Statistical QoS Guarantee and Energy-efficiency in Web Server Clusters *

Luciano Bertini, J.C.B. Leite
Instituto de Computação
Universidade Federal Fluminense
{lbertini, julius}@ic.uff.br

Daniel Mossé
Department of Computer Science
University of Pittsburgh
mosse@cs.pitt.edu

Abstract

In this paper we study the soft real-time web cluster architecture needed to support e-commerce and related applications. Our testbed is based on an industry standard, which defines a set of web interactions and database transactions with their deadlines, for generating real workload and benchmarking e-commerce applications. In these soft real-time systems, the quality of service (QoS) is usually defined as the fraction of requests that meet the deadlines. When this QoS is measured directly, regardless of whether the request missed the deadline by an epsilon amount of time or by a large difference, the result is always the same. For this reason, only counting the number of missed requests in a period avoids the observation of the real state of the system. Our contributions are theoretical propositions of how to control the QoS, not measuring the QoS directly, but based on the probability distribution of the tardiness in the completion time of the requests. We call this new QoS metric Tardiness Quantile Metric *(TQM). The proposed method provides fine-grained control over the QoS so that we can make a closer examination of the relation between QoS and energy efficiency. We validate the theoretical results showing experiments in a multi-tiered e-commerce web cluster implemented using only open-source software solutions.*

1. Introduction

The tremendous grow in the Internet usage in the last decade is in most part due to the increasing importance that e-commerce and related applications have been showing. Most people today rely on these applications in their day by day lives. Because of this growing importance, better models and more detailed specifications for e-commerce are becoming a new focus of research. In this way, many works have modeled e-applications with real-time characteristics. For example, [11] shows the real-time requirements of e-commerce, addressing mainly the timeliness, among other aspects; [26] presents protocols that can be used to detect when real-time constraints are violated; and [14] describes a real-time middleware to support e-commerce applications.

In addition to real-time characteristics, large systems to host e-commerce applications can show a huge electricity consumption [20], which means high costs of ownership, making power management necessary. In [10], for example, the authors point out that data centers operate at power densities of around 100 Watts per square feet. With all the real-time and energy efficiency issues in mind, architectural challenges arise when we try to deploy architectures to support e-applications which also need to satisfy QoS specifications.

The reasons to study energy-efficient web clusters for e-commerce applications are threefold. First, there is a need for a boost in e-commerce web servers efficiency, as the main goals of the e-commerce use in the enterprise are cost reduction and the creation of competitive advantage, and that is what makes energy-efficiency mandatory. Second, the number and variety of e-commerce applications are growing. An example is the integration and customization of such applications, such as the idea of web shopping malls, support for comparative shopping and business-to-business [16]. The third reason is that the recent work on performance evaluation for web server clusters, specially for power- and energy-efficiency, has clamored for more realistic test workloads.

Our objective is to have a means of exploring the trade-off between energy and QoS in complex web systems, and for this we need to have a fine grain control of the QoS. Instead of using a QoS measure based on the counting of missed deadlines, we use the on-line measurement of tardiness in the completion time of the requests, because we verified in practice that counting missed deadlines results in poor accuracy and broad confidence intervals. Our contribution is the statistical guarantee that we can achieve for the QoS based on approximations for the probability density function of the tardiness random variable. We show that the average tardiness is directly related to the QoS. To maintain the user specified QoS level, we used feedback control logic, based on a PID

*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.

controller[1]; the control variable used was the average tardiness instead of number of missed deadlines. Thus, we can show the consequences to the system when the QoS is maintained in a specified level, which is very important for the energy efficiency, because a lower QoS level is generally associated with less resource usage.

We prove the correctness of the proposed theoretical relation between tardiness and QoS. The performance evaluation we present in this paper is based on a real implementation of a web store, using commodity hardware and open-source software. The workload is from an industry standard transactional benchmark for e-commerce, the TPC-W [13], installed on a heterogeneous cluster running Linux.

2. Related Work

Several classical papers addressing energy-efficient web server clusters [8, 9, 15, 21, 22] introduce cluster reconfiguration techniques and local or global DVS policies (for a good review see [7]). We focus on those that have QoS awareness.

Two works are closest to ours. The first [24] used a feedback loop to regulate the voltage and frequency as a means of providing QoS awareness. Their controller uses utilization as the control variable aiming to keep it around a derived utilization bound that was shown to be a sufficient condition of schedulability. As exceeding this bound does not necessarily imply in missed deadlines, having this utilization bound as a control set-point achieves good results in guaranteeing the QoS close to 1.0.

The second [23] presents a cluster-wide QoS aware technique based on local DVS and cluster reconfiguration. They guarantee a QoS level of 0.95 by setting the maximum load of a server as the maximum number of requests that the system can handle meeting the 95% of deadlines, and always turn on a new machine when needed to reach this limit. The local DVS scheme sets the frequency periodically on a $10\ ms$ basis to $\lceil U f_{max} \rceil$, and the utilization U is calculated based on the current enqueued requests.

Both works [23, 24] do not allow the maintenance of the QoS at a precise user predefined value. Our work differs from these two approaches because we apply a statistical inference solution to guarantee the exact desired QoS level aside from the fact that our target environment is e-commerce. In addition, most previous work dealt only with requests with independent deadlines, which are not typically representative of e-commerce applications.

In [5], feedback control is used to achieve overload protection, performance guarantee, and service differentiation, based on the same concept of utilization bound [24], thus aiming to meet all deadlines. However, that work applies adaptation of QoS to server side load conditions, where the controller actuator can offer degraded service levels accomplished by content adaptation. The content is preprocessed a priori and stored in multiple copies that differ in quality and size. Hence, the approach is different, besides the fact that their architecture is primarily aimed for static web content.

Similarly, an autonomic system is described in [27] to allow administrators to set system properties like QoS. For this, they apply control theory with complex feedback optimization techniques where future environment inputs and the future consequences of the control actions are taken into account during optimization, which is multiobjective including power optimization goals. The QoS is defined as response time and is used directly as the controller set-point. However, the focus is more at the control theory rather than the implementation of a real e-commerce environment; the workload is derived from an Internet service provider and they assume continuous DVS settings. For e-commerce environments, an average response time goal alone cannot tell much about the fulfillment of the real-time rules. In this sense, this work is complementary to ours, because it may be applied to our statistical inference to achieve the desired QoS proportion.

3. Application and Web Cluster Model

Our cluster model is shown in Figure 1, with a front-end server acting as a reverse proxy. The front-end is capable of SSL encryption/decryption, and will distribute the requests to the web server nodes without encryption between front-end and web servers.

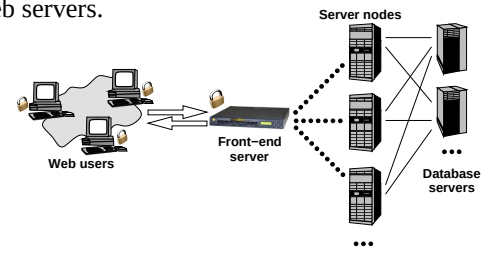

Figure 1. Cluster model

Our cluster has two layers after the front-end, with the application server and web server running at the same machine, and a second layer for the databases. As the purpose of this work is to focus on the power management of the web cluster, we replicate the web store in many database servers to avoid bottlenecks at that layer.

3.1. TPC-W Benchmark

TPC-W is a transactional web benchmark, produced by the Transaction Processing Performance Council [1], where the workload is performed in a controlled Internet commerce environment. The workload tests several system components associated with this environments, such as multiple on-line browser sessions, dynamic page generation, secure connections, and a database consisting of many tables with a wide variety of sizes, attributes, and relationships.

[1]A proportional-integral-derivative controller (PID controller) is a common feedback loop component in industrial control systems.

A possible environment for the TPC-W is depicted in Figure 2. The workload is generated by the remote browser emulator, responsible for managing the emulated browsers (EB) and the emulated sessions. The EBs access the web server using HTTP and HTTPS connections. The system under test is composed of three components, the web server for static pages, the application server for the execution of the application (e.g., using PHP), and the database server.

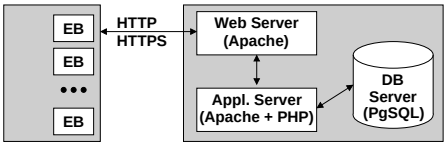

Figure 2. TPC-W environment

The performance metric reported by TPC-W is the number of *web interactions per second* (WIPS). TPC-W specifies 14 different interactions necessary to simulate the activity of a retail store, and each interaction is subject to a deadline that must be met with a specified QoS (as a percentage of deadlines met). There are three different profiles for the test, with a mix of interactions for shopping, browsing and ordering. The primary metric (WIPS) is intended to reflect an average shopping scenario with a mix of 80% of browsing interactions and 20% of ordering interactions (for a review about the TPC-W benchmark see [13]).

One aspect of the TPC-W specification is that one web interaction is not composed of a single request, but of a request to a dynamic page followed by several static requests for the embedded objects that are part of the dynamic generated page. The *web interaction response time* (WIRT) is defined by the time elapsed between sending the dynamic request until receiving the last byte of the last embedded object. This specification makes it impossible to measure the QoS locally in one server node, because each embedded object request may be sent to different nodes.

In the TPC-W real-time specification, each class of web interaction has a different deadline, as shown in Figure 3, with a minimum deadline hit ratio defined by the standard as 90% for all classes. Although it is not specified in the standard, a system should not a priori discard 10% of the requests just because the goal is to service 90%, but rather it should attempt to service all requests and provide for all an equal probability of meeting the deadline. Also, it is worth to note that these values for the deadlines include only local area access, according to the TPC-W specification, so that the Internet access can be disregarded.

	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
90% WIRT Constraint (deadline in seconds)	***20***	***3***	***5***	***5***	***3***	***3***	***3***	***5***	***3***	***3***	***3***	***3***	***10***	***3***

Figure 3. Deadlines as defined by TPC-W

4. QoS Control

The goal of the system is to maintain/control QoS at a certain level. This can be done by controlling the QoS directly, as in [19] and [25], but it turned out to be problematic because with the QoS defined as a ratio of deadlines met to the total requests, a reasonable number of requests is necessary to obtain narrowed confidence intervals. Furthermore, the QoS will saturate at 1.0, causing an asymmetry problem and instability, as will be shown in Section 6. In [19] and [25], however, they used a more complicated control, based on a second control loop for the utilization, that can solve the problem of deadline miss ratio saturation at 0, because the saturation condition of both controllers are mutually exclusive. In contrast, we propose to control the QoS based on the average tardiness of the web interactions. For each web interaction i, we define tardiness by the ratio *web interaction response time* (WIRT) to the respective deadline. That is, $tardiness_i = \frac{wirt_i}{deadline_i}$. A more detailed definition of WIRT will be given in Section 5.2. In this section we show the relation between QoS and the average tardiness.

The block diagram for the control logic is shown in Figure 4. As will be shown in Section 4.1, the user specified level of QoS is applied to a statistical inference method to obtain the necessary average tardiness for that QoS, and if the system is kept with this average tardiness, the QoS is statistically guaranteed to be in the specified value. This average tardiness value is the set-point to the controller.

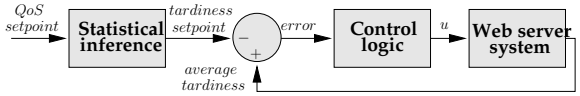

Figure 4. QoS control logic block diagram

Described in Section 4.2, our PID controller outputs a single frequency scaling factor u to be used to control the DVS of all the servers. For each server, u specifies the computing capacity. When $u = 0$, the server will run at the minimum frequency, and when $u = 1$, at the maximum frequency. Any value in between will cause the server to cycle periodically between two available discrete frequencies, so that the average frequency is a value proportional to u (see Section 4.3 for more details).

4.1. Statistical Inference: Tardiness Quantile Metric (TQM)

In this section we show some statistical tests of goodness of fit between the data and the chosen probability distributions. We study more than one distribution in order to choose the best approximation. We also present the theoretical QoS formulation for each distribution.

Using a large dataset, the authors in [12] showed that web traffic, such as response time, can be modeled using

heavy-tailed probability density functions, which have self-similarity property, specially the Pareto distribution. We then verified in practice that e-commerce traffic (i.e., WIRT and tardiness) do present a probability distribution close to Pareto. Based on this distribution, we formulated the requirements for the system to meet the specified QoS.

We also propose the use of a second distribution, the Log-normal, which has two parameters that can be easily estimated on-line. The intuition behind using the Log-normal distribution is the fact that the ratio execution time to the deadline has an unreachable lower limit of 0, but has no upper limit, like some variables usually modeled by Log-normal (e.g., personal incomes, tolerance to poison in animals, etc) [17].

The QoS and tardiness value are directly related. The bigger the average relative tardiness, the lower is the resultant QoS. The reason we chose tardiness as a control variable, aside from the problems mentioned earlier, is that tardiness does not carry only a boolean information about QoS: whether the deadline was met or missed, but it is a continuous value possible to be calculated for each web interaction, and its value shows how close the execution was to the deadline. The relation between tardiness and QoS is obtained from the probability density function for the tardiness value. We derive this relation from the p-quantile calculation, that is, the tardiness value x such that $P[X \leq x] = p$. Based on the tardiness definition, if the p-quantile is 1.0, then the QoS is p. Hence, we call this method of QoS measuring *Tardiness Quantile Metric* (TQM). In the rest of this section we will show the QoS-tardiness relationship for both Pareto and Log-normal distributions.

TQM with Pareto Distribution

In Figure 5, we show the p.d.f. obtained from an experiment run for $2,000$ seconds and $26,255$ web interactions. There is a visual fit between the data and the Pareto distribution, but the Kolmogorov-Smirnov goodness of fit test returns a maximum value between the empirical cumulative distribution and the expected Pareto value of 0.08, while the threshold necessary to accept the data as coming from a Pareto distribution would be 0.01. Figure 5 shows that the first bar close to zero is smaller than the second bar, which does not happen in a Pareto distribution. However, as we will show later, Pareto is still a good approximation to use.

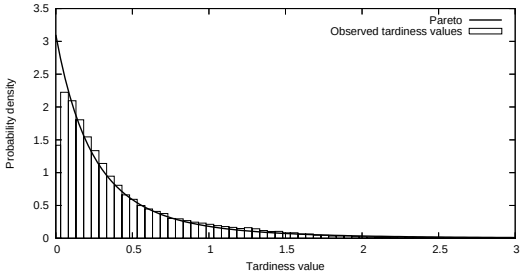

Figure 5. Tardiness p.d.f and Pareto p.d.f.

The representation of a Pareto probability density function is given by $f(x) = k\frac{x_m^k}{x^{k+1}}$, where the parameter k is related to the average μ of the distribution by $\mu = \frac{kx_m}{k-1}$, and x_m is the necessarily positive minimum possible value of X. Note that the tardiness value has a minimum value of 0. For this reason, we use $x_m = 1$ and the transformation $x' = x - 1$. Then we obtain the following equation for the tardiness distribution:

$$f(x) = \frac{k}{(x'+1)^{(k+1)}} \tag{1}$$

where $k = \frac{\mu+1}{\mu}$. Let p be the level of QoS desired, that is, $0 \leq p \leq 1$ denotes the fraction of deadlines that must be met. We can formulate the following theorem:

Theorem 4.1 (QoS based on Pareto) *If the tardiness value, defined in Section 4, is a random variable with Pareto distribution, a level p of QoS will be achieved, with a confidence level of $1 - \frac{c}{2}$, where $1 - c$ is the confidence level for the sample mean μ obtained from the system, if the following relation holds:*

$$\mu - z_{\frac{c}{2}}\frac{\sigma}{\sqrt{N}} = \frac{1}{log_2\left(\frac{1}{1-p}\right) - 1} \tag{2}$$

where μ is the average value for a set of N samples obtained for the tardiness, σ is the standard deviation for the same set, and $z_{\frac{c}{2}}\frac{\sigma}{\sqrt{N}}$ is the confidence limit for the mean with the desired significance level c.

Proof We will do the proof in two parts. First we show that the right side of the equation represents the value of the real mean of the data that makes the p-quantile equals 1.0. The web interactions with missed deadlines are those for which tardiness resulted bigger than 1. To have p deadlines met, we need the probability of $0 < tardiness < 1$ to be p. Thus we need $\int_0^1 \frac{k}{(x'+1)^{(k+1)}}dx = p$, resulting in $1 - 2^{-k} = p \Rightarrow k = log_2\left(\frac{1}{1-p}\right)$. As the average μ in a Pareto distribution with minimum value positioned at $x = 1$ is given by $\frac{k}{k-1}$, with the transformation $x' = x - 1$ we have $\frac{k}{k-1} = \mu + 1$, giving $k = \frac{\mu+1}{\mu}$. Solving for μ the equation $\frac{\mu+1}{\mu} = log_2\left(\frac{1}{1-p}\right)$, and adding the confidence limit, we obtain equation 2.

The second part is to consider the confidence level. The sample mean μ obtained does not represent the real mean of the data, but in half of the cases where the sample mean is obtained, this value will fall below the real mean, and for the other half will fall above. To guarantee the QoS, we need the real mean below or equal to the right side of the equation. Thus, if the sample mean is controlled in the lower limit given by the confidence interval, the unfavorable cases will happen only in $\frac{c}{2}$ of the cases. This limit is represented in the left side of equation 2 by the term $z_{\frac{c}{2}}\frac{\sigma}{\sqrt{N}}$. ∎

TQM with Log-normal Distribution

Now we will show the same idea for another distribution, the Log-normal. A data has Log-normal distribution if the

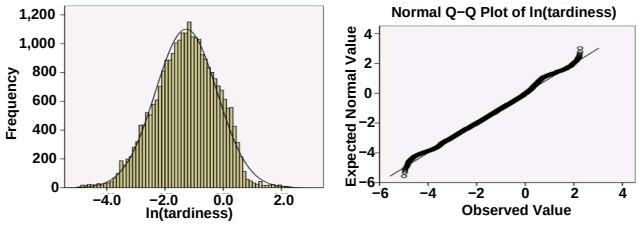

Figure 6. P.d.f. of $ln(tardiness)$ with the theoretical normal and the Q-Q plot

natural logarithm of the data has a Normal distribution. Figure 6 shows the histogram of the natural logarithm of the tardiness data and the theoretical Normal distribution, and also shows the *Quantile-Quantile* plot (right side) obtained using SPSS [4]. The *Q-Q* plot is used to verify the deviation of a given data to the normality. The normality of the data will cause a straight line in the *Q-Q* plot. The plot is showing that the data is very close to normal, with some variation on both end tails. We also applied the Kolmogorov-Smirnov goodness of fit test in this case and obtained a better fit, with 0.03 maximum difference between the measured and theoretical cumulative distributions, against 0.08 for Pareto (same 0.01 threshold). Thus, we have the following theorem:

Theorem 4.2 (QoS based on Log-normal) *If the tardiness value, defined in Section 4, is a random variable with Log-normal distribution, a level p of QoS will be achieved, with a confidence level of $1 - \frac{c}{2}$, where $1 - c$ is the confidence level for the sample mean μ obtained from the system, if:*

$$\frac{\mu - z_{\frac{c}{2}} \frac{\sigma}{\sqrt{N}}}{\sigma} = -\sqrt{ln\left(\frac{1}{1-p^2}\right)} \tag{3}$$

where μ and σ are the average value and the standard deviation of the natural logarithm of the tardiness value, considering N samples.

Proof Similarly to the Theorem 4.1, we have the p-quantile calculation and the addition of the same confidence limit. The proof of the right side of the equation follows. Let $f(x)$ be a normal distribution with average 0 and standard deviation σ. Let b be the value of x that results in $\int_{-\infty}^{b} f(x)dx = p$. We have to solve:

$$\frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{b} e^{-\frac{x^2}{2\sigma^2}} dx = p$$

which is solved using the square of this integral equation and the substitution $r^2 = x^2 + y^2$:

$$\int_{-\infty}^{b} e^{-\frac{x^2}{2\sigma^2}} dx \int_{-\infty}^{b} e^{-\frac{y^2}{2\sigma^2}} dy = 2\pi\sigma^2 p^2$$
$$\int_{-\infty}^{b}\int_{-\infty}^{b} e^{-\frac{x^2}{2\sigma^2}} e^{-\frac{y^2}{2\sigma^2}} dxdy = 2\pi\sigma^2 p^2$$
$$\int_{0}^{\sqrt{b^2+b^2}} e^{-\frac{r^2}{2\sigma^2}} 2\pi r dr = 2\pi\sigma^2 p^2$$

Using $u = r^2$ and $du = 2rdr$, we obtain $1 - e^{-\frac{b^2}{\sigma^2}} = p^2$, resulting in $b = \sigma\sqrt{ln(\frac{1}{1-p^2})}$. This result is for a normal distribution with $\mu = 0$. In order to have p of the deadlines met, we need a shifted normal distribution so that $b = 0$, because the natural logarithm of the tardiness will be less than 0 whenever the deadline is met. Thus, for this to happen, we need the average of the natural logarithm of the tardiness to be $\mu = -\sigma\sqrt{ln(\frac{1}{1-p^2})}$, which is equation 3 without the confidence limit. ∎

Discussion

For the TPC-W specification, where $p = 0.9$, the tardiness average is $\mu = 0.43068$ using the Pareto distribution, and the ratio $ln(tardiness)$ average to the standard deviation of $ln(tardiness)$ is $\frac{\mu}{\sigma} = -1.28869$. In the Pareto distribution, the on-line estimation of the tardiness average has a simpler implementation than in the Log-normal, but both can be done with a low complexity ($O(1)$ for time and $O(N)$ for space). We will show results for many values of specified QoS in Section 6, where we used a confidence limit of $\frac{2\sigma}{\sqrt{N}}$ to test both assumptions, yielding a confidence interval of 95.45% for the sample mean, and consequently 97.725% confidence level that the QoS will be equal or higher than the specified value.

4.2. Control Logic

We will make use of the classic $z-$transform methodology to derive the equations for the control logic. The $z-$transform is used in signal processing to convert a discrete time domain signal, which is a sequence of numbers, into a frequency domain representation. To make this conversion, the z variable, in the definition of the $z-$transform showed in equation 4, must be replaced by $z = e^{sT_s}$, where s is the complex parameter of the Laplace transform and T_s is the sampling interval.

$$X(z) = \sum_{n=0}^{\infty} x_n z^{-n} \tag{4}$$

In equation 4, where x_n is the n^{th} sample of the signal x, the signal is composed by the most up to date sample, multiplied by z^0, the previous sample, multiplied by z^{-1}, and so on. Thus, this definition can be used to discover the approximate frequency domain representation of a sampled signal. This is used in control theory to build digital filters with the same behavior of the equivalent analog filter.

We applied the $z-$transform to discretize the Laplace equation of a PID controller, given by $G(s) = K_P + \frac{K_I}{s} + K_D s$, where K_P, K_I, and K_D are the proportional, integral, and derivative PID constants, respectively. Using the simplest approximation[2] to find z as a function of s, we obtained the

[2]Called the backward difference, which is given by $z = \frac{1}{1-sT_s}$, and is obtained from a first order series approximation to the $z-$transform

following equation for the controller, which is $O(1)$ in time and space for implementation.

$$out_k = out_{k-1} + \left(\frac{K_D}{T_s} + K_P + T_s K_I\right) error_k - \left(\frac{2K_D}{T_s} + K_P\right) error_{k-1} + \frac{K_D}{T_s} error_{k-2} \quad (5)$$

where out_k is the k^{th} sample for the output (i.e., the frequency factor u) of the controller, and $error_k$ is the k^{th} sample for the error, which is the difference between the set-point and the actual value of the output (see Figure 4). Using equation 5, it is necessary only to keep in memory the two latest error values, $error_{k-1}$, and $error_{k-2}$.

The average and standard deviation were obtained by using a sliding window of size N. The implementation is $O(1)$ in time for both the average and the standard deviation. At each sample, the average value is updated by the sum of the new value and the subtraction of the oldest value. The space complexity is $O(N)$ for both.

As the focus of this paper is not the controller itself, we will not address it here. In [6] we address the controller showing an analysis of sensitivity to the parameters, and with improved control dynamics applying filters in the derivative part. Here, for the proof of concept, we use values $K_P = 0.02$, $K_I = 0.05$, and $K_D = 0.02$, and also the number of samples $N = 200$ that resulted in good responsiveness and stability.

4.3. Speed Setting

We use a simple DVS scheme that consists in switching between the two discrete values adjacent to the desired frequency [18]. This scheme is a good solution to the case of a controller actuator, because it offers a continuous, rather than discrete, operating point, so that the controller can have a continuous output. In this scheme, a high priority daemon executes periodically with a duty cycle α with the exact width to stay in the higher frequency, and the remaining of the period in the lower frequency.

The frequency scaling factor u output by the QoS controller is broadcast to each server node and each server node i calculates the desired frequency f_i given by $f_i = u(F_{max} - F_{min}) + F_{min}$. The duty cycle of the DVS mechanism is α, so that $\alpha||f_i||^- + (1-\alpha)||f_i||^+ = f_i$, where $||f_i||^-$ is the highest available discrete frequency smaller than f_i, and $||f_i||^+$ is the lowest available discrete frequency bigger than f_i.

5. Implementation Issues

We describe the system components used in the implementation of our web store on the cluster, and show some implementation issues not directly related to the QoS control, such as the request distribution mechanism, important time measurements, and servers turn-on/turn-off policy.

5.1. Hardware and Software

The hardware used in the testbed, summarized in Table 1, is composed of the front-end, four machines for the web server tier, and three machines for the database tier, besides one machine to execute the emulated browsers, in the same configuration as Figure 1. We chose this configuration so that we were able to focus on the web/application server layer. This configuration puts a load, including SSL processing, of 64% on the front-end and about 80% on the database servers, avoiding bottlenecks.

Table 1. Hardware used

Node	Function	Freq. available (MHz)	Specifications
yellow	front-end	Not applicable	AMD Athlon 64 X2 Dual Core 4200+ 2GB RAM
pm1	web server	600, 800, 1000, 1200, 1400, 1600, 1800	Pentium M 1GB RAM
black	web server	1000, 1800, 2000	AMD Athlon 64 3000+ 1GB RAM
silver	web server	1000, 1800, 2000, 2200, 2400	AMD Athlon 64 3400+ 1GB RAM
green	web server	1000, 1800, 2000	AMD Athlon 64 3000+ 1GB RAM
antimony	database	Not applicable	1 CPU Intel Xeon 3.80GHz 8GB RAM
oxygen	database	Not applicable	4 CPUs Intel Xeon 3.60GHz 4GB RAM
hydrogen	database	Not applicable	4 CPUs Intel Xeon 3.60GHz 4GB RAM

The software used was the Apache web server, the PHP scripting language, and the database PostgresSQL. For the TPC-W we used the specification compliant implementation available at the PgFoundry PostgreSQL development group [2]. The front-end works as a reverse proxy, with the load-balancing Apache module mod_backhand [3], which allows easy addition of new request distribution policies. For the database, it is mandatory to have a distributed database solution in this architecture. In spite of that, as our focus was to study the power management in the web server layer, we used multiple databases without replication. In each database, we deployed an independent web store with $10,000$ items and $1,000$ customers each. For example, for a load of 600 EBs, we start 200 EBs accessing each independent web store. For the web servers it makes no difference. That is, any request is treated equally, and any server is able to process any request, regardless of what database server will respond to the queries.

5.2. Time Measurements

The main problem that makes the implementation in [23], and others cited in Section 2, inappropriate to the TPC-W application is that we need to have a way to measure the web interaction response time (WIRT) as a whole, and it is impossible to be made locally in one web server node. The WIRT is defined by the TPC-W specification as the time from the sending of the PHP request by the EB until the receiving of the last byte of the last image embedded in that PHP request.

The problem is that the requests to the embedded objects may be sent to different web server nodes in the cluster.

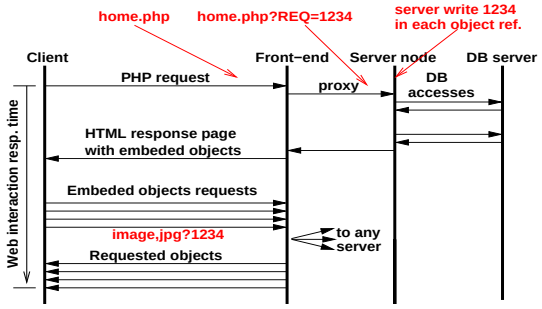

Figure 7. WIRT time components

We measure the approximate WIRT at the front-end, excluding only the local network time between the EBs and the front-end. For this, we implemented a new Apache module that labels the requests before sending them to the server nodes, as shown in Figure 7. When the PHP request arrives at the front-end (e.g., home.php), the module creates a unique number and attaches it as a new parameter in the URI of the PHP request. When the web server node receives the request, it gets the label and puts it, also as a parameter, in every embedded object reference. Each subsequent request for every embedded object will come with the label to which PHP request it belongs to. When the request for the last object finishes, the front-end knows the time for the whole web interaction and can compute the QoS and tardiness. We note that this solution does not modify the client at all, and therefore is backward compatible with existing systems.

Another implementation issue was that we needed to know the average CPU time spent, in user space and kernel space, by each PHP request, for the load estimation in the front-end. We attempted measuring them with direct measurements, but the precision is very poor, because the minimum CPU time, given by a system call called from the PHP script, had resolution of the same order than the execution time itself. Our solution was to design microbenchmarks using functionality from the EBs implementation [2], namely to have them generate specific interactions, in order to exercise each of the interactions separately.

The methodology for the microbenchmarks woks as follows. During a period T seconds, N_r requests type r are issued and the CPU achieves an utilization U. This way, the average CPU time t_r for request r is $\frac{UT}{N_r}$. However, there is a restriction. The TPC-W benchmark specifies a transition diagram with the possible set of transitions allowed after one specific web interaction, and thus, it is not possible to generate all kinds of interactions in isolation. For example, the request to display an order the client has made cannot be issued before the customer actually asks for that order. Similarly, the *Buy Confirm* interaction cannot happen before the *Buy Request* interaction. For the cases with this type of precedence restriction, we used the average value of the precedent interaction to calculate the average CPU time of the next interaction. In a sequence of n interactions, the CPU time of interaction r_i, say t_i, is given by $\sum_{i=1}^{n} N_i t_i = UT$.

The average value measured by this methodology, with $T = 20$ minutes is shown in Table 2. This resulted in about $10,000$ interactions in each measurement. The scripts *admin_confirm* and *admin_request* could not be determined with precision because they are not requested very often. In a 20-minute experiment, only 250 such interactions occurred, along with $40,000$ other precedent interactions. In fact these interactions are not important, because typical customers do not change or administer the database. Their CPU time, though, is approximately 4 ms, measured directly inside the script for one execution. Again, this measurement is not precise because the granularity of the time function used is 4 ms.

Table 2. Average CPU time (system + user) for each PHP script

PHP script	avg. time (ms)	PHP script	avg. time (ms)
admin_confirm	–	new_products	5.417
admin_request	–	order_display	5.456
best_sellers	5.578	order_inquiry	4.126
buy_confirm	6.929	product_detail	4.643
buy_request	6.039	search_request	4.576
customer_reg	4.242	search_result	5.406
home	5.012	shopping_cart	5.336

5.3. Request Distribution

As the PHP application depends on the session ID that the server generates and writes in the browser cookies, requests with the same session ID must go to the same server. This is implemented by the mod_backhand software, and is commonly called as a distribution with *sticky sessions*. The web request distribution adopted is based on current load, that is, the amount of work outstanding at the server. The web request is sent to the web server with lowest load, providing that the sticky session rule is not violated. The front-end estimates the load of each server as follows: for each web request, the average CPU time is added to the load estimator when the request arrives at the front-end, and the same value is subtracted after sending the response to the client.

5.4. On/Off Policy

The policy used to turn servers on and off affects the QoS control limiting the maximum load of the system and determining the moment to turn a node on, as in [23]. The difference is that we use suspend to RAM, and Wake on LAN and therefore we needed to adopt new values of overhead of time and energy when turning a machine on and off. In Figure 8 the activity line is the output of one parallel port pin measured by the same data acquisition system used to measure the power (in other words, clock skew is zero). A process is started at the same time of the command to shut down the machine ($t = 4$), switching this output. After that, any state different than switching (black part) means that the machine

is not operational. It can be seen in the plot that the time overhead to turn off is the period between 7 and 10 seconds. Similarly, the time to turn on goes from 18 to 24 seconds.

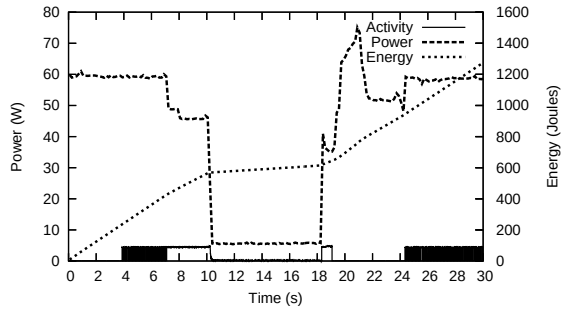

Figure 8. Overhead of time and energy for turning on/off the Pentium M server

6. Performance Evaluation

Before evaluating the proposed method, we will show empirical proofs for the impracticality of controlling the QoS using a direct measure of the QoS. The plot on Figure 9 shows the QoS being measured in a sliding window of size sufficient to store 10 seconds of web interactions information about whether it met or missed the deadline. This size was the biggest size that showed not to compromise the responsiveness of the control. The control set-point was set to 0.98, shown in the plot as a reference line. The plot also shows the control output (u in Figure 4) for the two cases: based on the direct QoS measure (sliding window), and based on the tardiness measure (with Pareto distribution).

The first of two problems of measuring the QoS is the broad confidence interval. The confidence interval in this experiment, not shown in the plot, resulted in values up to 0.06. For 0.98, for example, the confidence interval is 0.04, meaning that the real mean will lay between 0.96 and 1.0. For this reason, as can be observed in the plot, more often than not the QoS measure assumes the value 1.0 (for example, between $t = 370$ and $t = 470$), even though the real mean value (not the sample average) is something different, resulting in instability.

The second problem is that the maximum value of QoS is 1.0. The plot shows several intervals (e.g., $370 < t < 470$ and $t > 550$) where the measured QoS is bigger than the set-point 0.98, giving an error limited to 0.02, resulting in a long decreasing output, because 0.02 is too small. After this period, in most cases the output reached a position that caused an error much larger than 0.02 (e.g., $t = 250$, $t = 350$, and $t = 530$), resulting in a fast increasing of the output. On the other hand, the curve for the output based on tardiness shows a more constant behavior, and there is no asymmetry related to the set-point. Furthermore, the QoS measured in a sliding window during the control with tardiness is more constant, although higher than 0.98, because of the broad confidence interval. As a result, the control based on the direct QoS measure gives periods of high probability of meeting the deadline, followed by periods where it is more likely of missing the deadline than the previous period. Even though the final accumulated QoS for the whole experiment were correct for the two cases (close to 0.98), what is expected is that every web interaction have the same probability of meeting its deadline, uniformly, and the use of tardiness achieves this goal. The energy consumption is higher in the case of controlling the QoS without tardiness, because of the higher variability of the output u.

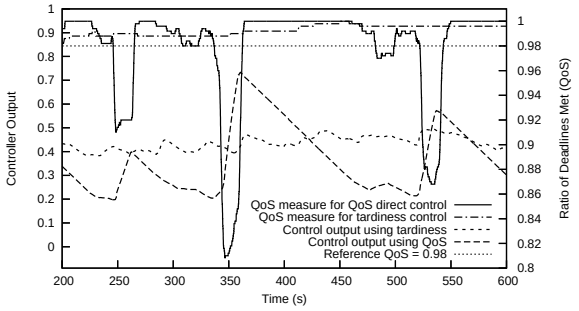

Figure 9. Control using direct QoS measure

The most important evaluation we made is to prove the correctness of Theorems 4.1 and 4.2, for the Pareto and Log-normal distributions. We executed the tests with 360 EBs, a number that represents half of full load and requires 4 servers turned on, divided equally into the three database servers and monitored the QoS obtained for each value of specified QoS. The obtained QoS (accumulated) was measured by the ratio $\frac{missed\ deadlines}{total\ requests}$ for each class of web interaction, and the tardiness values were from the web interaction class with the minimum QoS. In other words, the controller is directed to control the worst QoS among all classes of web interactions. Although conservative, this is to guarantee that all web interactions will stay with a QoS above the specified limit, as it is stated in the TPC-W specification.

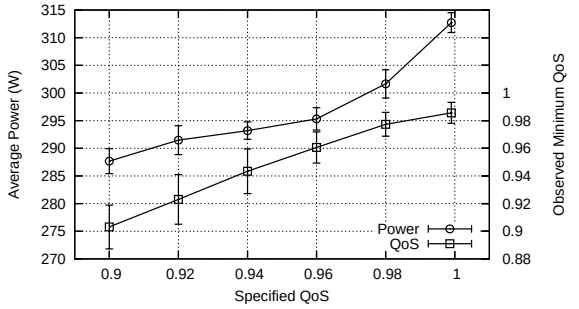

Figure 10. Evaluation of the Pareto distribution

The plots in Figure 10 and Figure 11 show average power and the minimum QoS obtained by our scheme as a function of the specified QoS when using Pareto distribution and Log-normal distribution, respectively. The confidence interval plotted is obtained in each measure by the confidence in-

terval for a proportion, given by $\pm 1.96\sqrt{\frac{p(1-p)}{N}}$, for a 95% interval, where p is the proportion, or the QoS measured.

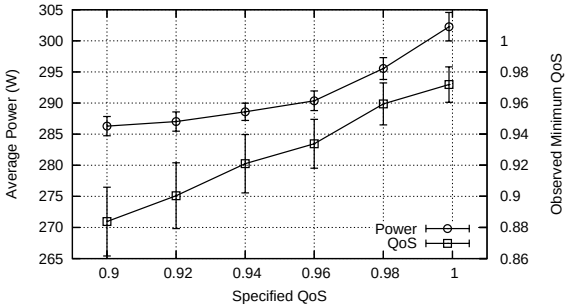

Figure 11. Evaluation of the Log-normal distribution

The Pareto distribution showed very accurate results for QoS values not close to 1.0. The Log-normal showed an error approximately constant of 0.02, and was consistently worse for all values. This is because the Pareto distribution has a better goodness of fit for the tail, which contains most of the requests with missed deadlines. On the other hand, the log-normal distribution had the worse fit exactly in both tails.

Both models, based on Pareto and on Log-normal, have some difficulty to be correct for QoS close to 1.0, as it can be expected examining the theorems. The points in both plots (Figures 10 and 11) close to 1.0 were actually user specified QoS of 0.999. This happens because in the case of QoS 1.0 the distributions will have no tail at all.

The TPC-W specifies 0.90 for QoS. Normally, when using the TPC-W to measure an e-commerce system performance, the number of items in the database must be scaled up until the server has minimum QoS of 0.90, and it is found the maximum scaling factor that the system under test can sustain. Thus, to get TPC-W results the system must be in full load. Our system, when not at full load, will slow down to stay in a similar condition of load with the accurate QoS of 0.90, and consequently will reduce the energy cost.

We compared the results, with QoS control based on Pareto, with the implementation in [23]. We made some few modifications in that implementation to accommodate the new real-time model and to support the bigger number of request types. The first result is shown in Figure 12, where the TPC-W test was executed for 30 minutes, with a load of 400 EBs. The QoS in the proposed scheme was set to 0.95 ([23] also had a target QoS of 95%). The QoS for [23] is not plotted, because, for this load, the QoS remained very close to 1.0 for all requests. The average power for the scheme proposed in [23] was 320.9 W, while it was 303.2 W for our scheme. This shows that our scheme can accurately specify QoS in a fine-grain manner.

We also compared the new scheme with [23] for several different loads (see Figure 13), using the specified QoS of 0.95, the same as in [23]. The experiments show that we can save power by having an accurate control of QoS.

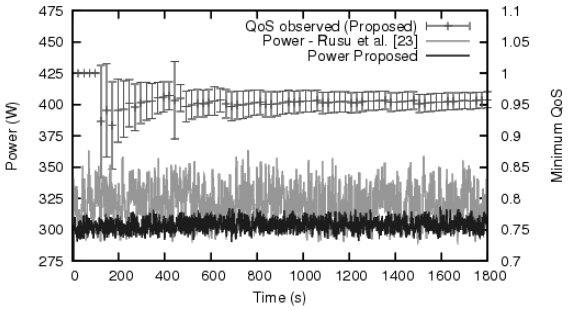

Figure 12. QoS and power in the TPC-W test

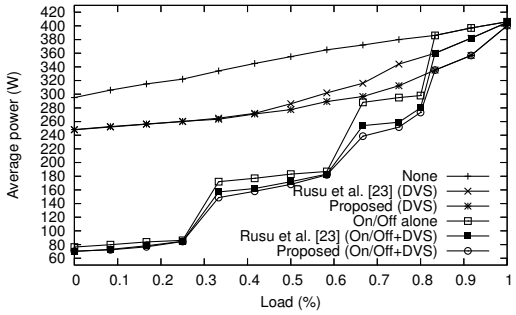

Figure 13. Evaluation of the proposed scheme

In Figure 14 we show that, even though we are focusing on the web server layer, the energy consumption of the web servers depends on the load of the database layer. We executed the same load in two different scenarios. In the first, all clients were directed to only one database, and in the second the clients were distributed to the three available databases. In the first scenario the database showed almost full utilization, against about 30 percent in the other option. When the load at the databases is higher, the web server layer has to speed up to compensate the response time increase at the database layer. Thus, the question arises on how to cleverly integrate the power management among the different tiers in a multi-tiered architecture. Preliminary data suggests that the configuration with 3 DB is better; this evaluation is outside the scope of this paper and left for future work. Figure 14 also shows the QoS obtained for both scenarios. We omitted the confidence intervals for better clarity, because they were superposed. It is important to note that both stayed close and above the QoS specified at 0.95.

The recomended performance metric by TPC-W is WIPS, which we measured for our proposed scheme and for the scheme proposed in [23]. For the experiment shown in Figure 12, the averages were 49.79 and 54.99 WIPS for our proposed and for [23], respectively. We had 10.4% less performance, but with a controlled quality of service in a value that attends the minimum level specified by TPC-W for achieving customer satisfaction. We used 95% in this experiment, but we can achieve even more power savings with the TPC-W requirement of 90%. The scheme proposed in [23] achieved better performance because of overprovisioning the system with respect to the real-time specifications.

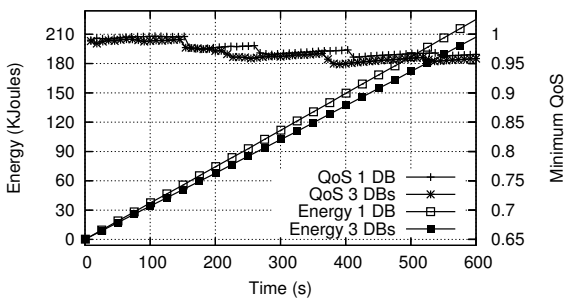

Figure 14. QoS and energy consumption for two scenarios with different database load

7. Conclusions and Future Work

In this work we presented a scheme to relate QoS to tardiness in a multi-tiered e-commerce environment, based on the statistical distribution of the tardiness of web interactions. This QoS metric was shown to be very useful because some practical difficulties arose when we tried to use the measured QoS in the control. On the other hand, tardiness is a continuous value that can be calculated for each web interaction, and its value depicts how close the execution was to the deadline.

We proposed two approaches, based on the probability density function adopted to represent the tardiness data: using the Pareto distribution and using the Log-normal distribution. We showed that the Pareto distribution achieves better results in the accuracy of the resultant system QoS, for values of user defined QoS not close to 1.0, and Log-normal showed to have a constant error due to differences in the fit of the data to the distribution. Our proposed scheme using Pareto was shown to be better than existing schemes like [23] and [24], because it meets with precision the real-time specification, not overprovisioning the system, and thus saving energy. A shortcoming of our approach is when the goal is to meet all deadlines, the tardiness would have an upper bound of 1, and thus the assumption on the tail distribution does not hold. In this case, the cited existing schemes would be more precise.

We plan to extend this work to focus also on the database layer, studying integrated DVS and On/Off schemes in a distributed database. As we have shown, having more load at the database layer causes the power consumption at the web server layer to increase, and thus we need to answer the question on how to minimize the overall aggregate power when all layers are DVS and On/Off enabled.

References

[1] http://www.tpc.org/. Transaction Processing Performance Council.

[2] http://pgfoundry.org/projects/tpc-w-php. TPC-W PHP Implementation Project Page.

[3] http://www.backhand.org. The Backhand Project.

[4] http://www.spss.com/. SPSS Statistical Analysis Software.

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

[6] L. Bertini, J.C.B. Leite, and D. Mossé. 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 Systems and Networks (FeBID)*, Munich, Germany, May 2007.

[7] R. Bianchini and R. Rajamony. Power and energy management for server systems. *IEEE Computer*, 37(11):68–74, 2004.

[8] J. Chase, D. Anderson, P. Thakur, and A. Vahdat. Managing energy and server resources in hosting centers. In *18th Symp. on Operating Systems Principles*, pages 103–116, October 2001.

[9] J. Chase and R. Doyle. Balance of power: Energy management for server clusters. In *Eighth Workshop on Hot Topics in Operating Systems*, May 2001.

[10] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gauta. Managing server energy and operational costs in hosting centers. In *ACM SIGMETRICS*, pages 303–314, 2005.

[11] A. M. K. Cheng. *Enterprise Information Systems II*, chapter E-Commerce and its Real-Time Requirements: Modeling E-Commerce as a Real-Time System. Kluwer Academic, 2001.

[12] M. E. Crovella and A. Bestavros. 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, May 1996.

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

[14] L. C. DiPippo, V. Fay-Wolfe, L. Nair, E. Hodys, and O. Uvarov. A real-time multi-agent system architecture for e-commerce applications. In *Intl. Symp. on Autonomous Decentralized Systems*, pages 357–364, March 2001.

[15] M. Elnozahy, M. Kistler, and R. Rajamony. Energy conservation policies for web servers. In *4th USENIX Symp. on Internet Technologies and Systems*, March 2003.

[16] A. Eyal and T. Milo. Integrating and customizing heterogeneous e-commerce applications. In *Workshop on Technologies for E-Services (TES 2000)*, pages 16–38, September 2000.

[17] T. Hill and P. Lewicki. *Elementary Concepts in Statistics*, chapter Distribution Fitting. StatSoft, Inc., 2006. WEB: http://www.statsoft.com/textbook/stathome.html.

[18] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *ISLPED '98: Intl. Symp. on Low power electronics and design*, pages 197–202, 1998.

[19] C. Lu, J. A. Stankovic, S. H. Son, and G. Tao. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Syst.*, 23(1-2):85–126, 2002.

[20] T. Mudge. Power: A first-class architectural design constraint. *IEEE Computer*, 34(4):52–58, April 2001.

[21] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Dynamic cluster reconfiguration for power and performance. In *Compilers and Operating Systems for Low Power*. Kluwer Academic Publishers, 2003.

[22] K. Rajamani and C. Lefurgy. On evaluating request-distribution schemes for saving energy in server clusters. In *IEEE Intl. Symp. on Performance Analysis of Systems and Software*, March 2003.

[23] C. Rusu, A. Ferreira, C. Scordino, A. Watson, R. Melhem, and D. Mossé. Energy-efficient real-time heterogeneous server clusters. In *IEEE Real-Time and Embedded Technology and Applications Symp.*, April 2006.

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

[25] S. H. Son and K.-D. Kang. Qos management in web-based real-time data services. In *Proc. of the Fourth IEEE Intl. Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS)*, pages 129–136, 2002.

[26] S. Subramanian and M. Singhal. Real-time aware protocols for general e-commerce and electronic auction transactions. In *ICDCS Workshop on Electronic Commerce and Web-Based Applications*, pages 65–70, May 1999.

[27] M. Wang, N. Kandasamy, A. Guez, and M. Kam. Adaptive performance control of computing systems via distributed cooperative control: Application to power management in computing clusters. In *Intl. Conf. on Autonomic Computing*, pages 165–174, June 2006.