Enhancing Energy Efficiency in Multi-tier Web Server Clusters via Prioritization

Tibor Horvath and Kevin Skadron
Department of Computer Science
University of Virginia
Charlottesville, VA 22904
{tibor, skadron}@cs.virginia.edu

Tarek Abdelzaher
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
zaher@cs.uiuc.edu

Abstract

This paper investigates the design issues and energy savings benefits of service prioritization in multi-tier web server clusters. In many services, classes of clients can be naturally assigned different priorities based on their performance requirements. We show that if the whole multi-tier system is effectively prioritized, additional power and energy savings are realizable while keeping an existing cluster-wide energy management technique, through exploiting the different performance requirements of separate service classes. We find a simple prioritization scheme to be highly effective without requiring intrusive modifications to the system. In order to quantify its benefits, we perform extensive experimental evaluation on a real testbed. It is shown that the scheme significantly improves both total system power savings and energy efficiency, at the same time as improving throughput and enabling the system to meet per-class performance requirements.

1 Introduction

Multi-tier, clustered web server architectures are routinely used for high-performance web service provision. Serious problems caused by the extremely high energy consumption of large-scale systems, including the cost of power delivery and cooling, thermal issues, and long-term reliability challenges, are well-known today. Several power reduction techniques based on low-power hardware states and load distribution strategies have been devised to address these problems in web server clusters [7, 10, 14, 15].

One of the principal techniques used to address the problem is dynamic voltage scaling (DVS), which exploits the capability of processors to rapidly change their operating frequency and voltage with very low overhead. The list of valid frequency-voltage pairs constitute the CPU's performance states or ACPI P-states. A software controller can transition between P-states in order to increase or decrease the CPU's power consumption.

Defining an optimal DVS policy for throughput-oriented server cluster workloads is especially challenging because the power-performance relationship varies between individual machines. Further, in multi-tier (functionally distributed) clusters, the workload performance characteristics also vary between tiers since different software is used for each. Hence, similar power management decisions affect each tier differently, and policies that are locally optimal for each tier do not combine into a *globally* optimal power management strategy. Therefore, multi-tier servers require global, coordinated energy management to jointly optimize power and end-to-end performance. In our previous work, we developed a DVS control algorithm for multi-tier servers based on theoretical optimization [11].

In this paper, we further increase the energy efficiency of server systems that use our DVS policy by introducing prioritization. In many applications, requests can be classified into different priority classes such that deadlines of lower-priority requests can be relaxed. Then, if the server is prioritized and the DVS algorithm is made aware of the relaxed performance requirements of some classes of requests, it may be able to operate some of the server machines in slower P-states, resulting in overall power savings.

We make two contributions in this paper. First, we demonstrate an inexpensive design of a prioritized multi-tier web server cluster running commodity software requiring no application or OS modifications. Second, we quantify the improvement in the system's overall energy efficiency due to such prioritization through experiments on our testbed with a realistic multi-tier web server workload. Prioritization saves up to 15% additional energy with only 3% increase in average deadline miss ratio (DMR) and an up to 4% *decrease* in DMR for high-priority requests.

2 System Architecture

2.1 Motivation

One of the successful existing power and energy management techniques involves power-aware QoS, a combination of dynamic power management and service prioritization [16]. The essence of such a technique is that lower-priority tasks (requests) with longer deadlines can be satisfied with lower system performance. Consequently, the power usage of the system can be reduced.

The power-aware QoS approach was shown to yield good results. However, prioritization in itself has not been evaluated from an energy management standpoint. This makes it difficult to predict how it would affect a system with an existing dynamic power management solution already in place. Therefore, our work is motivated by the question of what the added energy efficiency gain of prioritization is.

2.2 Assumptions

We address server farms running a multi-tier service such as an e-business web site. The general architecture is shown in Figure 1. Based on the expected load, each tier is statically preallocated a number of machines (m_i), among which the total offered load arriving at the tier is evenly distributed.

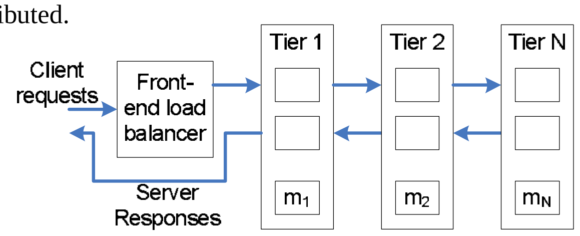

Figure 1. General model of multi-tier system

We assume that, at a given throughput, the most important performance metric for the multi-tier service is end-to-end server latency, measured from the point a client request enters the first tier until a response is sent back to the client. Since response latency is the primary factor behind user satisfaction with web sites [3], maximizing server throughput alone is not sufficient. Hence, we assume that the site owner defines end-to-end deadlines that the server application has to meet. As it is typical in web services, these deadlines are considered soft, where a user-selectable maximum miss rate must statistically be met.

2.3 Design

2.3.1 Ideal design

The complete prioritization of a traditional multi-tier server is very costly in terms of implementation effort. Consider the model of a typical server application's architecture shown in Figure 2. A request processor accepts requests from the input socket and places them in a task queue. The next available service thread starts executing the task. The task may create subrequests to the next tier, which are

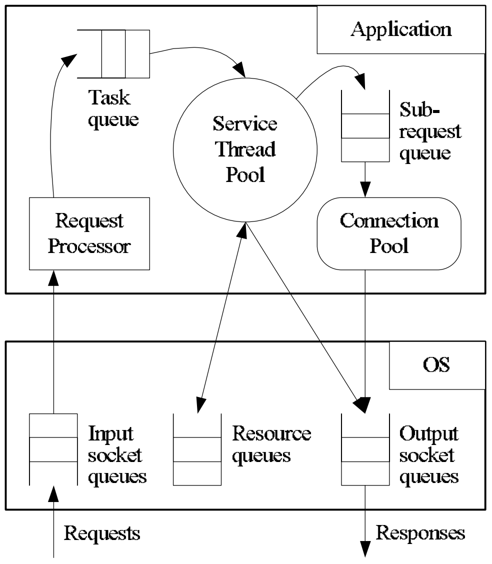

Figure 2. Simplified model of typical server in multi-tier system

queued up in the subrequest queue until a free connection is available. A thread may also become blocked and placed in an OS resource queue (e.g., CPU, disk, or semaphores). Finally, the response is buffered in an output socket queue until the OS is able to send it to the network.

Ideally, all of these queues should support priorities and all subcomponents of each task (such as subrequests, disk IO, resource locks, etc.) should inherit its priority. However, widely used server OSs such as Linux still lack many of these features by default. Moreover, application support is typically largely missing. Hence, to implement the ideal design, one would have to modify critical parts of OS resource management and scheduling, each server application, as well as the communication protocols to propage priority information between them.

2.3.2 Inexpensive design

Our design goal is to find the least intrusive prioritization solution that is effective for the multi-tier system. We avoid application modifications since most server applications are large and complex, and the source code is often not available. It is nevertheless critical that the task queue and subrequest queue behave as if they were prioritized because in general both can become a bottleneck. The reason is that both of these queues are served by a usually small number of pooled resources (threads or connections) and the service times can be relatively large. The local service time of a task primarily depends on how resource-intensive the given tier is, while the latency of a subrequest is the total task service time of the next tier (including its subrequests).

As a simple solution, in many cases it is possible to run multiple instances (one for each service class) of the same server application and prioritize them on the process level. This in effect creates separate queues for each class, which are served by each instance in FIFO order. Assuming the OS supports preemptive real-time process scheduling (such as SCHED_FIFO in Linux), by assigning real-time priorities to the instances we ensure that higher-priority queues are served first whenever there are idle threads or free connections available. Since now we have separate processes for each class, we also have separate communication channels for each class between the tiers. Thus, fortunately there is no need to modify the communication protocols to add priority information, since it is implicitly preserved by the structure: each instance only issues subrequests to next-tier instances of the same priority.

There are many situations where this design must be simplified even further. For instance, database servers are generally not possible to run as multiple instances operating on the same data set. In addition, their disk-intensive workload leaves process-level prioritization less effective, unless OS support for prioritized asynchronous IO was also added. Following our design goal, we solve the issue by leaving the database server unprioritized while trying to minimize task queuing in it. By selecting restrictive connection pool sizes in the previous tier, we ensure that it becomes a bottleneck instead of the unprioritized database server. This works because now the bottleneck stage (typically a CPU-bound application server) is prioritized. It also limits the number of concurrent database queries, reducing priority inversion due to transaction and IO locking. If the original connection pool size was already restrictive, it does not need to be further reduced, only partitioned between instances, since this reserves sufficient connections for high-priority requests to prevent their starvation. Following this principle, in our testbed we assigned 8 connections to 3 instances each, in place of our original pool of 24 connections.

Naturally, the inexpensive design has certain limitations. First, if the server's bottleneck is an OS resource that is not prioritized (e.g., disk or network), then process priorities are not helpful. In most cases, however, the bottleneck can be shifted to the previous tier as discussed above. Second, if an application that does not support multiple instances on the same machine is not the last tier, then this design has no way of propagating request priority to the subsequent tiers. Fortunately, in typical 3-tier setups this is not the case: web and application servers allow multiple instances. Finally, if different-priority instances in the same tier need to perform intensive communication (e.g., to maintain fast-changing shared state coherent), then under heavy overload as the low-priority instances are starved they cannot respond to any messages. However, such shared state poses scalability limitations in itself, and thus avoiding overload is important even without prioritization.

The inexpensive design represents a great reduction in complexity compared to the ideal design and still results in effective prioritization in our experiments.

3 Implementation

3.1 Overview

We deployed a three-tier web-serving system on Linux, with Apache on the first tier, JBoss on the second, and MySQL on the third. As the front-end load balancer, we used the Linux Virtual Server solution. Static content (e.g., images) are served directly from the first tier. Forwarding dynamic requests to JBoss and balancing them across the second tier is done using the Apache module mod_jk. Finally, database requests are issued by JBoss through the Connector/J MySQL driver. To avoid the complexity of database clustering, the third tier consisted of only one server. Since in our setup database performance was not the bottleneck, improving database scalability via clustering would not significantly affect our findings.

3.2 Energy Management

As the existing energy management solution, we employ the following simple feedback control-based DVS policy, discussed in more detail in our previous work [11]. The system periodically determines its load using the following user-defined criteria: if less than 5% of response latencies exceed the 50% of the deadline, the system is considered underloaded, while if more than 5% exceed the 90% of the deadline, we consider it overloaded. The algorithm increases the speed (i.e., P-state) of the most utilized server if the system is overloaded, or decreases the speed of the least utilized server if it is underloaded. Since in our workload (described in Section 4.1) the majority of deadlines were 3 or 5 seconds, a 5-second feedback period was chosen to allow control actions to have an effect by the next period. The latency statistics are smoothed by exponentially weighted moving averaging to reduce measurement noise.

Our implementation has two components: an Apache module to measure the end-to-end latencies, and a daemon on each server that measures its CPU utilization, runs the feedback controller, and sets its P-state.

3.3 Server Replication

Simply replicating a server such that multiple identical instances are executed on the same machine can result in OS resource conflicts. Since many of these are exclusive (bound sockets, output files, server state, etc.), separate resources must be configured for each instance.

Another issue is that server instances that are assigned real-time process priorities can starve regular (non-realtime) processes such as our user-space DVS daemon. Hence, it may not get a chance to increase the machine's speed when needed. Therefore, the daemon was assigned an even higher real-time priority than the server instances.

4 Experimental Evaluation

4.1 Workload

We used the TPC-W benchmark, a very realistic model of an online bookstore application. On the first tier are 2 web servers serving image files, on the second tier we have 4 application servers maintaining session state and executing business logic, and on the third tier we have a database server handling all persistent data. Client machines are running the Remote Browser Emulator, which emulates specified numbers of web clients. The database was populated with the scaling factors of 100 Emulated Browsers and 1,000 Items. The workload profile was the TPC-W Shopping Mix (the basis for the primary TPC-W metrics).

Experiments were repeated 5 times, and consisted of a 5-minute ramp-up period (to warm up the system), a 15-minute measurement interval, and finally a 30-second ramp-down period (to maintain load as measurement finishes). The error bars show the standard deviation.

4.2 Experimental Setup

Our testbed consists of 8 AMD Athlon64 PCs with 512 MB RAM. The processor has two P-states with frequencies at 1.8 GHz and 1.0 GHz. The machines are connected with 100 Mbit Ethernet. For measuring power, we use a Watts Up Pro power analyzer (with 3% accuracy).

We performed our experiments on three different test configurations. The Baseline configuration has no prioritization or DVS. In the NP-DVS configuration, we add our DVS solution, and all clients are treated equal. In the P-DVS configuration, however, we differentiate three clients by relaxing some of their deadlines, assigning them to corresponding priority classes, and prioritizing the multi-tier server cluster as described in Section 2.3.2. For example, if a request type has a 3-second standard deadline, the three clients are assigned deadlines of 3, 6, and 9 seconds, and priority classes 1 (most important), 2, and 3, respectively.

4.3 Evaluation Metrics

Our primary metrics to evaluate performance are the system's throughput (web interactions per second, WIPS) and its deadline miss ratio (DMR).

Our main results evaluate the energy efficiency of the system, showing total system power, and total system energy per web interaction (Joules/WI). The first metric is important since many clusters are thermally constrained and lowering the average power typically lowers related costs. Further, since it is measured for equal intervals, it is directly proportional to total system energy, which translates to electricity costs. The second metric is important because it is a measure of energy efficiency (more precisely its reciprocal), which represents the amount of "useful" energy spent executing the successful web interactions. It also equals to average power divided by average throughput, which demonstrates that it also accounts for performance.

4.4 Results

4.4.1 Performance

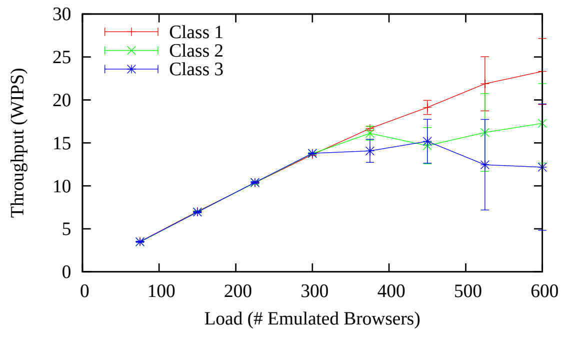

Figure 3. Throughputs of the P-DVS setup

As expected of the prioritized system, Figure 3 shows that the throughput of higher-priority classes is better as the load increases toward peak capacity. When the load is low, the system has enough capacity to satisfy requests from all classes equally, hence the throughput is also identical.

Figure 4 compares the combined throughput of all classes achieved by each setup. Note that in the higher load region the Baseline setup sustains a higher throughput

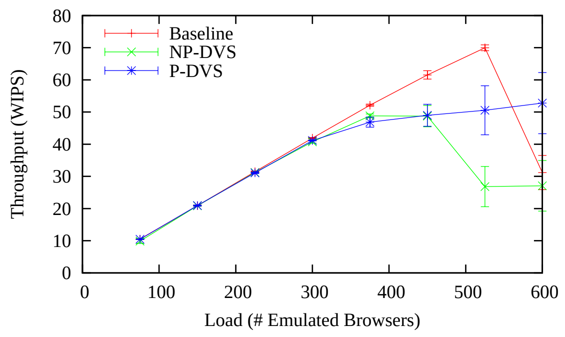

Figure 4. Total throughput comparison (summed across all classes)

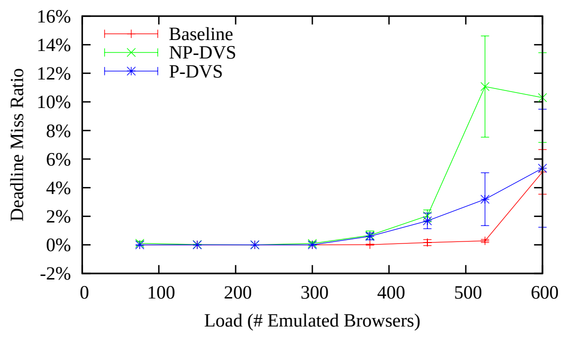

Figure 5. Overall deadline miss ratio comparison (including all classes)

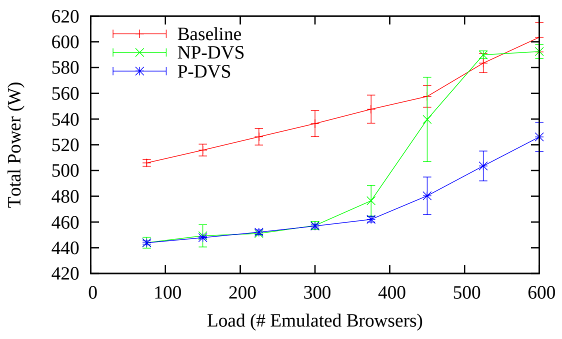

Figure 6. Total system power comparison

than both NP-DVS and P-DVS. This behavior is expected because, with many concurrent requests in the system, response times are streched out more as the DVS policy slows down some servers, hence the web interactions become more spread out, resulting in lower throughput. In other words, since the think times in the emulated browsers do not depend on the response times, if the latter are longer, the next requests will arrive later, effectively reducing throughput. This is, however, not a problem as long as deadlines are honored; in fact, the goal of the DVS policy is to use up all slack time to save energy. It is a major advantage in that the DVS-capable server automatically paces client behavior without missing deadlines, as opposed to operating faster and bringing more requests upon itself that ultimately results in lower energy efficiency.

We make two key observations from Figure 4. First, the Baseline setup's performance quickly falls off under very high loads due to inefficient scheduling. In contrast, P-DVS maintains a more consistent throughput by still allowing most high-priority requests to complete. Second, P-DVS clearly wins over NP-DVS, which degrades even earlier than the Baseline because the stretched execution times increase concurrency, exacerbating the scheduling inefficiencies. The fact that performing DVS in a throughput-oriented soft real-time system places more stress on the scheduler points to a fundamental performance advantage of prioritization in conjunction with energy management.

Comparing the overall deadline miss ratios between setups in Figure 5, we can see that when heavily loaded, NP-DVS has a higher miss ratio since treating all clients equally causes it to saturate earlier. P-DVS, on the other hand, successfully maintains an acceptable miss ratio by reducing scheduler contention via prioritization and by relaxing deadlines of low-priority requests.

4.4.2 Energy Efficiency

The average power of each experiment measured on the whole cluster is shown in Figure 6. As expected, NP-DVS follows the curve seen with conventional DVS algorithms, converging with the Baseline as load increases. In contrast, P-DVS keeps power usage lower even as the system becomes overloaded. As we saw earlier, the differentiated client classes result in a better overall deadline miss ratio, which allows as much as 15% greater power savings. Note that since all experiments had identical measurement intervals, this also equals to the total energy savings.

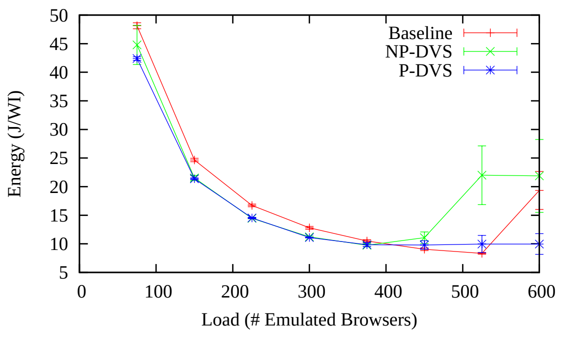

Figure 7. Total system energy per web interaction comparison

Our final goal is to quantify and compare the energy efficiency benefits of the DVS setups. As discussed in Section 4.3, energy efficiency is important since it factors in throughput in addition to energy usage. Figure 7 plots the average energy spent per web interaction, or the reciprocal of energy efficiency of each setup. While the graph shows no significant difference between the DVS setups at lighter loads up to 375 EBs (7–14% savings over Baseline with NP-DVS vs. 6–13% with P-DVS), around 525 EBs we see a large increase (i.e., drop in energy efficiency) with NP-DVS, which is due to the earlier saturation we observed in Section 4.4.1 that causes throughput reduction. P-DVS, on the other hand, avoids wasting energy on less important requests and even achieves 48% savings over the Baseline at 600 EBs because of its ability to use less power and still have higher throughput.

5 Related Work

Several papers address priorities in individual servers [2, 6, 13, 16]. These solutions, however, require modifying the server application, which is costly and often not feasible. Other efforts [1, 17] have been directed at middleware QoS solutions that do not require application or OS changes. These do not, however, address multi-tier servers or energy consumption. In contrast, our work is concerned with the interaction of service differentiation with power management and the energy consumption of clusters. Closely related to our research is the work of Sharma *et al.* [16]. While we build on some of their results, we have a multi-tier system model that requires coordinated energy management.

There is significant recent research on energy management in server clusters [4, 8–10, 12, 14, 15]. However, they either do not deal with service differentiation or they are restricted to a single-tier cluster. An economically-driven energy and resource management framework was presented for clusters in [5]. This research is closely related to our work since it allows service differentiation in conjunction with energy management. However, extending it to the case of multi-tier servers with *end-to-end* latency constraints is not straightforward. Our work is distinguished from the above literature in that we address *multi-tier* clusters with different service classes, and we focus on how prioritization affects the energy efficiency of such clusters.

6 Conclusion

This paper investigated how much additional benefit prioritization has on a multi-tier server system's energy efficiency with an existing DVS-based power management policy. We prioritized the multi-tier system without application or kernel modifications, and performed experiments using three client classes, two with relaxed deadlines. Our results clearly illustrate that the main benefit of prioritization is not only more stable performance as the system nears overload, but also greatly improved energy savings and efficiency, resulting in energy savings up to 15%.

Acknowledgment

This work was funded in part by NSF grant no. CNS-0306404, NSF grant no. CNS-0615277, ARO grant no. W911NF-04-1-0288, and a grant from Intel MRL.

References

[1] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for Web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):80–96, 2002.

[2] J. Almeida, M. Dabu, A. Manikutty, and P. Cao. Providing differentiated levels of service in web content hosting. Technical Report CS-TR-1998-1364, 1998.

[3] N. Bhatti, A. Bouch, and A. Kuchinsky. Integrating user-perceived quality into web server design. In *Proc. 9th International World Wide Web Conference on Computer Networks*, pages 1–16, Amsterdam, The Netherlands, May 2000. North-Holland Publishing Co.

[4] R. Bianchini and R. Rajamony. Power and energy management for server systems. In *IEEE Computer, Special issue on Internet data centers*, volume 37, Nov. 2004.

[5] J. S. Chase, D. C. Anderson, P. N. Thakar, A. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *Symposium on Operating Systems Principles*, pages 103–116, 2001.

[6] L. Eggert and J. S. Heidemann. Application-level differentiated services for web servers. *World Wide Web*, 2(3):133–142, 1999.

[7] E. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. In *Proc. Workshop on Power-Aware Computing Systems*, Feb. 2002.

[8] M. Elnozahy, M. Kistler, and R. Rajamony. Energy conservation policies for web servers. In *Proc. 4th USENIX Symposium on Internet Technologies and Systems*, Mar. 2003.

[9] V. W. Freeh and D. K. Lowenthal. Using multiple energy gears in mpi programs on a power-scalable cluster. In *PPoPP '05: Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 164–173, New York, NY, USA, 2005. ACM Press.

[10] T. Heath, B. Diniz, E. V. Carrera, W. M. Jr., and R. Bianchini. Self-configuring heterogeneous server clusters. In *Proc. Workshop on Compilers and Operating Systems for Low Power*, Sept. 2003.

[11] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu. Dynamic voltage scaling in multi-tier web servers with end-to-end delay control. In *IEEE Trans. Comput.* In press.

[12] L. Mastroleon, N. Bambos, C. Kozyrakis, and D. Economou. Autonomic power management schemes for internet servers and data centers. 2, 2005.

[13] D. McWherter, B. Schroeder, A. Ailamaki, and M. Harchol-Balter. Priority mechanisms for OLTP and transactional web applications. In *Proc. 20th International Conference on Data Engineering*, page 535. IEEE Computer Society, 2004.

[14] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath. Dynamic cluster reconfiguration for power and performance. In L. Benini, M. Kandemir, and J. Ramanujam, editors, *Compilers and Operating Systems for Low Power*. Kluwer Academic Publishers, 2002.

[15] C. Rusu, A. Ferreira, C. Scordino, and A. Watson. Energy-efficient real-time heterogeneous server clusters. In *Proc. 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, pages 418–428, Washington, DC, USA, 2006. IEEE Computer Society.

[16] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu. Power-aware QoS management in web servers. In *Proc. IEEE International Real-Time Systems Symposium*, page 63, Cancun, Mexico, 2003.

[17] P. B. Srinivas. Web2k: Bringing qos to web servers, 2000.