[1]

Autonomic Power & Performance Management for Large-Scale Data Centers

Bithika Khargharia[1], Salim Hariri[1], Ferenc Szidarovszky[1], Manal Houri[2], Hesham El-Rewini[2], Samee Ullah Khan[3], Ishfaq Ahmad[3], and Mazin S. Yousif[4]

Email: [1]{bithikak, hariri,}@ece.arizona.edu,[1]szidar@sie.arizona.edu, [2]{mhouri, rewini}@engr.smu.edu, [3]{sakhan,iahmad}@cse.uta.edu, [4] mazin.s.yousif@intel.com

Abstract

With the rapid growth of servers and applications spurred by the Internet, the power consumption of servers has become critically important and must be efficiently managed. High energy consumption also translates into excessive heat dissipation which in turn, increases cooling costs and causes servers to become more prone to failure. This paper presents a theoretical and experimental framework and general methodology for hierarchical autonomic power & performance management in high performance distributed data centers. We optimize for power & performance (performance/watt) at each level of the hierarchy while maintaining scalability. We adopt mathematically-rigorous optimization approach to provide the application with the required amount of memory at runtime. This enables us to transition the unused memory capacity to a low power state. Our experimental results show a maximum performance/watt improvement of 88.48% compared to traditional techniques. We also present preliminary results of using Game Theory to optimize performance/watt at the cluster level of a data center. Our cooperative technique reduces the power consumption by 65% when compared to traditional techniques (min-min heuristic).

1. Introduction

Automatic modeling and online analysis of multiple objectives and constraints such as power consumption and performance of large-scale distributed data centers is a challenging research problem due to the dynamic and heterogeneous nature of workloads & applications, continuous changes in topology, the variety of services & software modules being offered and deployed, and the extreme complexity & dynamism of their computational workloads. In order to develop an effective autonomic control and management system for power & performance management, it becomes highly essential for the system to have the functionality of online monitoring; adaptive modeling and analysis tailored for real-time processing and proactive management mechanisms. As part of this work, we develop innovative management techniques that address the following research challenges:

1. How do we efficiently and accurately model power and energy consumption from a system level perspective that involves the complex interactions of different classes of devices such as processor, memory, network and I/O?

2. How can we predict in real-time the behavior of system resources and their power consumptions as workloads change dynamically by orders of magnitude within a day or a week?

3. How to design efficient and self-adjusting optimization mechanisms that can continuously and endlessly learn, execute, monitor, and improve themselves in meeting the collective objectives of power & performance improvement?

The development of the models and solution methods consist of the following steps: First, a mixed programming model is developed to minimize the power consumption while maintaining performance requirements of a memory system which is at the lower-most layer (component-level) in a data center. Based on this model a game model is constructed which takes the competition of the different systems/platforms within the data-center taking into account the limited available electric power budget. Non-cooperative and cooperative solutions are determined and compared in order to find the most satisfying outcome for the entire system. In the next step each element of each data center will be considered as an agent in an agent-based gaming approach. Using simulation and sensitivity analysis, the most satisfying strategies of the agents will be

[1] 1-4244-0910-1/07/$20.00 2007 C IEEE

determined with respect to the overall performance of the entire system. In addition to developing a practical methodology, several theoretical issues have to be examined such as existence and uniqueness of non-cooperative Nash equilibrium and cooperative solution concepts. Since different solution concepts would lead to different outcomes, one objective of the proposed research going forward would be to find the solution concepts which fit best the particular problems under investigation.

The rest of the paper is organized as follows. In Section 2, we present a brief overview of the main methods used to address power issues in computing systems. In Section 3 we introduce the hierarchical framework and discuss how we achieve autonomic *performance/watt* management for a memory system at the component-level and for a cluster of machines at the system-level (using Game Theory). In Section 4 we discuss experimental results and conclude in Section 5.

2. Background and Related Work

Most power management techniques exploit the over-provisioning of components, devices or platforms for power savings. This technique also known as Dynamic Power Management (DPM) is extensively used for reducing power dissipation in systems by slowing or shutting-down components when they are idle or underutilized.

Most DPM techniques utilize power management features supported by the hardware. For example, frequency scaling, clock throttling, and dynamic voltage scaling (DVS) are three processor power management techniques [1] that are extensively utilized by DPM. [2] for example, extends the operating system's power manager by an *adaptive power manager (APM)* that uses the processor's DVS capabilities to reduce or increase the CPU frequency thereby minimizing the overall energy consumption. [3] combines the DVS technique at the processor-level together with a turn on/off technique at the cluster-level to achieve high power savings while maintaining the response time. [4] introduces a scheme to concentrate the workload on a limited number of servers in a cluster such that the rest of the servers can remain switched-off. for a longer time. [5] proposes power-aware QoS management in web servers where the algorithms reduce processor voltage and frequency as much as possible but not enough to cause per-class response time constraint violations. Other techniques use a utilization bound for schedulability of a-periodic tasks [6] [7] to maintain the timeliness of processed jobs while conserving power. Similarly, for dynamic memory power management [8] uses multiple power modes of RDRAM memory and dynamically turns off memory chips with power-aware page allocation in operating system.

Researchers have also explored joint power management techniques that involve techniques to jointly maintain power consumption of multiple system components such as the memory and the hard disk. For example, [9] has used the relationship between memory and disk (smaller the memory size, the higher the page misses and the higher the disk accesses) to achieve power savings by proactively changing disk I/O by expanding or contracting the size of the memory depending on the workload. [10] addresses base power consumption for web servers by using a *power-shifting* technique that dynamically distributes power among components using work-load sensitive polices.

Most techniques for dynamic power management justify the need to consider components in isolation. For example, [11] makes the case that processor is the major power consuming factor in servers. Following this thread [12] presents a request-batching scheme where jobs are forwarded to the processor in a batch after certain time such that the response time constraint is met for all classes of customers. This lets the processor be in a lower power state for a longer period of time. [13] on the other hand states that data center storage devices can consume over 25% power. This has spawned research in memory power management. However there has not been much effort to exploit these existing techniques for different classes of resources (processor, memory, cache, disk, network card etc) in a unified framework from a whole system perspective. While the closest to combining device power models to build a whole system has been presented in [14], our approach aims at building a general framework for autonomic power and performance management where we bring together and exploit existing device power management techniques from a whole system's perspective. We introduce a hierarchical framework for power management that starts at individual devices within a server to server clusters and cluster of clusters enabling power management at every level of the hierarchy of a data center with the solutions being more and more refined as we travel down the hierarchy from cluster of heterogeneous servers to independent devices. The closest to our approach is the work done by [15] that solves the problem of hierarchical power management for an energy managed computer (EMC) system with self-power managed components while exploiting application level scheduling.

While most power management techniques are either heuristic-based approaches [16] or stochastic optimization techniques [17] we explore Game Theory to seek radically fast and efficient solutions compared with the traditional approaches (e.g., heuristics, genetic algorithms, linear and dynamic programming, branch-and-bound etc) that are either impractical or are applicable only in a static fashion. With game theoretical techniques the solution may not be globally optimal in the *traditional* sense, but would be optimal in *given* circumstances [18]. This fits perfectly in

the context of large-scale distributed data centers since we find the best solution given the state of the system; we do not acquire global solutions, which are meaningless given the dynamic nature of distributed systems. With the aid of Game Theory we can use lower level information to dynamically tune the high-level management policies freeing the need to execute complex algorithms [19].

3. Autonomic Management Framework (AMF): Hierarchical Power and Performance Management

We define an Autonomic Management System (AMS) as a system augmented with intelligence to manage and maintain itself under changing circumstances impacted by the internal or external environment. In previous work we have laid the foundation for an Autonomic Computing System [20].

We apply our general *AMF* for power and performance management of a network of geographically dispersed Internet data centers with Autonomic Managers (*AM*s) at each level going from the inter-data center level to the intra-data center level, cluster-level, server-level and finally the component-level within a single server. This is shown in Figure 1. The *MS* (Managed System) changes depending on the domain being managed. For example the *MS* could be a network of data centers, a single data center, a Front End server cluster, a Web Server or a Memory System within a Web Server. The *AM*s may share a distributed or hierarchical management relationship (based on the *MS* and its parent if any). For example, in Figure 1 an *AM* managing a whole data center may share a distributed relationship with another *AM* at the data center level such that they compete for power budget. However, with a single data center the *AM*s share a hierarchical relationship where it tries to work within the allocated power budget while maintaining the application-level performance. As shown in Figure 1, within a data center the *MS* can be logically organized into three distinguishable hierarchies *i)* cluster level, where the whole data center is modeled as collection of networked clusters *ii)* server level, where each cluster is modeled as a collection of networked servers and *iii)* device level, where each server is modeled as a collection of networked devices. The top-level *AM* deals with the incoming data center workload. It consists of three sub-components. The *Profiler* profiles the power and performance characteristics of the incoming workload based on the current data center system configuration. It performs the statistical analysis and forwards the results of the analysis to the *Analyzer*. The *Analyzer* is responsible for predicting the power budget for the data center for the next observation interval based on the statistical analysis forwarded by *Profiler* as well as history-based knowledge. The *Regulator* regulates the distribution of incoming workload onto the system such that the system can meet the power budget determined by the *Analyzer*. Each lower-level *AM* in the hierarchy performs similar tasks to maintain the system within the allocated power budget (from the top-level) and still deliver the required performance.

In this work, we model the *MS* (at any hierarchy) as a set of states and transitions. Each state is associated with power consumption and performance values. A transition takes *MS* from one state to another. It is the task of the *AM* to enable MS to transition to a state where power consumption is minimal without violating any performance constraints. Our autonomic management approach relies on *MS* states at each level of the hierarchy to proactively manage power consumption and also maintain the *QoS* requirements.

In what follows, we first discuss a specific scenario of power and performance management at the component/device level for an FBDIMM memory system using optimization technique. We then discuss system-level (cluster-level) power and performance management using Game Theory.

3.1. AMF: Power and Performance Component Manager

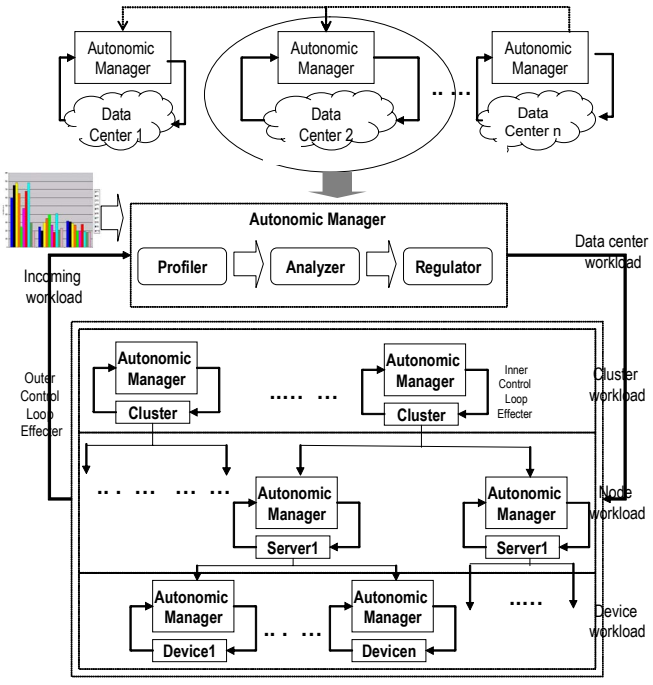

Figure 1: Hierarchical Power and Performance Management

We apply the autonomic computing paradigm to architect an intelligent memory controller (*MC*) that continuously reconfigures and scales the memory system

for maintaining power and performance. The objective of the *MC* is to always maintain the size and configuration of the memory system in a *state* where power consumed is minimal and the system still meets the threshold values for the performance parameters. Scaling the memory size to the minimum would give huge savings in power but may impact performance by increasing the *miss* ratio as well as the delay experienced by a single memory *access time*. Hence, the task of the *MC* is to allocate as much memory as is required by the application and the unused amount of memory can then be transitioned to one of the low-power states as supported by an FBDIMM (Figure 2). We can estimate the application's memory requirement at runtime by measuring the application's current *heap usage* and the total number of memory accesses going to each Rank. Based on the monitored values, at the end of each time epoch the MC maintains the system at the *maximum performance/watt* by determining 1) what is the minimum number of memory Ranks to be maintained in an *active* state? 2) Which Ranks should be selected to be active?

We formulate the MC decision-making process as an optimization problem where we index time into equidistant epochs of value t_{obs}. The MC searches for an optimal solution at the beginning of each epoch. Let us consider a state transition from state S_j to state S_k where S_j has n_j Ranks ($Rank_0$ *to* $Rank_j$) and S_k has n_k Ranks ($Rank_0$ *to* $Rank_k$). The data migration process during this state transition involves a Rank pair, one from the source pool of Ranks in state S_j and the other from the destination pool of Ranks in state S_k. In what follow, we discuss how to determine the target state S_k among all possible states. Data is then migrated from a source Rank to a destination Rank.

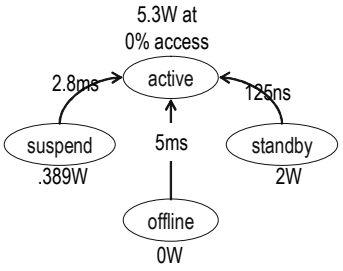

Figure 2: FBDIMM Power States [21]

3.1.1. Formulating the Optimization Problem. At the beginning of time epoch *i* the MC searches for the state where the sum of the transition energy consumed ($c_{jk} * t_{trans_{jk}}$) and the energy consumed in the target state ($p * n_k * t_{obs}$) by the memory system is the smallest given that in the target state S_k, the system can meet all the constraints. The objective function is given by Minimize energy for interval i,

$$e_i = \sum_{k=0}^{N} (c_{jk} * t_{trans_{jk}} + p * n_k * t_{obs}) * x_{jk}$$

Such that,

$$n_k * size / Rank >= N_{opt} * pageSize$$

$$Max(\bigvee_{ch:A,C} chBW_{ch}) <= threshold\_chBW$$

$$Min(\bigvee_{rank:0}^{n_n/2} arrTime_{rank}) >= threshold\_arrTime_{rank}$$

$$\sum_{k=0}^{n} x_{jk} = 1$$

$$\vee\ x_{jk} = 0 \mid 1$$

Where,

N:	Maximum Ranks in the system
n_k:	Total Ranks in state S_k
p:	Power consumed per Rank
t_{obs}:	Unit of time epoch
c_{jk}:	Power consumed in transition
$t_{trans_{jk}}$:	Time taken to transition
x_{jk}:	Decision variable for transition S_j to S_k.
$chBW_{ch}$, $arrTime_{Rank}$	: Ch BW and inter-arrival time in state S_k
$threshold\_chBW$, $threshold\_arrTime_{Rank}$	:Threshold values
$pageSize$	: Size of a single page (4KB for our system)
N_{opt}:	:Optimal pages for maximum hit ratio [22]
$Size/Rank$	: 512 MB for our system.

The first constraint states that the target state should have enough memory to hold all the N_{opt} *pages*. The second constraint states that in the target state, the maximum of the *percent channel BW* on a channel should be smaller than the threshold value set for the channel BW. Ideally it can be the theoretical upper limit. The third constraint states that in the target state, the minimum *request inter-arrival* among all the active Ranks should be larger than the threshold value set for the Rank where the threshold value is a percentage of the *access time*. This is to be experimentally determined. The fourth constraint states that the optimization problem leads to one and only one solution. The decision variable corresponding to that is 1 the rest are 0. The fifth constraint states that the decision variable is a 0-1 integer.

Evaluation of Migration Time, $t_{trans_{jk}}$ *and Energy* c_{jk} .

During migration, the MC stalls all memory access requests and consequently, the time for data migration is a sum of the data migration time (read time, transfer time and write time) and the time needed to make power transition. Given that, fraction of a page per Rank is given by

$$ppr = \frac{[n_k / 2]}{[pageSize/\ CLSize]}$$

The migration time per Rank (MTR) pair ($Rank_j$, $Rank_k$) is given by

$$MTR = \frac{ppr * N_{opt} * pgSize}{CLSize_{Rank_j} / 2 * tRead * 1024} + \frac{ppr * N_{opt} * pgSize}{CLSize_{Rankk} / 2 * tWrite * 1024} + \frac{ppr * N_{opt} * pgSize}{MaxThPut_{Ch_{Rank_j}}} + t_{act_{jk}}$$

Power State Transition Overhead. Figure 2 gives the power state transition overhead per DIMM. Migration Energy is the sum of the power consumed by two sources,

$$c_{jk} = n_k * p_{trans} + * t_{trans_{jk}} * p_{MC}$$

Where p_{trans}: Transition power consumed by a Rank
p_{MC}: Power consumed in buffers during data migration.

3.2. AMF: System-level Autonomic Power and Performance Management using Game Theory

In this Section we discuss how we apply Game Theory for system-level power/performance management.

3.2.1. Use Case: Power and Performance Optimization of a Computing Cluster. Consider a cluster consisting of a set of machines, $M = \{m_1, m_2, \ldots, m_m\}$. Each machine is equipped with a DVS module. Each machine is described by the following characteristics:

1. The frequency of the CPU, f_j, in cycles per unit time. With the help of DVS, f_j can vary from f_j^{min} to f_j^{max}, where $0 < f_j^{min} < f_j^{max}$. CPU speed, S_j, is simply the inverse of the frequency.
2. The specific machine architecture, $A(m_j)$ for machine m_j. The architecture includes the type of CPU (Intel, AMD), bus types and speeds in GHz, I/O, and Memory in Bytes.

Consider a meta-task, $T = \{t_1, t_2, \ldots, t_n\}$, where t_i is a task. Each task is characterized by:

1. *Computational cycles* c_i that it needs to complete.
2. *Machine architecture*, $A(t_i)$, that it needs to complete its execution.
3. The *deadline*, d_i, before it has to complete its execution.

It is obvious that the meta-task, T, also has a deadline, D, which is met if and only if the deadlines of all its tasks are met. Now suppose we are given a cluster and a meta-task, T, and we are required to map T on the cluster such that all the characteristics of the tasks and the deadline constraint of T are fulfilled. We term this fulfillment as a *feasible task to machine mapping*. A *feasible task to machine mapping* happens when:

1. Each task $t_i \in T$ can be mapped to at least one m_j subject to the fulfillment of all the constraints associated with each task: a) Computational cycles b) Architecture c) Deadline.
2. The deadline constraint of T is also satisfied, such that no task finishes after its deadline.

The number of cycles required by t_i to execute on M_j is assumed to be a finite positive number, denoted by c_{ij}, and is known *a priori*. The execution time of t_i under a constant speed S_{ij}, given in cycles per second is $t_{ij} = c_{ij}/S_{ij}$.

A task, t_i, when executed on machine m_j draws, p_{ij} amount of power. Lowering the power, will lower the CPU frequency and consequently will decrease the speed of the CPU, and hence cause t_i to possibly miss its deadline. For simplicity assume that the overhead of switching the CPU frequency is minimal and hence ignored. The architectural requirements of each task is recorded as a tuple with each element bearing a specific requirement, such as what does the task require for its execution, the architectural affinity matching of the task to the machine. We assume that the mapping of architectural requirements is a Boolean operation i.e. the architectural mapping is only fulfilled when all of the architectural constraints are satisfied, otherwise not.

3.2.2. Management Solution Using Game Theory. Game theory has two major branches namely, cooperative and non-cooperative games. Based on these two types of games we classify our solutions as

1. Static, centralized & controlled approach using cooperative games.
2. Dynamic, distributed & autonomous approach using non-cooperative games.

The system-level *AM* will decide on the mapping of tasks onto each and every machine. The *AM* has all the necessary information to execute such a decision. The necessary information about each machine includes: (a) Load of the CPU, *i.e.*, what tasks are currently scheduled on the CPU; (b) The range of CPU clock frequency, $[f_j^{min}, f_j^{max}]$; (c) The architecture of the machine.

To fully utilize the computing cluster, the *AM* has to arrive at a decision (mapping of tasks onto machines) in a controlled environment, which is pareto-optimal, beneficial and fair to all of the machines. This can be achieved very efficiently using cooperative games. The *AM* will simulate a cooperative game among the machines. The goal of such a game would be to find task to machine mapping such that the system as a whole can benefit. Although, some machines may not be satisfied with their particular allocation of tasks, but they overlook that since the goal is for the system to prosper. The prosperity of the system is measured by its ability to execute tasks within their deadlines and also minimize the power used by the machines.

Let us consider a scenario where an *AM* manages a data center cluster. It optimizes power consumption while maintaining a high task turn-around time often completing the tasks before the deadline. Given a computing cluster and a meta-task T the *AM* has to find the task to machine mapping where the total power utilized is minimized such that the makespan of the meta-task, T, is minimized.

Mathematically, the above scenario can be stated as:

$$\min \sum_{i=1}^{n} \sum_{j=i}^{m} p_{ij} x_{ij} \text{ such that } \min \max_{1 \le j \le m} \sum_{i=1}^{n} t_{ij} x_{ij} \text{ subject to}$$

1. $x_{ij} \in \{0,1\}, i = 1,2,\ldots,n; j = 1,2,\ldots m.$
2. if $t_i \to m_j, \forall i, \forall j,$ such that $A(t_i) = A(m_j),$ then $x_{ij} = 1$
3. $t_{ij} x_{ij} \le d_i, \forall i, \forall j, x_{ij} = 1$
4. $(t_{ij} x_{ij} \le d_i) \in \{0,1\}$

5. $\prod_{i=1}^{n}(t_{ij}x_{ij} \le d_i) = 1, \forall i, \forall j, x_{ij} = 1$

Constraint 1 is the mapping constraint, such that x_{ij} =1, if a task, t_i, is mapped to machine, m_j. Constraint 2 expands on this mapping in conjunction to the architectural requirements and it states that a mapping can only exists if the architecture is mapped. Constraint 3 relates to the fulfillment of the deadline of each task and constraint 4 tells us about the Boolean relationship between the deadline and the actual time of execution of the tasks. Constraints 5 relates to the deadline constraints of the meta-task, which will hold if and only if all the deadlines of the tasks, t_i, i =1, 2, …n, are satisfied.

4. Experimental Results

4.1. Memory Power and Performance Management

We evaluate the performance of the autonomic component manager on a server with four different memory configurations running SPEC JBB 2005 benchmark [23]. For each configuration we measured the *percent active pages*, *percent channel BW utilization* and *request inter-arrival time* at the end of each additional *warehouse* launched by the SPEC JBB benchmark. We used the *percent active pages* and current memory configuration to compute the *migration overhead* in going to a target memory configuration. Notice that our performance management scheme individually maintains the applications *hit ratio* and the system's memory *access time.* However that maintains the application-level performance as can be seen in Figure 4. Our power and performance management scheme gives the maximum *performance/watt* for the platform during the entire run of SPEC JBB.

4.1.1. Analysis of Performance/Watt. Figure 4 gives the performance/watt of a power and performance managed memory system (Scheme 2) as we discussed in Section 3.1 compared to one without (Scheme 1). Clearly Scheme 2 outperforms Scheme 1 at all points during the run of SPEC JBB.

In summary, the dynamic data migration yields energy savings of about 48.8 % (26.7 kJ) compared to traditional techniques (without migration) measured at a meager 4.5%. Furthermore, the transition overhead is about 18.6 ms which leads to energy savings of 1.44 kJ per millisecond of transition overhead time. Consequently, our scheme gives additional savings of another 50% (2 GB) for an overhead of 16.9 ms during zero workload and gives the maximum *performance/watt* at all points during the run of the application with a maximum improvement of 88.48%.

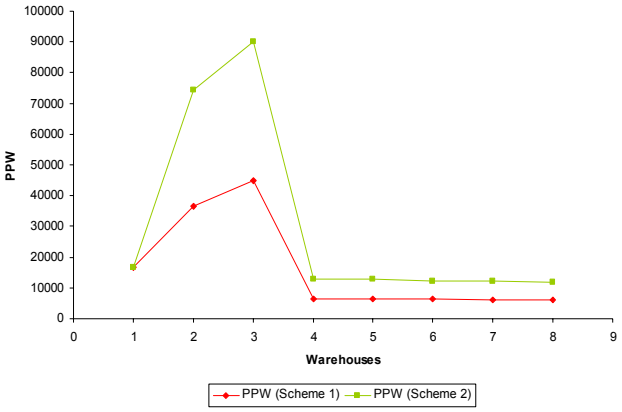

Figure 4: Performance/Watt Comparison

4.2. Game Theory

Based on the size of the problems, the experiments were divided in two parts. For small size problems, we used an Integer Linear Programming tool called LINDO [24]. LINDO is useful to obtain optimal solutions, provided the problem size is relatively small. Hence for small problem sizes, the performance of the cooperative game is compared against 1) the optimal solution using LINDO and 2) the min-min heuristic [25]. The min-min heuristic does not consider power as an optimization constraint; however, it is very effective for the optimization of the makespan. Thus, the comparison provides us with a wide range of results. On one extreme we have the optimal algorithm, on the other a technique which focuses on the optimization of makespan. For large size problems, it becomes impractical to compute the optimal solution by LINDO. Hence, we only consider comparisons against the min-min heuristic.

The system heterogeneity is captured by the distribution of the number of CPU cycles, c_{ij}, required by different t_is on different m_js. Let C denote the matrix composed by c_{ij}, where i = 1, 2, …, n and j = 1, 2, …,m. The C matrix was generated using a Gamma distribution based method [26]. The Gamma distribution method requires input of mean value along the task axis, w, which we set to 200. d_i, the deadline task t_i was generated based on the C matrix. Let w_i be the largest value among the elements in the i-th row of C and let $X = n/m$, where n is the number of tasks and m is the number of m_js. d_i is calculated as $K \times w_i \times X$, where K is a pre-specified positive value for adjusting the relative deadlines of tasks.

For small size problems, the number of m_js was fixed at 5, while the number of tasks, t_i, varied from 20 to 40. The number of DVS levels per m_j was set to 4. The maximum CPU speed of each m_j was set to 1.0 and the minimum speed was set to 0.25. It is assumed that other CPU speeds are distributed uniformly between the maximum and minimum speeds. Therefore, the other two

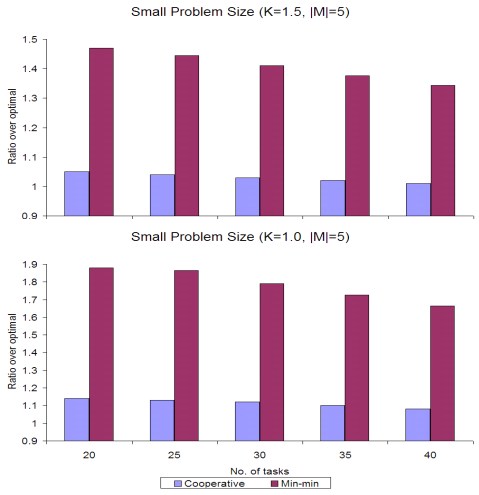

Figure 5: Comparison with Optimal (K=1.5, 1.0)

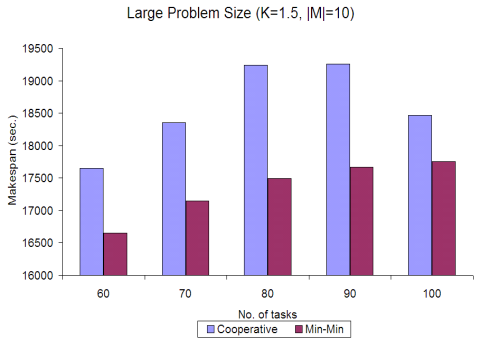

Figure 6(a): Power Savings (K=1.5)

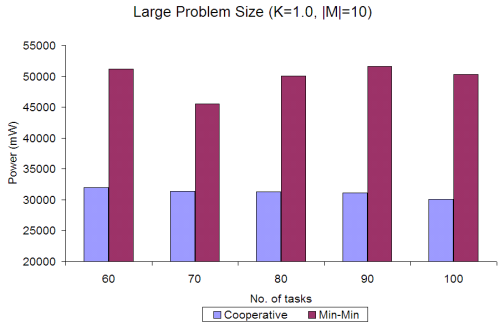

Figure 6(b): Power Savings (K=1.0)

levels of CPU speed are set to 0.75 and 0.5. For large size problems, the number of m_js was fixed at 10, while the number of tasks varied from 50 to 100. The number of DVS levels per m_j was set to 8. Other parameters were the same as those for small size problems. The experimental results for small size problems when K is set to 1.5 and 1.0 are shown in Figure 5. It shows the ratio of the overall system power consumption obtained from the two techniques and the optimal. Each case was run several times so as to gain enough confidence in the results reported here. The plots clearly show that the cooperative technique performs extremely well and achieves a performance level of 10%-15% of the optimal, when the relative deadline factor K was set at very tight bound 1.0.

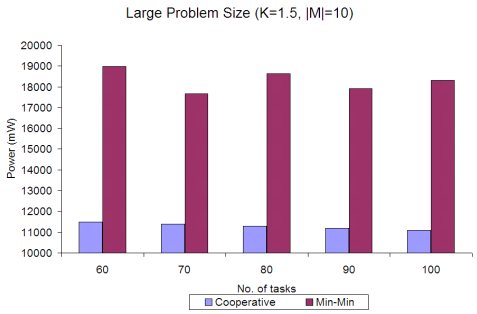

Figure 7(a): Makespan Comparison (K=1.5)

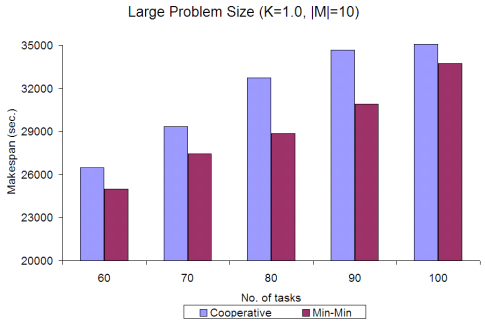

Figure 7(b): Makespan Comparison (K=1.0)

Figures 6(a) and 6(b) show the relative performance of the cooperative technique and the min-min heuristic for large size problems when K = 1.5 and 1.0, respectively. From these plots, we can clearly see the savings in power. The min-min heuristic maps tasks onto machines which are running on full throttle whereas the cooperative technique is continuously optimizing power. Finally, we compare (for the same setup) the makespan found by the cooperative and min-min heuristic. It can be seen that the cooperative technique brings the power consumption down to 65% of the min-min heuristic, identifies a task to machine mapping that produces a makespan that is within 10% of min-min. The results are shown in Figure 7.

5. Conclusions

In this paper, we presented a theoretical and experimental framework to optimize power and performance at runtime for e-business data centers. We presented *performance/watt* management of a memory system using optimization technique. Our scheme gives a maximum *performance/watt* improvement of 88.48%. We

also develop a Game Theory model for *performance/watt* optimization in a data center server cluster. Our cooperative technique brings the power consumption down to 65% of traditional techniques.

We are currently performing comprehensive modeling and analysis of large-scale e-business data centers. We are also analyzing the comparative performance of stochastic, predictive and heuristic techniques on power and performance management applied to the data center domain.

References

1. A. Miyoshi , C. Lefurgy , E. Van Hensbergen , R. Rajamony , R. Rajkumar, Critical power slope: understanding the runtime effects of frequency scaling, Proceedings of the 16th international conference on Supercomputing, June 22-26, 2002, New York, New York, USA
2. Application Specific and Automatic Power Management Based on Whole Program Analysis, http://cslab.snu.ac.kr/~egger/apm/final-report.pdf, August 20th, 2004
3. E. N. Elnozahy, M. Kistler, R. Rajamony. Energy-Efficient Server Clusters. In Proceedings of the 2nd Workshop on Power-Aware Computing Systems, February 2002.
4. E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath, "Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems," *Proceedings of the Workshop on Compilers and Operating Systems for Low Power*, September 2001; *Technical Report DCS-TR-440*, Department of Computer Science, Rutgers University, New Brunswick, NJ, May 2001.
5. V. Sharma , A. Thomas , T. Abdelzaher , K. Skadron , Z. Lu, Power-aware QoS Management in Web Servers, Proceedings of the 24th IEEE International Real-Time Systems Symposium, p.63, December 03-05, 2003
6. T. Abdelzaher and V. Sharma. "A synthetic utilization bound for aperiodic tasks with resource requirements". In *Euromicro Conference on Real Time Systems*, Porto, Portugal, July 2003.
7. T. F. Abdelzaher and C. Lu. "Schedulability analysis and utilization bounds for highly scalable real-time services". In *IEEE Real-Time Technology and Applications Symposium*, TaiPei, Taiwan, June 2001.
8. A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis. *Power Aware Page Allocation*. In ASPLOS, pages 105{116, 2000.
9. Cai, L., Yung L., *Joint Power Management of Memory and Disk*, IEEE, 2005
10. W. Felter, K. Rajamani, T. Keller (IBM ARL), and C. Rusu, A Performance-Conserving Approach for Reducing Peak Power Consumption in Server Systems, *ACM International Conference on Supercomputing (ICS), Cambridge, MA, June 2005*
11. P. Bohrer, E. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony. *The case for power management in web servers. Power Aware Computing*, 2002. Klewer Academic Publishers.
12. M. Elnozahy, M. Kistler, and R. Rajamony. "Energy Conservation Policies for Web Servers". In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, March 2003.
13. Q. Zhu, F. M. David, C. Devaraj, Z. Li, Y. Zhou, and P. Cao. "Reducing Energy Consumption of Disk Storage Using Power-Aware Cache Management". In HPCA, pages 118-129, 2004.
14. S. Gurumurthi, A. Sivasubramaniam, M.J. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, L.K. John, "Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach," In Proceedings of the International Symposium on High Performance Computer Architecture (HPCA-8), Cambridge, MA, pages 141-150, February, 20.
15. P. Rong, M. Pedram, "Hierarchical Power Management with Application to Scheduling", ISLPED (International Symposium on Low Power Electronics and Design) 2005.
16. M. Srivastava, A. Chandrakasan, and R. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE Trans. VLSI Systems*,Vol. 4, pp. 42–55, Mar. 1996.
17. Paleologo, L. Benini, A. Bogliolo, and G. De Micheli, "Policy optimization for dynamic power management," *IEEE Trans. Computer-Aided Design*, Vol. 18, pp. 813–33, Jun. 1999.
18. S.U. Khan and I. Ahmad, "A Powerful Direct Mechanism for Optimal WWW Content Replication," in *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Denver, CO, U.S.A., April 2005, p. 86.
19. S.U. Khan and I. Ahmad, "Non-cooperative, Semi-cooperative, and Cooperative Games-based Grid Resource Allocation," in *20th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Rhodes Island, Greece, April 2006. (To appear)
20. S. Hariri, Bithia Khargharia, Manish Parashar and Zhen Li, ``The Foundations of Autonomic Computing, edited by Albert Zomaya, CHAPMN, 2005.
21. DDR2 FBDIMM Technical Product Specifications, http://www.samsung.com/Products/Semiconductor/DDR_DDR2/DDR2SDRAM/Module/FBDIMM/M395T2953CZ4/ds_512mb_c_die_based_fbdimm_rev13.pdf
22. P. Zhou , V. Pandey , J. Sundaresan , A. Raghuraman , Y. Zhou , S. Kumar, Dynamic tracking of page miss ratio curve for memory management, ASPLOS 11, October 07-13, 2004, Boston, MA, USA.
23. SPEC JBB 2005, http://www.spec.org/jbb2005/docs/WhitePaper.html
24. L. Schrage, *Linear, Integer, and Quadratic Programming with LINDO*, The Scientific Press, Redwood city, CA, 1986
25. T. D. Braun, S. Ali, H. J. Siegel and A. A. Maciejewski, "Using the Min-Min Heuristic to Map tasks onto Heterogeneous High-performance Computing Systems," in *2nd Symposium of the Los Alamos Computer Science Institute*, Oct. 2001.
26. S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen and S. Ali, "Task Execution Time Modeling for Heterogeneous Computing Systems," in 9th *Heterogeneous Computing Workshop,* May 2000 , pp. 185-199.