

## Dynamic Frequency Scaling Schemes for Heterogeneous Clusters under Quality of Service Requirements

JIAN-JIA CHEN<sup>1</sup>, KAI HUANG<sup>2</sup> AND LOTHAR THIELE<sup>3</sup>

<sup>1</sup>*Department of Informatics  
Karlsruhe Institute of Technology  
76131 Karlsruhe, Germany*

*E-mail: j.chen@kit.edu*  
<sup>2</sup>*Department of Informatics  
Technical University of Munich  
80333 Munich, Germany*

*E-mail: kai.huang@tum.de*  
<sup>3</sup>*Computer Engineering Group  
ETH Zurich  
8092 Zurich, Switzerland  
E-mail: thiele@tik.ee.ethz.ch*

Nowadays, both the performance and power consumption for modern server clusters and data centers must be considered to reduce the maintenance cost for quality of service guarantees, as power dissipation affects the cost of both the power delivery subsystems and cooling facility. Considering the popularity of heterogeneous clusters, this paper proposes efficient and effective power management schemes for large scale server farms. Distinct from existing heuristic approaches, we propose dynamic frequency scaling schemes with approximation factor guarantees, compared to the optimal power management. By considering systems with discrete frequency levels on every server, our schemes can be applied for different power consumption models. Our greedy power management schemes have 1.5 or 2 approximation guarantees depending on the complexity. Our dynamic-programming approach can trade the quality, in terms of power consumption, of the resulting solutions with the time/space complexity. We provide extensive simulation results to show that the proposed schemes are effective for the minimization of the power consumption for large scale clusters.

**Keywords:** power management, soft real-time systems, dynamic voltage scaling, quality of service, heterogeneous clusters

### 1. INTRODUCTION

Power and energy consumption nowadays has become the key concern in server clusters or data centers. For example, a high-performance server with 300Watt power consumption consumes 2628 kiloWatt hours. Therefore, within one year, the annual power cost of the server is around \$263, if the electricity cost is \$0.1 per kiloWatt hour. Even without considering the cost of the power delivery subsystems and the cooling facility, for maintaining a cluster with hundreds of servers, the electricity cost is significant. Another fact is that the performance per watt remains roughly flat over time [2], although advanced hardware technology has improved the performance per hardware dollar. As a result, the electricity cost of server clusters will be more than the hardware cost and be-

---

Received May 31, 2011; accepted March 31, 2012.  
Communicated by Junyoung Heo and Tei-Wei Kuo.

come a major fraction of the total cost of ownership.

To reduce the power consumption without sacrificing the performance, power-aware and energy-efficient scheduling has been extensively explored in the literature, especially for real-time systems, *e.g.*, [1, 5, 15]. Low-power opportunity for one single web server is observed in [3, 10] to reduce the energy consumption by applying dynamic voltage scaling (DVS) with minimal performance impact. For server clusters, by applying workload prediction for predicting the workload or the average request arrival rate for the next scheduling interval, the power management routine in the front-end server has to decide whether a back-end server should be activated for serving requests, deactivated for power saving, accelerated for serving more requests, or decelerated for serving less requests. For homogeneous server clusters with identical servers, Chase *et al.* [4] develop a load balancing framework to dynamically turn on/off servers, Xu *et al.* [14] propose algorithms to determine the number of servers to turn on by applying both DVS and DPM, and Wierman *et al.* [13] explore how to balance the mean energy consumption and the mean response time under processor sharing scheduling.

Considering the popularity of heterogeneous clusters, power management for heterogeneous server clusters under quality of service (QoS) guarantees has been recently explored in [7, 9, 12]. Specifically, Wang and Lu [12] develop a power management algorithm to order heterogeneous servers in a pre-defined order. After deciding the activation and deactivation of servers, Lagrange Multiplier Method is applied to decide the execution frequency. Similarly, for servers with discrete speeds, Rusu *et al.* [9] use two tables for deciding which servers to be activated and which frequency levels to be executed. Moreover, Guerra *et al.* [7] model the problem as an integer linear programming (ILP) problem by applying ILP solvers with high complexity to get a table for storing decisions for different workloads under pre-defined granularity.

For most commercial computing systems, the available frequency levels are fixed. As a result, the approaches in [12] might not be suitable, in which using a higher available frequency level might sacrifice the optimality. Because storing decisions in tables requires exponential space complexity in the worst case, researchers in [7, 9] discretize the possible amount of the to-be-served workload into pre-defined granularity and derive the scheduling tables based on the granularity. As a result, the quality of the derived solution heavily depends on the granularity. Moreover, if a server in the cluster is out of service due to some maintenance reasons, recomputing the scheduling tables by applying algorithms in [7, 9] might be time-consuming.

This paper explores the power management problem for heterogeneous clusters under QoS constraints. Distinct from the heuristic approaches in [7, 9, 12], we propose algorithms to provide different approximation guarantees for power consumption minimization under different time/space complexity. By considering systems with discrete frequency levels on servers, our schemes can be applied for general power consumption models and QoS models. Our greedy power management schemes have 1.5-approximation or 2-approximation guarantees depending on the complexity. Our dynamic-programming approach can trade the quality, in terms of power consumption, of the resulting solutions with the time/space complexity. As our approaches do not rely on global tables, they are more robust against failure of servers. If building a scheduling table is necessary for system designers, our proposed dynamic programming approach can be applied and extended to build tables with different granularity. Simulation results show that the pro-

posed schemes are effective for minimizing the power consumption. Note that this paper is an extended version of the conference paper in [6]. Compared to the conference version, we provide extensive simulation results by clusters with up to 800 servers to demonstrate the effectiveness of our algorithms.

The rest of this paper is organized as follows: Section 2 provides system models and problem definition along with hardness analysis. Section 3 presents our greedy power management schemes with different approximation factor guarantees. Section 4 demonstrates how to use dynamic programming to trade the quality of the derived solution with the time/space complexity. Simulation results are presented in section 6. Section 7 concludes this paper.

## 2. SYSTEM MODELS

This section presents the model of a cluster, the power consumption model of a server in the cluster, and the problem definition.

### 2.1 System Model

We consider a cluster with a front-end server, which arbitrarily distributes workload of requests to a cluster of back-end servers. The front-end server is assumed not to participate in processing any requests, but only to decide the power states of back-end servers and how to distribute the requests to those back-end servers that are activated. Fig. 1 illustrates an example for a cluster of web servers. The cluster consists of  $M$  heterogeneous back-end servers, denoted by  $m_1, m_2, \dots, m_M$ , providing CPU-bounded services. The heterogeneity comes from different hardware architectures, different manufacturing techniques, different vendors, *etc.* However, all these  $M$  back-end servers have the same functionality, *i.e.*, a request can be served in any of these back-end servers. As a result, once a back-end server is activated (turned on), it can serve any assigned request.

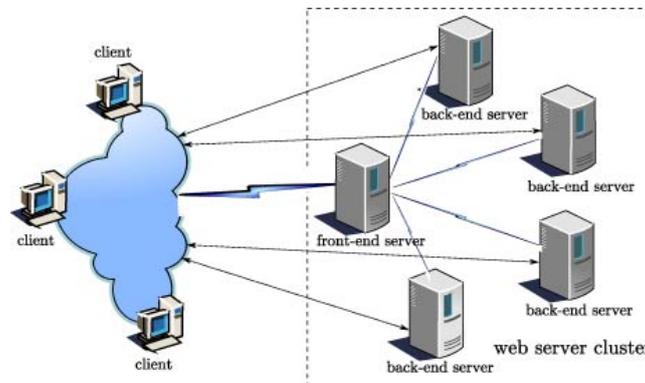


Fig. 1. An example for a web server cluster.

To satisfy the performance requirement, the cluster has to provide its services under some quality of service (QoS) constraint. Due to heterogeneity of back-end servers, the

performance of servers  $m_i$  and  $m_j$  at frequency  $f$  might be different. To have the same performance index under different servers, each server  $m_i$  is associated with a performance co-efficient  $\alpha_i$  such that  $\alpha_i f$  is the throughput (in terms of executed number of cycles or requests per time unit) of server  $m_i$  at frequency  $f$ .

To measure quality of servers, we might either apply analysis for soft real-time systems for the (average) percentage of requests that miss their timing constraints [9], or apply Queueing Theory to guarantee the average response time, e.g., M/M/1 model in [12] or M/G/1 PS model in [13]. For example, as shown in [12], if the quality of service is on the average response time under the M/M/1 queuing model, to serve workload with average request rate  $\lambda$  on server  $m_i$ , the average response time at frequency  $f$  is  $\frac{1}{\alpha_i f - \lambda}$ , where  $\alpha_i f$  is the number of requests finished per time unit. For the rest of this paper, suppose that  $q_i(f, \lambda)$  is the quality of service provided by server  $m_i$  at frequency  $f$  when the average request rate assigned on server  $m_i$  is with arrival request rate  $\lambda$ . Note that, the metric of  $q_i(f, \lambda)$  depends on the definition of the QoS of a server.

The derivation of  $q_i(f, \lambda)$  is not a focus of this paper. One might apply existing results in the literature, e.g., [9, 12, 13]. We only assume that  $q_i(f, \lambda)$  is not worse than  $q_i(f - \eta, \lambda)$  for any  $\eta > 0$ . In other words, this paper focuses on a more general setting, in which for a fixed average request rate assigned to a server, the QoS provided by the server is not worse when the server is operated at a higher frequency.

## 2.2 Power Consumption and DVS Models

We consider servers with discrete dynamic voltage scaling levels. The number of available frequency levels on server  $m_i$  is  $K_i$ . (Note that, if there is no possibility for dynamic voltage scaling on server  $m_i$ , we can simply set that  $K_i$  to 1 and  $f_{i,1}$  is the frequency of the server.) For brevity, we order the available frequency levels on server  $m_i$  from the lowest one to the highest. Let  $f_{i,j}$  be the  $j$ th lowest available frequency for server  $m_i$ , in which  $f_{i,1} < f_{i,2} < \dots < f_{i,K_i}$  when  $K_i > 1$ . The power consumption for server  $m_i$  on frequency  $f_{i,j}$  is  $P_{i,j}^\dagger$ .

When a server  $m_i$  is activated, it must operate at least at frequency  $f_{i,1}$ . However, if it is not necessary to turn a server on, we can deactivate the server (if it is activated) to reduce the power consumption. If a server  $m_i$  is not activated, its power consumption is assumed to be a constant  $P_i^\delta$ . If the power consumption at frequency  $f_{i,j}$  is less than  $f_{i,k}$  for some  $k < j$ , we can simply remove the power-inefficient frequency  $f_{i,k}$ . Therefore, we consider systems with  $P_i^\delta < P_{i,1}^\dagger < P_{i,2}^\dagger < \dots < P_{i,K_i}^\dagger$ . As we cannot reduce the power consumption  $P_i^\delta$ , it can be subtracted from the power consumption  $P_{i,j}^\dagger$ . For the rest of this paper, we will only focus on the *manageable power consumption*  $P_{i,j}$  of server  $m_i$  at frequency  $f_{i,j}$ , in which  $P_{i,j}$  is  $P_{i,j}^\dagger - P_i^\delta$ .

When a server is activated for execution, both efficiency and power consumption issues must be considered. For example, if a server has high power consumption, activating the server might consume too much power even though we might not have to activate the other servers. On the other hand, if a server has lower power consumption, activating the server might not be enough, and, hence, we might have to activate many servers to satisfy the QoS requirements.

As a result, we have to consider the *power density* of a server, which is defined as the power consumption of the server divided by the request rate it can serve under the QoS

requirement, denoted by  $R$ , of the cluster. Note that as we focus on general settings of QoS requirement,  $R$  could be average response time, average waiting time, or worst-case response time<sup>1</sup>, *etc.* Suppose that  $L_{i,j}(R)$  is the (average) request arrival rate that server  $m_i$  can serve at frequency  $f_{i,j}$  with quality of service no worse than  $R$ . That is,  $q_i(f_{i,j}, L_{i,j}(R))$  is not worse than  $R$ . For example, if the quality of service is the average response time in the M/M/1 queuing model,  $q_i(f_{i,j}, L_{i,j}(R)) = \frac{1}{\alpha_i f_{i,j} - L_{i,j}(R)} \leq R$ . For notational brevity, for the rest of this paper, we will define both  $P_{i,0}$  and  $L_{i,0}$  as 0. As  $R$  is assumed to be a fixed parameter, we will use  $L_{i,j}$  to represent  $L_{i,j}(R)$  for the rest of this paper. Fig. 2 illustrates an example for the power consumption and the power density, in which the power consumption is an increasing function of the average request arrival rate but the power density is not.

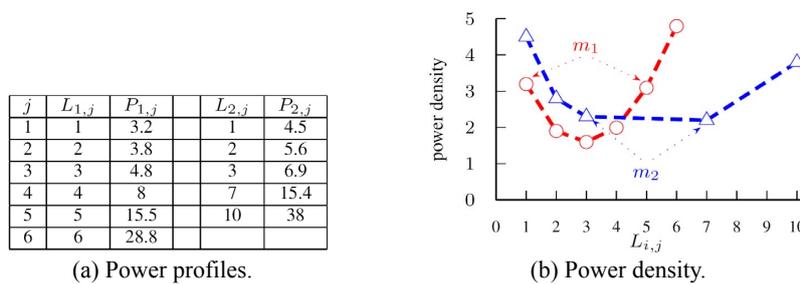


Fig. 2. An example for power consumption and power density of servers.

Furthermore, we assume that the front-end server is responsible for estimating the average request rate for the next time interval for scheduling, and for distributing the requests to back-end servers such that the quality of service is satisfied and the power consumption is minimized. To estimate the average request rate, one can apply a feedback-control with queuing theoretic prediction approach, which is also adopted in [12]. Throughout this paper, we assume that the workload prediction is known a priori, in which the average request rate of the cluster is  $\lambda$ , and our task is to decide the activation and frequency levels of back-end servers to minimize the power consumption under the QoS constraint.

### 2.3 Problem Definition

This work explores power management for a heterogeneous cluster under the quality of service requirement  $R$ . As the average request rate (workload) changes over time, the power management must be done dynamically to cope with dynamic rates. Suppose that the current average request rate of the cluster is  $\Lambda$ . Our objective is (a) to activate/deactivate servers to distribute the average request rate to those servers that are activated for execution, and (b) to decide the operation frequency of activated servers, such that the power consumption is minimized and the QoS requirement  $R$  is satisfied. For brevity, we denote the above problem as the Power Management for hEterogeneous server Clusters (PMEC) problem.

A solution  $S$  to the PMEC problem distributes  $\Lambda$  into  $\lambda_1(S), \lambda_2(S), \dots, \lambda_M(S)$  and de-

<sup>1</sup> For worst-case response time, we can only guarantee under the prediction of estimated worst cases instead of average cases.

cludes the frequency levels  $s_1(S), s_2(S), \dots, s_M(S)$  of these  $M$  back-end servers. A solution  $S$  is said *feasible* for the P MEC problem if the rate distribution is no less than  $\Lambda$ , i.e.,  $\sum_{i=1}^M \lambda_i(S) \geq \Lambda$ , and the frequency level on server  $m_i$  is sufficient to provide the quality of service requirement, i.e.,  $\lambda_i(S) \leq L_{i,s_i(S)}$ . For brevity, we denote the power consumption of a solution  $S$  by  $\Phi(S)$ , in which  $\Phi(S) = \sum_{i=1}^M P_{i,s_i(S)}$ . A solution  $S$  is said *optimal* for the P MEC problem if its power consumption is the minimum among the feasible solutions. For a solution, if  $\sum_{i=1}^M L_{i,s_i(S)}$  is no less than  $\Lambda$ , we can easily distribute  $L_{i,s_i(S)}$  average request rate to server  $m_i$  when  $s_i > 0$  without violating the QoS guarantee. Therefore, for the rest of this paper, we only focus our discussions on how to decide the frequency levels of servers to guarantee that the solution  $S$  satisfies  $L_{i,s_i(S)} \geq \Lambda$  with adoption of the above request distribution strategy.

If activating all the back-end servers at the highest frequency level cannot satisfy the QoS requirement, one has to augment the back-end servers, and there does not exist any feasible solution for the P MEC problem. For the rest of this paper, we focus on the optimality issue by considering cases with  $\sum_{i=1}^M L_{i,K_i} \geq \Lambda$ . Obviously, the studied problem is  $\mathcal{NP}$ -hard, as it can be reduced to the Knapsack problem.

Due to the  $\mathcal{NP}$ -hardness of the P MEC problem, this paper pursues polynomial-time approximation algorithms with worst-case guarantees on the quality of the derived solutions. A  $\rho$ -approximation algorithm for the P MEC problem (or, an algorithm with a  $\rho$ -approximation factor) guarantees to derive solutions with at most  $\rho$  times of the power consumption of the corresponding optimal solutions [11].

### 3. GREEDY SCHEMES

This section presents our proposed greedy power management schemes for the P MEC problem. We will first present the construction of a *decision tree* to decide whether we should turn on a server, or accelerate or decelerate the execution frequency of a server. Then, we will present our proposed greedy power management schemes with a 2-approximation factor, followed by improved schemes.

#### 3.1 Constructing Decision Trees

Before presenting our proposed greedy power management schemes, we will first describe the construction of a decision tree for a server  $m_i$ . Our proposed greedy power management schemes will decide whether we should activate a server for serving requests or accelerate a server for accommodating more request rate.

Suppose that server  $m_i$  is activated and  $c_i$  is the *critical frequency level* with  $D_{i,c_i} \leq D_{i,j}$  for  $1 \leq j \leq K_i$  and  $c_i > 0$ . The most power-efficient execution on server  $m_i$  is to distribute  $L_{i,c_i}$  amount of workload to server  $m_i$  at execution frequency level  $c_i$ . If we want to distribute more workload to server  $m_i$ , it will require more additional power density to serve these additional workload. If we want to reduce the allocated workload, we can definitely reduce the power consumption, but the resulting solution will have more power density than executing at the critical frequency level.

For the decision tree  $\mathcal{T}_i$  of server  $m_i$ , suppose that each vertex  $v$  of the tree has the following fields: *density*, *index*, *start*, *end*, *right*, and *left*, in which *density* is the increased

power density of this selection, *index* is the index of frequency level this vertex represents for, *start* (*end*, respectively) is the low bound of the frequency level (highest frequency level, respectively) used for the subtree rooted by *v*, *left* (*right*, respectively) is the left-hand child (right-hand child, respectively) of vertex *v*. For clarity, we will use *density*(*v*), *index*(*v*), *start*(*v*), *end*(*v*), *right*(*v*), and *left*(*v*) to present the corresponding values.

We use Algorithm 1, *i.e.*, calling Algorithm DT(*m<sub>i</sub>*, 0, *K<sub>i</sub>*), to construct  $\mathcal{T}_i$ . In the specified range between frequency level indexes *a* and *b* given as part of input parameters of Algorithm DT, we find the index  $j^*$  such that the increased power density by operating at frequency  $f_{i,j^*}$  is the minimum, in which the increased power density at frequency  $f_{i,j}$  is defined as  $\frac{P_{i,j} - P_{i,a}}{L_{i,j} - L_{i,a}}$ . Then, for the vertex (root of a sub-tree), we set *density*(*v*) to  $\frac{P_{i,j^*} - P_{i,a}}{L_{i,j^*} - L_{i,a}}$ , *index*(*v*) to  $j^*$ , *start*(*v*) to *a*, *end*(*v*) to *b*, the right child by calling DT(*m<sub>i</sub>*,  $j^*$ , *b*) recursively, and the left child by calling DT(*m<sub>i</sub>*, *a*,  $j^* - 1$ ) recursively. If Algorithm DT is called with  $a \leq b$ , this is a termination condition, in which we just simply return a null pointer. The construction of the decision tree  $\mathcal{T}_i$  for server *m<sub>i</sub>* takes  $O(K_i^2)$  time complexity and has  $O(K_i)$  space complexity. Note that the decision trees of servers are constructed only once in off line.

**Algorithm 1** DT  
**Input:** (*m<sub>i</sub>*, *a*, *b*);  
**Output:** a decision tree in the feasible range [*a*, *b*] for server *m<sub>i</sub>*;  
1: **if**  $a \leq b$  **then**  
2:     return **null**;  
3: **end if**  
4: construct a vertex *v*;  
5:  $j^* \leftarrow \operatorname{argmin}_{a < j \leq b} \frac{P_{i,j} - P_{i,a}}{L_{i,j} - L_{i,a}}$ ; (break ties arbitrarily)  
6:  $\text{density}(v) \leftarrow \frac{P_{i,j^*} - P_{i,a}}{L_{i,j^*} - L_{i,a}}$ ;  
7:  $\text{index}(v) \leftarrow j^*$ ,  $\text{start}(v) \leftarrow a$ ,  $\text{end}(v) \leftarrow b$ ;  
8:  $\text{right}(v) \leftarrow \text{DT}(m_i, j^*, b)$ ;  
9:  $\text{left}(v) \leftarrow \text{DT}(m_i, a, j^* - 1)$ ;  
10: return *v*;

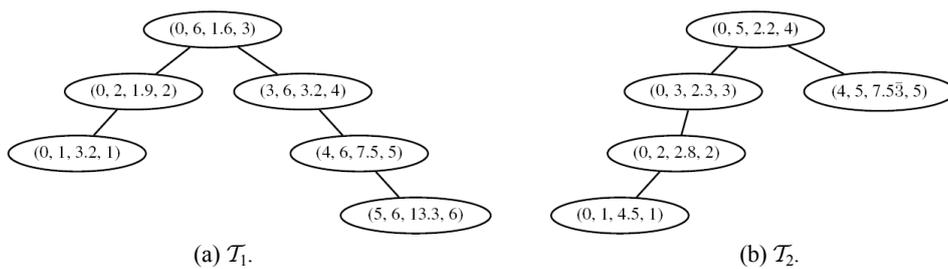


Fig. 3. An example for the decision tree of the example in Fig. 1, where the numbers on a vertex denote its fields *start*, *end*, *density*, and *index* accordingly.

Fig. 3 illustrates an example for the decision tree for the power consumption of a server presented in Fig. 2. Note that, if the power consumption is a convex function of average request rate as the case in Fig. 2, the resulting decision tree is skew. Based on the definition of the decision, we will have the following lemmas.

**Lemma 1** Given a vertex  $v$  in the decision tree  $\mathcal{T}_i$  of server  $m_i$ . For any index  $j$  with  $start(v) < j \leq index(v)$ ,

$$P_{i,j} + density(v)(L_{i,index(v)} - L_{i,j}) \geq P_{i,index(v)}.$$

**Lemma 2** Given a vertex  $v$  in the decision tree  $\mathcal{T}_i$  of server  $m_i$ . For any index  $j$  with  $index(v) < j \leq end(v)$ ,

$$\frac{P_{i,j} - P_{i,index(v)}}{L_{i,j} - L_{i,index(v)}} \geq \frac{P_{i,j} - P_{i,start(v)}}{L_{i,j} - L_{i,start(v)}}.$$

Note that we skip all the proofs in this paper. The detailed proofs are referred to the conference version.

### 3.2 Algorithm Greedy

By adopting the decision tree, we propose a greedy algorithm, denoted as Algorithm Greedy, to decide the activation and operation frequencies of servers. The pseudo-code of Algorithm Greedy is presented in Algorithm 2. The basic idea is to try to accommodate more request rate with the smallest increased power density. We will start from the case that none of the servers is activated at the beginning, and, in each step, we try to activate or accelerate a server at a power-efficient frequency level by using the given decision trees of servers.

Initially, the frequency level  $s_i'$  of server  $m_i$  is set to 0, and the vertex  $v_i$  of server  $m_i$  in the decision tree is set to the root of decision tree  $\mathcal{T}_i$ . The variable  $\ell$  is used to record the amount of total request rate served by executing at frequency level  $s_i'$  on server  $m_i$ , i.e.,  $\ell$  is  $\sum_{i=1}^M L_{i,s_i'}$ . While there exists some non-null  $v_i$ , we enter the loop to activate or accelerate a server  $m_{i^*}$ , in which the increased power density is the smallest in step 8 in Algorithm 2. Clearly, if we increase  $s_{i^*}'$  to  $index(v_{i^*})$ , we will set increase the served request rate by  $L_{i^*,index(v_{i^*})} - L_{i^*,(v_{i^*})}$ , abbreviated by  $q_{i^*}$ . If  $\ell + q_{i^*}$  is less than  $\Lambda$ , we set  $\ell$  to  $\ell + q_{i^*}$ ,  $s_{i^*}'$  to  $index(v_{i^*})$ , and  $v_{i^*}$  to  $right(v_{i^*})$ . Otherwise, we know that the solution  $S'$  by setting server  $m_{i^*}$  at frequency level  $index(v_{i^*})$  and the others servers  $m_i$  at frequency level  $s_i'$  is a feasible solution for the P MEC problem. If  $S'$  is better than the best solution  $S^\dagger$  so far, we replace the best solution  $S^\dagger$  with  $S'$ . Moreover, we also have to set  $v_{i^*}$  to  $left(v_{i^*})$  for the case  $\ell + q_{i^*} \geq \ell$ . The solution  $S^\dagger$  is then returned when every  $v_i$  of server  $m_i$  is null.

The time complexity of Algorithm Greedy is  $O(M \sum_{i=1}^M K_i)$ , since each iteration in the while loop in Algorithm 2 takes  $O(M)$  time and there are at most  $O(\sum_{i=1}^M K_i)$  iterations.

Take the input instance in Fig. 1 for example. Suppose that  $\Lambda$  is 9. For the first iteration in the loop of Algorithm Greedy, we will greedily choose  $i^*$  as 1 by updating  $\ell$  to 3 and  $s_1$  to 3. The second iteration chooses  $i^*$  as 2, and then  $\ell + q_2 = 10 > \Lambda$ . Therefore, we

**Algorithm 2** Greedy

**Input:** average request arrival rate  $\Lambda$ ,  $M$  servers with decision trees  $\mathcal{T}_1, \dots, \mathcal{T}_M$ , QoS requirement  $R$ ;

**Output:** a feasible solution under QoS requirement  $R$ ;

1. **If**  $\Lambda > \sum_{i=1}^M L_{i,K_i}$  **then**
2.     return “no feasible solution”;
3. **end if**
4. let  $S^\dagger$  be the solution by activating all servers at their highest frequency levels;
5. set  $s_i'$  to 0 and  $v_i$  to the root of tree  $\mathcal{T}_i$ ;
6.  $\ell \leftarrow 0$ ;
7. **While** there exists  $v_i \neq \text{null}$  **do**
8.      $i^* \leftarrow \operatorname{argmin}_{1 \leq i \leq M \text{ and } v_i \neq \text{null}} \text{density}(v_i)$ ;
9.      $q_{i^*} \leftarrow L_{i^*, \text{index}(v_{i^*})} - L_{i^*, \text{start}(v_{i^*})}$ ;
10.    **if**  $\ell + q_{i^*} < \Lambda$  **then**
11.      $\ell \leftarrow \ell + q_{i^*}$ ;
12.      $s_{i^*}' \leftarrow \text{index}(v_{i^*})$ ;
13.      $v_{i^*} \leftarrow \text{right}(v_{i^*})$ ;
14.    **else**
15.     let  $S'$  be the solution by setting server  $m_{i^*}$  at frequency level  $\text{index}(v_{i^*})$  and the others servers  $m_i$  at frequency level  $s_i'$ ;
16.     **if**  $S'$  has less power consumption than  $S^\dagger$  **then**
17.          $S^\dagger \leftarrow S'$ ;
18.     **end if**
19.      $v_{i^*} \leftarrow \text{left}(v_{i^*})$ ;
20.    **end if**
21. **end while**
22. return  $S^\dagger$  as the solution;

have a solution  $S'$  with  $(s_1, s_2) = (3, 4)$  and 20.2 power consumption by updating  $v_2$  to the left child of the root of decision tree  $\mathcal{T}_2$ . In the third iteration, we will then choose  $i^*$  as 2 again by setting  $s_2^*$  to 3 and  $\ell$  to 6. For the next iterations, the algorithm goes to the right-most child of decision tree  $\mathcal{T}_1$ , and then  $\ell + q_{i^*} = 9$ , where  $S'$  in this case is with power consumption 35.7. As a result, Algorithm Greedy will return the solution  $S^\dagger$  with  $(s_1, s_2) = (3, 4)$  for this example.

### 3.3 Analysis of Algorithm Greedy

Based on Algorithm Greedy, we have the following lemma for feasible solutions.

**Lemma 3** For any feasible solution, there must be at least one activated server  $m_i$  with frequency level higher than  $s_i'$ .

This lemma comes from the definition of  $s_i'$  in Algorithm Greedy where we can guarantee that  $\sum_{i=1}^M L_{i,s_i'} < \Lambda$  at any moment.

For the optimal solution  $S^*$  (it exists but is unknown), suppose that  $s_i^*$  is the assigned frequency level of server  $m_i$ . Note that if server  $m_i$  is not activated for serving requests in

$S^*$ , then  $s_i^*$  is set as 0. By Lemma 3, there must be at least one faster server  $m_i$  in solution  $S^*$ , in which  $s_i^* > s_i'$ .

We now analyze the power consumption of the derived solution of Algorithm Greedy, compared to the power consumption of solution  $S^*$ . We first decompose optimal solution  $S^*$  by running Algorithm Greedy in the loop between steps 7 and 21 in Algorithm 2 as follows:

- If the condition  $\ell + q_{i^*} < \Lambda$  in step 10 in Algorithm 2 is false and  $s_i^* \geq \text{index}(v_{i^*}) > s_{i^*}'$ , let this server be  $m_{k^*}$  and break the loop before step 19 in Algorithm 2.
- Let  $s_i^b$  ( $v_i^b$ , respectively) be the frequency level  $s_i'$  ( $v_i$ , respectively) before breaking the loop.
- Let  $D^*$  be  $\text{density}(v_{i^*})$ , which is the increased power density when we break the loop.

Let  $S^b$  be the solution by activating server  $m_i$  at frequency levels  $s_i^b$  with average request rate  $L_{i,s_i^b}$ . Moreover, let  $S^\dagger$  be the solution by activating server  $m_{k^*}$  at frequency level  $\text{index}(v_{k^*})$  with average request rate  $L_{k^*,\text{index}(v_{k^*})}$  and the other servers  $m_i$ s at frequency levels  $s_i^b$  with average request rate  $L_{i,s_i^b}$ . For brevity, let  $s_i^\ddagger$  ( $s_i^b$ , respectively) be the frequency level of server  $m_i$  in solution  $S^\ddagger$  ( $S^b$ , respectively).

By the definition of  $S^\ddagger$  and  $S^b$ , we know that

$$\Phi(S^\ddagger) - \Phi(S^b) \leq \Phi(S^*). \tag{1}$$

We use the example in Fig. 2 for demonstrating how to construct  $S_i^b$  and  $S^\ddagger$ . Suppose that  $S^*$  is with  $(s_1^*, s_2^*) = (2, 4)$ . For constructing  $S^\ddagger$  and  $S_i^b$ , we have the situation that  $\ell + q_{i^*} \geq \Lambda$  in the second iteration of the loop, and then we know that  $s_2^* = 4 \geq \text{index}(v_{2^*}) = 4 \geq s_2' = 4$ . Therefore, solution  $S^\ddagger$  is with  $(s_1^\ddagger, s_2^\ddagger) = (3, 4)$  and solution  $S^b$  is with  $(s_1^b, s_2^b) = (3, 0)$ .

**Lemma 4** Solution  $S^\ddagger$  is a feasible solution for the PMEC problem, and the power consumption  $\Phi(S^\ddagger)$  is no less than the power consumption  $\Phi(S^\dagger)$  of the solution  $S^\dagger$  derived from Algorithm Greedy.

Based on Lemma 4, to show the 2-approximation factor of Algorithm Greedy, we will simply show

$$\Phi(S^\ddagger) \leq 2\Phi(S^*). \tag{2}$$

By the feasibility of solution  $S^*$  and infeasibility of solution  $S^b$ , we have the following lemma.

**Lemma 5**  $\sum_{i=1}^M L_{i,s_i^*} \geq \Lambda > \sum_{i=1}^M L_{i,s_i^b}$ .

By comparing solutions  $S^b$  and  $S^*$ , we divide these  $M$  back-end servers into two sets  $K_1$  and  $K_2$ , in which

$$K_1 \leftarrow \{m_i \mid s_i^* < s_i^b\}, \tag{3a}$$

$$K_2 \leftarrow \{m_i \mid s_i^b \leq s_i^*\}. \quad (3b)$$

For sets  $K_1$  and  $K_2$ , the following lemmas show important properties resulting from the decision trees.

**Lemma 6** For any server  $m_i$  in set  $K_1$ , we have

$$P_{i,s_i^b} \leq P_{i,s_i^*} + D^*(L_{i,s_i^b} - L_{i,s_i^*}).$$

**Lemma 7** For any server  $m_i$  in set  $K_2$ , we have

$$P_{i,s_i^*} \geq P_{i,s_i^b} + D^*(L_{i,s_i^*} - L_{i,s_i^b}).$$

Based on the above lemmas, we show the approximation factor of Algorithm Greedy in the following theorem.

**Theorem 1** Algorithm Greedy is a polynomial-time 2-approximation algorithm for the PMEC problem, provided that all  $L_{i,j}$ s on server  $m_i$  at frequency  $f_{i,j}$  are given.

### 3.4 Algorithm E-Greedy

Based on the 2-approximation of Algorithm Greedy, we are going to present an improved greedy algorithm, called Algorithm E-Greedy. The approach is to force a server  $m_i$  to run at a specified frequency  $f_{i,j}$ , and then the rest  $M - 1$  servers are used to serve the rest  $\Lambda - L_{i,j}$  request rate. Among all (at most  $\sum_{i=1}^M K_i$  feasible) solutions under the above restriction, we return the best one. The algorithm is illustrated in Algorithm 3.

#### Algorithm 3 E-Greedy

**Input:** average request arrival rate  $\Lambda$ ,  $M$  servers with decision trees  $\mathcal{T}_1, \dots, \mathcal{T}_M$ , QoS requirement  $R$ ;

**Output:** a feasible solution under QoS requirement  $R$ ;

1. let  $\hat{S}$  be the solution derived from Greedy;
2. **for**  $i \leftarrow 1; i \leq M; i \leftarrow i + 1$  **do**
3.   **for**  $j \leftarrow 1; j \leq K_i; j \leftarrow j + 1$  **do**
4.     let  $S'$  be the solution by activating server  $m_i$  at frequency  $f_{i,j}$  and the other servers by calling Algorithm Greedy with arrival rates  $\Lambda - L_{i,j}$ ;
5.     **if**  $S'$  is feasible and  $\Phi(S') < \Phi(\hat{S})$  **then**
6.        $\hat{S} \leftarrow S'$ ;
7.     **end if**
8.   **end for**
9. **end for**
10. return  $\hat{S}$  as the solution;

**Theorem 2** Algorithm E-Greedy is a polynomial-time 1.5-approximation algorithm for the PMEC problem, provided that all  $L_{i,j}$ s on server  $m_i$  at frequency  $f_{i,j}$  are given.

**Algorithm 4** DP

**Input:**  $\epsilon$ , average request arrival rate  $\Lambda$ ,  $M$  servers, QoS requirement  $R$ , solution of Algorithm Greedy  $\Psi(S^\dagger)$ ;

**Output:** a feasible solution under QoS requirement  $R$ ;

1.  $P_{ij}^b = \left\lfloor \frac{2MP_{i,j}}{\epsilon \Phi(S^\dagger)} \right\rfloor, \forall 1 \leq M, 1 \leq j \leq K_i$ ;
2. **for**  $p \leftarrow 0$ ;  $p \leftarrow p + 1$  **do**
3.   **for**  $i \leftarrow 1$ ;  $i \leq M$ ;  $i \leftarrow i + 1$  **do**
4.     derive  $\Psi_i(p)$  by Eqs. (6) and (7);
5.   **end for**
6.   **if**  $\Psi_M(p) \geq \Lambda$  **then**
7.      $P' \leftarrow p$ ;
8.     back-track the dynamic programming entries from  $\Psi_M(P')$  to find the solution  $S^\epsilon$  contributing to  $\Psi_M(P')$ ;
9.     return solution  $S^\epsilon$ ;
10.  **end for**
11. **end for**

#### 4. DYNAMIC PROGRAMMING

This section provides a fully polynomial-time approximation scheme (FPTAS) for the PMEC problem by applying dynamic programming. An FPTAS for the PMEC problem is a  $(1 + \epsilon)$ -approximation algorithm with polynomial-time complexity by treating  $1/\epsilon$  as an input parameter for any positive  $\epsilon$ . Unless  $\mathcal{NP} = \mathcal{P}$ , fully polynomial-time approximation schemes are the best in terms of polynomial-time approximation algorithms with worst-case guarantees.

Suppose that  $\Phi(S^\dagger)$  is power consumption of the solution derived by applying Algorithm Greedy in section 3. To derive (more precise) approximated solution, we first derived the rounded power consumption  $P_{ij}^b$  as follows:

$$P_{ij}^b = \left\lfloor \frac{2MP_{i,j}}{\epsilon \Phi(S^\dagger)} \right\rfloor, \quad (4)$$

where  $\epsilon$  is a user-specified parameter for the tolerable approximation factor. Then, we perform dynamic programming based on the rounded power consumption. Suppose that  $\Psi_i(p)$  is the maximum average request rate that can be served by using only servers  $m_1, m_2, \dots, m_i$  with rounded power consumption no more than  $p$ . Hence, for brevity, for  $1 \leq i \leq M$ , we define

$$\Psi_i(p) = -\infty \text{ when } p < 0. \quad (5)$$

Suppose that  $j_p$  is the frequency level  $j$  with  $P_{1,j}^b \leq p < P_{1,j+1}^b$  for  $j < K_1$ . Furthermore, when  $p$  is no less than  $K_1$ , let  $j_p$  be  $K_1$ . The boundary condition of  $\Psi_1(p)$  for  $p \geq 0$  is:

$$\Psi_1(p) = L_{i,j_p}. \quad (6)$$

Then, for  $i \geq 2$ , the value of  $\Psi_i(p)$  can be calculated by the following recursive function:

$$\Psi_i(p) = \max_{j=0}^{K_i} \{\Psi_{i-1}(p - P_{ij}^b) + L_{ij}\}. \quad (7)$$

Suppose that  $P'$  is the minimum value with  $\Psi_M(P') \geq \Lambda$ . By back-tracking the dynamic programming table, we can derive a solution  $S^\epsilon$  with  $\sum_{i=1}^M P_{i,s_i^\epsilon}^b = P'$  and  $\sum_{i=1}^M L_{i,s_i^\epsilon} \geq \Lambda$ , in which the frequency level on server  $m_i$  in the solution is  $s_i^\epsilon$ . Algorithm 4 presents the dynamic programming, denoted by Algorithm DP, in which the detail for back-tracking is omitted due to space limitation.

The following theorem shows that the quality of the derived solution  $S^\epsilon$  from the above dynamic programming is not too far away from the optimum, even in the worse case.

**Theorem 3** Deriving  $S^\epsilon$  takes  $O\left(\frac{MK_{\max}}{\epsilon} + MK_{\max}\right)$  time complexity and  $O\left(\frac{M}{\epsilon} + M\right)$  space complexity, where  $K_{\max}$  is  $\max_{i=1,2,\dots,M} K_i$ . For any input instance with feasible solution  $S^*$ ,

$$\Phi(S^\epsilon) \leq (1 + \epsilon)\Phi(S^*).$$

## 5. REMARKS AND EXTENSIONS

We so far assume that the front-end server makes decisions without considering the overhead for activating or deactivating a server or the current state of servers. The consideration of the overhead can be achieved by building multiple decision trees for a server. That is, if server  $m_i$  is currently not activated, activating  $m_i$  takes additional energy and some fixed time for booting. Therefore, we can calculate the overhead in both average power and quality of service requirement. By adding this overhead to  $P_{ij}$  and  $L_{ij}$ , we can construct a new decision tree  $\mathcal{T}_{i,off}$ , which is used by Algorithm Greedy when server  $m_i$  is currently not activated. Similarly, we can also use the same strategy to decide whether we should turn off a server that is currently activated.

Suppose that the derived solution  $S^\dagger$  of Algorithm Greedy provides higher service than the average cluster request rate, *i.e.*,  $\sum_{i=1}^M L_{i,s_i^\dagger} > \Lambda$ . If the average cluster request rate  $\Lambda'$  is with  $\sum_{i=1}^M L_{i,s_i^\dagger} \geq \Lambda' \geq \Lambda$  for the next scheduling interval, Algorithm Greedy derives the same solution  $S^\dagger$ . Therefore, we do not have to change the configuration of the back-end servers for such cases, which can reduce the run-time overhead.

Moreover, if the preference of the system designer is to build a scheduling table for reference, *e.g.*, the approach in [7, 9]. The dynamic programming approach in section 4 can be applied by using different rounding precisions  $\epsilon$  and suitable values of  $\Psi(S^\dagger)$ .

## 6. PERFORMANCE EVALUATION

This section provides performance evaluation for the proposed power management schemes, including Greedy, E-Greedy, and DP. To demonstrate the generality of our approach, three QoS models are applied, *i.e.*, the M/M/1 [12] queuing model, the M/G/1 PS

[13] queuing model, and a soft real-time model similar to [9]. All results in this experiment are mean values of 10 different runs on an Intel Xeon CPU with 3.06 GHz.

### 6.1 Simulation Setting

To evaluate how heterogeneity affects the power consumption, a 4-tuple  $(f_{i,\max}, c_i, \alpha_i, \beta_i)$  is used to compute the power consumption. Variables  $f_{i,\max}$ ,  $c_i$ ,  $\alpha_i$  and  $\beta_i$  are random variables within range  $[1, 4]$ ,  $[20, 80]$ ,  $[200, 400]$  and  $[2, 5]$ , denoting the maximum speed, the constant power consumption, the CPU performance coefficient, and the frequency coefficient of server  $m_i$ , respectively. The power consumption of a deactivated server is assumed to 0. The operating frequencies are discretized into 10 scaling levels by uniform distribution within range  $(0, f_{i,\max})$ . The power consumption for server  $m_i$  at frequency  $f$  is  $P_i(f) = c_i + \beta_i \cdot f^3$ , as adopted in [5, 8, 12] as well. For evaluation, we evaluate cases with 100 and 200 back-end servers, considering three QoS models as follows.

- **M/M/1 Queuing Model** [12]: In this model, the average response time is used as the QoS constraint. As a result,  $L_{i,j}(R) = f_{i,j} \cdot \alpha_i - 1/R$  where  $R$  is the average response time given for the QoS control. Since  $R$  only introduces constant offset, the setting of  $R$  only has minor effect. Therefore, we set  $R$  as 1 in our experiment. For comparison, we also simulate an algorithm extended from the TP-CP-OP algorithm developed in [12] which assumes continuous frequencies. To find a feasible solution for discrete frequencies, the closest upper frequency on each server is used, denoted as R-TP-CP-OP.
- **M/G/1 PS Queuing Model** [13]: In this model, job arrivals to the servers follow a Poisson distribution. The QoS constraint is the mean response time  $E[R] = 0.38$  sec. The resulting  $L_{i,j}(R) = (\mu \cdot r - 1/E[R])(1/f_{i,j})$  where  $1/\mu = 38ms$  is the mean job-execution time and  $r = f_{i,j}/f_{i,\max}$  is the speed ratio of the execution speed to the maximum speed of server  $m_i$ . We do not compare with the approaches in [13] since they focus on homogeneous servers.
- **Soft Real-Time Request (SRR) Model**: The SRR model is similar to the one in [9] and considers only dynamic requests. The execution time of a request follows a normal distribution with mean  $\lambda = 24.5ms$  and deviation  $\delta = 60ms$ . The deadline of a request is  $D = 200ms$ . The QoS constraint is that the probability of all requests that will not miss their deadlines is  $R = 95\%$ . The  $L_{i,j}(R)$  is thus defined as the maximal  $L_{i,j}$  such that the probability of  $L_{i,j} \cdot \lambda < f_{i,j} \cdot D$  is 0.95. In this experiment, we use the inverse cumulative distribution function of the normal distribution  $(L_{i,j} \cdot \lambda, \sqrt{L_{i,j} \cdot \delta^2})$  coupled with a binary search to find  $L_{i,j}(R)$ .

To vary the average request rate, we first compute the maximum tolerable request rate  $\Lambda_{\max}$  of the cluster,  $\sum_{i=1}^M L_{i,K_i}(R)$ . For an input average request rate  $\Lambda$ , the load ratio is defined as  $\Lambda/\Lambda_{\max}$ . A lower bound of the optimal solution is computed as the baseline, which is obtained by adding  $density(v_i^*) \ell \frac{\Lambda - \ell}{q_i^*}$  to the solution when Algorithm 2 hitting the condition  $\ell + q_i^* \geq \Lambda$ . For comparison, all power consumption reported are normalized with respect to the computed lower bounds.

### 6.2 Simulation Results

Fig. 4 illustrates the normalized power consumption of a 400-server cluster for the aforementioned three models. As shown in the figure, our schemes reasonably approximate the lower bounds for all cases. In general, better results are achieved when the load ratio increases. Especially for cases of load ratio larger than 0.4, our schemes derive solutions that consume less than 1% additional power consumption for all three models, compared to the lower bounds. The second observation is that since Algorithm R-TP-CP-OP uses a fixed order of servers according to high workload (80% of the maximal average request rate on servers), the decision for activating servers might be only sub-optimal, as depicted in Fig. 4 (a). Note that, in Figs. 4 (b) and (c), we only compare our results with the computed lower bounds, because the approaches presented in [9, 13] apply exhaustive search and exact method to compute the optimum, respectively, the complexity of which constrains these approaches to clusters with small scales.

We also present the impact of the  $\epsilon$  to Algorithm DP for all three QoS models in Fig. 5 for a cluster with 300 servers. As expected, the smaller  $\epsilon$ , the better approximation is obtained, at the cost of longer computation time. One observation is that the impact of varying the  $\epsilon$  becomes more significant as the load ratio increases. The reason is that with a larger load, the exploration space is larger, and a higher  $\epsilon$  would result in more errors for rounding down the power consumption in Eq. (4). From the figure, we can conclude that 0.05 is a proper value for  $\epsilon$ . Further smaller values are not necessary.

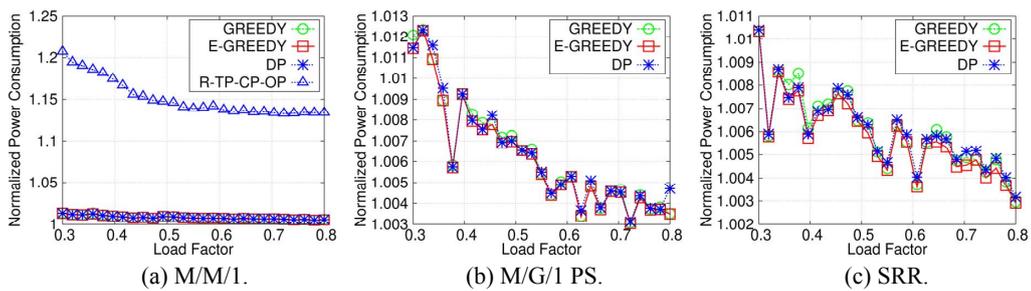


Fig. 4. Normalized power consumption of a 400-server cluster for the three QoS models with  $\epsilon = 0.05$  for algorithm DP.

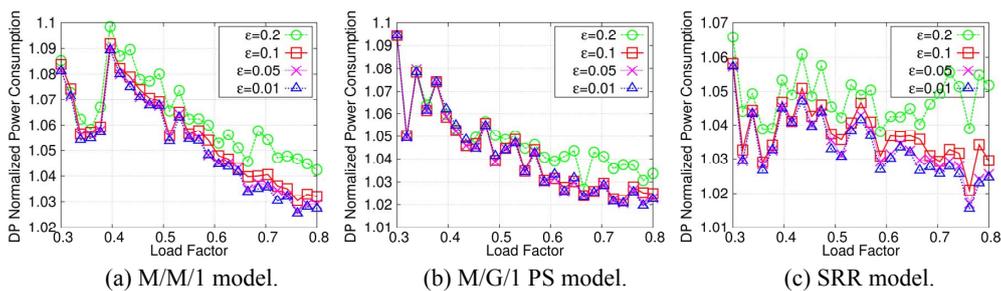


Fig. 5. Varying the  $\epsilon$  for all three models for a 300-server cluster.

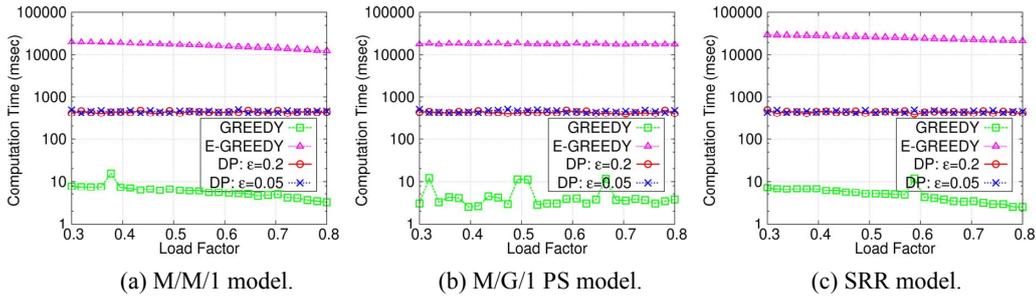


Fig. 6. Computation time for all three models for a 200-server cluster.

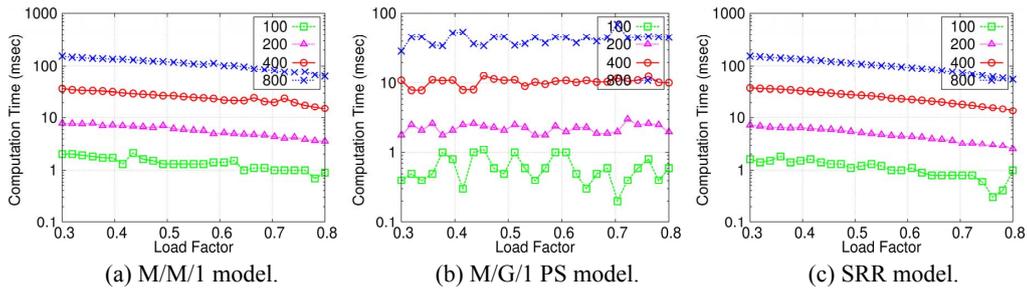


Fig. 7. Computation time of algorithm greedy for all three models for 100/200/400/800 servers.

Fig. 6 depicts the computation time of our algorithms for a 200-server cluster. From the figure, one can observe that the time to compute a solution for this cluster is reasonably fast for all three algorithms. Algorithm GREEDY takes only a few milliseconds while the slowest one, *i.e.*, Algorithm E-GREEDY, is still in the range of seconds. Note that the computation time for Algorithm R-TP-CP-OP is not included, because it takes hours even for the 100-server case. The computation time of Algorithm GREEDY for clusters with up to 800 servers is reported in Fig. 7. As the figure shown, even with 800 servers, the computing time of our algorithm is still around 100 milliseconds. From these figures, we can conclude that our algorithms are also suitable for time-critical large-scale clusters.

## 7. CONCLUSION

This paper explores the power management problem for a heterogeneous cluster to minimize the power consumption while guaranteeing quality of service constraints. We propose approximation algorithms to provide tradeoffs of approximation guarantees in power consumption minimization with time/space complexity. Extensive simulation results with up to 800 servers show that the proposed schemes are effective for minimizing the power consumption large scale server farms or data centers.

## REFERENCES

1. H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, 2001, pp. 95-105.
2. L. A. Barroso, "The price of performance," *Queue*, Vol. 3, 2005, pp. 48-53.
3. P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony, "The case for power management in web servers," *Power Aware Computing*, 2002, pp. 261-289.
4. J. S. Chase, D. C. Anderson, P. N. Thakar, A. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centres," in *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, 2001, pp. 103-116.
5. J. J. Chen, H. R. Hsu, and T. W. Kuo, "Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems," in *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2006, pp. 408-417.
6. J. J. Chen, K. Huang, and L. Thiele, "Power management schemes for heterogeneous clusters under quality of service requirements streams," in *Proceedings of the 26th ACM Symposium on Applied Computing*, 2011, pp. 546-553.
7. R. Guerra, J. Leite, and G. Fohler, "Attaining soft real-time constraint and energy-efficiency in web servers," in *Proceedings of ACM Symposium on Applied Computing*, 2008, pp. 2085-2089.
8. R. Jejurikar, C. Pereira, and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," in *Proceedings of the 41st Annual Design Automation Conference*, 2004, pp. 275-280.
9. C. Rusu, A. Ferreira, C. Scordino, and A. Watson, "Energy-efficient real-time heterogeneous server clusters," in *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2006, pp. 418-428.
10. V. Sharma, A. Thomas, T. F. Abdelzaher, K. Skadron, and Z. Lu, "Power-aware QoS management in web servers," in *Proceedings of the 24th IEEE International Real-Time Systems Symposium*, 2003, pp. 63-72.
11. V. V. Vazirani, *Approximation Algorithms*, Springer, 2001.
12. L. Wang and Y. Lu, "Efficient power management of heterogeneous soft real-time clusters," in *Proceedings of the 29th IEEE Real-Time Systems Symposium*, 2008, pp. 323-332.
13. A. Wierman, L. L. H. Andrew, and A. Tang, "Power-aware speed scaling in processor sharing systems," in *Proceedings of IEEE INFOCOM*, 2009, pp. 2007-2015.
14. R. Xu, D. Zhu, C. Rusu, R. Melhem, and D. Mossé, "Energy-efficient policies for embedded clusters," in *Proceedings of ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, 2005, pp. 1-10.
15. F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, 1995, pp. 374-382.



**Jian-Jia Chen** joined Department of Informatics at Karlsruhe Institute of Technology in Germany as a Juniorprofessor for Institute for Process Control and Robotics in 2010. He received his Ph.D. degree from Department of Computer Science and Information Engineering, National Taiwan University, Taiwan in 2006. He received his B.S. degree from the Department of Chemistry at National Taiwan University 2001. He was a postdoc researcher at Computer Engineering and Networks Laboratory in ETH Zurich, Switzerland between Jan. 2008 and April 2010. His research interests include real-time systems, embedded systems, energy-efficient scheduling, power-aware designs, temperature-aware scheduling, and distributed computing. He received Best Paper Awards from ACM Symposium on Applied Computing in 2009 and IEEE International Conference on Embedded and Real-Time Computing Systems and Applications in 2005.



**Kai Huang** is currently a research associate in the Technical University of Munich in Germany. He was a research group leader in the fortiss GmbH Germany before joining TUM. He received his Ph.D. degree in the Computer Engineering and Networks Laboratory of ETH Zurich, Switzerland, in 2010. He received B.Sc. degree in Computer Science at Fudan University, China, in 1999 and M.Sc. degree in Computer Science at Leiden University, The Netherlands, in 2005. His research interests include methods and techniques for the analysis, design, and optimization of embedded systems. He received Chinese Government Award for Outstanding Self-Financed Students Abroad 2010, Best Paper Awards from International Symposium on Systems, Architectures, Modeling and Simulation, and General Chairs' Recognition Award for Interactive Papers in the IEEE Conference on Decision and Control in 2009.



**Lothar Thiele** joined the Swiss Federal Institute of Technology, Zurich (ETH) as a full Professor of Computer Engineering in 1994, where he currently leads the Computer Engineering and Networks Laboratory. His research interests include models, methods, and software tools for the design of embedded systems, embedded software, and bioinspired optimization techniques. He received the 1986 Dissertation Award of the Technical University of Munich, the 1987 Outstanding Young Author Award of the IEEE Circuits and Systems Society, the 1988 IEEE Browder J. Thompson Memorial Award, and the 2000-2001 IBM Faculty Partnership Award. In 2004, he joined the German Academy of Natural Scientists Leopoldina. He received the 2005 Honorary Blaise Pascal Chair of Leiden University, The Netherlands. Since 2009 he is a member of the Foundation Board of Hasler Foundation, Switzerland. Since 2010, he is a member of the Academia Europaea.