Online Response Time Optimization of Apache Web Server

Xue Liu, Lui Sha
Department of Computer Science
Univ. of Illinois at Urbana-Champaign
Urbana, IL, 61801, USA
Email: {xueliu, lrs}@cs.uiuc.edu

Yixin Diao, Steven Froehlich,
Joseph L. Hellerstein, Sujay Parekh
IBM T. J. Watson Research Center
Hawthorne, NY 10532, USA
Email: {diao, stevefro, hellers, sujay}@us.ibm.com

Abstract— **Properly optimizing the setting of configuration parameters can greatly improve performance, especially in the presence of changing workloads. This paper explores approaches to online optimization of the Apache web server, focusing on the `MaxClients` parameter (which controls the maximum number of workers). Using both empirical and analytic techniques, we show that `MaxClients` has a concave upward effect on response time and hence hill climbing techniques can be used to find the optimal value of `MaxClients`. The underlying intuition is that `MaxClients` controls the trade-off between delays in the TCP Accept Queue and delays due to contention for operating system resources. We investigate two optimizers that employ hill climbing—one based on Newton's Method and the second based on fuzzy control. A third technique is a heuristic that exploits relationships between bottleneck utilizations and response time minimization. In all cases, online optimization reduces response times by a factor of 10 or more compared to using a static, default value. The trade-offs between the online schemes are as follows. Newton's method is well known but does not produce consistent results for highly variable data such as response times. Fuzzy control is more robust, but converges slowly. The heuristic works well in our prototype system, but it may be difficult to generalize because it requires knowledge of bottleneck resources and an ability to measure their utilizations.**

Index Terms— **web server; Apache; response time; optimization; queueing; fuzzy control**

I. Introduction

The widespread use of eCommerce systems has focused attention on quality of service, especially response time. One challenge here is adapting systems to changing workloads by online optimization of configurations. This paper explores approaches to such online optimization in the Apache web server with an emphasis on techniques that are minimally invasive and are applicable to a wide range of parameters and systems.

TABLE I
Response time (sec.) for different workloads

MaxClients	Workload: *Dynamic*	Workload: *Dynamic+Static*
150	50	
650	1	15
900	30	2

How much can be achieved by optimizing the settings of configuration parameters? Consider the Apache `MaxClients` parameter, which governs the number of requests being processed in parallel by the web server. In Table I, we show average response times measured at different `MaxClients` settings under different workloads [1]. The testbed used to collect this data is discussed later in this paper. We see clearly that the best `MaxClients` value to use depends on the type of pages being accessed at the site. Since real-world workloads can change rapidly, there is the potential of obtaining order-of-magnitude improvements by doing online optimization of such key parameters.

This paper describes a generic approach to the online optimization of response times for the widely used Apache web server[2]. One area of related work is differentiated service in which the intent is to achieve response time objectives for different classes of work. In [3], [4], the authors use proportional-integral controllers to achieve response time regulation and differentiation. In [5], multiple-input multiple-output controller design is used to regulate server CPU and memory utilization within specified QoS value. [6] describes an approach that combines queueing theory and control theory to regulate response times for response time regulation. Unfortunately, the regulation problems addressed by

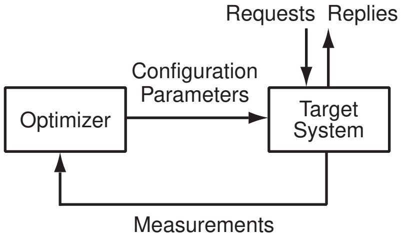


Fig. 1. General architecture for online optimization. The target system is controlled by configuration parameters that are dynamically changed by the optimizer in response to changing workloads.

these approaches is quite different from optimization. In essence, regulation (e.g., ensuring target response times for gold and silver service) determines how to "cut the pie" whereas optimization (e.g., minimize response time across service classes) addresses how to "make the pie bigger" by adjusting resource allocations to the workload. Some work has been done in the area of online optimization of resources in computing systems. [7] describes an Apache implementation that manages web server resources based on maximizing revenue (e.g., responding within 8 seconds so that users are not discouraged). While the results are encouraging, the approach requires substantial modifications to the Apache resource management schemes. [8] considers maximizing SLA profits for web server farms, but this is done in a way that depends on having an accurate analytic model of the system being controlled. More recently, [1] proposes a fuzzy control approach to minimize response time using a combination of feedback control system and qualitative insights into the effect of tuning parameters on QoS. Unfortunately, the convergence times are long.

Figure 1 displays the architecture we propose. The target system (e.g., Apache) exposes one or more configuration parameters (e.g., `MaxClients`) that are dynamically modified by the optimizer to optimize measured variables (e.g., response times). In the specific case of Apache and `MaxClients`, we proceed as follows. First, we show that `MaxClients` has a concave upward effect on response time and hence hill climbing techniques can be used to find the optimal value of `MaxClients`. We investigate two optimizers that employ hill climbing—one based on Newton's Method and the second based on fuzzy control. A third technique is a heuristic that exploits the observed relationships between bottleneck utilizations and response time minimization. Newton's method does better than the default Apache scheme but yields inconsistent results because response times are variable. Fuzzy control is more robust, but converges slowly. The heuristic works well in our prototype system, but it may be difficult to generalize because it requires knowledge of bottleneck resources and an ability to measure their utilizations.

The remainder of the paper is organized as follows. Section 2 discusses the Apache architecture and response time measurements. Section 3 describes a queueing system that explains how `MaxClients` affects response times. Section 4 presents and evaluates several approaches to online optimization. Our conclusions are contained in Section 5.

II. Apache Architecture and Measurements

Apache [2] is typically structured as a pool of workers that handle HTTP requests. Our studies use release 1.3.19 in which workers are processes, although we believe that the central ideas are broadly applicable.

The flow of requests in Apache is displayed in Figure 2. Requests enter the TCP Accept Queue where they wait for a worker. A worker processes a single request to completion before accepting a new request. The number of worker processes is limited by the parameter `MaxClients`.

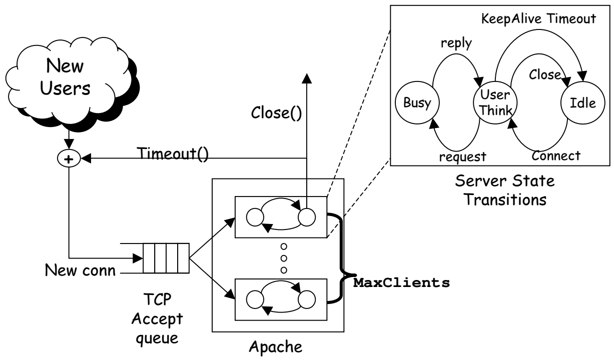


Fig. 2. Apache architecture and session flow.

Many of the insights in this paper are based on experimental results. Throughout, these experiments use the Apache 1.3.19 server software running on a Pentium III 600 MHz server with 256 MB RAM running the Linux 2.4.7 kernel. We use synthetic workload generators running on a set of similar machines, connected via a 100Mbps LAN. The distribution of files sizes is the same as Webstone 2.5 [9]. We employ both a static and a dynamic workload. Requests for dynamic pages are processed by the Webstone 2.5 CGI script. A detailed description of the Apache testbed and workload generator can be found in [5].

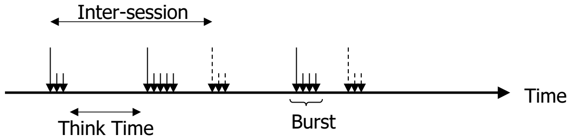


Fig. 3. Elements of the WAGON workload model. Two sessions are shown, as depicted by the solid and dashed lines. The longer arrows indicate the HTML text of a web page, and the short arrows indicate requests for objects in the web page.

Further details are needed to describe the manner in which requests are generated. Our workload model is based on WAGON [10], which has been shown to apply to a wide range of web requests. This model structures the workload into multiple sessions (which represent a series of user interactions). As illustrated in Figure 3, a session consists of multiple page requests. A page contains a number of embedded objects, which is parameterized by the burst length. Thus, the workload parameters are: session arrival rate, session length (number of clicks or web page requests in a session), burst length (number of objects in a burst), and think time (time between successive clicks). Table II summarizes the parameters used in this paper, which are based on data reported in [10] for a public web site. We use the httperf program [11] to generate HTTP/1.1 requests based on a synthetic web log that is generated according to the WAGON model.

TABLE II
WORKLOAD PARAMETERS

Parameter Name	*Distribution*	*Parameters*
$SessionRate$	Exponential	$mean = 0.1$
$SessionLength$	LogNormal	$mean = 8$, $\sigma = 3$
$BurstLength$	Gaussian	$mean = 7$, $\sigma = 3$
$ThinkTime$ (s)	LogNormal	$mean = 30$, $\sigma = 30$

The metric we choose to minimize is the server-side response time. Although client side response time measure is the most user-relevant metric, this information is generally not available at the web server in real-time. Moreover, even if the client side response time can be approximated using server side measurements and a TCP model [12], only the server side response time can be controlled by the server. Hence, we consider the server side response time, in particular, per page response time (RT) in this paper. Since delivering a page may involve multiple requests, this quantity needs to be estimated. We use the following equation:

$$RT = AQT + BL \times ST \tag{1}$$

The accept queue time (AQT) is collected from the Linux kernel where we have added instrumentation for measuring the average delay for connections that enter the accept queue within a time window. The service time (ST) is measured by instrumenting the first and last steps in the Apache request processing loop (i.e., at the points where the request enters and the reply is sent). The average number of embedded requests per page is also known as the burst length and denoted as BL. It can be calculated as the number of requests serviced in all the worker processes divided by the number of connections serviced in the TCP accept queue. Due to persistent connections in HTTP/1.1, an established TCP connection remains open between consecutive HTTP requests of a burst so that only the first request needs to set up the TCP connection and enter the TCP accept queue. This gives us the above equation.

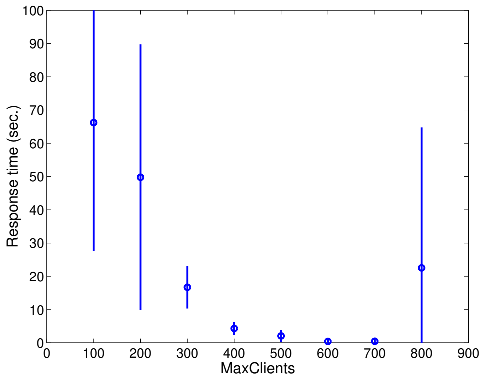


Fig. 4. Effect of `MaxClients` on response times.

Figure 4 displays the results of experiments in which Apache is configured with different settings of `Max-Clients`. The circles indicate mean response times, and the vertical lines specify the standard deviations. Note that the circles line is a pronounced concave upward shape to this curve. Further, the curve indicates that `MaxClients` has in excess of a ten-fold effect on response times for this workload.

This concave shape can be explained in terms of the Apache architecture. If `MaxClients` is too small, there is a long delay due to waits in the TCP Accept Queue. 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`. In essence, worker processes can be viewed as **logical resources** that are needed to process a request. However, to do so, **physical resources** are required, such as CPU, memory, and input/output bandwidth.

Our interest is in online optimization and thus we must change `MaxClients` dynamically. To this end, a mechanism similar to graceful restart was implemented to allow for a way to change `MaxClients` without shutting down Apache. Doing so required having an agent on the Apache system. This agent also provides instrumentation such as CPU utilizations, memory utilizations, and server side response times.

While this paper focuses on `MaxClients`, we believe that the approach taken has broader application. For example, we are currently investigating `KeepAliveTimeOut`, another Apache parameter that determines the time that a session may remain idle if persistent connections are used. Other systems have configuration parameters similar to `MaxClients`, such as the number of servlet threads and EJB threads in an application server.

III. ANALYTIC MODEL

This section develops an analytic model for how `MaxClients` affects response times in the Apache web server. We have two objectives. First, we want to demonstrate that the concave upward effect of `MaxClients` on response times can be explained in terms of a simple model. Second, we believe that this model provides a general framework in which other similar studies can be done.

We model interactions with Apache using an $M/M/m/K/N$ queueing system [13]. That is, interarrival times are exponentially distributed with a rate of λ, there are m servers (Apache workers), service times are exponentially distributed with a rate of μ_m (i.e., the service rate depends on the number of servers), the buffer size is K, and there are N customers. The model is evaluated using data from our Apache testbed.

Figure 5 depicts the $M/M/m/K/N$ queueing system as applied to Apache using the WAGON workload model. User sessions are modeled as the N customers in the queueing system, which is assumed to be constant. Users remain idle (in "think time") for an exponentially distributed period with mean $1/\lambda$. At the end of a think time, a page request is generated and enters the queue.

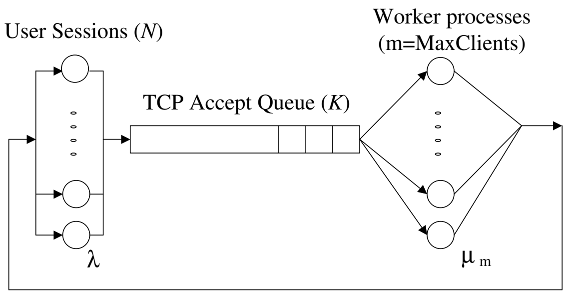


Fig. 5. Modeling Apache with an $M/M/m/K/N$ queueing network model.

(Note that a request represents a page, not a session.) This queue represents the TCP Accept Queue. The queue has a finite length of K. Requests wait until one of the m servers is available, where a server represents a worker and $m =$ `MaxClients`. The time a request spends in service is exponentially distributed with rate μ_m.

We calibrate the model parameters based on the Apache architecture and the workload model.

- Request generation rate λ: The reciprocal of λ is the user think time, which is the time between bursts (clicks) within a user session. Thus, for our testbed, we use the (reciprocal of) the mean of the think time distribution of the WAGON model.
- Number of servers m: This is the number of worker processes in Apache, i.e., `MaxClients`.
- Service rate μ_m: The service rate for a page is computed as the reciprocal of the inter-page request time. That is,

$$\mu_m = \frac{1}{BurstLength \times ST_m + KeepAliveTimeOut} \quad (2)$$

  where ST_m is the service time for an HTTP request processed by Apache. The subscript indicates that service time depends on the value of `MaxClients`. Once all objects in a page have been processed, the worker waits for the minimum of a think time and the `KeepAliveTimeOut` time. We assume that the user think time distribution generates values that are mostly larger than the `KeepAliveTimeOut` value (default value is 15 seconds). We obtain burst length from the workload model, service time is computed as an average based on our instrumentation of the Apache request processing loop.
- Buffer size K: This is the buffer size of the TCP Accept queue. We can either get its value from the Linux kernel, or assume it to be ∞ if there is

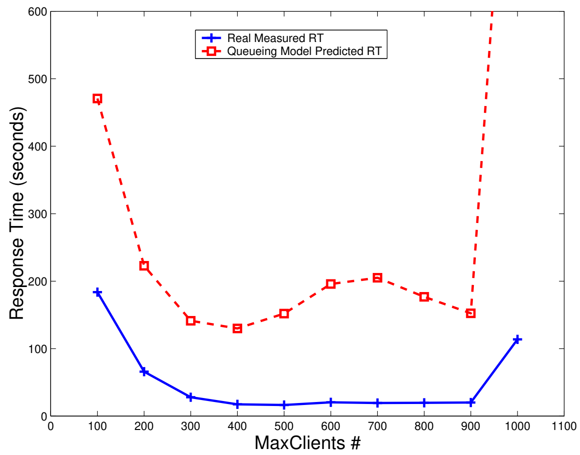


Fig. 6. Comparison of measured and predicted response times.

almost no request lost. We assume $K = \infty$ for the simplicity of calculation, since in our experiments no requests were found to be dropped.

- Number of customers N: The number of customers in the system should approximate the number of concurrent user sessions. This can be estimated using Little's Result.

$$\begin{aligned} N &= SessionRate \times SessionDuration \\ &= SessionRate \times (\frac{1}{\mu_m} \times SessionLength \\ &\quad + ThinkTime \times (SessionLength - 1)) \end{aligned}$$

These parameters can all be obtained from the WAGON workload model.

Based on the $M/M/m/K/N$ queueing formula in [13] and the foregoing description of how to obtain the model parameters, response time can be computed. Figure 6 plots predicted response times using this model and the measured values obtained from experiments at different values of `MaxClients`. We see that the model consistently overestimates the true response times, sometimes by a large amount. Possible reasons for the model's inaccuracy include the distributional assumptions (e.g., exponential inter-arrivals and service times) and the assumption that the number of sessions present is constant.

Even though the model is inaccurate in an absolute sense, it does *track* measured response times well. In particular, we see that predicted response times are concave upward in `MaxClients`. This gives us assurance that Figure 4 is not an artifact of the measurement environment. Further, it may be that the model can be applied more broadly, such as to the configuration of application servers (e.g., the maximum number of servlet or enterprise Java Bean (EJB) threads).

Another potential application of the queueing model is to aid in online optimization. This is possible since the model tracks the measurement results reasonably well. Unfortunately, we cannot use the model in this way, at least not without additional online measurement data. The problem is the manner in which model parameters are obtained. Many depends on characteristics of the workload that are not known a priori. Other model parameters require new instrumentation, such as state-dependent service time information to estimate service rates.

IV. OnLine Optimization

This section describes ways to minimize response times by dynamically adjusting `MaxClients` based on the insight that response time is concave upward in `MaxClients`. Several schemes are explored: Newton's Method, fuzzy control, and a heuristic. These techniques are compared as to how well they minimize response time and the speed of convergence to the steady state value. The former is desirable in terms of improving quality of service. The latter is important in order to adapt to changing workloads.

All of the approaches considered involve feedback. Thus, `MaxClients` is adjusted based on the observed effect on response time or other metrics. An alternative would be to employ a feed-forward scheme in which the optimal value of `MaxClients` is computed based on an analytic model. Feed-forward is appealing in that it avoids problems with stability and speed of convergence. However, such a scheme requires an analytic model that (a) tracks the measured values of response time and (b) has inputs that can be readily estimated. While the model developed in Section 3 satisfies (a), it does not satisfy (b). For example, the service rate μ_m depends on the number of servers, i.e., $\mu_m = f(m)$, but we do not know this function. Possibly, a model could be developed that satisfies both (a) and (b). However, lacking such a model, we focus on feedback approaches.

A. Newton's Method

Figure 7 displays an architecture in which Newton's Method [14] is employed for online optimization of Apache response times by dynamically adjusting `Max-Clients`. Newton's method, a widely used approach to optimization, uses the gradient of the function to be minimized (e.g., Figure 4) to estimate the value of `Max-Clients` that minimizes response times. For example,

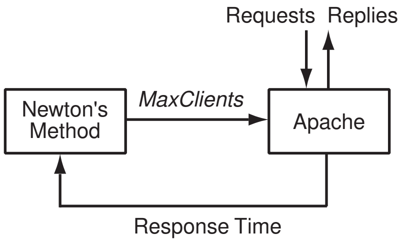


Fig. 7. Architecture of online optimization of Apache using Newton's Method to dynamically adjust `MaxClients` based on measurements of response times.

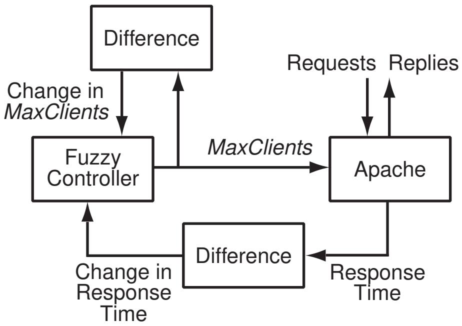


Fig. 8. Architecture of online optimization of Apache using Fuzzy Control to dynamically adjust `MaxClients` based on changes in `MaxClients` and response times.

if y is response time and x is `MaxClients`, we might use the approximation $y = f(x) \approx a(x - x^*)^2 + b$, where $a, b \geq 0$ are unknown constants estimated from the data and x^* is the value of `MaxClients` that minimizes response time. Newton's method is described by the following equation

$$x_{k+1} = x_k - (\bigtriangledown^2 f(x_k))^{-1} \bigtriangledown f(x_k) \tag{3}$$

where x_k is the value of x at discrete time k. Equation (3) starts from an initial value x_0 at $k = 0$. The gradient $\bigtriangledown f(x_k)$ is computed at x_k, and its negation indicates the direction of steepest descent. The value $\bigtriangledown^2 f(x_k)$ (the second partial derivative of $f(x)$) indicates the update step size. The introduction of the second partial derivative removes the local linear search, allowing a potentially faster convergence. But this also makes the algorithm more sensitive to measurement noise.

B. Fuzzy Control

Fuzzy control is another approach for online optimization [1]. We explore this approach in the context of the present study. Figure 8 displays an architecture in which fuzzy control is employed for online optimization of Apache response times by dynamically adjusting `MaxClients`. The fuzzy controller uses changes in `MaxClients` and response times to dynamically optimize `MaxClients`.

The actions of the fuzzy controller are guided by a set of IF-THEN rules. 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 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. Fuzzy control systems provide a way to map between numeric variables and linguistic variables (referred to as fuzzification and de-fuzzification). More details on fuzzy control can be found in [15].

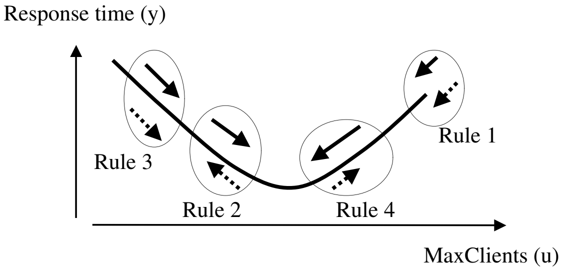


Fig. 9. Illustration of fuzzy rules.

TABLE III
FUZZY RULES

Rule	*IF*			*THEN*
	change in MaxClients	*AND*	*change in Response Time*	*change in next MaxClients*
1	*neglarge*	*AND*	*neglarge*	*neglarge*
2	*neglarge*	*AND*	*poslarge*	*poslarge*
3	*poslarge*	*AND*	*neglarge*	*poslarge*
4	*poslarge*	*AND*	*poslarge*	*neglarge*

The optimization problem we address can be easily represented in fuzzy rules. The rules, which are listed in Table III, are structured as follows. (They are also illustrated in Figure 9 where the dashed arrow lines indicate the *IF* premise parts and the solid arrow lines indicate the *THEN* consequent parts.) The *IF* part determines the position on the response time curve relative to the optimal `MaxClients`. For example, *Rule 4* considers the case in which `MaxClients` is increased and the result is a larger response time. This suggests that we are to the right of the optimal `MaxClients`.

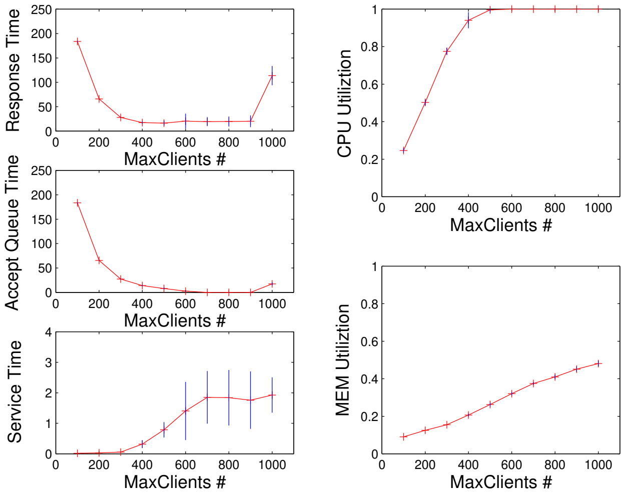


Fig. 10. Apache measurements for a dynamic workload.

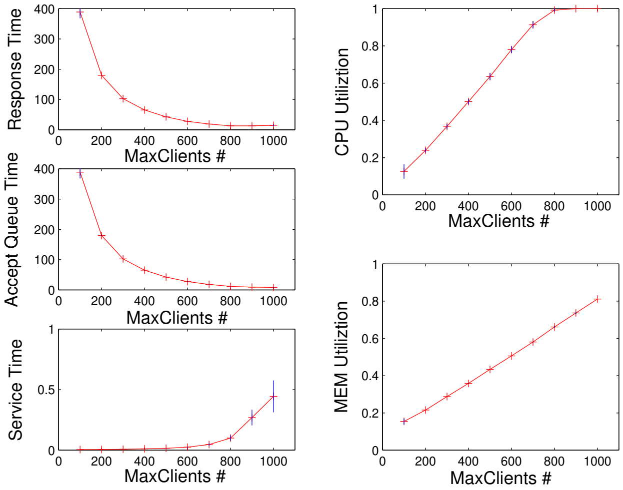


Fig. 11. Apache measurements for a static workload.

The *THEN* part indicates how `MaxClients` should be changed—*neglarge* is a decrease, and *poslarge* is an increase. *Rule 1* and *Rule 3* describe situations in which `MaxClients` was last changed in the correct direction in that the result is a decrease in response time. Conversely, *Rule 2* and *Rule 4* handle "incorrect actions", where the previous action caused the response time to increase. The magnitude of the change in `MaxClients` determines the speed of convergence and the extent of any oscillation at steady state. Clearly, if the curve is steep, small changes in `MaxClients` are best. For a more gradual slope, larger changes are better.

C. Saturation-Based Heuristic Optimization

This method is motivated by a desire to achieve fast convergence while being robust to noise and the specifics of the function being optimized.

Our heuristic is based on the following observation: response time is minimized when `MaxClients` is increased to the point where the CPU is 100% utilized. This is apparent in the measurements of static and dynamic workloads in Figure 10 and Figure 11. For different `MaxClients` values the average accept queue time and service time are measured, and the response time is computed using Equation (1). The CPU and memory utilizations are also measured for monitoring system resource usage. In Figure 10, response time decreases as `MaxClients` is increased from 200 to 400, at which point CPU utilization is approximately 100%. In Figure 11, this saturation occurs when `MaxClients` is approximately 800.

Our intuition as to why this works is as follows. `MaxClients` determines a set of logical resources—

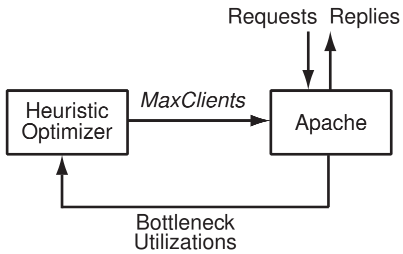


Fig. 12. Architecture of online optimization of Apache using a saturation-based heuristic to dynamically adjust `MaxClients` based on changes in bottleneck utilizations.

the Apache workers. These logical resources share the same physical resources, such as CPU, memory, and input/output bandwidth. By increasing `MaxClients` up to the point at which a physical resource saturates, we allow more of the logical resources to operate in parallel. However, once a physical resource saturates, further increases in the logical resource do not increase parallelism. Instead, such increases add overhead (e.g., due to process switches).

The foregoing observations motivate the architecture displayed in Figure 12 in which the heuristic controller dynamically determines the minimum value of `Max-Clients` that maximizes the utilization of the bottleneck resource. The major steps are given as follows.

1) Given an initial `MaxClients` value MC_0, measure the CPU utilization CPU_0 and the memory utilization MEM_0.
2) Use a linear model to predict the `MaxClients` values when CPU and memory will saturate. Note that since the CPU and memory utilizations are always zero when `MaxClients`=0, we need only

one observation to generate this prediction. The predicted `MaxClients` limit values are $\frac{MC_0}{CPU_0}$ and $\frac{MC_0}{MEM_0}$, respectively (where the subscript denotes discrete time).

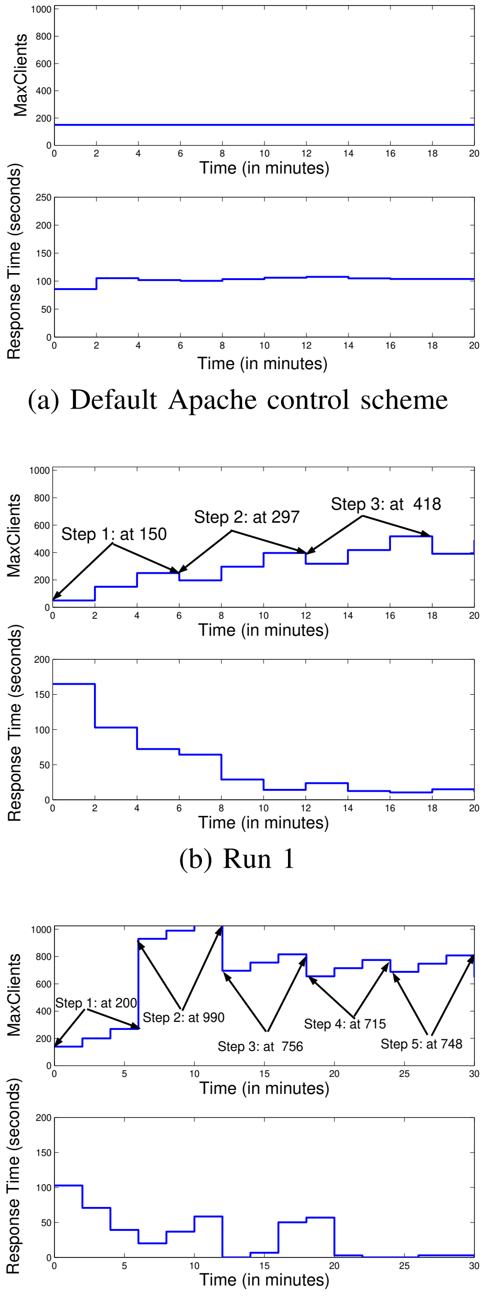
3) Set `MaxClients` to $\min(\frac{MC_0}{CPU_0}, \frac{MC_0}{MEM_0})$. This goes to a vicinity of the real optimal value of `MaxClients`.
4) Keep measuring CPU and memory utilizations, and if they vary significantly due to workload changes, go to step 2 to find a new `MaxClients` value.

Our heuristic does require measurements of utilizations for all potential bottleneck resources. There are two issues here. First, internal metrics are sometimes difficult to acquire in practice because of limitations of the measurement system. (In contrast, response time measurements can readily be provided by an external probing station.) Second, it is often difficult to determine which are the bottleneck resources. For example, a disk may be 100% utilized but have no queueing delays because it is used by only one single-threaded application. Never-the-less, if the utilization measurements are available and the heuristic applies, then it can provide fast, robust convergence to the minimal response times.

D. Experimental Results

This section compares the techniques for online optimization in terms of the minimum value of response time that is achieved, the speed of convergence, and robustness.

Figure 13 compares the performance of Newton's Method with the default Apache scheme. The figure contains three sub-figures, each with two plots. In each sub-figure, the upper plot shows the trajectory of `Max-Clients`, and the bottom plot displays the associated response times. Note that Newton's Method does improve response times compared to those in the default Apache scheme. However, because of the variability of response times, different runs of Newton's Method can produce very different results. This is because obtaining the Hessian matrix requires three samples to compute the second derivative, *at each step of the algorithm*. This increases the convergence time and also the algorithm is more sensitive to noise in response time measurements. Unfortunately, response times are typically quite noisy, unless they are averaged over many samples (something that reduces the speed with which the controller can respond to changes in workloads). Because of this noise sensitivity, we do not consider Newton's Method in the remaining comparisons.



(a) Default Apache control scheme

(b) Run 1

(c) Run 2

Fig. 13. Comparison of the default Apache scheme with Newton's method under a dynamic workload. While Newton's method does achieve lower response times, its behavior is not consistent due to the variability of response times.

Next, we compare the default Apache scheme with fuzzy control and the heuristic method presented earlier. Figure 14 displays the results for a dynamic workload. (The results are structured in the same manner as Figure 13.) We see that the heuristic converges its `MaxClients` value after 2 minutes. For fuzzy control, convergence takes approximately 14 minutes. On the other hand, fuzzy control does achieve a smaller response time. Figure 15 displays the results for a static workload. Once again, the heuristic converges faster than fuzzy control. Here, however, the steady state response time achieved by the heuristic is about the same as that achieved by fuzzy control.

TABLE IV

QUALITATIVE COMPARISONS OF TECHNIQUES.

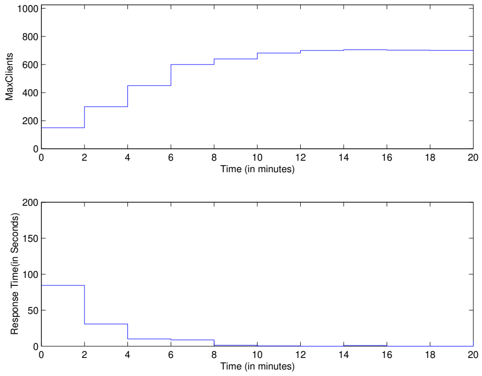
	Optimization	*Speed*	*Robustness*
Default Apache	Poor	Fast	Good
Newton's Method	Fair	Slow	Poor
Fuzzy Control	Good	Slow	Good
Heuristic	Good	Fast	Fair

Table IV provides a qualitative comparison of the four schemes considered. The default Apache scheme does a poor job of minimizing response times, in large part because this is not what it is designed to do. Newton's method improves on this, but it converges slow and has poor robustness to noise. Fuzzy control is quite robust because it makes few assumptions, but it converges slowly. Our heuristic provides good optimization and converges quickly, but it makes assumptions about the bottleneck resources that may not always hold.

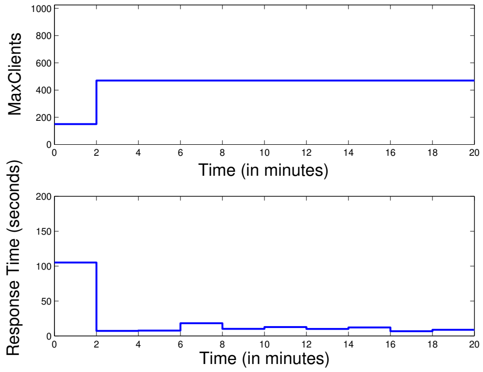
V. CONCLUSIONS

This paper explores approaches to online optimization of configuration parameters of the Apache web server with an emphasis on techniques that are minimally invasive and are applicable to a wide range of parameters and systems. We focus on the `MaxClients` parameter, which controls the maximum number of workers. First, we show that `MaxClients` has a concave upward effect on response time and hence hill climbing techniques can be used to find the optimal value of `MaxClients`. This is demonstrated both in measurements and with an analytic model. The underlying intuition is that `MaxClients` controls the trade-off between delays in the TCP Accept Queue and delays due to contention for operating system resources.

We investigate two optimizers that employ hill climbing—one based on Newton's Method and the sec-



(a) Fuzzy control

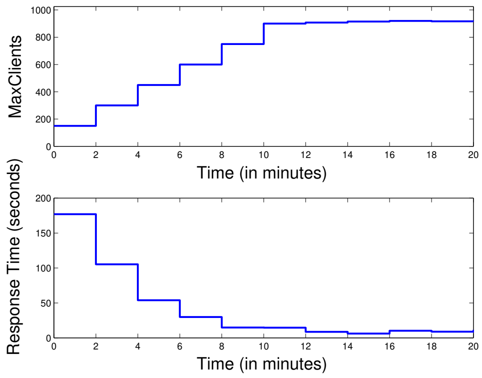


(b) Heuristic control

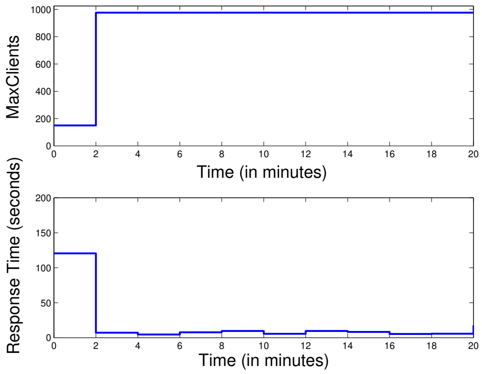
Fig. 14. Comparison of schemes for online optimization under a dynamic workload.

ond based on fuzzy control. A third technique is a heuristic that exploits relationships between bottleneck utilizations and response time minimization. In all cases, online optimization reduces response times by a factor of 10 or more compared to using a static, default value. The trade-offs between the online schemes are as follows. Newton's method is well known but does not produce consistent results for highly variable data such as response times. Fuzzy control is more robust, but converges slowly. The heuristic works well in our prototype system, but it may be difficult to generalize because it requires knowledge of bottleneck resources and an ability to measure their utilizations.

Our future work will address a number of issues. Foremost, we want to simultaneously optimize multiple parameters. This may involve other dynamically adjustable parameters in Apache such as `KeepAliveTimeOut`, which specifies how long the TCP connection to be kept for a client before the connection is torn down. Second, while we have studied the Apache web



(a) Fuzzy Control



(b) Heuristic control

Fig. 15. Performance comparison of schemes for online optimization under a static workload.

sever performance tuning, there are other more complex systems such as database servers and application servers where online optimization have a more dramatic effect on end-user response times. Last, we want to explore the effect of distributed architectures, especially the trade-off between doing local optimization with accurate knowledge of local state versus global optimization with somewhat dated information.

REFERENCES

[1] Y. Diao, J. L. Hellerstein, and S. Parekh, "Optimizing quality of service using fuzzy control," in *Proceedings of Distributed Systems Operations and Management*, 2002.

[2] Apache Software Foundation. http://www.apache.org.

[3] Y. Diao, J. L. Hellerstein, and S. Parekh, "A business-oriented approach to the design of feedback loops for performance management," in *Proceedings of Distributed Systems Operations and Management*, 2001.

[4] C. Lu, T. Abdelzaher, J. Stankovic, and S. Son, "A feedback control approach for guaranteeing relative delays in web servers," in *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, 2001.

[5] 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.

[6] L. Sha, X. Liu, Y. Lu, and T. Abdelzaher, "Queuing model based network server performance control," in *Proceedings of the IEEE Real-Time Systems Symposium*, 2002.

[7] D. Menasce, V. Almeida, R. Fonsece, and M. Mendes, "Busines oriented resource management policies for e-commerce servers," *Performance Evaluation*, vol. 42, pp. 223–239, Oct. 2000.

[8] Z. Liu, M. S. Squillante, and J. L. Wolf, "On maximizing service-level-agreement profits," in *Proceedings of the ACM Conference on Electronic Commerce (EC'01)*, 2001.

[9] I. Mindcraft, "Webstone 2.5 web server benchmark," 1998. http://www.mindcraft.com/ webstone/.

[10] Z. Liu, N. Niclausse, C. Jalpa-Villanueva, and S. Barbier, "Traffic model and performance evaluation of web servers," Tech. Rep. 3840, INRIA, Dec. 1999.

[11] D. Mosberger and T. Jin, "httperf: A tool for measuring web server performance," in *First Workshop on Internet Server Performance (WISP 98)*, pp. 59—67, ACM, June 1998.

[12] D. P. Olshefski, J. Nieh, and D. Agrawal, "Inferring client response time at the web server," in *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 2002.

[13] S. S. Lavenberg, ed., *Computer performance modeling handbook*. Orlando, FL: Academic Press, INC, 1983.

[14] A. L. Perssini, *The Mathematics of Nonlinear Programming*. Springer-Verlag, 1988.

[15] K. M. Passino and S. Yurkovich, *Fuzzy Control*. Menlo Park, CA: Addison Wesley Longman, 1998.