SISO PIDF Controller in an Energy-efficient Multi-tier Web Server Cluster for E-commerce

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

Daniel Mossé
Department of Computer Science
University of Pittsburgh
Pittsburgh PA, USA
Email: mosse@cs.pitt.edu

***Abstract*—In this paper we describe a simplified way to implement performance control in a multi-tier computing system designed for e-commerce applications. We show that the simpler SISO (*Single Input Single Output*) controller, rather than a more complex distributed or centralized MIMO (*Multiple Input Multiple Output*) controller, works well, regardless of the presence of multiple cluster nodes and multiple execution time deadlines. Our feedback control loop acts on the speed of all server nodes capable of dynamic voltage scaling (DVS), with QoS (*Quality of Service*) being the reference setpoint. By changing the speed, we change the position of the *p*-quantile of the tardiness probability distribution, a variable that enables to measure QoS indirectly. Then, the control variable will be the average tardiness, and the setpoint the tardiness value that will position this *p*-quantile at 1.0, value at which a request finishes exactly at the deadline. Doing so will guarantee that the QoS will be statistically *p*. We test this new Tardiness Quantile Metric (TQM) in a SISO PIDF control loop implemented in a multi-tier cluster. We use open software, commodity hardware, and a standardized e-commerce application to generate a workload close to the real world. The main contribution of this paper is to empirically show the robustness of the SISO controller, presenting a sensibility analysis of the four controller parameters: damping factor *zeta*, derivative filter factor *beta*, integral gain *ki*, and zero time constant *tau*.**

I. Introduction

As people increase their trust on Internet means for services like banking and commerce, electronic applications become everyday more popular and widespread. The complexity of the computing systems for these applications are increasing fast, both for well established popular kind of applications such as e-banking and e-commerce, and also for less known business-to-business applications, such as e-sourcing, where businesses auction the willingness to purchase from the seller who can offer lowest prices and best contracts. Due to the needed complexity and size, computing systems are becoming complicated, dense, and of high cost of ownership. As pointed out in [1], because of this growing complexity, the computing systems for today's applications need to be able to do self-configuration and self-optimization, and act in an autonomic way, such that it can optimize itself seamlessly to the desired performance objectives. With the motivation that control theory will play a crucial role in the development of complex and large scale computing systems, we present in this paper a practical use of control theory for multi-tier clusters to host e-commerce and related applications.

Following the work in [2], where the authors discussed the scaling aspects of control problems that arise in large computer systems, our control borrows some characteristics from the centralized MIMO (*Multiple Input Multiple Output*) models. They used as a target architecture a multi-tier e-commerce system composed of multiple layers of web clusters, each layer used to process a different part of the web request, namely, request distribution (layer 1), static and dynamic requests (layer 2), and database access (layer 3). In their classification, for any performance control, an e-commerce system has to be either MIMO centralized, where there is a centralized controller with multiple actuators and multiple sensors, or MIMO distributed, with several distributed independent controllers. The authors claim that the controller for an e-commerce system has to be MIMO by necessity, for example, because of the existence of multiple web request types with different response time objectives. However, in our practical implementation of a multi-tier e-commerce web cluster, the industry standard e-commerce application used presented some restrictions that make it impracticable to read the control metric from the multiple servers. The reason is that the information, or control metric, is distributed across the cluster, and the only way to measure it is at the front-end server where the controller runs. This prompted us to build a SISO *Single Input Single Output* controller, using a normalized response time among classes of requests to obtain a single control metric that normalizes the several different time constraints.

In this paper we show a real implementation of an e-commerce computing system based only on open source software and industry standard workloads. Open-source software offers a huge advantage for controlled computing systems, because virtually any metric or measurement can be derived from the system, as we have total access to the source code, from the core kernel level to the application user level. Our objective is to accomplish energy consumption minimization and QoS (*Quality of Service*) guarantee. We build a feedback control loop that regulates the performance

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.

of all dynamic voltage scaling (DVS) capable server nodes (i.e., layers 2 and 3), with QoS being the reference control objective. But rather than sensing the QoS directly, which is measured as a ratio of number of requests that executed within their deadlines to the total number of requests, we use a new metric of QoS based on the tardiness of the completion of web requests proposed in [3], where tardiness, the control variable, is defined as the ratio of web request response time to the deadline. This metric is based on the probability distribution of tardiness, and because it presents more information about the completion of tasks than the QoS, it offers a better metric for using in a feedback control loop.

We will apply the theory of a PIDF controller, which is basically a proportional-integral-derivative (PID) controller augmented with a low pass filter (F) in the derivative part. The workload of a web system is a composition of random variables, and consequently, present the random fluctuations that is characteristic of any stochastic process. We consider the unpredictability of the workload as being similar to sensor noise. With the low pass filter, the process disturbance caused by random oscillation will be rejected by the controller. In such a web system, it is desirable to have the derivative component, because as the plant dynamic presents a dead time delay, it is important to have the predictive characteristic given by the derivative part. Besides, we need also to include averages in the control variable to handle the intrinsic randomness. We will measure the plant dynamics after the inclusion of the averages and apply some tuning rules for the controller.

Our main contribution is the practical implementation and robustness evaluation of the control loop for a real e-commerce web server cluster, with sensitivity analysis to the parameters of the PIDF controller. The workload is generated by an e-commerce benchmarking industry standard. We show the solution for some practical issues, such as the difficulty in measuring the end-to-end delay of e-commerce requests that are defined as a sequence of smaller web requests that can be serviced in a distributed way or in parallel inside the cluster.

In this paper, Section II presents some concepts related to the cluster model, workload generation, the control input metric, and the DVS based actuator mechanism. In Section III we derive the controller equations. Section IV presents evaluation results, and in Section V we discuss the implementation and compare with other similar implementations.

II. Background

Our goal was to deploy a cluster environment to serve as a testbed for e-commerce applications, specifically to test energy-efficient policies. In this section we present the cluster model, the industry standard TPC-W used to create the e-commerce environment, the statistical inference method adopted to measure the control variable, and the DVS policy used. For more details see [3].

A. Cluster Model

The cluster architecture is composed of a central web server that serves as a front-end to the whole system (layer L1), a layer L2 of servers to process dynamic and static requests, and the L3 layer to execute a distributed database that will store all the information related to the application. The front-end node implements a request distribution policy based on the amount of work that each second-tier server has. The front-end server acts as a reverse proxy, that is, it redirects requests to other servers and also returns the server's response to the client. The front-end is capable of SSL encryption/decryption as required for the e-commerce application. The load distribution among the database servers is done statically. We replicate the web store in many independent database servers to avoid bottlenecks, and the total load is divided equally to each database. To implement this architecture we used in layer L1 the Apache web server with the module *backhand* [4] for load balancing and a new module to implement the controller, in layer L2 we have Apache with PHP scripting language support for the dynamic pages, and in L3, PostgreSQL for the databases.

B. Workload Generation

The TPC-W standard [5] is a transactional web benchmark where the workload is performed in a Internet commerce environment. The workload is generated by a software entity that runs in the local network, outside the cluster. It is responsible for managing the emulated browsers (EB) and the emulated sessions. Each EB is a thread implemented in Java that makes access the web server, with HTTP and HTTPS connections, emulating a real customer performing some browsing, searching and purchases.

The performance metric defined 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 book store, and each interaction has a different time constraint and a specified QoS (as a percentage of requests that do not violate the time constraint). For a good review about the TPC-W benchmark see [6].

In the TPC-W, one web interaction is defined as a sequence of one HTTP dynamic request followed by many static requests. The time constraint is related to the end-to-end execution time of a whole web interaction, from the arrival of the dynamic request to the time the server sends the last byte of the last static request. This specification prohibits to measure the control metric from a single server in isolation, because as soon as the client receives the response for the dynamic request, the client will issue many requests for the static requests, and these requests may be serviced in a distributed way and in parallel. This restriction guided us to implement a SISO controller, because the information is located at a centralized location.

C. Controller Input Metric

The input metric to the controller is based on the tardiness of a web interaction. For each web interaction i, we define tardiness by the ratio *web interaction response time* (WIRT) to the respective deadline. That is, $tardiness_i = \frac{wirt_i}{deadline_i}$.

Doing this, we normalize all tardines values from all web interactions in only one measure.

As the goal is to control the QoS, not tardiness, we need a translation from tardiness to QoS. We implemented a statistical model based on the probability distribution for the workload. To do this, we make the assumption that the workload has a Pareto distribution. For specific probability distributions, the relation between the tardiness and the QoS can be obtained analytically. We show briefly the expression for the Pareto distribution in Equation 1 (we show demonstrations and also tests of goodness of fit in [3]). The assumption that web traffic presents a Pareto distribution is common. For example, in [7] it has been shown that the commonly assumed model for Web traffic based on Poisson distributions and Markovian arrival processes does not hold in practice, but rather they present the statistical characteristic of self-similarity, which is the property that the appearance of an object is always the same if looking at any scale. They showed that web traffic, such as response time, can be modeled using heavy-tailed probability density functions, such as Pareto.

The Pareto probability density function is given by $f(x) = k\frac{x_m^k}{x^{k+1}}$, where k is related to the average μ by $\mu = \frac{kx_m}{k-1}$, and x_m is the positive minimum possible value of X. As tardiness has a minimum value of 0, we use $x_m = 1$ and use $x + 1$ to locate the function. Then we obtain $f(x) = \frac{k}{(x+1)^{(k+1)}}$ for the tardiness probability density function, where $k = \frac{\mu+1}{\mu}$.

To relate the tardiness with QoS with a known distribution, we need to calculate the *p*-quantile and make it equal 1.0, the tardiness value after which a web interaction will miss its deadline, so that the probability to miss a deadline will be $1 - p$, and the QoS will be p. This allows to relate the mean μ with the value of p, as follows:

$$\mu = \frac{1}{log_2\left(\frac{1}{1-p}\right) - 1} \tag{1}$$

With Equation 1, we have a statistical inference method to relate a QoS setpoint to a tardiness setpoint.

D. DVS Actuator mechanism

The actuator of the control system is based on dynamic voltage scaling (DVS). Changing the voltage and frequency of all L2 and L3 servers, we can speed up the system, pushing the average tardiness to values closer to zero or slow down the system, resulting in bigger average tardiness.

Our goal is to maintain the voltage/frequency at the lowest level that maintains the QoS at the specified level. Because the controller outputs a continuous value and because every DVS capable processor has discrete levels of voltage and frequency, we adopted a periodic switching DVS scheme to match the speed of the continuous actuator. Our scheme consists of switching between the two discrete values adjacent to the desired continuous value, as proposed in [8]. To implement this scheme, a high priority daemon executes periodically with a duty cycle α.

To implement a controller with single output, we used a frequency scaling factor u output by the QoS controller, which is multicast to all L2 and L3 servers, and each server node i calculates its 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.

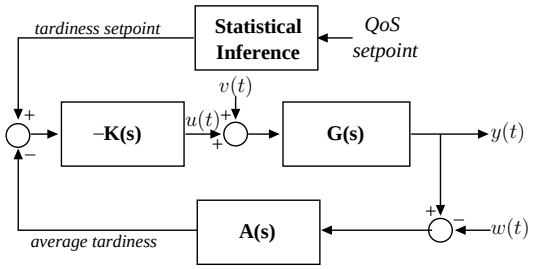


Fig. 1. Control logic block diagram

III. Control Logic

Figure 1 shows the control logic block diagram adopted. As suggested in [9], we model the noise as the input signal $w(t)$; in our model, noise is present in the measure because of the stochastic nature of the workload $v(t)$ (the process disturbance), which will cause the randomness present in the tardiness measure. The controller output is $u(t)$, and the transfer function $K(s)$ of the controller has a minus because it has to invert the output related to the input error. When the error is negative, the *p*-quantile for the QoS p is bigger than 1.0, and the deadline miss ratio is bigger than $1 - p$, and therefore the server must increase the speed. $G(s)$ is the unknown plant transfer function; we will measure its dynamics in Section IV-B. $A(s)$ represents the averaging included in the control variable.

We have used in [3] a simple PID controller given by $K(s) = k_P + \frac{k_i}{s} + k_D s$. To improve it, as suggested in [10], we insert a lowpass filter in the derivative part to make it reduce the noise, and we change the parametrization of the controller as proposed in [10]. With only the lowpass filter, the controller becomes: $K(s) = k_P + \frac{k_i}{s} + \frac{k_D s}{1+sT_f}$. The new parametrization will use the four parameters: dumping factor (ζ), derivative filter factor (β), integral gain (k_i), and zero time constant (τ). The advantage of using these parameters is better stability, because it reduces the freedom of the traditional parameters in a way that the controller is easily kept in a stable region. This parametrization also makes the controller tuning procedure easier. The resultant controller is:

$$K(s) = k_i \frac{1 + 2\zeta\tau s + \tau^2 s^2}{s\left(1 + s\frac{\tau}{\beta}\right)} \tag{2}$$

where $\beta = \frac{k_\infty}{\tau k_i}$, and $k_\infty = \lim_{s\to\infty} K(s)$.

The damping factor ζ dictates the responsiveness of the controller. With a increased ζ, the system becomes slower to achieve steady state, and with a small ζ, the overshoot increases. The zero time constant τ is dependent on the plant dynamics. In [9] a very simple method of tuning the controller is to make $\tau = \frac{T}{3}$, where T is the time constant of the plant

(for a first order plant, the time the output takes to achieve 63.2% of the input in the step response). The filter factor β is related to the high-frequency gain, or control activity, $k_\infty = \beta\tau k_i$. If β is small, the system may lose control activity and perform as if in a positive retrofit (see Section IV). Increasing k_i will increase the performance of the controller.

For the controller, we implemented Equation 2 in the discrete domain. We used the backward difference, given by $1 - sT_s = z^{-1}$, that is obtained from a first order series approximation to the $z-$transform, with T_s being the sampling period. The controller equation relating the discrete output u_k to the discrete error e_k becomes:

$$K(z) = k_i \frac{T_s + 2\zeta\tau + \frac{\tau^2}{T_s} - \left(\frac{2\tau^2}{T_s} + 2\zeta\tau\right) z^{-1} + \left(\frac{\tau^2}{T_s}\right) z^{-2}}{T_s + \frac{\tau}{\beta} - \left(T_s + 2\frac{\tau}{\beta}\right) z^{-1} + \left(\frac{\tau^2}{T_s}\right) z^{-2}} \quad (3)$$

The discrete equation obtained by straightforward manipulation of Equation 3 is in the recurrence formula in Equation 4.

$$u_k = \frac{(\beta T_s + 2\tau)\, u_{k-1}}{\beta T_s + \tau} - \frac{\tau^2 k_i \,(u_{k-2} - e_{k-2})}{T_s \left(T_s + \frac{\tau}{\beta}\right)} + \frac{\left(T_s + 2\zeta\tau + \frac{\tau^2}{T_s}\right) k_i e_k}{T_s + \frac{\tau}{\beta}} - \frac{\left(\frac{2\tau^2}{T_s} + 2\zeta\tau\right) k_i e_{k-1}}{T_s + \frac{\tau}{\beta}} \quad (4)$$

IV. Evaluation and Sensitivity Analysis

In this section we present a set of experiments with the controller. The first step is to measure the process dynamics in open loop and then tune the controller accordingly. We adopted the tuning procedure given by Equation 11 of [9]. For the closed loop, all tests use a QoS setpoint of 0.95.

A. Process Dynamics

We adopt the first order plant with delay $G(s) = k\frac{e^{-sL_d}}{1+sT}$, where L_d is the lag delay, or the time it takes for the output to change after a step response, and T is the time constant.

We will show results (how the process dynamics change) for two different time windows for computing average tardiness (which are also the sampling period T_s). We will use an average of 10 seconds plus an additional filter with constant $T_f = 10s$, and to test bigger averages, we use window average of 30 seconds plus an additional filter with constant $T_f = 30s$ (a sampling period and average of 30 seconds was also used in [11]). This lowpass filter in the measurement is required for smoothing and improving the measurement of the control variable. With this averaging scheme implemented, we measured the step response for both cases, and the result is in Figure 2. We will use this figure in the next section for fitting with the plant model adopted.

B. Tuning

We did curve fitting from the results in Figure 2 to extract the parameters of the plant model. We obtained $L_d = 10s$, $T = 12s$, and $k = 0.35$ for the $10s$ case and $L_d = 30s$, $T = 36s$, and $k = 0.33$ for the $30s$ case. Applying these

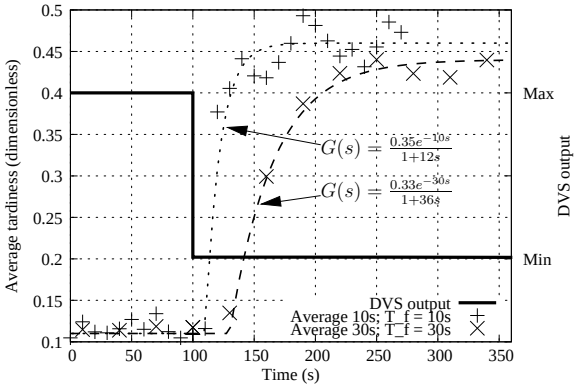


Fig. 2. Step response in open loop, for $10s$ average with $T_f = 10$ s and $30s$ average with $T_f = 30$

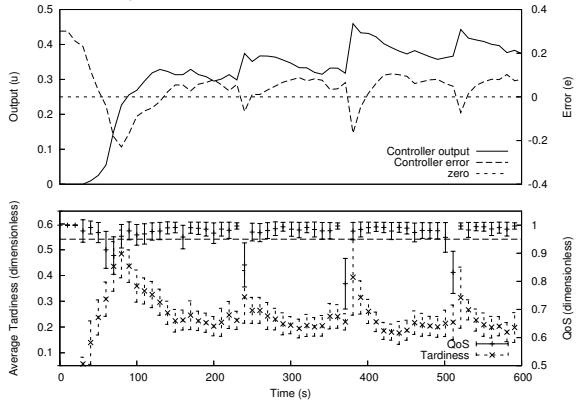


Fig. 3. Control performance with $10s$ average

values to the tuning rule described by Equation 11 of [9], we obtain $\zeta = 0.83$, $\tau = 6.52$, $k_i = 0.29$, and $\beta = 3.91$, for $10s$ case, and $\zeta = 0.83$, $\tau = 19.56$, $k_i = 0.10$, and $\beta = 3.68$, for the $30s$ case. The study in [9] showed that these values yield closed-loop behavior close to optimal, for first order plants with moderate time delay. In our case, with $10s$ delay resulted in good stability, but a $30s$ delay was too large and did not yield good results (see Section IV-C).

C. Results

The experimentation results are shown in Figures 3, 4, and 5. In all experiments, the control variable used is not only the average tardiness, but the average tardiness added to the confidence limit calculated every sampling interval. For example, if in one given sampling interval the average tardiness measured with its confidence interval is 0.30 ± 0.05, the control variable will be 0.35 rather than 0.30. This is to guarantee, with the confidence level adopted (95%), that the QoS will lay above the specified value.

In Figure 3, the tuning rules resulted in stable operation of the controller with $10s$ average. The QoS measured every interval remained above, in most cases, the specified value of 0.95, as expected, because we controlled by the confidence limit. The points close to $t = 240s$, $t = 380s$, and $t = 510s$ with low QoS were caused by load imbalancing that is difficult to avoid when all servers run almost with full utilization.

Figure 4a shows the $30s$ case. As the lag delay was too

big, the tuning rules failed. With a too small β, the integral part is not sufficient to recover from a negative error. The effect is of a positive retrofitted system. We solved this by increasing β and increasing k_i, for better performance and better control activity. The result is in Figure 4b, which also shows the increase in control activity with higher β. For the remaining experiments, one parameter will be changed, while the others will remain the same given by the tuning rules.

In Figure 5a, we show that increasing the integral gain k_i, the performance increases. The curve with $k_i = 0.1$ is much slower than with $k_i = 0.3$. However, $k_i = 1.0$ is too big, and resulted in instability.

Figure 5b shows the effect of varying the damping factor ζ. As was expected, an increase in ζ lowers the overshoot of the system, but increases the time to reach the setpoint.

In Figure 5c we show the effect of the parameter τ. The zero constant must be tuned with the plant dynamics. The value $\tau = 6.5$ was the value returned by the tuning rule. We also experimented with $\tau = 3$, which was too small and did not allow the system to correct the positive error, and $\tau = 12$, which caused difficulty in correcting a negative error.

In this work we have not shown any energy measurement because we focused more in the stability analysis and sensitivity to parameters, issues that we could not assess in [3]. In that work we compared the energy consumption with other interval based DVS mechanisms and we showed that extra energy savings can be achieved with the fine-grain QoS control proposed. We did not evaluate, however, the energy-efficiency of the system during the settling time, which will depend on the tuning rules. This is not an important issue because the settling time of 150 seconds, observed in Fig 3, about half the settling time obtained in [11], is sufficiently small to accommodate the workload variation.

V. Discussion and Related Work

Control theory has been used many times, in the last decade, as the solution for performance control in computing systems. A seminal work appears in [12], where the authors change the paradigm of scheduling, applying control theory to maintain the performance of the system stable. Moreover, as pointed out in [1], the computing systems for today's applications will rely on control theory to make systems that can achieve the desired performance objectives.

In this work we followed the general framework for describing control problems presented in [2]. They use a multi-tier e-commerce system as illustration and classify the possible control architectures, including *SISO, MISO*, and *MIMO*, which refer to the number of inputs and outputs of the controller (S = single, M = multiple). MIMO, in particular, can be further divided in centralized and distributed. The authors argue that e-commerce systems are MIMO by necessity, because the target system must have multiple inputs in order to achieve multiple objectives, and must have multiple outputs in order to measure the multiple objectives (see Fig. 6a).

However, although this classification is very reasonable, there are practical issues to implement the e-commerce web

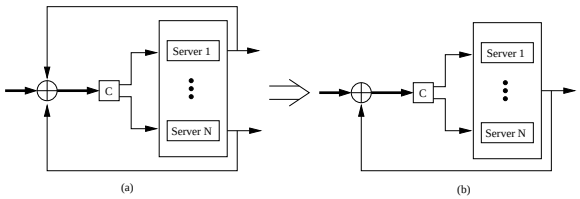


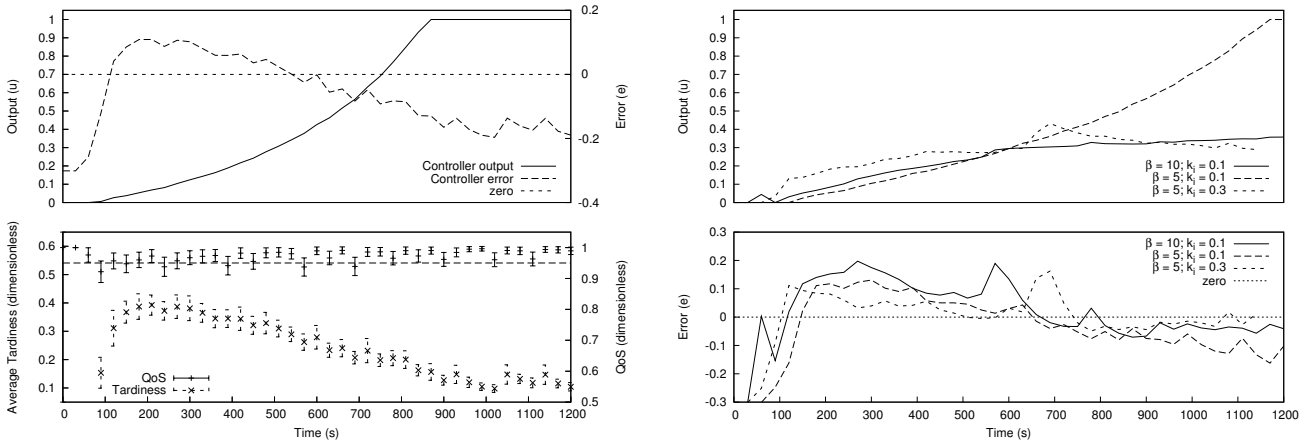
Fig. 6. Comparison with the classification in [2]. (a) The expected MIMO-C controller for QoS control. (b) The simplified SISO controller implemented

system, and it turns out that it is possible to use a simpler SISO architecture, as shown in Figure 6b. As the chosen metric to be used in the controller was the tardiness of web interactions, and because of the definition of web interaction given by the TPC-W standard, the MIMO model is not convenient. The reason is that the TPC-W standard defines a web interaction as a sequence of several HTTP requests, and the real-time requirements in this standard determine that a certain level of QoS must be achieved for the end-to-end service time of each web interaction. Since the metric must account for the whole web interaction, and since each of the HTTP subrequests may be serviced by different L2 server nodes with a certain level of parallelism, it is impossible to obtain the response time at the server nodes. In our implementation, the centralized controller runs in the front-end server, where all requests and responses go through and the end-to-end time is measured.

In [11], different classes of requests are considered. The actuator does not use DVS, but enforces desired relative delays among classes via dynamic connection scheduling and process reallocation, with the goal of providing differentiated services. That work shows clearly the problem of having an unpredictable workload: the sampling period used was $30s$, and the settling time achieved was $270s$, which is the time for the Web server to enter steady state.

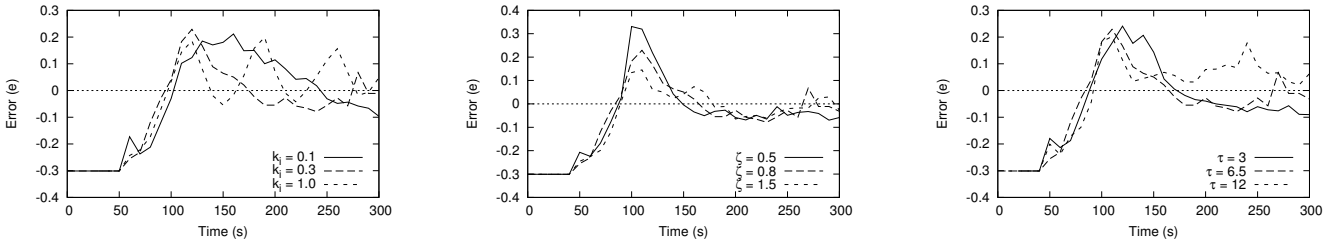
QoS control can also be done by sensing QoS directly [13], [14] rather than by a statistical approach like ours. However, this may be problematic, because the QoS measure will have a saturation point in 1.0 very close to the desired setpoint. This asymmetry can cause instability, as we have shown in [3]. In [13] and [14], they solved this problem with a more complicated control, based on a second control loop for the utilization, and the saturation condition of utilization and QoS was proved to be mutually exclusive. These works use actuators that change the scheduling of the system, performing admission control. They also do not apply DVS.

In [15] the authors 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. However, it differs from our work because their technique is conservative, providing a QoS guarantee always close to 1.0, not controlling QoS at a fine-grain setpoint. Computing systems with utilization control have usually a different goal, which is to enforce a certain utilization by means of admission control, not DVS, to prevent overload conditions. Other recent works in this area are [16], [17], [18], [19].



(a) (b)

Fig. 4. Control performance with $30s$ average



(a) (b) (c)

Fig. 5. Experimentation with parameters k_i, ζ, and τ

VI. Conclusion

In this paper we showed a practical implementation of a feedback control loop in a multi-tier web server system for e-commerce. We used DVS to adjust the system performance to save energy, but with the QoS specification being guaranteed by the control loop. We showed practical issues that arise in the implementation of a controller in a real web cluster application. The experiments showed that the parametrized controller is easy to tune, because tuning has a limited degree of freedom, which helps stability. Our experiments showed an analysis of sensitivity to the controller parameters that can help in achieving the best performance for the controlled system. The fine-grain QoS control showed in this work is useful in achieving extra energy savings for interval based DVS schemes where the goal is to meet all deadlines, avoiding overprovisioning the system according to the real-time specifications.

References

[1] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[2] Y. Diao, J. L. Hellerstein, and S. Parekh, "Control of large scale computing systems," *SIGBED Rev. Special Issue on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID 2006)*, vol. 3, no. 2, pp. 17–22, 2006.

[3] L. Bertini, J. C. B. Leite, and D. Mossé, "Statistical QoS guarantee and energy-efficiency in web server clusters," in *19th Euromicro Conference on Real-Time Systems*, Pisa, Italy, 2007, to appear.

[4] "http://www.backhand.org," the Backhand Project.

[5] http://www.tpc.org/, transaction Processing Performance Council.

[6] J. G. Daniel F. Garcia, "TPC-W e-commerce benchmark evaluation," *IEEE Computer*, vol. 36, no. 2, pp. 42–48, February 2003.

[7] 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 Comp. Sys.*, 1996, pp. 160–169.

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

[9] B. Kristiansson and B. Lennartson, "Robust tuning of PI and PID controllers," *Control Systems Magazine*, vol. 26, no. 1, pp. 55–69, 2006.

[10] P. Falkman, A. Vahidi, and B. Lennartson, "Convenient, almost optimal and robust tuning of PI and PID controllers," in *15th IFAC World Congress*, Barcelona, Spain, July 2002.

[11] C. Lu, Y. Lu, T. F. Abdelzaher, J. A. Stankovic, and S. H. Son, "Feedback control architecture and design methodology for service delay guarantees in web servers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 9, pp. 1014–1027, September 2006.

[12] J. A. Stankovic, C. Lu, and S. H. Son, "The case for feedback control real-time scheduling," in *11th Euromicro Conference on Real-Time Systems (ECRTS)*, York, England, 1999, pp. 11–20.

[13] C. Lu, J. A. Stankovic, S. H. Son, and G. Tao, "Feedback control real-time scheduling: Framework, modeling, and algorithms," *Real-Time Syst.*, vol. 23, no. 1-2, pp. 85–126, 2002.

[14] S. H. Son and K.-D. Kang, "Qos management in web-based real-time data services," in *4th IEEE Intl. Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems*, 2002, pp. 129–136.

[15] 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.*, December 2003, pp. 63–72.

[16] Y. Fu, H. Wang, C. Lu, and R. S. Chandra, "Distributed utilization control for real-time clusters with load balancing," in *27th IEEE Intl. Real-Time Systems Symposium*, December 2006, pp. 137–146.

[17] X. Wang, D. Jia, C. Lu, and X. Koutsoukos, "Decentralized utilization control in distributed real-time systems," in *26th IEEE Intl. Real-Time Systems Symposium*, Washington, DC, USA, 2005, pp. 133–142.

[18] X. Wang, C. Lu, and X. Koutsoukos, "Enhancing the robustness of distributed real-time middleware via end-to-end utilization control," in *26th IEEE Intl. Real-Time Systems Symposium*, Washington, DC, USA, 2005, pp. 189–199.

[19] Y. Lu, T. F. Abdelzaher, C. Lu, L. Sha, and X. Liu, "Feedback control with queueing-theoretic prediction for relative delay guarantees in web servers," in *IEEE Real Time Technology and Applications Symposium*, 2003, pp. 208–218.