Adaptive Admission Control for Web Applications with Variable Capacity

Vipul Mathur, Preetam Patil, Varsha Apte and Kannan M. Moudgalya
Indian Institute of Technology–Bombay
Powai, Mumbai, Maharashtra 400076, India
{vipulmathur,pvpatil,varsha,kannan}@iitb.ac.in

***Abstract*—The system capacity available to a multi-tier Web based application is often a dynamic quantity. Most static threshold-based overload control mechanisms are best suited to situations where the system's capacity is constant or the bottleneck resource is known. However, with varying capacity, the admission control mechanism needs to adapt dynamically.**

We propose and implement an adaptive admission control mechanism that adjusts the admitted load to compensate for changes in system capacity. The proposed solution is implemented as a proxy server between clients and front-end Web servers. The proxy monitors 'black-box' performance metrics—response time and rate of successfully completed requests (goodput). With these measurements as indicators of system state, we employ a control theory based feedback loop to dynamically determine the rate of admitted requests. The objective is to balance changes in response time and changes in goodput, while preventing overloads due to reduction in available system capacity. We evaluate our mechanism with experiments on a test-bed and find that it is able to maintain higher productivity than a static admission control scheme.

I. Introduction

A client-server application running on a distributed system is said to be overloaded when the load offered to it exceeds the limit of its processing ability, i.e. capacity. This leads to increase in delays and blocking due to congestion, and rate of successfully completed requests (goodput) can drop due to impatient users timing out, abandonment of work, increase in overheads and other degenerate causes such as thrashing.

Allocating more resources will resolve an overload condition by increasing capacity, but doing so implies that we have knowledge of where the bottleneck is in a system. This is often infeasible in a distributed system with multiple tiers and many resources on each tier. An admission control mechanism, on the other hand, limits the amount of load entering a system. This effectively makes the system robust to unchecked increase in offered load, preventing any unwanted performance degradation. Various overload control solutions based on these approaches have been proposed [1]–[6] in literature.

Data center infrastructure is often sized to support peak load, invariably leading to grossly underutilized server resources [7] due to the high variance of offered load. In an effort to reduce such wasted capacity, recent trends have been towards dynamic provisioning, on-demand resource allocation, and shared virtualized server hardware [7] allowing system capacity to be scaled based on demand. Another source of changes in capacity of a system is failure of nodes in a cluster.

Overload control techniques based on *static thresholds*—be it admitted load or resource limits—are not effective in environments with varying capacity since they do not compensate for capacity changes. Hence we are motivated to look for *adaptive* approaches to overload control. Existing literature deals with heuristic or rule-based methods [3], dynamic programming based parameter optimization [6], and control theory based methods [2], [4], [5] for such adaptation. The common factor in all of these is the use of *feedback* where measurements of current system state are used to reconfigure inputs to drive the system towards a performance target. This is unlike *feedforward* methods that need accurate models for predicting inputs needed to get desired performance.

In this paper we present an adaptive admission control scheme that dynamically adjusts the amount of load admitted into the system. The primary goal is to protect the system from adverse effects of overload cased due to changes in capacity. Additionally, our solution aims to be robust to variations in offered load, not need a very detailed model of the distributed system, not be specific to any tier or resource, have low operational overhead, and accommodate short-term peaks in offered load.

We note that it is non-trivial to obtain static performance targets (e.g., maintain response time at 200ms) for a Web application. Such targets are typically not specified by Web users, even though users are impatient and will timeout and abandon requests if response time is too long. Static performance targets also raise feasibility questions. For instance, enforcing a strict response time limit by dropping load may cause excessive loss of requests if processing time itself exceeds the limit.

Main contributions of our work are as follows. We address the problem of ensuring stable performance of a distributed Web application in an environment with variable system capacity, as distinct from the often-studied case of variable incoming load. Rather than tracking some externally specified reference value for the performance metrics, our solution achieves a balance between changes in response time and changes in goodput, keeping the system close to a *maximum power* point. Our solution is not specific to any bottleneck resource or application tier. We use a control theory based feedback loop [8], [9] to tune token-bucket regulators for limiting load entering the system when offered load is more than current capacity—thus preventing overload.

The rest of the paper is organized as follows. Section II motivates our system model and solution architecture. In Section III we present the details of our adaptive admission control solution for Web applications. Results of experiments on a test-bed are presented in Section IV, followed by a look at related work. We conclude in Section V.

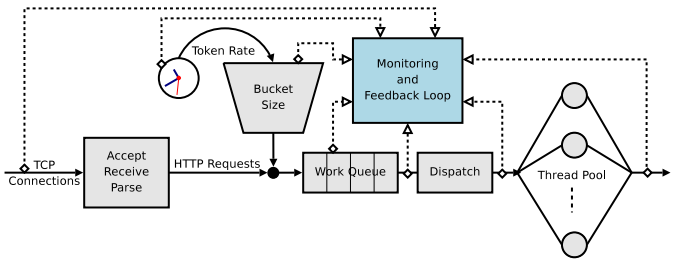

Fig. 1. Architectural overview of our proxy-based solution

II. System Model and Solution Architecture

We implement our feedback based adaptive admission control mechanism in a Web proxy [10] that sits between clients and front-end Web servers, transparent to clients, co-located with a load-balancer if required. The proxy keeps measurements of performance metrics, implements load control and request traffic shaping mechanisms, and supports queues of HTTP requests to better manage bursty arrival patterns. The block-diagram of this proxy's architecture is given in Fig. 1. The proxy accepts incoming TCP connections, receives HTTP requests and adds them to the work queue. Requests are picked up by a dispatch thread and sent to upstream servers when a worker thread is free. This might add some queuing delay at the proxy, but it prevents unconstrained increase in service time at server due to congestion during overload.

A token-bucket regulator placed at the ingress point of the work queue (Fig. 1) controls the rate of requests to be admitted into the system. This admission rate (λ_{ADM}) is the parameter that is set by our proposed feedback controller. Token-bucket regulators have the desirable property of allowing bursts of requests (up to a defined limit), while still enforcing a limit on the average rate of requests admitted. A request is allowed to enter the queue only if sufficient tokens (one request is equivalent to 100 tokens) are available in the bucket. If not, the request is dropped and a message returned to the user discouraging repeated retries that only add to the load.

To measure the performance delivered to the user and the 'productivity' of the Web Application, we use: *Response Time* R—the duration between arrival of request and successful completion of a reply to the user; and *Goodput* Λ—the average rate of requests successfully completed, excluding requests that were abandoned by impatient users due to a time-out. These are 'black-box' performance metrics and can be measured directly at the proxy and thus our solution is not specific to any application tier or bottleneck resource. A drop in the Web application's performance due to overload is reflected in these two metrics as a rise in mean response time, and more severely, a drop in goodput. The *power metric* for a queue, defined by Kleinrock [11], combines the behavior of response time and goodput into one convenient metric.

$$\text{Power} = \frac{\text{goodput (normalized to capacity)}}{\text{response time (normalized to service time)}} \quad (1)$$

We use *power* to aid in selection of the baseline performance desired and as an indicator of system productivity.

A. Fixed-rate Admission Control

If the capacity of the system is fixed ($\Lambda_{\mathrm{static}}$), then a *static* admission control mechanism with a fixed admission rate $\lambda_{\mathrm{ADM}} < \Lambda_{\mathrm{static}}$ prevents performance degradation if offered load exceeds system capacity by rejecting surplus load.

Figure 2 shows results from experiments on our test-bed (described later) for a case where the capacity $\Lambda_{\mathrm{static}}$ was 13 req/s. Without admission control, goodput dips to a low value and response time shoots up as load increases beyond 13 req/s. Using a token-bucket regulator with a fixed admission rate λ_{ADM} of 12 req/s, the goodput is maintained at 12 req/s and response time is kept low even as offered load increases to more than twice the capacity. The third plot in Fig. 2 shows that the maximum power point is near 10 req/s. Without admission control, power drops drastically as response time rises and goodput falls, while fixed-rate control gives higher power in overloaded region.

Here we had assumed that system capacity is static. There are many sources of 'disturbance' in a shared system, such as changes in resource usage by other applications and background tasks, that are manifested as a change in *available* capacity. Also, virtualized hardware and dynamic resource allocation environments have variable resource capacity as a feature. In such situations, our goal is to adapt the admitted load λ_{ADM} to compensate for capacity change, such that system *productivity* is maintained at an acceptable level while preventing performance degradation due to overload. Given our power metric (Equation 1) based notion of productivity, this goal translates to ensuring that response times are kept low and goodput is as high as feasible. Since it is difficult to measure or model the capacity of a distributed system directly, we choose to use feedback control to adapt the admitted load.

III. Control Theory Based Solution

The central idea of a feedback controller (FBC) is to measure the current state of the system using a set of performance metrics. Then this state is used to iteratively refine the input parameters of the system such that the system state moves towards some definite target. This enables the system to adapt to changes in its environment. Control theory provides tools for designing FBCs such that properties like stability, accuracy and convergence are ensured. When dealing with computing systems, measurements and system dynamics are discrete in nature. Hence we restrict ourselves to the domain of digital control [8], [9]. In this view, time is measured in discrete steps (of fixed size T_s) denoted by index k where measurement and controller action takes place.

The system's state is represented by state variables $x_1(k)$ capturing mean response time (R ms) and $x_2(k)$ based on mean goodput (rate Λ req/s of successfully completed requests). The tunable parameter $u_1(k)$, termed as *control input*, is based on admitted load λ_{ADM} tokens/s. The variance (stochastic noise) in measured performance metrics increases as we move towards higher loads. Hence exponential weighted averaging of the observed metrics is done to smooth noise:

$$\begin{aligned} R_{\mathrm{filtered}}(k) &= \alpha_1\, R_{\mathrm{filtered}}(k-1) + (1-\alpha_1)\, R(k) \\ \Lambda_{\mathrm{filtered}}(k) &= \alpha_2\, \Lambda_{\mathrm{filtered}}(k-1) + (1-\alpha_2)\, \Lambda(k) \end{aligned} \quad (2)$$

where α_i are filtering factors. To express our intention of pushing the admitted load and goodput close to capacity, we use difference from maximum goodput Λ_{MAX} as u_1 and x_2:

$$u_1(k) = \Lambda_{\mathrm{MAX}} - \lambda_{\mathrm{ADM}}(k)$$

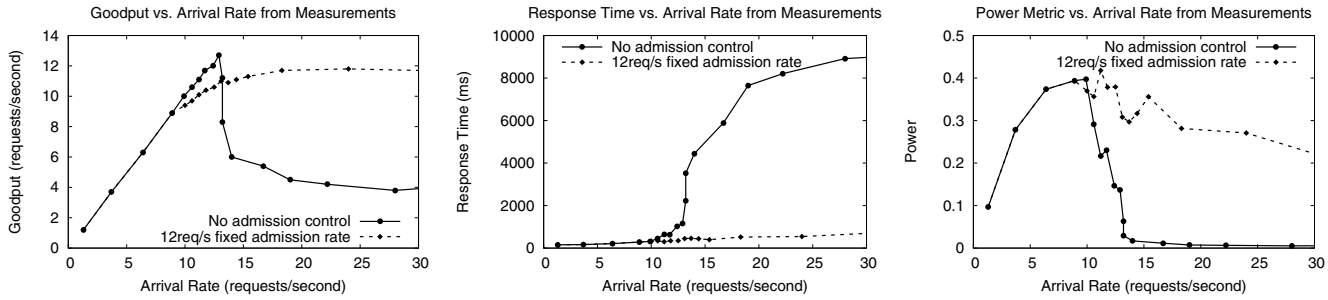

Fig. 2. Effect of fixed-rate admission control on goodput and response time. Capacity $\Lambda_{\text{static}} = 13$ req/s, $\lambda_{\text{ADM}} = 12$ req/s.

$$x_2(k) \quad = \quad \Lambda_{\text{MAX}} - \Lambda_{\text{filtered}}(k) \tag{3}$$

We define the *operating point* θ of the system as the steady-state values of control input ($\bar{u}_1$) and state variables ($\bar{x}_1$, $\bar{x}_2$) such that the system is stable in the absence of any disturbance. To determine the operating point, a set of experiments are conducted with successively increasing load levels to identify the desired operating region, i.e. low load to just before peak capacity (Fig. 2). Typically, the maximum power point $\lambda^\star$ (Equation 1) is a good baseline value of admitted load $\bar{\lambda}_{\text{ADM}}$. However, if $\lambda^\star$ is too low compared to system capacity, then the amount of lost work in the form of rejected requests may be significant. On the other hand if we set $\bar{\lambda}_{\text{ADM}}$ very close to capacity, then stochastic variation in load or reduction in capacity can push the system into overload. Thus we choose $\bar{\lambda}_{\text{ADM}}$ in the region between maximum power point $\lambda^\star$ and peak goodput Λ_{MAX} depending on system characteristics (Fig. 2). For the results presented in Sec. IV, $\bar{\lambda}_{\text{ADM}}$ was fixed at 10 req/s = 1000 tokens/s. This corresponds to a mean response time of 400 ms and goodput of 9.6 req/s.

Evolution of the system is represented by an iterative model written in the form of a difference equation

$$\boldsymbol{x}(k+1) \quad = \quad \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}\boldsymbol{u}(k) \tag{4}$$

where matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are the model's parameters to be determined according to the system. $\boldsymbol{A}$ relates the next value of state vector $\boldsymbol{x}(k)$ to its own previous values, while $\boldsymbol{B}$ relates it to the values of the control input $\boldsymbol{u}(k)$. We use deviations of control input and system state from operating point θ as our control input vector $\boldsymbol{u}(k)$ and state space vector $\boldsymbol{x}(k)$

$$\begin{aligned} \boldsymbol{u}(k) &= \left[\ u_1(k) - \bar{u}_1\ \right] = \left[\ \big(\Lambda_{\text{MAX}} - \lambda_{\text{ADM}}(k)\big) - \bar{u}_1\ \right] \\ \boldsymbol{x}(k) &= \begin{bmatrix} x_1(k) - \bar{x}_1 \\ x_2(k) - \bar{x}_2 \end{bmatrix} = \begin{bmatrix} R_{\text{filtered}}(k) - \bar{x}_1 \\ (\Lambda_{\text{MAX}} - \Lambda_{\text{filtered}}(k)) - \bar{x}_2 \end{bmatrix} \end{aligned} \tag{5}$$

to denote our intention to operate the system close to the selected baseline performance in the absence of any disturbance.

The feedback loop is completed by setting the control input $\boldsymbol{u}(k)$ based on the state of the system $\boldsymbol{x}(k)$ via a *control law*

$$\boldsymbol{u}(k) = -\boldsymbol{K}\boldsymbol{x}(k) \tag{6}$$

where $\boldsymbol{K}$ is termed as the *controller gain*.

A. System Characterization

Our scheme for deriving the model parameters $\boldsymbol{A}$ and $\boldsymbol{B}$ is based on doing least-square regression on measurements [9]. To obtain the measurements for this step, we conduct another experiment where we use a constantly high (say λ_{IN} = 18 req/s) arrival rate and vary the control input $\boldsymbol{u}(k)$ in a sinusoidal pattern with cycle time 3000s and amplitude over its entire operating range [50, 1150] tokens/second as identified earlier. The sinusoidal pattern of varying $\boldsymbol{u}(k)$ ensures that the perturbation of the system covers the entire operating region evenly. We collect observations of performance metrics for a total duration of 9000s and use linear least-squares regression fitting of $\boldsymbol{x}(k)$ and $\boldsymbol{u}(k)$ to Equation (4) to yield:

$$\boldsymbol{A} = \begin{bmatrix} 0.69321 & 0 \\ 0 & 0.32734 \end{bmatrix} \quad \boldsymbol{B} = \begin{bmatrix} -0.0917293 \\ 0.0066773 \end{bmatrix} \tag{7}$$

We have implemented an internal ready-to-use module to automate this step, within the `MASTH Proxy` platform [10].

B. Linear Quadratic Regulator Design

We employ the Linear Quadratic Regulator (LQR) [8], [9] optimal control framework to design our controller gain $\boldsymbol{K}$ such that a weighted sum of deviations in metrics and control input is minimized. This keeps our system as close to the baseline operating point θ as possible. Rise in response time is a strong indicator of impending overload and the we want the system to react strongly to variations in response time to protect against overload. Large variations in the admitted load and goodput are also undesirable. These trade-offs are encoded in the cost function for the LQR setup, defined as

$$J = \sum_{k=0}^{\infty} J(k) = \sum_{k=0}^{\infty} \big(q_1 x_1^2(k) + q_2 x_2^2(k) + r_1 u_1^2(k)\big) \tag{8}$$

by fixing weights q_1, q_2 and r_1 according to the desired sensitivity of FBC to changes in response time, goodput and admission rate respectively. We normalize the contributions of state variables and control input to the cost (Equation 8) by dividing with the maximum values of each parameter in the operating range. For our test-bed, we set the relative contribution of each cost term in the ratio $q_1 x_1^2 : q_2 x_2^2 : r_1 u_1^2 :: 7 : 1 : 1$. This gives us weights:

$$q_1 = 8.75 \times 10^{-6},\ q_2 = 5 \times 10^{-3},\ r_1 = 7.69 \times 10^{-7} \tag{9}$$

We use the standard LQR solution available in `GNU Octave` (http://octave.org/) that finds controller gain $\boldsymbol{K}$ by minimizing long term cost J, as a function of model parameters (Equation 7) and weights (Equation 9) to give:

$$\boldsymbol{K} = \begin{bmatrix} -0.81782 & 10.27185 \end{bmatrix} \tag{10}$$

This concludes the design of our controller.

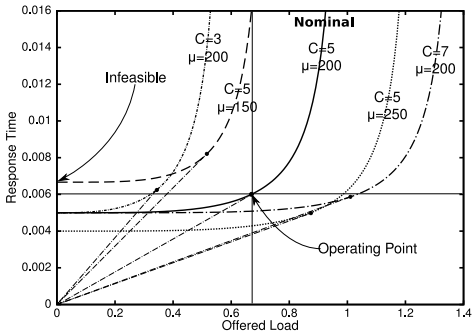

Fig. 3. Changes in response time behavior due to changing capacity of an $M/M/C$ queue. C denotes number of servers and μ denotes service rate.

C. Effects of Variable Capacity

A decrease in available capacity leads to an increase in time taken to serve requests and is thus reflected in increasing response time R and and decreasing goodput Λ. To understand this further, Fig. 3 shows the behavior of R vs. offered load for different system capacities in an $M/M/C$ queue simulation. Two kinds of disturbances are depicted here: a change in service rate μ (e.g. caused by change in allocated resource capacity in a virtualized server system), and a change in number of servers C (e.g. due to dynamic server provisioning or failures). The maximum power points are marked by tangents from origin to each curve. For reasonable magnitudes of capacity change, the FBC has the desired effect of keeping the system close to the knee of the curve which is close to operating point θ for most cases.

Fig. 3 also depicts cases where the operating point is an infeasible target. For instance, the dashed line corresponding to (C=5, μ=150) has a service time 0.0066 that is more than the operating point for response time (0.006). A single-metric based feedback loop would shut down the system in this case, since the minimum difference between the measurement and operating point for R is at load zero. In contrast, by balancing change in response time with change in goodput, our two-metric system settles close to the operating point, i.e. somewhere on the knee of the dashed curve, thus preventing the system from shutting down.

Note that our solution would not be very effective if the capacity change is such that a decrease in number of servers C is accompanied by an increase in service rate μ (e.g. when migrating from five slow servers to two fast servers). In such cases, a new operating point needs to be set.

An increase in response time is indicator of approaching overload, even before goodput gets affected. Our controller reacts to an increase in R by lowering the request admission rate λ_{ADM} (Table I). In the underloaded region, goodput follows admission rate since almost all requests entering the system are completed successfully. Hence an observed reduction in goodput from its operating point means one of three things: (a) overload has set-in causing degradation of goodput Λ, (b) offered load λ_{IN} is less than admitted rate λ_{ADM}, or (c) admitted load λ_{ADM} itself has been reduced by the FBC. Case (a) signals trouble, but is guarded-against in our setup by the response time metric that reacts faster than goodput to approaching overload. Case (b) is benign and the controller need not react at all. Case (c) is where having goodput as a metric in our system state helps. A reduction in λ_{ADM} due to the response time metric will be balanced with a change in the opposite direction suggested by the goodput metric. This action is summarized in Table I.

TABLE I
SUMMARY OF CONTROLLER ACTION

Metric	Change	$x_i(k)$	$u_1(k)$	$\lambda_{\mathrm{ADM}}(k)$
$R(k)$	+	+	+	−
$\Lambda(k)$	−	+	−	+

IV. EXPERIMENTS, RESULTS AND DISCUSSION

We deployed our proxy-based solution on a test-bed consisting of one server with two Intel Quad-core Xeon E5405 2Ghz CPUs and 8GB RAM, and three AMD64 X2 1.6Ghz 2GB RAM client machines. The server hosted both our `MASTH Proxy` [10] and the Apache 2.2.4 Web server. The workload generators were a modified version of `httperf` 0.8 (http://httperf.sourceforge.net/) with added feature of exponentially distributed user think-times. We used our in-house developed `MASTH Proxy` platform since it has been built with a detailed measurement instrumentation and is easily extended to implement a feedback loop. We used a set of Perl CGI scripts doing CPU-intensive work to emulate the Web application. Note that though we have used CPU-bound payload scripts, our method is not specific to CPU as a bottleneck.

We use a `TuningInterval` of $T_S = 10$ seconds for taking control action and `MeasurementInterval` of 5 seconds for recording measurements in the proxy. These values of time-scale are found to address our objective of reacting to (slower) capacity changes while rejecting short-term disturbances due to stochastic noise. The controller gain $\boldsymbol{K}$ (Equation 10) and the operating point θ chosen earlier are inputs to the feedback loop in our proxy-based implementation of the controller. The filtering factors used were: α_1=0.5 for response time and α_2=0.4 for goodput.

To demonstrate the effectiveness of our mechanism, we conduct experiments where we emulate a change in capacity of the server by suitably scaling the service times of the payload scripts. Thus a 33% decrease in capacity is represented by a 50% increase in service time, a 50% decrease in capacity by a 100% increase in service time, and so on.

One can note that the peak goodput of the system is 12 req/s. A 25% reduction in capacity reduces this to 9 req/s, and thus the system operating at 10 req/s will be in an overloaded state. Arrival rate is fixed at λ_{IN}=18 req/s for these experiments.

Fig. 4 shows plots of response time and power with varying disturbance levels. We can see that, even with 50% reduction in capacity, our adaptive admission control scheme is able to keep the response times much lower (max observed 4000ms vs 20000ms for static control) by adapting the admitted rate as compared to a fixed-rate control case configured to operate at 10.4 req/s. The plot of power metric in Fig. 4 shows that for a 33% to 50% reduction in capacity, the (smoothed average) delivered power in our adaptive scheme is up to 6 times higher than that of a static admission control scheme.

There are, however, some oscillations and variance observed in the admitted rate, which are undesirable. Preliminary results suggest that adapting the operating point itself, in response to

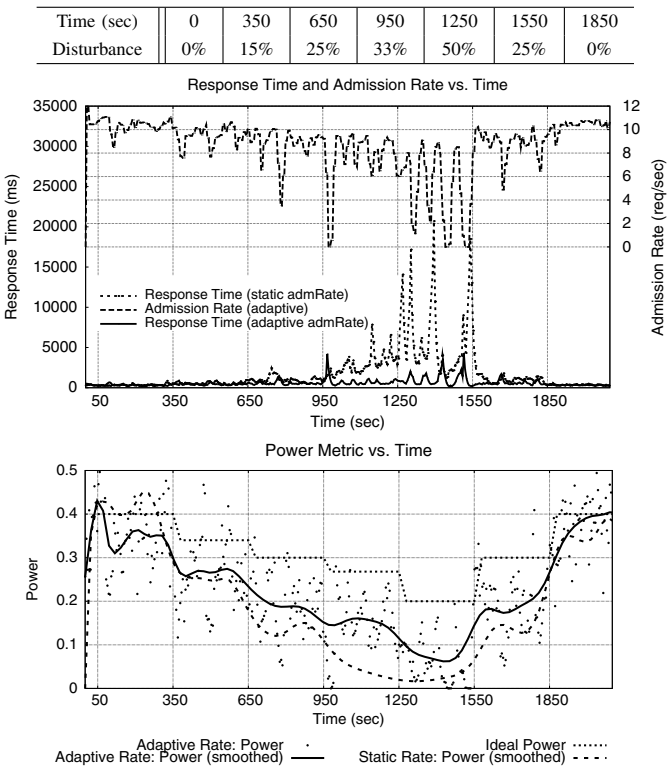

Fig. 4. Comparison of our adaptive scheme with fixed rate control.

changes in capacity can improve the delivered performance further. We are currently exploring such methods.

A. Related Work

Abdelzaher *et al.* have published a series of papers [5] on the use of control theory to provide QoS in Web servers. In these works, the authors provide solutions to providing delay guarantees using control theory based feedback loops to tune per-class resource allocation. However, it is not specified how such delay requirements are obtained for a real application. Diao *et al.* describe [4] a control theory based feedback controller tuned by inputs from a modelling agent, which continuously computes model parameters by observing the system. They control the CPU and memory utilization of an Apache server by tuning the number of worker processes and length of keep-alive requests. Voigt and Gunningberg [1] describe a resource-based adaptive approach to admission control. They implement token-bucket rate regulators in the kernel to avoid over-utilization of resources such as CPU and disk. Their solution is resource and server dependent unlike our proxy-based resource and tier independent mechanism.

The approach taken by the Yaksha [2] controller is the closest to our present work. Yaksha maintains mean response time close to an externally given reference value by controlling the probability of accepting requests. In such a setup, the admitted load is a function of the offered load and sudden unconstrained bursts of traffic can enter the system possibly sending it into overload. In contrast, we use a token-bucket regulator to directly set a limit on the average rate of admitted requests. This allows bursts of only a limited number of requests and the average rate is still maintained at the desired value, independent of the offered load. Yaksha compensates for load variations and inaccuracies in the model parameters, through online tuning of model parameters from measurements. This self-tuning approach lowers the need for system characterization experiments in designing the controller, but raises questions of stability if system capacity is changing. Response time target may no longer be feasible, leading to acceptance probability essentially becoming zero effectively shutting down the system. This is undesirable for Web applications and our two-metric balance setup guards against this.

V. Conclusions and Future Work

We have proposed and implemented an adaptive admission control mechanism for Web applications. Our solution based on control theory adapts the rate of requests admitted into the system when capacity available to the application changes. Instead of enforcing an absolute value for a single metric, our controller maintains a balance between two competing metrics in a manner that the system can remain productive, instead of going into an overloaded state.

Adapting to capacity changes has become an important problem, given the recent trends towards virtualization, shared data centers and cloud computing. However, measuring a distributed system's capacity directly is infeasible making this a challenging problem. Our solution does not assume any fine-grained knowledge of the resources/ topology of the distributed system hosting the application.

Future work involves extending this technique to a session based system with multiple performance classes. Early results indicate that controlling the entry point of the session, based on each class's performance, can effectively control overload.

References

[1] T. Voigt and P. Gunningberg, "Adaptive resource-based Web server admission control," in *Proceedings of the Seventh International Symposium on Computers and Communications*. Washington, DC, USA: IEEE Computer Society, 2002, pp. 219–224.

[2] A. Kamra, V. Misra, and E. M. Nahum, "Yaksha: a self-tuning controller for managing the performance of 3-tiered Web sites," in *Proceedings of the Twelfth IEEE International Workshop on Quality of Service (IWQOS'04)*, June 2004, pp. 47–56.

[3] M. Welsh and D. Culler, "Adaptive overload control for busy internet servers," in *Proceedings of the 4th USENIX Conference on Internet Technologies and Systems (USITS'03)*, Seattle, WA, USA, March 2003.

[4] Y. Diao, J. L. Hellerstein, S. Parekh, and J. P. Bigus, "Managing Web server performance with AutoTune agents," *IBM Systems Journal*, vol. 42, no. 1, pp. 136–149, 2003.

[5] Y. Lu, T. Abdelzaher, C. Lu, L. Sha, and X. Liu, "Feedback control with queueing-theoretic prediction for relative delay guarantees in Web servers," in *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium*. Los Alamitos, CA, USA: IEEE Computer Society, May 2003, pp. 208–217.

[6] D. A. Menascé, M. N. Bennani, and H. Ruan, *On the Use of Online Analytic Performance Models in Self-Managing and Self-Organizing Computer Systems*, ser. Lecture Notes in Computer Science. Springer Verlag, 2005, vol. 3460.

[7] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," University of California at Berkeley, Tech. Rep. UCB/EECS-2009-28, February 2009.

[8] K. M. Moudgalya, *Digital Control*. John Wiley & Sons Ltd., 2007.

[9] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[10] V. Mathur, S. Dhopeshwarkar, and V. Apte, "MASTH proxy: An extensible platform for Web overload control," in *Proc. of the 18th International World Wide Web Conference*, Spain, 2009.

[11] L. Kleinrock, "On flow control in computer networks," in *Proceedings of the IEEE International Conference on Communications*, vol. II, Toronto, Canada, June 1978, pp. 27.2.1–27.2.5.