Optimizing Quality of Service Using Fuzzy Control

Yixin Diao, Joseph L. Hellerstein, and Sujay Parekh

IBM T.J. Watson Research Center, Hawthorne, NY 10532
{diao, hellers, sujay}@us.ibm.com

Abstract. The rapid growth of eCommerce increasingly means business revenues depend on providing good quality of service (QoS) for web site interactions. Traditionally, system administrators have been responsible for optimizing tuning parameters, a process that is time-consuming and skills-intensive, and therefore high cost. This paper describes an approach to automating parameter tuning using a fuzzy controller that employs rules incorporating qualitative knowledge of the effect of tuning parameters. An example of such qualitative knowledge in the Apache web server is "`MaxClients` has a concave upward effect on response times." Our studies using a real Apache web server suggest that such a scheme can improve performance without human intervention. Further, we show that the controller can automatically adapt to changes in workloads.

1 Introduction

The advent of eCommerce has created demand for high quality information technology (IT) services. For example, a "buy" transaction that takes more than few seconds may cause the customer to abandon the purchase. Hence, businesses seek ways to improve the quality of service (QoS) in a cost-effective way. For example, in the Apache web server, tuning parameters such as `MaxClients` can have a substantial impact on service levels, but the "best" value of the parameters depends on system capacity and workload (which can change over time). Properly adjusting tuning parameters for best values is time-consuming and skills-intensive. This increases the cost of ownership of the system, both in terms of the cost of hiring and maintaining the appropriately skilled personnel as well as the lost opportunity in terms of misconfigured systems.

There has been much interest in using feedback control theory for automating system configuration and performance management. The tuning parameters are desired for automatic, on-line adjustment so as to reduce human intervention. Typically, the feedback controller is used for regulatory control, that is, to keep the performance metric at a desired reference value. Some examples are controlling queue length in Lotus Notes [1], buffer length in Internet routers [2], CPU and memory utilizations in Apache [3], and response times for web service differentiation [4]. Our interest is in generic approaches to optimizing the setting of tuning parameters so as to improve quality of service (e.g., minimize response times). Put differently, regulatory control attempts to divide the "QoS

M. Feridun et al. (Eds.): DSOM 2002, LNCS 2506, pp. 42–53, 2002.

pie" accurately; we want to make the "pie" bigger. In [5] a system is designed to perform on-line optimization of a web server using hill climbing techniques. In [6] the authors describe a method to manage web server resources based on maximizing revenue. In [7] the authors consider maximizing SLA profits for web server farms. However, all of the above approaches require detailed knowledge of the system being optimized (e.g., in order to construct queueing models). A different approach [8] is to use the linear proportional-integral (PI) controller for response time management, but it requires hand-crafted construction for specific workload and profit model.

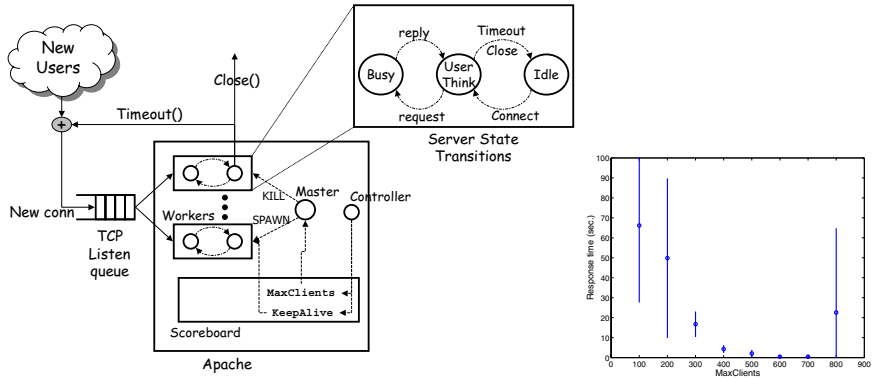
The QoS optimization problem is essentially a nonlinear control problem; hence, linear control techniques as used in [2,3,4,8] may not be appropriate. Among nonlinear control techniques, fuzzy control has been an active area of research with various applications [9]. In particular, fuzzy techniques have been applied to the optimization problem. In [10] the authors derive fuzzy control laws to maximize the profit of a simple queueing system. However, the results are specific to GI/M/1 tandem queues and require prior knowledge of arrival rates. In [11] fuzzy control is applied to maximize profits in service level management. Our current work is an extension of that approach applied to directly managing QoS and applied to a running system rather than a simulation.

In this paper we describe an approach to optimizing quality of service using a combination of feedback control system and qualitative insights into the effect of tuning parameters on QoS. This approach is applicable to a variety of systems as long as they satisfy some simple concave conditions on the effect of the tuning control. For the Apache web server studied in this paper, experimental studies suggest that `MaxClients` has a concave upward effect on the response times, that is, increasing `MaxClients` will reduce the response times at first but further `MaxClients` increases can lead to longer response times. We use fuzzy rules to encode this knowledge, thereby avoiding difficulties with skills-intensive considerations such as building a queueing model (e.g., [10]) or hand-crafting a PI controller (e.g., [8]). A fuzzy control loop is then constructed that interprets the rules to adjust `MaxClients`. It can be proven that such a controller converges to a neighborhood of the best `MaxClients` value. Our studies of a real Apache server under a synthetic workload suggests that our approach can result in considerable reductions in the mean and variance of response times.

The remainder of the paper is organized as follows. Section 2 describes the background of performance management for eCommerce sites with multi-tier servers. Section 3 introduces the fuzzy control architecture and components that automate dynamic performance tuning. The testbed setup and experimental results are shown in Section 4 to demonstrate the performance of the fuzzy control algorithm. Our conclusions are contained in Section 5.

2 System Background

The general situation we consider is the multi-tier system. Web requests (e.g., "browse", "buy", etc.) arrive at a Request Router that selects a Web Server



(a) Apache architecture and session flow. (b) Response time curve.

Fig. 1. Apache response time.

(e.g., by balancing load) to process the request. Most requests are for dynamic pages that require the execution of business logic in the Application Server, such as computing discounts, and so the Web Server routes requests appropriately. These computations in turn often require access to information in a database and hence the request may be further routed to one or more database servers.

Managing the response times of such a system is complex. While some of this is handled by the load balancing done by the Request Router, the Web Servers can aid in this as well. In particular, the Web Servers can control the rate at which requests flow into the remaining tiers by adjusting the number of client processes available to receive requests. For example, in the Apache Web Server, the parameter `MaxClients` is used to exert this control. Moreover, in the system we describe, the Web Server control is extended to use response time information from simulated users (the "RT Probe" elements in the figure) to choose an optimal value of `MaxClients`.

We focus on the Apache web server. Apache v1.3 is structured as a pool of worker processes monitored by a master process [12]. As shown in Figure 1 (a), the master process monitors the health of the worker processes and manages their creation and destruction. The worker processes are responsible for handling the communications with the web clients, including the work required to generate the responses. A worker process handles at most one connection at a time, and it continues to handle only that connection until the connection is terminated. Thus, the worker is idle between consecutive requests from its connected client.

The Apache `MaxClients` parameter limits the size of this worker pool, thereby providing a kind of admission control in which pending requests are kept in the TCP Listen queue. Clearly, we want `MaxClients` to be large enough so that idle resources can be used by in-coming requests, but it should not be so large that resource contention (or in extreme cases, memory thrashing) occurs. The optimal value of `MaxClients` depends on server capacity and the nature of the workload. Typically, this is a difficult choice to make, especially since work-

loads change frequently in commerce sites and so decisions must be made in real time.

To demonstrate the effect of `MaxClients` on response time, we conducted experiments in which the value of `MaxClients` is varied. (The details of the testbed and the workloads are discussed later.) Figure 1 (b) displays the results. The circles indicate the average response times measured at different `MaxClients` values and the vertical lines indicate the standard deviations around the mean. (Note that at `MaxClients`=800 even if all response times are positive, the variation is so drastic that the standard deviation is larger than the mean.) If `MaxClients` is too small, there is a long delay due to waits in the TCP accept queue (see in Figure 1 (a)). Indeed, it is possible that the queue overflows, which causes requests to be rejected. On the other hand, if `MaxClients` is too large, resources become over-utilized, which degrades performance as well. In extreme cases, there may be an internal server error if the limit on the number of processes is exceeded. The combined effect of these factors is that response time is a concave upward function of `MaxClients`, which is confirmed by Figure 1 (b). Indeed, other systems have similar relationship as well, such as the effect of `MaxUsers` on SLA profits in Lotus Notes [11].

Our experimental studies have also verified that for different workloads the `MaxClients` vs. response time curve slope and the optimal value of `MaxClients` are different, even if the shape is still concave upward. In the next section we show how to use a fuzzy controller to automatically converge to a satisfactory value of `MaxClients`. Since the fuzzy controller operates based on qualitative knowledge of the concave upward shape but not the specific curve data, it is not workload sensitive and is able to adapt for workload changes.

3 Performance Optimization with Qualitative Knowledge and Fuzzy Control

Fuzzy control provides a means for expressing control laws as high level rules. Using fuzzy rules for response time minimization can both simplify the controller design (no need for building the model and specifying the reference value as in using the PI controller) and facilitate incorporation of the human expert knowledge on server tuning.

The architecture of our proposed fuzzy control system is shown in Figure 2. The feedback loop operates in discrete time. The first component in the feedback loop is a probing station that measures response time from the end user's perspective. To reduce the effect of measurement noise, multiple probe response times are averaged over a measurement interval. Next in the feedback path is a differentiator whose output is the change in the response time (dy) between the current and previous intervals. Moving further along the feedback loop, there is the fuzzy controller that determines the change in `MaxClients` for the next time interval. The fuzzy controller has two inputs: the change in response time (from the differentiator) and the change in `MaxClients` value (du) between intervals. The controller's output is the change in `MaxClients`for the next interval. An inte-

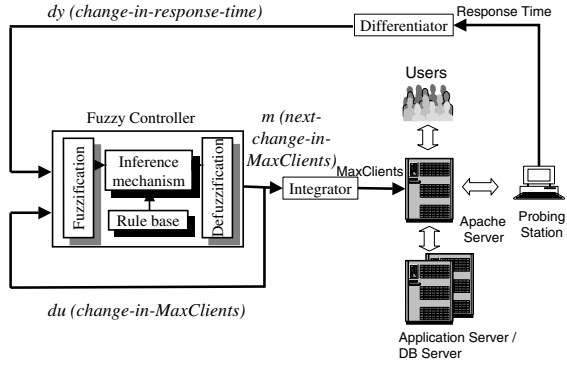


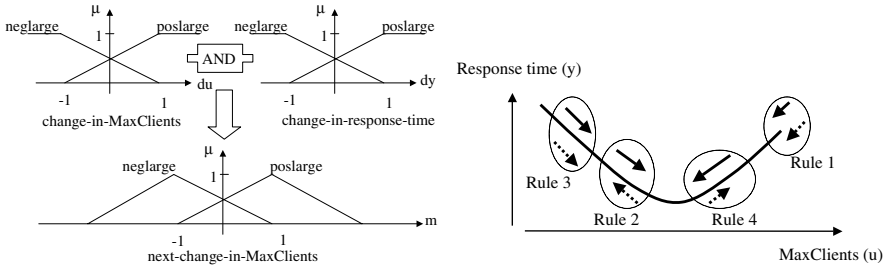
Fig. 2. Architecture of the service-oriented fuzzy controller.

grator element (accumulator) converts this *du* output into an actual `MaxClients` value which is applied to the Apache server.

The actions of the fuzzy controller are guided by a set of IF-THEN rules that are stored in a rule database (or just rule base). For example, "IF *change-in-MaxClients* is *neglarge* and *change-in-response-time* is *neglarge*, THEN *next-change-in-MaxClients* is *neglarge*." The terms *change-in-MaxClients* and *change-in-response-time* are linguistic variables; *neglarge* is a linguistic value. Linguistic variables and values are central to fuzzy control [9]. In particular, linguistic variables are a natural way to handle uncertainties created by the stochastics present in most computer systems.

Linguistic variables exist in one-to-one correspondence with numeric variables. For example, *change-in-MaxClients* is a linguistic variable corresponding to the numeric variable for the change in `MaxClients`. (Similarly for *change-in-response-time* and *next-change-in-MaxClients*.) A linguistic variable takes on linguistic values, such as *poslarge* and *neglarge*. (Note that "*neglarge*" is generally used in fuzzy control literature as an abbreviation for "negative large in size" and so on for others such as "*poslarge*" or "*possmall*").

In order to do control, we convert the value of numeric variables into linguistic values of linguistic variables, and do the reverse as well. A simple approach is to use a triangular membership functions (other functions such as trapezoid or gaussian may also be used), as shown in Figure 3 (a). For example, we map between the numeric variable *du*, the change of `MaxClients` value, and the linguistic variable *change-in-MaxClients*. The y-axis indicates the "degree of truth" for each of the linguistic values *neglarge* and *poslarge*. For example, $du = 0.5$ maps to 0.75 for *poslarge* and 0.25 for *neglarge*. The degree of truth for each possible linguistic value actually quantifies the certainty that a numeric variable can be classified into a linguistic value. This is represented as a continuous value between 0 to 1 where 0 is false, 1 is true, and 0.5 indicates we are halfway certain. Also note that the measured variables are normalized before applying the mapping function, which is why the x-axis shows -1 and 1 for all the membership functions. This is done by multiplying the measured numeric quantities



(a) Membership functions. (b) Illustration of fuzzy rules.

Fig. 3. Operation of the fuzzy controller.

by factors known as the normalizing gains, and denoted g_{du} and g_{dy}. For example, in the membership function shown in Figure 3 (a), we would like that *change-in-MaxClients* map to *poslarge* if it is larger than "1." If $g_{du} = 1/10$, *change-in-MaxUsers* is *poslarge* if 10 more users are allowed to enter.

Due to the stochastic nature of computing systems, the online measured response time feedback will contain significant variability, particularly when the sample intervals are short. Hence, there is some uncertainty as to how much of the change in response time from one interval to the next is caused by the setting of `MaxClients` and how much due to stochastics. Fuzzy set theory provides a natural way to handle such stochastic data, by using the normalizing gains and membership functions to characterize these uncertainties (as shown in Figure 3 (a) between -1 and 1). In the example above, the mapping of $du = 0.5$ indicates that we're more certain (0.75) that the "actual" value of du is positive large than negative large (0.25).

The fuzzification component of the controller (bottom of Figure 2) implements the membership functions, as shown in Figure 3 (a), that convert the input numerical variables to their linguistic equivalents. Fuzzy inference involves the evaluation of the fuzzy rules in the rule set and combining the actions of rules (the THEN parts) to yield an output in terms of a linguistic variable. The defuzzification component is complementary to fuzzification and converts the output linguistic variable to a numerical value.

As observed in Section 2, response time is a concave upward function of `MaxClients`. Thus, response times can be minimized using a gradient descent procedure [13]. We describe this in terms of fuzzy rules since doing so facilitates the addition of expert heuristics. To see how this is done, suppose current `MaxClients` is to the left of the optimal value, as illustrated in Figure 3 (b), where the dashed arrow lines indicate premises (the *IF* parts) and the solid arrow lines indicate consequents (the *THEN* parts). Then, we would want to increase `MaxClients`. However, since the optimal `MaxClients` value is unknown, the fuzzy rules must first determine whether we are to the left or to the right. Moreover, we must adjust `MaxClients` in appropriate decrements since a too small change will cause slow convergence, whereas a too large step may cause oscillation. We encode this knowledge using the rules shown in Table 1. (Actu-

ally, from the interest of representation brevity these 4 rules can be condensed into a single meta-rule: *If the last change in* `MaxClients` *results in a lower response time, continue to change* `MaxClients` *in the same direction; Otherwise, change* `MaxClients` *in the opposite direction.* However, for the practical operation of fuzzy inference, it is more convenient to represent the knowledge in four separate rules.)

Table 1. Fuzzy rule base

Rule	*IF*			*THEN*
	change-in-MaxClients	*AND*	*change-in-response-time*	*next-change-in-MaxClients*
1	*neglarge*	*AND*	*neglarge*	*neglarge*
2	*neglarge*	*AND*	*poslarge*	*poslarge*
3	*poslarge*	*AND*	*neglarge*	*poslarge*
4	*poslarge*	*AND*	*poslarge*	*neglarge*

The *IF* part determines the position on the response time curve, as well as the distance from the optimal point. For the example of *Rule 4*, if we increased `MaxClients` and the response increased, then we are to the right of the minimum. The *THEN* part indicates the suggested action, to change `MaxClients` in the opposite direction. *Rule 1* and *Rule 3* indicate the "correct" situations in that the response time is decreasing. Conversely, *Rule 2* and *Rule 4* handle the "incorrect actions", where the previous action caused the response time to increase. Since we do not know the actual `MaxClients` value that minimizes the response time, these rules are easier to describe in terms of changes to the `MaxClients` and response time values. Afterwards, the consequents from all activated rules are weighted (e.g., using the "center of gravity" method) to compute the appropriate adjustment on `MaxClients`.

Note that the above four rules are only given to illustrate the tuning concepts, while more complicated rule sets can be defined to have a finer definition of the linguistic values (e.g, *neglarge, negmed, negsmall, zero, possmall, posmed, poslarge*) to increase the design flexibility, and to incorporate more tuning knowledge. such as how to get out of the flat saturation regions. Also worthy of mention is that for a typical tracking or regulation problem where the objective is to have zero control error, the inputs of the controller usually include both error and change-in-error. For the problem studied in this paper, the control objective is to minimize the response time. Since the minimum value is unknown in advance, the error (the difference between the current measured response time and the minimum value that we pursue) is also unknown and cannot be used as an input. Instead, we use *change-in-MaxClients* and *change-in-response-time* as control inputs, since they are both measurable.

The use of fuzzy inference allows us to separate the logic of gradient descent from the calculation of values of `MaxClients`. The rules encode the control logic, and the membership function defines how response times and `MaxClients` are weighted and adjusted during fuzzy inference. This separation facilitates the incorporation of semantic knowledge and simplifies the controller design work.

4 Experimental Assessment

Our experimental testbed consists of one server machine running Apache and one or more client machines running synthetic workload generators. The behavior of back end processing is simulated through *CGI* (Common Gateway Interface) codes. All of the above machines are connected through a 100Mbps LAN. We used a modified version of the Apache web server that allows us to adjust `MaxClients` without restarting the server.

For workload generation, we used the WAGON model that has been validated in extensive studies of production web servers [14]. This model characterizes web workloads in terms of several parameters, including the rate at which new sessions arrive (session arrival rate) and the number of URLs requested when a page is accessed (burst length). The httperf program [15] is used to generate synthetic HTTP requests that conform to this model. The file access distributions are from the Webstone 2.5 reference benchmark [16]. In order to create additional load and delay on the server to simulate back-end processing, we request the pages via the *CGI* codes that are part of Webstone. The session arrivals follow Poisson distribution with a rate of 10 sessions per second. (A more detailed description of the Apache testbed and workload generator can be found in [3].)

4.1 Controller Design

The fuzzy controller is designed to dynamically adjust the server tuning parameter `MaxClients` in order to minimize the response time. Only the concave upward assumption is required. However, the specific response time curve need not to be known. The design/implementation work for the fuzzy controller is to select the control interval and to choose the scaling factors (normalizing gains) g_{du}, g_{dy}, and g_m. We select the control interval to be 5 minutes since the response time data are not probed very frequently (i.e., the inter-request time for the probing station is 10 seconds). A large control interval reduces the variability in the probed response time data and gives a more accurate measure. Next, the normalizing gains are tuned based on experimentation and heuristic knowledge of the target system. Generally, the larger the value of g_{du}, the faster the control action, that is, the controller responds quickly. However, too quick a control response may cause system oscillation as the controller overreacts to small variations in measurements. We choose $g_{du} = g_m = 1/200$ and $g_{dy} = 1/5$ based on the system dynamics since we consider a change of 200 in `MaxClients` or of 5 seconds in response time are reasonably large.

4.2 Experimental Results

Figure 4 shows how the fuzzy controller seeks the optimal `MaxClients` value under a stationary workload (same as that shown in Figure 1 (b)). The bottom plot displays response times. The circles indicate where response times are so long that the connection timed-out. The top plot shows the value of `MaxClients`. We see that `MaxClients` starts at 200 and keeps increasing until it converges to

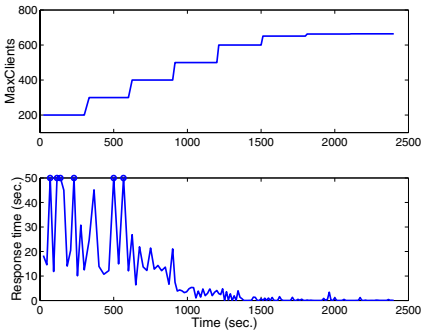


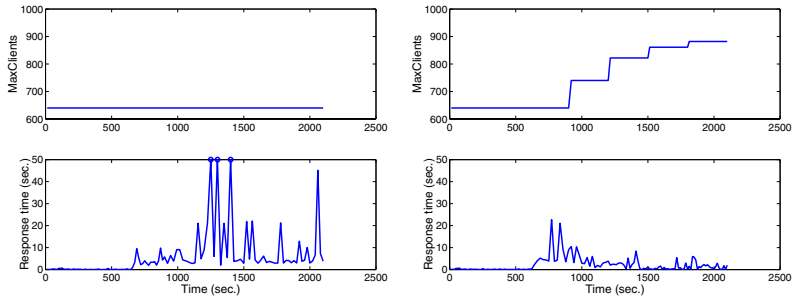
Fig. 4. Fuzzy control performance.

a value that results in a much lower mean and variance of response time. Note that the fuzzy controller operates in a manner comparing the current response time measures with the previous ones; the fuzzy controller will converge when the changes in response time is negligible. Also note that comparing to the standard hill climbing method the fuzzy control approach proposed in this paper uses variable step size, and comparing to the steepest descent method the fuzzy control approach does not use the difference of the tuning parameter as the denominator. This increases the convergence speed and also adds the robustness of the controller to the stochastics appearing in the response time measures.

Next, we study the effect of changes in workload (i.e., non-stationary workload). In the situation considered, workload is stationary for 600 seconds and then changes abruptly. By mixing the old workload with some additional static web pages requests, the new workload has a higher session rate but not as CPU/memory intensive as the old workload. Apparently, it is not appropriate to ignore this change and use static control—maintain `MaxClients` at the final value reached in Figure 4. Figure 5 (a) displays the results of such an approach. We see that after the change in workload occurs, response times increase dramatically. In contrast, Figure 5 (b) indicates that the fuzzy controller adapts to the change in workload by observing the variation in the response time and increasing `MaxClients` by over 50%. Once again, the fuzzy controller selects a `MaxClients` setting that results in a much lower mean and variance of response time.

4.3 Remarks

By looking into the operational mechanism of the Apache server (see Section 2), we know that the old workload (*CGI* codes with low request rates) is CPU/memory intensive; hence, a relatively small `MaxClients` value helps to avoid system over-utilization and balance the time spent in the TCP queue and worker processes. In contrast, the new workload (a mix of *CGI* codes and static HTML files with high request rates) is non-CPU/memory intensive but I/O intensive, which changes the response time bottleneck from worker processes to



(a) With a fixed value of `MaxClients`. (b) Under fuzzy control.

Fig. 5. Effect of a change in workload on the response time. The change in workload occurs at time 600.

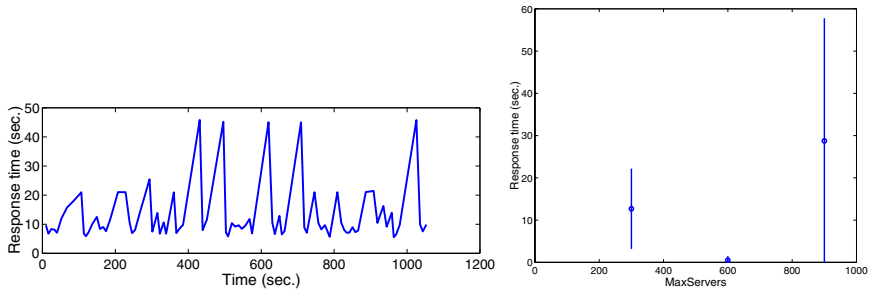
the TCP queue. Consequently, a relatively large `MaxClients` is desired for this case.

To apply the above detailed knowledge may be difficult for the automatic controller as measurements on the workload characteristics or the waiting times for each component (TCP queue, CPU queue) are needed. Instead, the fuzzy controller proposed in this paper only needs the measurement of overall user response time (from the probing station) and applies the basic qualitative knowledge that `MaxClients` has a concave upward effect on response times. Due to its simplicity and generality, the fuzzy controller herein proposed has the potential for wide-spread use.

4.4 Comparison with the Default Apache Control Scheme

It is interesting to compare the fuzzy control method proposed in this paper with the default Apache control scheme. In the original Apache HTTP server setting, the number of worker processes are regulated based on the policy that if the idle processes are fewer than `MinSpareServers` (the default value is 5), a new process is created; if they are more than `MaxSpareServers` (the default value is 10), some of the spares die off. The maximum number of processes is also limited by `MaxServers` (the default value is 300).

This default scheme is actually a heuristic feedback controller. It tries to maintain enough worker processes to handle current load and prepare a few extra ones to handle transient load spikes. However, it is an open question on how to optimally choose the default values. For example, if we use the default setting of `MaxServers`=300 for the workload shown in Figure 1 (b), the response times are generally large as shown in Figure 6 (a). Simply increasing the `MaxServers` value may not minimize the response time as a high `MaxServers` value can result in system over-utilization. The relationship between `MaxServers` and response times is shown in Figure 6 (b), which has a similar shape as that in Figure 1 (b). Note that in the default Apache control scheme `MaxServers` is the limit on the total number of worker processes, and the actual number of running worker



(a) Control performance for `MaxServers`=300. (b) Response time curve.

Fig. 6. Default Apache control scheme.

processes is usually smaller than that number. In our modified Apache control scheme `MaxClients` is the actual number of worker processes that are running.

Although the default scheme is simple and may work well for many web sites, the lack of QoS oriented measurements and control scheme may retard its performance especially for eCommerce sites with large workload variation and high QoS requirement. In contrast, the fuzzy control approach tries to improve QoS with qualitative understanding of the effect of tuning parameters on QoS metrics, and shows better performance in our experimental studies.

5 Conclusions

The advent of eCommerce has created demand for high quality information technology (IT) services, and hence the need to optimize tuning parameters to minimize response times. Unfortunately, properly adjusting tuning parameters is time-consuming and skills-intensive, which has caused many commerce sites to overprovision their hardware and to spend too much money on it. This paper describes a generic approach to automatic, on-line adjustment of tuning parameters so as to optimize quality of service. While our work has focused on the Apache web server, we believe that it has much broader application (e.g., maximizing SLA profits [11]).

Our approach requires a qualitative understanding of the effect of tuning parameters on QoS metrics. For example, the fuzzy controller only assumes the knowledge that the Apache tuning parameter `MaxClients` has a concave upward effect on response times (so that the gradient based local search results in global optimum), but the specific curve shape need not to be known and the minimum point can be different for different workloads. We construct a fuzzy controller that uses this knowledge to optimize the setting of `MaxClients`. Our studies of an experimental eCommerce site with a real Apache web server and simulated back-end processing reveal that the fuzzy controller can improve performance without human intervention and that the controller automatically adapts to changes in workloads.

Our future work will address a number of issues. Foremost, we want to expand the set of qualitative relationships that are described by fuzzy rules. A

trivial extension is to consider concave downward functions as well as concave upward. A more challenging task is to discover whether a concave function is upward or downward. A second extension is to simultaneously optimize multiple parameters. This may involve more usage of gradient information. Third, while our approach finds the best value of the parameter, it does not necessarily converge as rapidly. However, there is a trade-off here between the speed of convergence (e.g., by changing the tuning parameter more frequently) and robustness to noise. Last, we want to explore the effect of distributed architectures on the techniques we describe, especially the trade-off between doing local optimization with accurate knowledge of local state versus global optimization with somewhat dated information.

References

1. S. Parekh, N. Gandhi, J. L. Hellerstein, D. M. Tilbury, and J. P. Bigus, "Using control theory to achieve service level objectives in performance management," in *Proceedings of IEEE/IFIP Symposium on Integrated Network Management*, 2001.
2. V. Misra, B. G. Wei, and D. Towsley, "Fluid based analysis of a network of AQM routers supporting TCP flows with an application to RED," in *ACM SIGCOMM*, pp. 151–160, Oct. 2000.
3. Y. Diao, N. Gandhi, J. L. Hellerstein, S. Parekh, and D. M. Tilbury, "Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache web server," in *Proceedings of Network Operations and Management*, 2002.
4. C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, S. Son, and M. Marley, "Performance specifications and metrics for adaptive real time systems," in *Proceedings 21st IEEE Real Time Systems Symposium*, pp. 13–24, Nov. 2000.
5. D. Menasce, D. Barbara, and R. Dodge, "Preserving QoS of e-commerce sites through self-tuning: A performance model approach," in *Proceedings of 2001 ACM Conference on E-commerce*, 2001.
6. D. Menasce, V. Almeida, R. Fonsece, and M. Mendes, "Busines oriented resource management policies for e-commerce servers," *Performance Evaluation*, 2000.
7. Z. Liu, M. S. Squillante, and J. L. Wolf, "On maximizing service-level-agreement profits," in *Proceedings of the ACM Conference on Electronic Commerce*, 2001.
8. Y. Diao, J. L. Hellerstein, and S. Parekh, "A business-oriented approach to the design of feedback loops for performance management," in *Distributed Systems Operations and Management*, 2001.
9. K. M. Passino and S. Yurkovich, *Fuzzy Control*. Menlo Park, CA: Addison Wesley Longman, 1998.
10. R. Zhang and Y. A. Phillis, "Fuzzy control of arrivals to tandem queue queues with two stations," *IEEE Transactions on Fuzzy Systems*, pp. 361–367, 1999.
11. Y. Diao, J. L. Hellerstein, and S. Parekh, "Using fuzzy control to maximize profits in service level management," *IBM Systems Journal*, 2002.
12. Apache Software Foundation. http://www.apache.org.
13. R. Fletcher, *Practical Methods of Optimization*. John Wiley & Sons, 2000.
14. Z. Liu, N. Niclausse, C. Jalpa-Villanueva, and S. Barbier, "Traffic model and performance evaluation of web servers," Tech. Rep. 3840, INRIA, Dec. 1999.
15. D. Mosberger and T. Jin, "httperf: A tool for measuring web server performance," in *First Workshop on Internet Server Performance (WISP 98)*, ACM, 1998.
16. I. Mindcraft, "Webstone 2.5 web server benchmark," 1998. http://www.mindcraft.com/ webstone/.