frac{\hat{E}_k(e_j\lambda_j - q_j)}{\sum_{j\in\{l|q_l<e_l\lambda_l\}}(e_j\lambda_j - q_j)} & \text{if } q_j < e_j\lambda_j \end{cases}$$

where $j = 1, 2, \ldots, m$.

If a straightforward approach were followed, we need to carry out the aforementioned analysis and control for every possible service group. For a cluster serving M services, that means 2^M repetitions of the above procedure. This large time complexity is not acceptable. Therefore, our admission control algorithm instead follows a reactive and iterative approach, which repeats the procedure only if the solver repetitively fails to find a feasible solution for the optimization problem (i.e., Equations (4.11) and (4.12) or Equations (4.13) and (4.14)). That is, when the optimization solver fails for the first time. We start the analysis and control procedure for the first group G_1 of services (i.e., the group that includes all M services). If cutting the workload for G_1 resolves the overload situation and makes the optimization problem solvable, we are done with the admission control. Otherwise, the procedure is repeated for the next largest group that remains to be tested. Based on our experience, the overload is often eliminated after only 2 or 3 iterations.

4.4.2 Capacity-based Admission Control

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

According to the capacity-based algorithm (see Section 4.3.3), the total workload demand distributed to server i is:

$$\mu'_i = \sum_{j=1}^{M} e_j \lambda_{ij} = \sum_{j=1}^{M} e_j \frac{x_{ij}\mu_i}{\sum_{i=1}^{N} x_{ij}\mu_i} \lambda_j \tag{4.20}$$

while server i's total capacity is:

$$\mu_i = \alpha_i f_{i\_max} \tag{4.21}$$

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

$$t_j = max\{\frac{x_{ij}\hat{E}_i}{\mu'_i} | 1 \leq i \leq N\} \tag{4.22}$$

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

4.5 Performance Evaluation

This section evaluates the performance of the proposed algorithms in our second work.

Virtualized Cluster Configuration. A discrete simulator has been developed to simulate heterogeneous virtualized clusters that are compliant to models presented in Section 4.1. We simulate the following clusters:

- **C**luster1. First, we simulate a cluster that consists of 4 back-end servers. To derive server parameters, experimental data from [42, 11, 21] are referred. Table 4.1 lists the estimated server parameters. In addition, we assume that the processors only support discrete frequencies, i.e., a processor's frequency can only be set to one of ten discrete levels in the range $[f_{i\_min}, f_{i\_max}]$, where f_{i_min} = 25%f_{i_max}. This cluster is assumed to provide 4 different services.

- **C**luster2. Second, we simulate a cluster that has 16 back-end servers of 4 different types, whose parameters are the same as those listed in Table 4.1. The

Server	f_{i_max}	c_i	β_i	α_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 4.1: Parameters of 4 Types of Server

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 $f_{i\_min} = 25\% f_{i\_max}$. This cluster is assumed to provide 16 different services.

Virtual Machine Placement. Another key configuration is on the placement of VMs. A VM placement can be defined in terms of the total number of VMs, their types and their distribution among physical servers. We assume no two VMs in a physical server are the same. Thus, a physical server can host up to 4 VMs in Cluster1 and up to 16 VMs in Cluster2. It is expected that algorithms could perform differently under different VM placements. Thus, to fairly compare algorithms, we often evaluate their performance under serval different placements. In paper [49], we investigates the effects of VM placement on algorithm performance.

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

To generate requests for service j, we assume their inter-arrival time follows a series of exponential distribution with a time varied mean $\frac{1}{\lambda_j[k]}$. In Sections 4.5.1 and 4.5.2, we simulate cases where a cluster is not overloaded. As illustrated in Figure 4.1, the total workload of Cluster1 (i.e., $\lambda_{cluster}[k] = \sum_{j=1} 4\lambda_j[k]$) changes

Service	e_j	$\hat{R}_j$
1	1	0.5
2	2	0.9
3	3	1.4
4	4	2.0

Table 4.2: Cluster1: Service Parameters

in the range [10%, 90%] of the cluster capacity. Similar workload patterns are generated for Cluster2. In paper [49], we also generate overloaded workloads, where the total workload sometimes reaches 2.1 times of the cluster capacity.

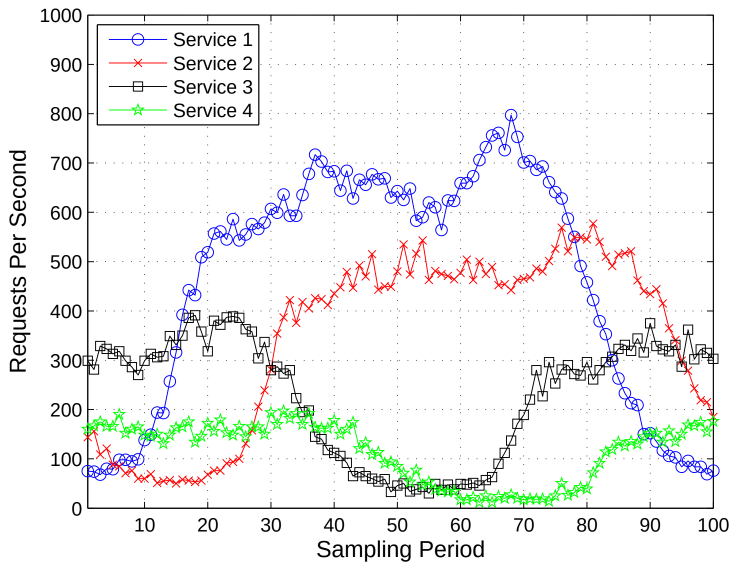


Figure 4.1: Average Request Rate

Request execution time E_i is assumed to follow exponential distribution, whose average is $\frac{e_j}{\mu_1}$ (i.e., $\frac{e_j}{\alpha_1 f_{1\_max}}$) for service j. Table 4.2 illustrates the parameters of the 4 services in Cluster1, including CPU demand factor e_j and average response time target $\hat{R}_j$.

Each simulation lasts 3000 seconds and periodically, i.e., every 30 seconds, the system measures the current workloads and predicts the average request rates $\lambda_j[k]$ for the next period. We adopt a method proposed in [22] for the workload prediction. Based on the predicted rates $\lambda_j[k]$, the corresponding workload distribution (i.e., $\lambda_{ij}[k]$) are derived by the algorithm. According to $\lambda_{ij}[k]$, the back-end server DVS mechanism decides the server's frequency setting $f_i[k]$. Since a CPU only supports discrete frequencies, we approximate the desired continuous frequency $f_i[k]$ by switching the CPU frequency between two adjacent discrete values, e.g., to approximate 2.65GHz frequency, during the 30-second sampling period, the CPU frequency is first set at 2.4GHz for 11.25 seconds and then at 2.8GHz for 18.75 seconds. To evaluate algorithm performance, we measure two metrics: average response time and power consumption.

4.5.1 Non-Overloaded Cluster1

This section evaluates the performance of workload distribution algorithms in Cluster1. We choose a virtual machine placement where there are 2 VMs on each physical server and each service has 2 corresponding VM instances. As mentioned, moderate workload is generated, which does not overload the cluster. Figure 4.2 presents the cluster power consumption, which demonstrates that our PE-based workload distribution algorithm consumes the least power in all sampling periods. We illustrate the resultant average response times in Figures 4.3, 4.4 and 4.5. From the data, we can see that all three algorithms ensure QoS, successfully keeping average response times around their targets: 0.5, 0.9, 1.4 and 2 seconds.

4.5.2 Non-Overloaded Cluster2

In this section, we evaluate algorithm performance in Cluster2, where we choose a VM placement that includes a total of 192 VMs. Experimental data are

Alg	Power (Watt)	
	Avg	Std
PE-based	274.8	1.31
Capacity-based	289.2	2.99
LB-based	285.2	0.87

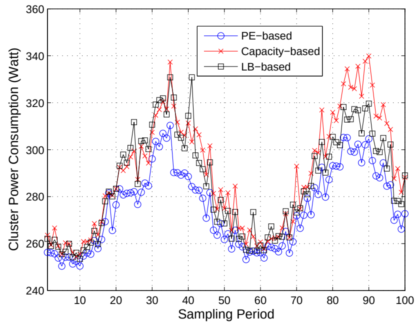


Figure 4.2: Non-Overloaded Cluster1: Power Consumption

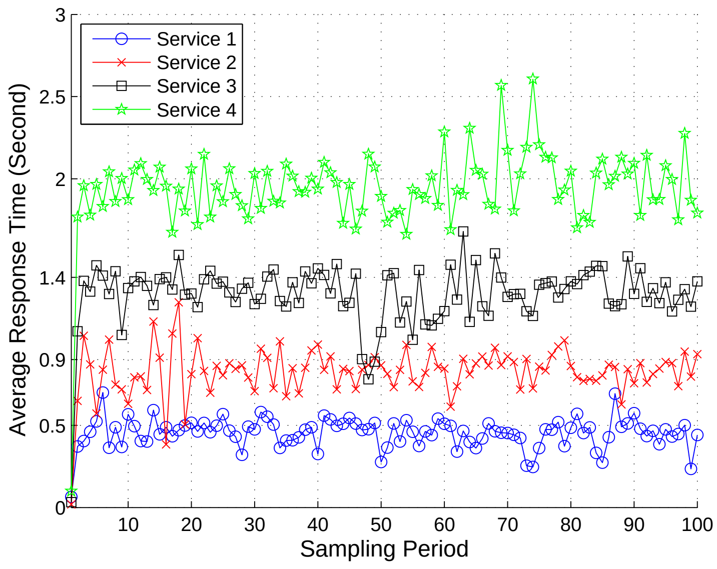


Figure 4.3: PE-based Non-Overloaded Cluster1: Average Response Time

similar to those shown in Section 4.5.1. Due to the space limitation, we only illustrate the power consumption data in Figure 4.6. Again, the PE-based algorithm leads to the smallest power consumption.

4.5.3 Effects of Virtual Machine Placement

It is expected that algorithms could perform differently under different VM placements. Therefore, in this section, we investigate the effects of VM placement on

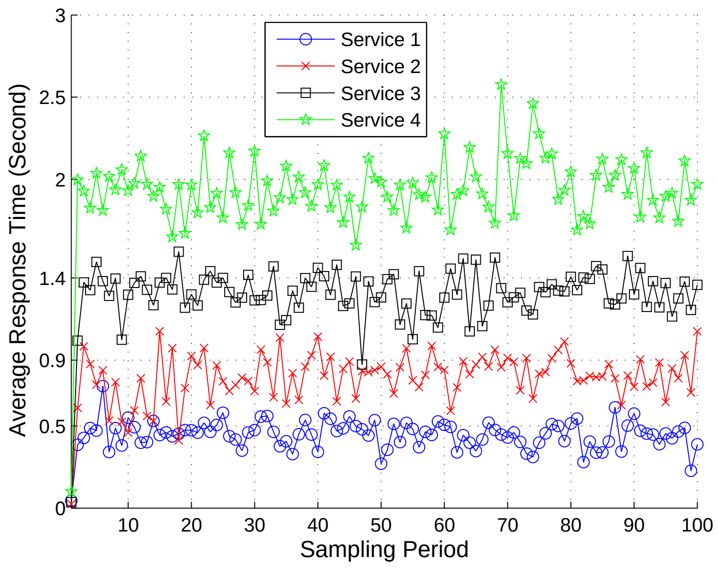


Figure 4.4: LB-based Non-Overloaded Cluster1: Average Response Time

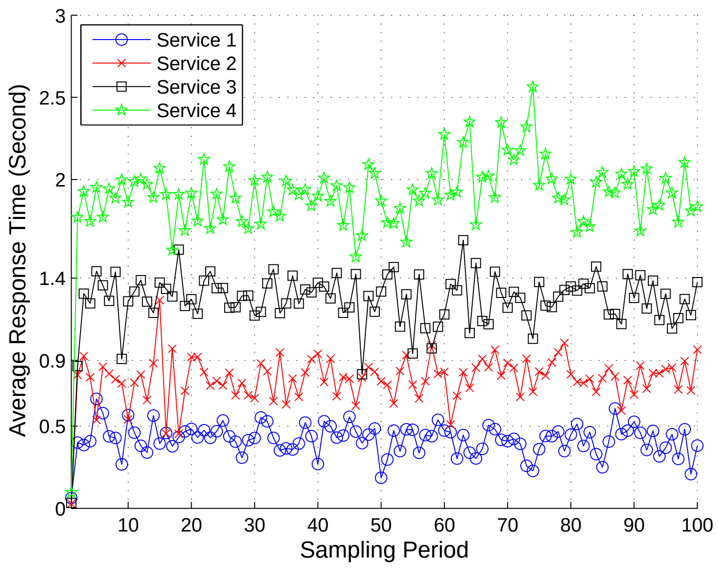


Figure 4.5: Capacity-based Non-Overloaded Cluster1: Average Response Time

Alg	Power (Watt)	
	Avg	Std
PE-based	1100	3.23
Capacity-based	1201	5.45
LB-based	1144	2.78

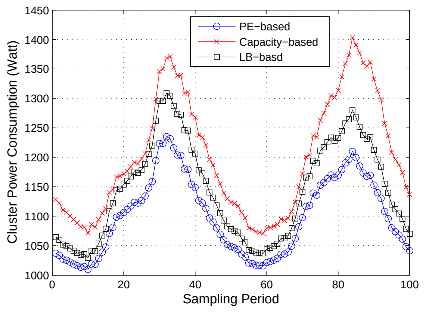


Figure 4.6: Non-Overloaded Cluster2: Power Consumption

algorithm performance.

The first experiment studies how the VM placement affects the feasibility of the workload distribution. Since PE and LB-based algorithms share the same workload constrains, scenarios where they have feasible solutions are the same. Thus, in this experiment, we compare PE & LB-based algorithms with capacity-based algorithm. We simulate Cluster1 with 19 different workload levels, where the total workload grows 5% each time from 10% to 100% of the cluster capacity. Since Cluster1 is composed of 4 physical servers and provides 4 services, there could be at most 16^4, i.e., 65536 number of different VM placements. We only test those *q*ualified placements where each physical server has at least 2 VMs and each service has at least 2 corresponding VMs. For each of the 19 workload levels, we simulate the workload distribution algorithms subject to the 7342 number of *q*ualified VM placements. The percentage of placements where an algorithm successfully finds a feasible solution is shown in Figure 4.7. From the figure, we can see that, when the workload level is below 10, all algorithms can find feasible workload distribution solutions under all VM placements. As the workload level increases, the capacity-based algorithm fails to find a feasible solution for an increasingly large portion of placements. On the contrary, the feasibility ratio of the other two algorithms still keeps stable until workload level 16. These data show that unlike the capacity-based algorithm, with the PE or LB-based algorithm, the VM placement has less effect on the effective cluster capacity.

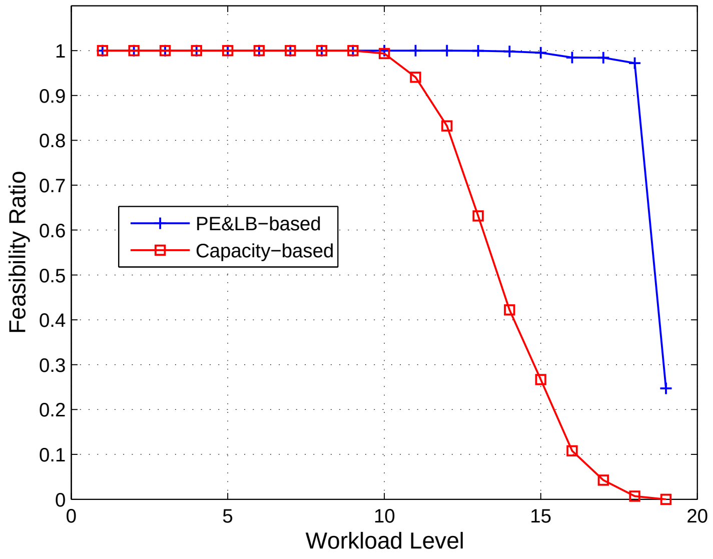


Figure 4.7: Feasibility of Algorithms under Different VM Placements

Next, we evaluate how VM placement affects the power consumption. This experiment runs on Cluster2. For each algorithm, we simulate the same workload under 4 different VM placement groups. Placement groups are distinguished by the total number of VMs, where placements with the same total number of VMs fall into one group. We choose 4 groups whose placements have 128, 176, 192 and 240 VMs respectively. From each group, 10 placements are randomly picked. We test how an algorithm performs under these placements. For each sampling period, we calculate the resultant power consumption averaged among each placement group. Figures 4.8, 4.9 and 4.10 respectively show the results for PE, LB and capacity-based algorithms. We can see that with the PE or LB-based algorithm, the resultant power consumption is insensitive to the VM placements. Despite that different placement groups have different numbers of VMs, their average power consumptions are pretty much the same. This is not the case for the capacity-based algorithm, where the difference of power consumption among the 4 groups are obviously much larger. This is because the capacity-based algorithm employs a workload distribution strategy that is not adaptive to the placement, while for PE and LB-based algorithms, they always strive to find the most power-efficient or balanced workload distribution based on the current placement.

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

Num.of VMs	Power (Watt)	
	Avg	Std
128	1107	4.15
176	1101	3.20
192	1101	3.23
240	1102	3.17

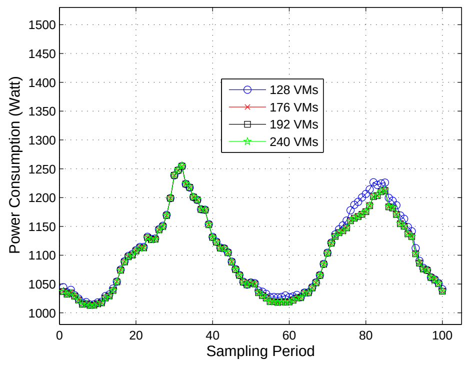


Figure 4.8: PE-based Algorithm: Power Consumption

Num.of VMs	Power (Watt)	
	Avg	Std
128	1147	2.67
176	1145	2.95
192	1145	2.55
240	1146	2.78

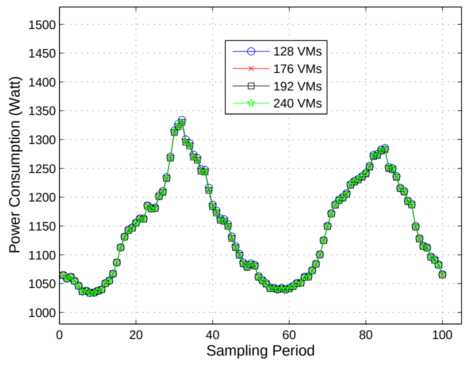


Figure 4.9: LB-based Algorithm: Power Consumption

Num.of VMs	Power (Watt)	
	Avg	Std
128	1182	8.02
176	1233	7.83
192	1202	7.36
240	1214	6.44

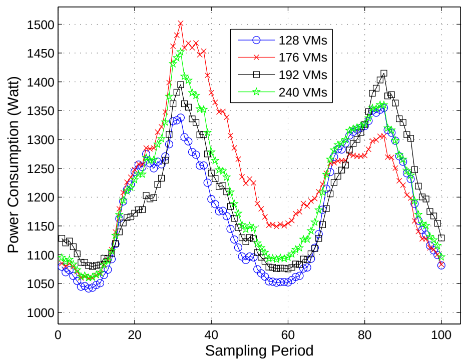


Figure 4.10: Capacity-based Algorithm: Power Consumption

Level	1.2	1.5	1.8	2.1
PE & LB-based	0.66%	7.80%	15.90%	23.39%
Capacity-based	3.43%	11.28%	19.09%	26.96%

Table 4.3: Request Reject Ratio under Different Workload Levels

	1.2	1.5	1.8	2.1
PE-based	348.4	399.2	420.1	472.2
LB-based	362.4	399.8	440.9	474.7
Capacity-based	357.5	416.4	440.0	470.2

Table 4.4: Power Consumption (Watt) under Different Workload Levels

4.5.4 Admission Control Performance

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

This experiment runs on Cluster1 with a fixed VM placement. We simulate each algorithm with 4 different workload levels. For a workload, the ratio of its peak volume vs. the total cluster capacity is used to represent its level. We test the following 4 workload levels: 1.2, 1.5, 1.8 and 2.1. The resultant request reject ratios are shown in Table 4.3. Since PE and LB-based algorithms share the same admission control module, they result in the same reject ratio. Compared with the capacity-based algorithm, they always reject less requests.

Figures 4.11, 4.12 and 4.13 respectively present the average response time of the three algorithms subject to the highest workload level 2.1. Since admission control modules have rejected extra requests, the average response times of admitted requests are successfully kept around their targets.

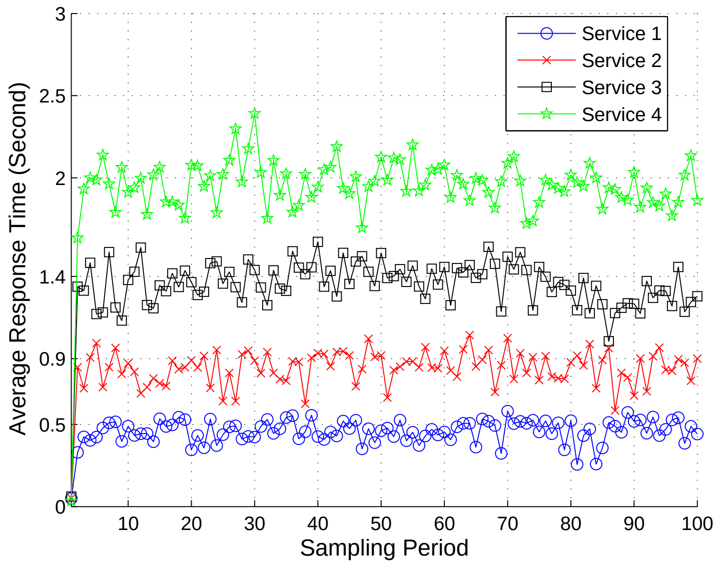


Figure 4.11: PE-based Overloaded Cluster1: Average Response Time

In Figures 4.14, 4.15, 4.16 and 4.17, we show the power consumption with the 4 workload levels respectively. Our PE-based algorithm still performs the best in most sampling periods. When request rejection happens, by rejecting the largest number of requests, the capacity-based algorithm sometimes leads to the least power consumption. Table 4.4 summarizes the power consumption data. From Tables 4.3 and 4.4, we conclude that the PE-based algorithm always serves the largest number of requests with nearly the least amount of power consumption.

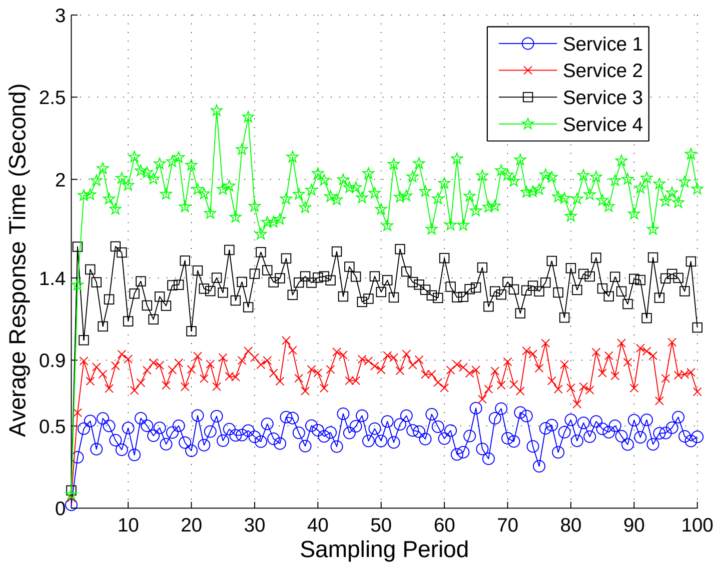


Figure 4.12: LB-based Overloaded Cluster1: Average Response Time

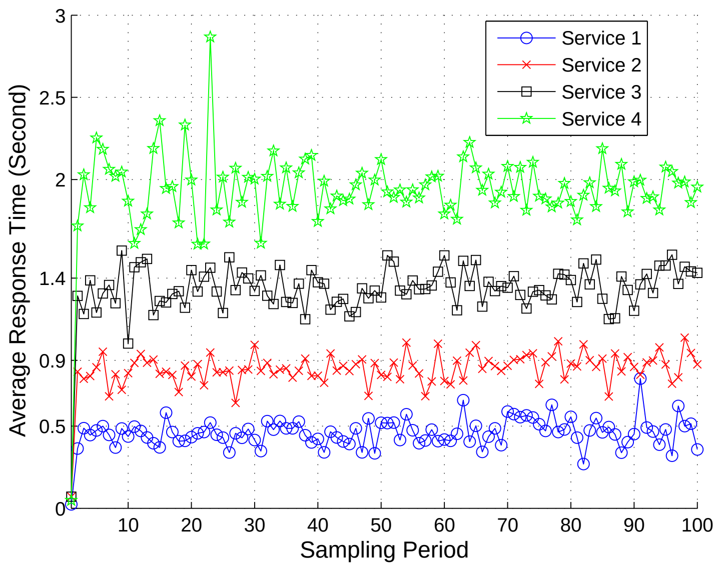


Figure 4.13: Capacity-based Overloaded Cluster1: Average Response Time

Alg	Power (Watt)	
	Avg	Std
PE-based	348.4	2.29
Capacity-based	357.5	3.19
LB-based	362.4	1.87

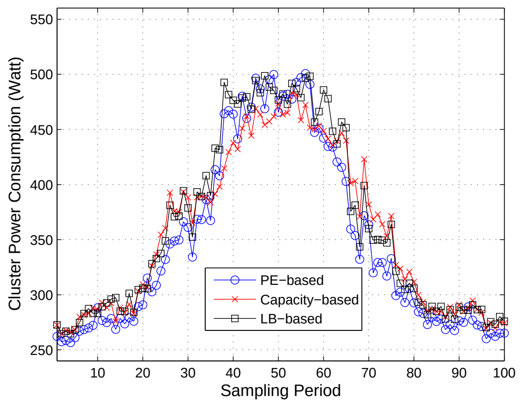


Figure 4.14: Power Consumption - Workload Level 1.2

Alg	Power (Watt)	
	Avg	Std
PE-based	399.2	2.64
Capacity-based	399.8	3.67
LB-based	416.4	2.52

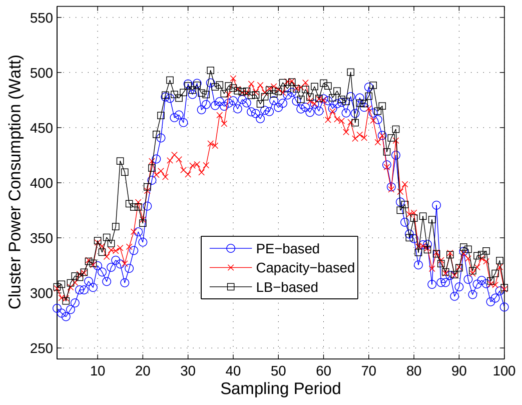


Figure 4.15: Power Consumption - Workload Level 1.5

Alg	Power (Watt)	
	Avg	Std
PE-based	420.1	2.55
Capacity-based	440.0	3.18
LB-based	440.9	2.03

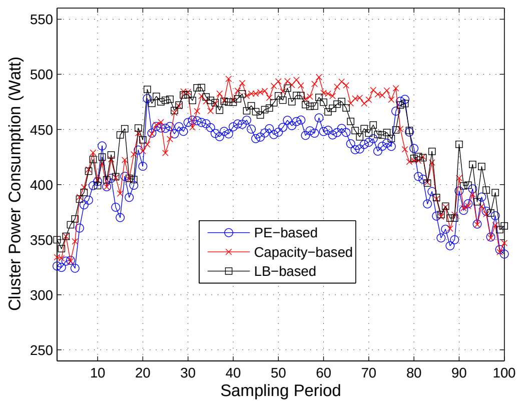


Figure 4.16: Power Consumption - Workload Level 1.8

Alg	Power (Watt)	
	Avg	Std
PE-based	472.2	1.76
Capacity-based	470.2	2.51
LB-based	474.7	1.22

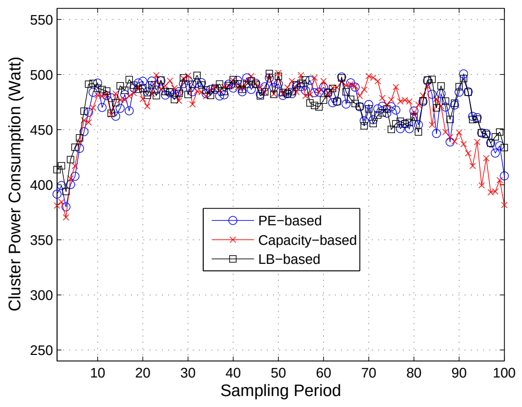


Figure 4.17: Power Consumption - Workload Level 2.1

Chapter 5

Conclusion

In this thesis, we first presented a threshold-based method for efficient power management of heterogeneous soft real-time clusters. Then, in the second part, we modified this algorithm such that it can be unprecedentedly deployed to a virtualized server cluster system.

Our methods have two advanced features. First, based on simple but effective mathematical models, the algorithm leads to low software customization costs when deployed to varied cluster platforms. Second, the algorithm is developed upon a solid theoretical foundation, where we integrate optimization, queuing theory and control theory techniques. By experiments, we compared the proposed algorithm with different baseline algorithms. We studied how these algorithms perform in different clusters, VM placements and workloads. Simulation results have demonstrated the strong advantages of our algorithms, which incur low overhead and lead to near-optimal power consumption.

Bibliography

[1] ftp://ftp.ircache.net/Traces/DITL-2007-01-09/rtp.sanitized-access.20070109.gz. 3.4, 3.4.4

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

[3] Luiz Andre Barroso, Jeffrey Dean, and Urs Holzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23(2):22–28, 2003.

[4] Luciano Bertini, Julius C. B. Leite, and Daniel Mosse. Generalized tardiness quantile metric: Distributed dvs for soft real-time web clusters. In *ECRTS '09: Proceedings of the 2009 21st Euromicro Conference on Real-Time Systems*, pages 227–236, Washington, DC, USA, 2009. IEEE Computer Society. 1

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

[6] Pat Bohrer, Elmootazbellah N. Elnozahy, Tom Keller, Michael Kistler, Charles Lefurgy, Chandler McDowell, and Ram Rajamony. The case for power management in web servers. *Power aware computing*, pages 261–289, 2002. 2, 3.1.2, 4.1.2

[7] T. D. Burd and R. W. Brodersen. Energy efficient cmos microprocessor design. In *HICSS '95: Proceedings of the 28th Hawaii International Conference on System*

Sciences (HICSS'95), page 288, Washington, DC, USA, 1995. IEEE Computer Society.

[8] Jeff Chase and Ron 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. 1, 2

[9] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centers. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 103–116, New York, NY, USA, 2001. ACM Press.

[10] Yiyu Chen, Amitayu Das, Wubi Qin, Anand Sivasubramaniam, Qian Wang, and Natarajan 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. 1, 2, 3.1.3, 4.1.3

[11] Mike Chin. M. chin, desktop cpu power survey, silentpcreview.com, april 2006. 3.4, 4.5

[12] Mike Chin. Desktop cpu power survey. *Silentpcreview.com*, Apr. 2006.

[13] Kihwan Choi, Ramakrishna Soma, and Massoud Pedram. Dynamic voltage and frequency scaling based on workload decomposition. In *ISLPED '04: Proceedings of the 2004 international symposium on Low power electronics and design*, pages 174–179, New York, NY, USA, 2004. ACM Press.

[14] E.K.P. Chong and Stanislaw H. Żak. *An Introduction to Optimization*. Wiley, 2001. 3.3.4

[15] Michele Colajanni, Valeria Cardellini, and Philip S. Yu. Dynamic load balancing in geographically distributed heterogeneous web servers. In *ICDCS '98: Proceedings of the The 18th International Conference on Distributed Computing Systems*, page 295, Washington, DC, USA, 1998. IEEE Computer Society.

[16] Mark E. Crovella and Azer Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes, 1996. 3.4.3

[17] Gaurav Dhiman, Giacomo Marchetti, and Tajana Rosing. vgreen: a system for energy efficient computing in virtualized environments. In *ISLPED*, pages 243–248. ACM, 2009. 2

[18] Mootaz Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy conservation policies for web servers. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems USITS'03*, March 2003.

[19] Mootaz Elnozahy, Mike Kistler, and Ram Rajamony. Energy-efficient server clusters. In *Proceedings of the Second Workshop on Power Aware Computing Systems*, February 2002. 1, 2, 3.1.3, 4.1.3

[20] Kinshuk Govil, Edwin Chan, and Hal Wasserman. Comparing algorithm for dynamic speed-setting of a low-power cpu. In *MobiCom '95: Proceedings of the 1st annual international conference on Mobile computing and networking*, pages 13–25, New York, NY, USA, 1995. ACM Press.

[21] Tom's Hardware. Cpu performance charts. *Tom's Hardware*, 2006. 3.4, 4.5

[22] John P. Hayes. Self-optimization in computer systems via on-line control: Application to power management. In *ICAC'04: Proceedings of the First Inter-*

national Conference on Autonomic Computing (ICAC'04), pages 54–61, Washington, DC, USA, 2004. IEEE Computer Society. 3.4, 4.5

[23] Taliver Heath, Bruno Diniz, Enrique V. Carrera, Wagner Meira Jr., and Ricardo 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. 1, 2, 3

[24] J. Heo, D. Henriksson, X. Liu, and T. Abdelzaher. Integrating adaptive components: An emerging challenge in performanceadaptive systems and a server farm case-study. In *IEEE International Real-Time Systems Symposium*, pages 227–238, December 2007. 3.1.3, 4.1.3

[25] Jin Heo, Dan Henriksson, Xue Liu, and Tarek 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.

[26] Fabien Hermenier, Nicolas Loriant, and Jean-Marc Menaud. Power management in grid computing with xen. *Frontiers of High Performance Computing and Networking, ISPA 2006 Workshops*, pages 407–416, 2006. 2

[27] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Online strategies for dynamic power management in systems with multiple power-saving states. *Trans. on Embedded Computing Sys.*, 2(3):325–346, 2003.

[28] Tohru Ishihara and Hiroto Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *ISLPED '98: Proceedings of the 1998 in-*

ternational symposium on Low power electronics and design, pages 197–202, New York, NY, USA, 1998. ACM Press.

[29] Dara Kusic, Jeffrey O. Kephart, James E. Hanson, Nagarajan Kandasamy, and Guofei Jiang. Power and performance management of virtualized computing environments via lookahead control. In *ICAC '08: Proceedings of the 2008 International Conference on Autonomic Computing*, pages 3–12, Washington, DC, USA, 2008. IEEE Computer Society. 1, 2

[30] Charles Lefurgy, Karthick Rajamani, Freeman Rawson, Wes Felter, Michael Kistler, and Tom W. Keller. Energy management for commercial servers. *Computer*, 36(12):39–48, 2003.

[31] 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. 1, 2

[32] John Markoff and Saul Hansell. Hiding in plain sight, google seeks more power. *New York Times*, June 2006.

[33] Bob Monkman. Cluster virtualization – revisiting beowulf-class linux cluster architectures. *HPCwire*, August 2006. 1

[34] Tridib Mukherjee, Ayan Banerjee, Georgios Varsamopoulos, Sandeep K. S. Gupta, and Sanjay Rungta. Spatio-temporal thermal-aware job scheduling to minimize energy consumption in virtualized heterogeneous data centers. *Comput. Netw.*, 53(17):2888–2904, 2009. 2

[35] Ripal Nathuji and Karsten Schwan. Vpm tokens: virtual machine-aware power budgeting in datacenters. In *HPDC '08: Proceedings of the 17th inter-*

national symposium on High performance distributed computing, pages 119–128, New York, NY, USA, 2008. ACM. 2

[36] Trevor Pering, Tom Burd, and Robert Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *ISLPED '98: Proceedings of the 1998 international symposium on Low power electronics and design*, pages 76–81, New York, NY, USA, 1998. ACM Press.

[37] Padmanabhan Pillai and Kang G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *ACM Symposium on Operating Systems Principles*, pages 89–102, 2001.

[38] 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. 1, 2

[39] Eduardo Pinheiro and Ricardo Bianchini. Energy conservation techniques for disk array-based servers. In *ICS '04: Proceedings of the 18th annual international conference on Supercomputing*, pages 68–78, New York, NY, USA, 2004. ACM Press.

[40] Eduardo Pinheiro, Ricardo Bianchini, Enrique V. Carrera, and Taliver Heath. Dynamic cluster reconfiguration for power and performance. pages 75–93, 2003.

[41] 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. 1, 2

[42] Cosmin Rusu, Alexandre Ferreira, Claudio Scordino, Aaron Watson, Rami Melhem, and Daniel Mosse. Energy-efficient real-time heterogeneous server clusters. In *Proceedings of the Twelfth Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, Apr. 2006. 1, 2, 3, 3.4, 3.4, 3.4.3, 4.1.2, 4.5

[43] Lui Sha, Xue Liu, Ying Lu, and Tarek Abdelzaher. Queueing model based network server performance control. In *IEEE Real-Time Systems Symposium*, Austin, TX, December 2002. 3.2, 4.2

[44] Vivek Sharma, Arun Thomas, Tarek Abdelzaher, Kevin Skadron, and Zhijian 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. 2

[45] David C. Snowdon, Sergio Ruocco, and Gernot Heiser. Power management and dynamic voltage scaling: Myths and facts. In *Proceedings of the 2005 Workshop on Power Aware Real-time Computing*, September 2005.

[46] Keisuke Toyama, Satoshi Misaka, Kazuo Aisaka, Toshiyuki Aritsuka, Kunio Uchiyama, Koichiro Ishibashi, Hiroshi Kawaguchi, and Takayasu Sakurai. Frequency-voltage cooperative cpu power control: A design rule and its application by feedback prediction. *Syst. Comput. Japan*, 36(6):39–48, 2005.

[47] Leping Wang and Ying Lu. Efficient power management of heterogeneous soft real-time clusters. *Technical Report TR-UNL-CSE-2008-0004, University of Nebraska-Lincoln*, 2008. 1, 3.4

[48] Leping Wang and Ying Lu. Efficient power management of heterogeneous soft real-time clusters. In *Real-Time Systems Symposium (RTSS08)*, pages 323–332, December 2008. 4.1.1

[49] Leping Wang and Ying Lu. Power-efficient workload distribution for virtualized server clusters. In *High Performance Computing (HiPC), 2010 International Conference on*, pages 1 –10, dec. 2010. 4.5

[50] Xiaorui Wang and Yefu Wang. Co-con: Coordinated control of power and application performance for virtualized server clusters. *Technical Report, UTK, http://pacs.ece.utk.edu/techreports/tech0908.pdf*, 2008. 2

[51] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced cpu energy. In *Proceedings of 1st USENIX Symposium on Operating System Design and Implementation*, pages 13–23, November 1994.

[52] Ming Xiong, Song Han, Kam-Yiu Lam, and Deji Chen. Deferrable scheduling for maintaining real-time data freshness: algorithms, analysis, and results. 57(7):952–964, 2008. 4.1.2

[53] Songnian Zhou, Xiaohu Zheng, Jingwen Wang, and Pierre Delisle. Utopia: a load sharing facility for large, heterogeneous distributed computer systems. *Softw. Pract. Exper.*, 23(12):1305–1336, 1993.