Coordinated DVS for QoS Control in Energy-efficient Web Clusters *

Luciano Bertini[1], **J. C. B. Leite**[1], **Daniel Mossé**[2]

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)
Rua Passo da Pátria 156, Bloco E, 3^o andar – 24.210-240 – Niterói – RJ – Brazil

[2]Department of Computer Science – University of Pittsburgh
Sennott Square – Pittsburgh, PA 15260 – USA

{lbertini,julius}@ic.uff.br, mosse@cs.pitt.edu

Abstract. *The design of energy-efficient computing systems involves seeking any slack that the system can offer. With real-time modeling, one avenue is to design the system's resource management to control the quality of service (QoS) at the lowest level that still meets the real-time specification, avoiding overprovisioning of the system, and thus saving energy. To achieve QoS control, we obtain dynamically a QoS metric and then actuate in the speed of the server nodes capable of* dynamic voltage scaling *(DVS). Because clusters are often heterogeneous, using different DVS settings in each server produces better energy savings. In this paper we investigate local nodes adjusting their DVS settings based on a global off-line optimization, achieving extra power reduction up to* 10%*. Our testbed is a Web cluster composed only of commodity servers, and the workload is from a standardized e-commerce application modeled with real-time properties, the TPC-W.*

1. Introduction

E-commerce has completely conquered its position in the worldwide economy, with a variety of different applications, such as e-sourcing and business-to-business applications. Some companies achieve outstanding success relying only on the Internet, and some purely Internet companies outperform well established rivals who are based only on the old model. Because of this success, the complexity of these applications increase everytime, and also increases the complexity of the systems to host them. On the other hand, the search for more energy-efficient architectures is a research area motivated by relevant issues such as system's cost of ownership, system dependability, and environmental issues.

This paper focuses on energy-efficient e-commerce architectures based on multi-tiered server clusters with optimized power management, and also controlled provisioning of the system. Two ways to save energy in a computing system have been well studied. The first one is to use optimization techniques so that the same performance is obtained with less power consumption. The second way is to model the application with real-time characteristics and reduce the performance while still meeting the real-time specification. Both are important in seeking slacks toward energy savings. There are also two ways of

*This research is being partially supported by the Brazilian Government, through Capes PDEE grant *BEX-3697053*, by CNPq, by the State of Rio de Janeiro Research Foundation (FAPERJ) under grant *E-26/150657/2004*, and also by the US federal research agency NSF, under grant *ANI 03-25353*, *S-CITI* project.

implementing DVS. One is to make a localized DVS in each server that will increase or decrease its speed according to the load it receives. The second class, to which our method belongs, is to have a coordinated DVS. The coordinator sets the DVS operating point and then the amount of work for each server is defined by a request distribution mechanism.

We show the implementation of a web server cluster with a standardized e-commerce application defined with real-time characteristics. Applying QoS control to maintain the QoS in the minimum level dictated by the real-time specification, the system can operate without overprovisioning, and of course this reduces the consumed power. Furthermore, we also apply global optimization in the QoS control process where each server adjusts locally its DVS operating point, to achieve better energy consumption while providing the same QoS. Our architecture uses a QoS control implemented using traditional control theory, which uses dynamic voltage scaling (DVS) as actuators in the system performance to regulate the QoS to the specified level. As a contribution, we show the implementation of the global Single to Multiple Output Optimization (SMOO) that can generate optimized multiple different DVS outputs out of a single controller output, and doing so, the simple and well known single input and single output (SISO) controller can be used.

2. Background

We will describe briefly the environment used as testbed for e-commerce applications.

2.1. Cluster Model

The cluster architecture is shown in Figure 1. It is composed of the front-end server, the application layer to process dynamic and static requests, and the distributed database that will store the e-commerce data. The front-end implements a request distribution policy based on the load of second-tier servers, and based on the overload, if any, of these servers. The front-end server redirects requests to the application servers, directs the server's response to the clients, and is capable of SSL encryption/decryption as required for the e-commerce application.

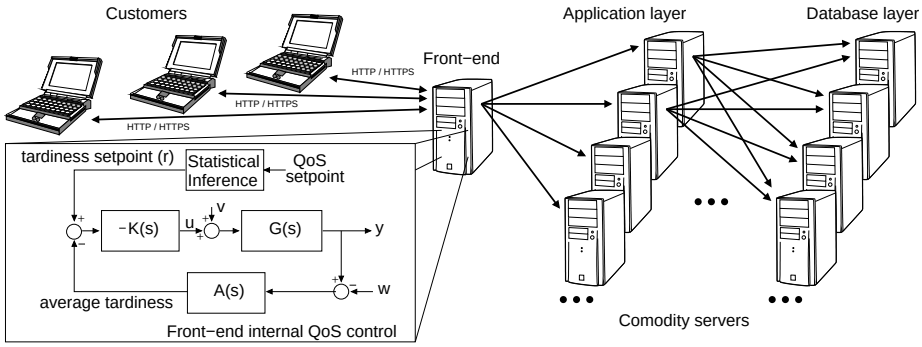


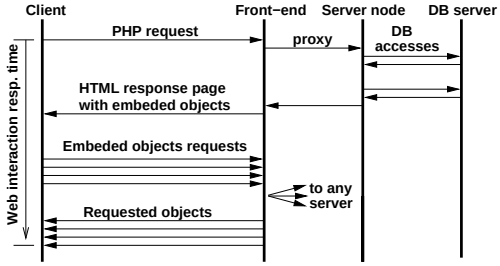
Figure 1. Cluster architecture

For simplicity, the load distribution among the database servers is done statically. We replicate the web store in many independent database servers, avoiding bottlenecks, and dividing equally the total load to each database. To implement the architecture of Figure 1 we used the Apache web server with the module *backhand* [Schlossnagle 2000] for load balancing, a new Apache module to implement the QoS controller, PHP scripting language for the dynamic pages, and PostgreSQL for the databases. The e-commerce application

is the TPC-W, an industry standard benchmark modeled with real-time characteristics (see Section 2.2). Figure 1 also depicts the QoS control logic implemented at the front-end (see Section 2.3), implemented in a new Apache module that is also responsible for measuring the end-to-end time delay of each web interaction.

2.2. Workload Generation

TPC-W is a transactional web benchmark [Daniel F. Garcia 2003] that defines a full e-commerce environment. The workload is generated by Emulated Browsers (EB) that run on the local network, outside the cluster. Each EB is a thread implemented in Java that accesses the web server through HTTP and HTTPS connections, emulating a real customer performing browsing, searching and purchases. TPC-W specifies 14 different interactions necessary to simulate the activity of a book store, and each interaction has a different deadline. With QoS defined as the ratio deadlines met to the total serviced interactions, the standard specifies a QoS level of 0.9 for all interactions. Figure 2b shows a table with the standard defined deadlines.



(a)

Admin Confirm	Admin Request	Best Sellers	Buy Confirm	Buy Request	Customer Regist.	Home	New Products	Order Display	Order Inquiry	Product Detail	Search Request	Shopping Cart	Search Results
20	***3***	***5***	***5***	***3***	***3***	***3***	***5***	***3***	***3***	***3***	***3***	***10***	***3***

(b)

Figure 2. TPC-W features. (a) characterization of one web interaction. (b) deadlines per web interaction type, in seconds

In order to perform QoS control, the front-end must be able to measure the time of one web interaction (the *Web Interaction Response Time* - WIRT). This measure is not trivial because of the definition of a web interaction given by TPC-W. Figure 2a shows the events involved in one web interaction. Upon the arrival of a PHP request, the request is proxied to an application layer server that will access the database. After the database response, the dynamic page is built up and an HTML response page with embedded objects (e.g., images) is sent to the client. The client will then issue the requests for all embedded objects, and they may be distributed to any of the application servers. The subsequent objects will be serviced with a certain level of parallelism, and the time must be measured from the arrival of the PHP request to the time the last embedded object is sent to the client. To make this measurement we implemented in the Apache QoS control module a labeling technique that can mark to what dynamic request belongs every static request, so that the front-end can wait for the sequence of static requests for the time measurement.

2.3. Statistical Inference and QoS Control Logic

We use the Tardiness Quantile Metric defined in [Bertini et al. 2007b]. The main idea is to measure the tardiness, defined as the ratio of web interaction response time to the deadline, and keep the *p*-quantile of this random variable in 1.0, value that means a web interaction finished its execution by the deadline. Doing so, a fraction p of web interactions will meet the deadlines, resulting in a statistical QoS guarantee. This mechanism allows us to specify QoS with very fine granularity.

To estimate the *p*-quantile, we will make assumptions on the workload probability distribution. It has been shown [Crovella and Bestavros 1996] that Web traffic response time can be modeled using heavy-tailed probability density functions, specially the Pareto distribution. Based on this suggestion, we adopted the Pareto distribution to model the tardiness variable, and we have shown in [Bertini et al. 2007b] some statistical tests of goodness of fit, such as the Kolmogorov-Smirnov test and Quantile-Quantile plots. The tests showed very good fit specially at the tail, the region of more interest to quantify the QoS. The result is an expression relating the average tardiness with the *p*-quantile. Furthermore, to make this expression more conservative, we used the confidence interval measured in each sample interval. For example, if the average tardiness measured with its confidence interval is 0.30 ± 0.05, it will be assumed 0.35 rather than 0.30. With a confidence level of 95%, this gives the confidence of 97.5% that the mean will lay below the setpoint value.

The front-end has a PIDF controller (see block diagram in Figure 1), that is, a *Proportional-Integral-Derivative* (PID) controller augmented with a lowpass filter (F) in the derivative part. In industrial plants the filter is needed to reject the noise present in the sensing process. In a e-commerce web cluster the control variable is stochastic, it presents randomness variations that are very similar to noise. We give more details in [Bertini et al. 2007a].

The actuator of the control system is based on dynamic voltage scaling (DVS). The controller broadcast a frequency factor and coordinate the DVS of the application servers. Our DVS scheme consists in switching between the two discrete values, as proposed in [Ishihara and Yasuura 1998], adjacent to the desired continuous value represented by the controller output frequency factor. The controller is single output, it broadcasts the frequency scaling factor r and each server node i in the application layer calculates the desired frequency f_i given by $f_i = r_i(F^i_{max} - F^i_{min}) + F^i_{min}$. The value r_i will be obtained from r by the SMOO method. Then each server calculates its duty cycle α_i for the DVS mechanism that will determine the fraction of the period to stay in the highest available discrete frequency smaller than f_i, denoted by $||f_i||^+$, and the fraction of time to stay in the lowest available discrete frequency bigger than f_i, denoted by $||f_i||^-$. More details in next section.

3. Problem Statement and Solution

Although it is believed that MIMO controllers (*Multiple Input Multiple Output*) are necessary when multiple control objectives exist [Diao et al. 2006], we adopted the simple model of a SISO controller (*Single Input Single Output*). Our model becomes possible because we control the variable tardiness, which aggregates all 14 web interaction response times in only one measure. Further, the DVS output can be also normalized between 0.0 and 1.0 and each server node calculates its desired frequency as explained in Section 2.3. In this section we will show how to calculate a different frequency factor r_i for each server node i in order to reduce the energy consumption related to the baseline, in which all servers use the same $r_i = r$ frequency factor. We have already shown in [Bertini et al. 2007b] that the coordinated DVS scheme with $r_i = r$, and an appropriate load balance, is better than other schemes that use local interval based DVS schemes as in [Rusu et al. 2006]. The SMOO problem is defined by the question: having heterogeneous processors, what is the combination of frequency scaling factors r_i, that will achieve the same performance as if $r_i = r$, but will minimize the total power? We can formalize this as the following:

Minimize

$$P = \sum_{i=1}^{N} P_i = \sum_{i=1}^{N} \left\{ \alpha_i P^i_{busy} \left(||f_i||^- \right) + (1 - \alpha_i) P^i_{busy} \left(||f_i||^+ \right) \right\}$$

subject to:

$$H = \sum_{i=1}^{N} \left\{ \alpha_i H_i \left(||f_i||^- \right) + (1 - \alpha_i) H_i \left(||f_i||^+ \right) \right\} \geq H_{base}$$

where f_i is the desired frequency in the continuous range between F^i_{min}, the minimum frequency of server i, and F^i_{max}, the maximum frequency of server i, given by $f_i = r_i(F^i_{max} - F^i_{min}) + F^i_{min}$, and α_i is the solution to $\alpha_i ||f_i||^- + (1 - \alpha_i)||f_i||^+ = f_i$. In the restriction part, H_{base} is the baseline performance, that is, with $r_i = r$, given by $H_{base} = \sum_{i=1}^{N} \{H^i_{min} + r\,(H^i_{max} - H^i_{min})\}$, with H^i_{min} and H^i_{max} being the performances obtained at F^i_{min} and F^i_{max} respectively, for each server i. This uses the assumption that the performance varies linearly with the frequency in one server, and the collected data showed this does happen (see in Table 1 that the Req/s performance is roughly linear with the frequency). The functions $P^i_{busy}(\cdot)$ and $H_i(\cdot)$ return values from Table 1, and the presented values were measured using the *httperf* benchmark [Mosberger and Jin 1998]. The problem then consists in finding a set of values $R = \{r_1, r_2, \ldots, r_N\}$ so that the power is minimized and the performance is the same as in the case $r_i = r$.

Table 1. Power and performance characteristics of each server node.

Server 1: pentium−m
CPU: Intel Pentium M 1.8 GHz

Freq. (MHz)	P_{busy} (W)	P_{idle} (W)	*httperf* (Req/s)
600	44	42	226.3
800	45	43	303.0
1000	47	43	378.2
1200	49	44	450.7
1400	51	45	535.9
1600	55	47	591.6
1800	60	49	675.3

Server 2: silver−athlon
CPU: AMD Athlon 64 3400+

Freq. (MHz)	P_{busy} (W)	P_{idle} (W)	*httperf* (Req/s)
1000	77	68	332.6
1800	89	70	572.5
2000	100	74	640.8
2200	115	79	673.2
2400	136	85	744.3

Server 3: black−athlon
CPU: AMD Athlon 64 3000+

Freq. (MHz)	P_{busy} (W)	P_{idle} (W)	*httperf* (Req/s)
1000	78	69	345.0
1800	101	73	599.5
2000	112	76	660.2

Server 4: green−athlon
CPU: AMD Athlon 64 3000+

Freq. (MHz)	P_{busy} (W)	P_{idle} (W)	*httperf* (Req/s)
1000	72	65	336.6
1800	105	75	583.5
2000	124	84	658.4

Server 5: blue−athlon
CPU: AMD Athlon 64 3000+

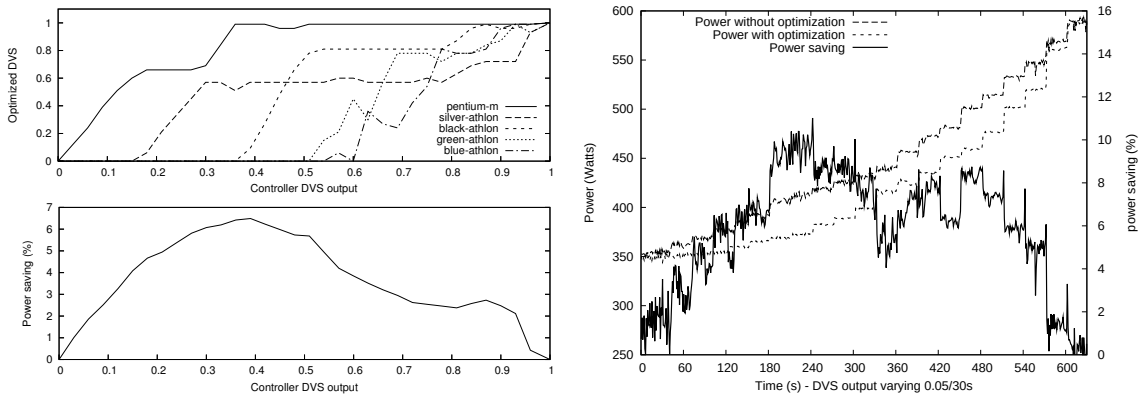
Freq. (MHz)	P_{busy} (W)	P_{idle} (W)	*httperf* (Req/s)
1000	73	64	329.2
1800	108	74	584.2
2000	124	81	653.9

We solved the SMOO problem for 5 servers using *exhaustive search*. Once constructed, the table is stored at each server node and after receiving the broadcast value r, each server i look up its value r_i, in $O(1)$ time. We built the table dividing the range $[0, 1]$ with bins of 0.03 and plotted the curves in Figure 3a. The bottom chart shows the theoretical power saving that the optimization process will achieve, based on the power and performance measures shown in Table 1. This optimization also reveals the most efficient machines, and it can be used to define a dynamic power on/off policy for servers, but this subject is beyond the scope of this paper.

4. Experimental Results

We will show the efficacy of the optimization by two experiments. First we will validate in practice the theoretical plot shown in Figure 3a, and secondly we will show that the Web

cluster can guarantee the same QoS but with reduced power consumption. The practical reproduction of Figure 3a (bottom) is being shown in Figure 3b. The experiment consisted of varying the DVS output r from 0.0 to 1.0 in 0.05 steps, staying in each step for $30s$. Each server was running a CPU intensive process so that the utilization is 100%. The daemon that switches the frequency had a period of $500ms$. The switching overhead is negligible. In [Rusu et al. 2006] the frequency is changed in a $10ms$ basis without noticed overhead. We measured the power with a data acquisition board that monitors current and voltage in AC for all application layer servers. Each point is the average for 60 cycles of instantaneous current $\times$ voltage, that is, one second. The plot is very similar to the theoretical, although the the power savings showed to be at most 10%, rather than 6.5% as shown in Figure 3a. This may be explained by differences in power values from Table 1 compared to the experiment, that used a different application.



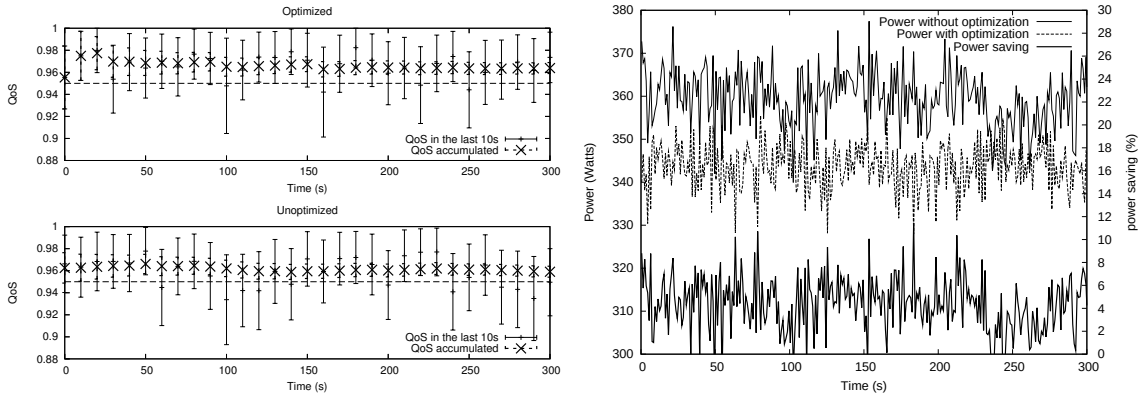
(a) (b)

Figure 3. (a) Theoretical optimization result. (b) Power consumption applying the optimization phase with 100% utilization.

The second experiment was done with 400 EBs (a number that resulted in a DVS output in the middle of the range). With a confidence interval of 95%, the average power in both cases were: 355.94 ± 0.36, and 338.81 ± 0.36, what gives an average saving of 4.8% over the base case (see Figure 4b), and the same QoS for both cases (see Figure 4a). Two QoS measures are being shown, the windowed QoS average, which used a window of $10s$, and the total accumulated QoS during the execution. The former has a bigger confidence interval, because the size of the sample is smaller, and the latter shows a more precise value. As the plot shows, in both cases it is above the QoS setpoint of 0.95. In Figure 4b there is a reduction in the savings near $t = 250s$. This is due to the randomness intrinsic to this system. A second run, even with the same parameters, will never be the same. However, the consistency of the power saving is clearly being shown by the experiment.

5. Related Work

The coordinated DVS scheme (CVS) was first used in [Elnozahy et al. 2002]. They compared an independent DVS, where each server node decides locally its frequency value, against CVS. The performance of CVS was shown to be better, but the with increased complexity for implementation that could not justify its use. In our case, besides being slightly better, its use was imposed by the use of a centralized controller.



(a) (b)

Figure 4. Optimization results. (a) QoS measured with and without optimization. (b) Power consumption in each cases and the percentage power saving.

The method of using off-line optimization to build a table for on-line look up also appear in [Xu et al. 2005], where a technique called LAOVS (Load-Aware On-off with independent Voltage Scale) was proposed, but there the intention of the optimization table was to determine the number of active nodes, in a on/off dynamic mechanism. They used local interval based DVS for changing the processors speed.

Applying queuing theory to model multi-tiered web architectures [Lien et al. 2004, Liu et al. 2005, Urgaonkar et al. 2005] is another possibility to compute and control the QoS probabilistically. We have shown in a previous work [Guerra et al. 2006] that it is possible to choose the system settings so that the power is minimized, and the average response time, given by queuing theory, is such that a predefined amount of deadlines is met. However, it is difficult to have a good queuing model for a real e-commerce environment that allows for a simple analytical formulation of the response time without having many non realistic assumptions about the workload generation and service times. The approach we use in this work is based on a real e-commerce scenario.

Use of control theory in large web systems is a trend. [Diao et al. 2006] gives a classification of the control problems for large systems, using e-commerce as an example. [Kephart and Chess 2003] suggests that autonomic systems will rely on control theory to achieve desired performance objectives. These works motivated ours. Control and power optimization is typically based on CPU utilization. In [Sharma et al. 2003] the authors provide QoS awareness with a feedback loop and DVS, but the goal is to guarantee high QoS levels, in a conservative way, not to control QoS at a fine-grain level.

6. Conclusion

In this paper we showed the implementation of a global optimization technique that convert a single DVS output to multiple DVS outputs (SMOO – Single to Multiple Output Optimization), applied to an e-commerce web server cluster with QoS control. The QoS controller adopts statistical inference to quantify the QoS indirectly, using the tardiness of web interactions related to its deadlines. To simplify the implementation, our controller has one single output. We obtain different outputs for each server in the cluster using the SMOO that is run off-line. The same QoS was obtained with power reduction of up to 10% when compared to an already power-minimized system, and it comes with no QoS penalty.

References

Bertini, L., Leite, J. C. B., and Mossé, D. (2007a). SISO PIDF controller in an energy-efficient multi-tier web server cluster for e-commerce. In *2nd IEEE Intl. Workshop on Feedback Control Impl. and Design in Computing Sys. and Networks*, Munich, Germany.

Bertini, L., Leite, J. C. B., and Mossé, D. (2007b). Statistical QoS guarantee and energy-efficiency in web server clusters. In *19th Euromicro Conf. on Real-Time Sys.* To appear.

Crovella, M. E. and Bestavros, A. (1996). Self-similarity in world wide web traffic: Evidence and possible causes. In *ACM SIGMETRICS Intl. Conf. on Measurement and Modeling of Computer Systems*, pages 160–169.

Daniel F. Garcia, J. G. (2003). TPC-W e-commerce benchmark evaluation. *IEEE Computer*, 36(2):42–48.

Diao, Y., Hellerstein, J. L., and Parekh, S. (2006). Control of large scale computing systems. *SIGBED Rev. Special Issue on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID 2006)*, 3(2):17–22.

Elnozahy, M., Kistler, M., and Rajamony, R. (2002). Energy-efficient server clusters. In *Second Workshop on Power Aware Computing Systems*, pages 179–196.

Guerra, R., Bertini, L., and Leite, J. C. B. (2006). Improving response time and energy efficiency in server clusters. In *VIII Workshop de Tempo Real*.

Ishihara, T. and Yasuura, H. (1998). Voltage scheduling problem for dynamically variable voltage processors. In *Intl. Symp. on Low Power Electronics Design*, pages 197–202.

Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *IEEE Computer*, 36(1):41–50.

Lien, C.-H., Bai, Y.-W., Lin, M.-B., and Chen, P.-A. (2004). The saving of energy in web server clusters by utilizing dynamic sever management. In *12th IEEE Intl. Conf. on Networks*, volume 1, pages 253–257.

Liu, X., Heo, J., and Sha, L. (2005). Modeling 3-tiered web applications. In *13th Intl. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2005)*, pages 307–310.

Mosberger, D. and Jin, T. (1998). httperf – a tool for measuring web server performance. *ACM SIGMETRICS Perform. Eval. Rev.*, 26(3):31–37.

Rusu, C., Ferreira, A., Scordino, C., Watson, A., Melhem, R., and Mossé, D. (2006). Energy-efficient real-time heterogeneous server clusters. In *RTAS'06*, pages 418–428.

Schlossnagle, T. (2000). mod_backhand: A load balancing module for the apache web server. In *ApacheCon Europe*, London, England.

Sharma, V., Thomas, A., Abdelzaher, T. F., Skadron, K., and Lu, Z. (2003). Power-aware QoS management in web servers. In *24th IEEE Real-Time Systems Symp.*, pages 63–72.

Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M., and Tantawi, A. (2005). An analytical model for multi-tier internet services and its applications. In *ACM SIGMETRICS Intl. Conf. on Measurement and Modeling of Computer Systems*, pages 291–302.

Xu, R., Zhu, D., Rusu, C., Melhem, R., and Mossé, D. (2005). Energy-efficient policies for embedded clusters. *SIGPLAN Notices*, 40(7):1–10.