Auto-Tune Design and Evaluation on Staged Event-Driven Architecture

Zhanwen Li, David Levy
School of Electrical and Information Engineering
University of Sydney
NSW, 2006 Australia
{li_zw, dlevy }@ee.usyd.edu.au

Shiping Chen, John Zic
Networking Technologies Laboratory
CSIRO ICT Centre
NSW, 2122 Australia
{shiping.chen, john.zic}@csiro.au

ABSTRACT
SEDA (Staged Event-Driven Architecture) is a middleware architecture designed to support massive concurrency demands of internet services. However, managing the resources manually to achieve high performance in such a computing system has proved difficult, time-consuming, error-prone and non-QoS-guaranteed. In this paper, we propose an adaptive control approach to automatic resource management and performance control for SEDA-based applications. This approach is based on a combination of a load balancing strategy and feedback auto-tune stages for global optimal performance. In addition, our control algorithms are able to automatically optimize the control parameters at runtime. The design has been built into a SEDA-based web sever and validated by benchmarking this web server. The experimental results demonstrate that our auto-tune design is able to yield superior adaptation performance for SEDA applications in dynamic working environments, achieving desired performance targets with simple control algorithms and automatic parameter tuning.

Categories and Subject Descriptors
D2.9 [**Software Engineering**]: Management – *Model Driven Architecture, Software Configuration Management;* D4.1 [**Operating Systems**]: Process management – *threads;* I.2.8 [**Artificial Intelligence**]: control methods – Control theory; G.4 [**Mathematical Software**]

General Terms
Algorithms, Management, Performance

Keywords
Event-based middleware, Model-driven Performance Management, Feedback Control, Self-tuning

1. INTRODUCTION
Thread-based concurrency model is a common middleware architecture used by server applications. However, this model is not good at handling large concurrent loads due to the overheads associated with resource contention and threading. It has been reported that the server performance would be greatly degraded when the threads/loads reach a certain degree [16, 20, 21].

Alternative to the thread-concurrent model, Staged Event-Driven Architecture (SEDA) is a new middleware architecture to support massive concurrency demands [21]. SEDA models applications as a series of event-driven stages interconnected with event queues and supported by non-blocking I/O [17], avoiding the resource contentions and the scalability limits of threads [16]. As demonstrated by Welsh etc al. [19, 21], this design can greatly benefit the system in massive current loads and service fairness. Although SEDA provides such self-tune control techniques as heuristic control and admission control in each stage, SEDA system performance is still determined by the controlled parameter configurations in each stage. When all stages are optimally set up, SEDA can perform in the best conditions; but managing a multiple-stage SEDA system manually is a very complicated and time-consuming job. Although considerable work has been done for automatic system resource management [3, 4, 5, 6, 7, 8, 10, 11, 19], most of them are based on thread-based concurrency model and do not support the multi-event-queue system as a global control strategy.

In this paper, we propose an adaptive control approach to automatic resource management and performance control for SEDA-based applications. Under the SEDA framework, we exploit global control strategy for load balancing and build each stage as a feedback control system composed of an adaptive controller and an event-driven thread pool. We also develop control algorithms that are able to optimize the control parameters at runtime. Based on the theoretical proofs and experimental results, we argue that our design can not only greatly enhance overall performance of the SEDA application to meet the dynamic desired demands by adaptively adjusting the resources, but it can also reduce a lot of manual work in system configuration.

The remainder of the paper is organized as follows. Section 2 describes the background of SEDA, especially its performance management. Section 3 presents our autonomous control system design for SEDA and exercises our design with three classical control algorithms. Section 4 shows the experimental results of benchmarking our design in a SEDA-based web server. We discuss the related work in Section 5 and give conclusions in Section 6.

2. STAGED EVENT-DRIVEN ARCHITECTURE

2.1 Overview
SEDA innovates from the traditional event-driven design patterns [21]. SEDA partitions a complex business logic into a set of simple basic tasks in order. Each task is processed by a sequence

MODDM '06, November 27-December 1, 2006 Melbourne, Australia

of stages separated by event queues. By means of non-blocking I/O, SEDA applications overcome the weaknesses of previous event-driven designs based on blocking I/O operations and thus are able to efficiently support high concurrent loads with less resources and contentions [1, 17, 20].

In a SEDA-based system, each stage consists of an event handler, an event queue, and a thread pool. Threads within a stage operate by pulling a number of events off the event queue and invoking the event handler. The event handler then processes the events and dispatches the events by enqueueing them in the event queue of the next stage.

The explicit event-queues between stages act as a mechanism for controlling the flow of requests in the whole system. Each stage therefore is isolated from each other and is responsible only for processing a subset of requests to avoid holding resources by single request/thread for too long. [18, 21]

2.2 Performance Management

The performance of each stage in the original SEDA design is subject to its resource control based on the heuristic control algorithm. The controller consists of a thread pool controller and batching controller. Thread pool controller is used to adjust the number of the threads in the stage by monitoring the incoming event-queue length, and the batching controller is designed to tune the batching factor by the performance (throughput) feedback. Whenever the number of events in the incoming queue is over the threshold, the thread pool controller will add a specific number of threads in the stage thread pool; similarly, when the throughput degrades to the value that is less than the recent running average, the batching control mechanism will increase the batching factor in the event handler.

The existing SEDA performance management has a simple design and potentials to handle dynamic loads. However, how to configure it to generate the desired performance requires administers to correctly set multiple relevant parameters such as threshold, thread pool size for multiple stages. It is a very time consuming and tedious manual operation. In addition, this configuration is not based on any mathematical relationships between the controlled parameters and the target performance. An 'optimal' configuration usually depends on an experienced administrator's good guess. Therefore, parameter configuration can easily result in over utilizing resources or under utilizing available resources. Moreover, whether to add or to remove threads, and the number of threads allowed to be changed in every sampling period in SEDA are statically controlled by the admission control mechanism according to the fixed policies, which may be not suitable in a dynamic environment.

3. AUTOMATIC PERFORMANCE CONTROL FOR SEDA

3.1 System Modeling

Generally, computing system performance can be measured with a variety of metrics. In this paper, we take throughput (i.e. expected workload) as the performance target. Since many other performance metrics such as resource utilization and response time can be calculated from system workload [9], if our design is able to control the workload to meet the desired target, it means that our approach also can be extended for other performance requirements.

In SEDA, a client request is processed along a staged pipeline. When there is a bottleneck stage in the process flow, the overall system performance would be limited. Picture a scenario where one stage has heavy database (DB) reads, but all others are essentially main memory manipulation. If the stage to process DB reads cannot improve its workload, the system final throughput might not be enhanced even though other tasks are isolated from DB reads by using different thread queues in SEDA. It implies that if loading cannot be balanced in stages, when other stages support higher workload than the bottleneck stage, it cannot help in improving system performance, but wastes system resources. Balancing workload in stages located in the same pipeline thus is very important for enhancing the whole system performances.

Based on the SEDA design pattern and the above analysis, our adaptive performance management in this paper is designed as the combination of load balance and self-tune stage models. The load balance model provides a framework to globally set up the desired workload on every stage as Figure 1 illustrated. Each stage model is a feedback self control system using adaptive control techniques to adjust stage resources to support the expected stage performance.

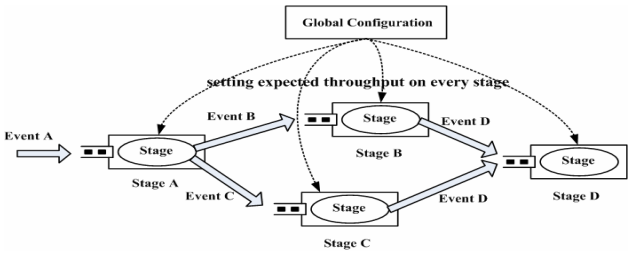

Figure 1. Global Control Framework

Since system throughput is chosen as the performance metrics in the current design, the principle of making the setting in the global control model is that the workload on every stage placed in serial is expected to be the same, i.e. the maximum workload that the system expects to support, for load balance, and the stages working in parallel should share the workload in a specified proportion. For example, when the SEDA application is structured as Figure 1, the desired workloads for each stage should be configured to hold the relationship of $\tilde{T}_A = \tilde{T}_B + \tilde{T}_C = \tilde{T}_D$, where $\tilde{T}$ represents the reference workload for each corresponding stage. After getting the reference control signal from global configuration, each stage can run the auto tune processes to self optimize the stage resources, meeting the need of its performance target. Each stage in Figure 1 can be modeled in detail as shown in Figure 2.

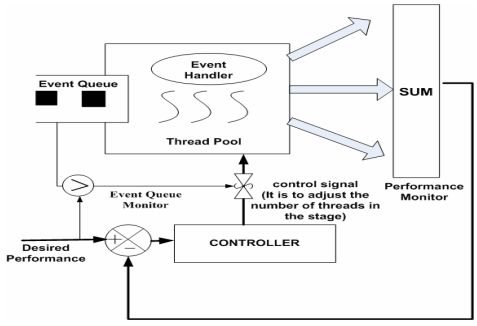

Figure 2. Self-Tune Stage

The stage control model consists of admission control and feedback control. Admission control is to monitor the number of requests that entered the event-queue in the last sampling period and to decide if the feedback controller should be enabled. Generally, this mechanism is specifically for the situation that the desired workload is much higher than the actual number of requests. Feedback control here is developed to auto tune the manipulated parameters to the best values by using automatic control theorems so that the controlled system can generate expected output.

Separated by event queues, in some degree, each stage in SEDA can be viewed as an independent thread-pool based system. Like other thread-based systems, the number of threads in the pool determines the workload of the stage. A higher value of threads size allows the system to process more requests concurrently, increasing the system capacity. So stage thread pool is chosen as the controlled target system in the feedback control system.

Although the relationship between throughput and thread pool size is in fact nonlinear and stochastic, we found that it can be regarded as linear before the number of threads reaches a certain degree. Therefore, by using least mean square (LMS) identification techniques, throughput-thread relationship can be modeled as a first-order ARX model [12, 13] as Equation 1.

$$y(k) = A * y(k-1) + B * u(k) \tag{1}$$

where $y(k)$ and $u(k)$ denote the estimated throughput and change of threads number in the stage at time k respectively; A and B are scalars obtained by system identification. This equation indicates that whenever the number of threads is varied, the output would be changed immediately in the same sample time without any delays, and the output is linear related to the number of the threads changing in this sampling period. In Figure 3, we show the comparison of the actual and estimated throughput of a stage in a SEDA based web server (Haboob [21]).

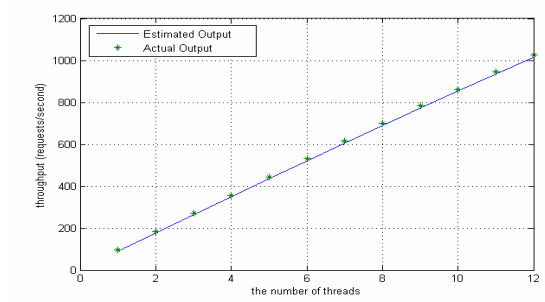

Figure 3. Estimated Output and Actual Output

Despite a difference between the actual and the model output, the comparison demonstrates that modeling the system in this way is correct and the model is sufficient to be used in the following system control.

3.2 Adaptive Control System Design

Base of the above analysis and modeling, we design the adaptive control system as follows. Since our control is discrete, we apply the standard z-transformation [15] on the system equation (EQ1) and we get:

$$\frac{BZ}{Z - A} \tag{2}$$

The stage control model in Figure 2 thus can be simplified to the control flowchart shown in Figure 4.

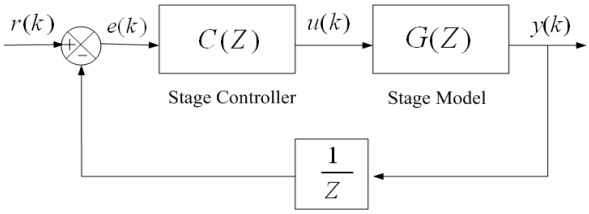

Figure 4. the Stage Control Model

In Figure 4, $C(Z)$ and $G(Z)$ represent the stage controller and stage model respectively. $u(k)$ is the control input to the plant, which is generated by the controller through computing the error lying between the reference and the last system output $y(k-1)$, in the current design, it physically means the change of threads number in the stage. The unit delay in the feedback path is to avoid the algebraic loop and to keep the information of the last output for the control function. When $u(k) > 0$, it means it will add threads in the stage; otherwise, when $u(k) < 0$, it is to say $u(k)$ threads are retired. Whenever the stage gets this control input, the performance output is immediately generated.

The stage control model can be drawn as a system transfer function as below:

$$f_{sys}(z) = \frac{C(z)G(z)}{1 + C(z)G(z)z^{-1}} \tag{3}$$

3.2.1 Proportional Control

We firstly apply the P (Proportional) control on the stage controller. We use K_p to represent the proportional constant in the controller, and the stage model described by EQ 3 can be transformed to be:

$$f_{sys}(z) = \frac{K_p \frac{BZ}{Z - A}}{1 + K_p \frac{BZ}{Z - A} Z^{-1}} = \frac{K_p B}{1 - (A - K_p B)Z^{-1}} \tag{4}$$

Applying inverse Z-Transformation on EQ 4, we achieve the form of the transfer function in time domain, shown as EQ 5,

$$X(n) = K_p B(A - K_p B)^n u[n] \tag{5}$$

where $X(n)$ represents the change of the output at time n, and the $u(n)$ represents the input signal to the system at this sample time. In the design here, $u(n)$ specifically denote the sampled reference input $r(k)$, which is the input to the control system.

According to the system stability requirements, the poles of the closed loop system transfer function should be located in the unit circle. Therefore, a constraint on the choice of the parameter values in EQ 4 is as below:

$$\left| A - K_p B \right| < 1 \tag{6}$$

Thus, from EQ 5 and EQ 6, we know that the system final steady output would be the maximum system output, i.e.:

$$\sum_{n=0}^{\infty} X(n) = \sum_{n=0}^{\infty} K_p B(A - K_p B)^n = \frac{K_p B}{1 - A + K_p B} \quad (7)$$

Whenever the EQ7 equals to the desired output, we can draw the value of K_p, which is definitely the optimal value of K_p. Therefore, we can follow the processes (EQ8~EQ10) as below to obtain this optimal values.

Firstly, let:

$$1 - A + K_p B = 1 \quad (8)$$

then we can get:

$$K_p = \frac{A}{B} \quad (9)$$

Since the system (Figure 3) final steady output is equal to A, we can achieve the desired output by developing a simply transformation equation as follows:

$$y(k) = \alpha A r(k) \quad (10)$$

And let

$$\alpha = \frac{1}{A} \quad (11)$$

The system model described in Figure 4 thus can be designed as shown in Figure 5 when it is working with proportional controller.

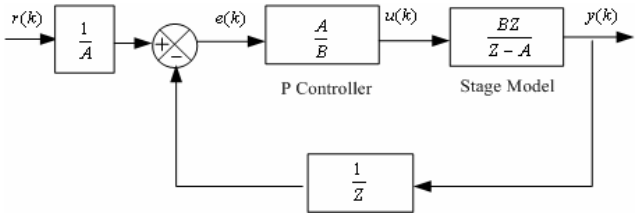

Figure 5. P- Control Based Pre-Compensator Control Model

3.2.2 PI / PD Control

PI (proportional-integral) and PD (proportional-derivative) are two well-known control techniques in classical control theory, which add integral and derivative components respectively on the proportional control [2, 10].

There have been a lot of discussions on how to get the best values of these controlled parameters to achieve the desired output. However, if the system is discrete and the zeros in the transfer function affect a lot on the system performance, there is little previous work to deal with this problem [10]. In this section, we propose a new approach to deriving the optimal parameter values.

An ideal controller usually enables the controlled system to perform as expected. In general, the overshoot and settling time are the two most important performance metrics for automatic control systems. In the following, we take these two parameters as our control targets to develop our PI/PD control algorithms.

We first present the design of PI control system. Similar to the above P control algorithm, from EQ 3, the PI control system model can be depicted in the equation as EQ 12

$$f_{PI}(z) = \frac{B(K_p + K_i)Z^2 - BK_p Z}{Z^2 - (A + 1 - K_p B - K_i B)Z + A - K_p B} \quad (12)$$

where K_p and K_i represent the proportional and integral constants.

Converting this model from Z-plane to the time domain, we have EQ 13 that shows the change of the system performance output $X_{PI}(n)$ resulted by the current input signal.

$$X_{PI}(n) = \frac{B(K_p + K_i)r^{n+1}\sin(\theta n + \theta) - BK_p r^n \sin(\theta n)}{r\sin\theta} u[n] \quad (13)$$

where r and θ mean the distance and angle of the pole in the z-plane respectively. Because of the stability and performance requirements, we can demand:

$$0 < r < 1 \text{ and } \theta \in [0, \frac{\pi}{2}] \quad (8)$$

Taking the desired settling time K_s into account, we can achieve the range of r by solving the equations of EQ 15 for PI.

$$[1 - 2\frac{\sin(\theta K_s + \theta)}{\sin(\theta K_s)}\cos\theta]r^2 + \frac{\sin(\theta K_s + \theta)}{\sin(\theta K_s)}(1 + A)r - A = 0 \quad (15)$$

Because the system output equals to

$$y(n) = \sum_{n=0}^{k} X(n) \quad (16)$$

with the r obtained from EQ 15, the desired settling time and the overshoot, we then can get the proper range of θ by calculating EQ 16, and finally we can use EQ17 to obtain K_p and K_i.

$$K_p = \frac{A - r^2}{B}$$

$$K_i = \frac{r^2 + 1 - 2r\cos\theta}{B} \quad (17)$$

with this algorithm, the K_p and K_i are a set of combined values that meet the control demands. Any pair applied in the adaptive controller can achieve the desired performance.

PD controller development in our algorithm uses the same design process as the above PI. Only because of the difference lying on the system transfer function, the equations related the system model (EQ 12, 13, 15 and 17) should be replaced by PD equations (EQ 18, 19, 20 and 21) respectively as below

$$f_{PD}(z) = \frac{B(K_p + K_d)Z^2 - BK_d Z}{Z^2 - (A - K_p B - K_d B)Z - K_d B} \quad (18)$$

where K_p and K_d represent the proportional and derivative constant respectively. The time domain behavior then is described as EQ 19:

$$X_{PD}(n) = \frac{B(K_p + K_d)r^{n+1}\sin(\theta n + \theta) - BK_d r^n \sin(\theta n)}{r\sin\theta} u[n] \quad (19)$$

EQ 15 of PI correspondingly is changed into EQ 18 for PD

$$r = \frac{A\sin(\theta K_s + \theta)}{2\cos\theta\sin(\theta K_s + \theta) - \sin(\theta K)} \quad (20)$$

Finally, we can use EQ 21 to replace the EQ 17 of PI to achieve the optimal parameters for PD control to meet the system performance requirements.

$$K_D = -\frac{r^2}{B}$$

$$K_p = \frac{A - 2r\cos\theta}{B} - K_D \quad (21)$$

4 TESTING AND ANALYSIS

We validate our design by implementing it with the above three control strategies into a SEDA-based web server [21] and evaluate our approaches by benchmarking the web server. The testbed consists of one server machine (2.8 GHz Pentium 4 systems with 1.5 GB of RAM) and a client machine (2.0 GHz Pentium 4 systems with 512MB of RAM). The SEDA web server is developed with SUN JDK 1.5 as the JAVA platform1 running Linux kernel v2.6. The client is running synthetic workload generator based on the SPECweb 99 testing suite [21, 22]. .

Figure 6 demonstrates the performance results of the tests. Here, we take the same performance tests on each control strategy. Without human intervention, each controller adaptively tunes the parameters to the values which achieve the desired throughput changed at runtime. As the figure shows, our control design is able to efficiently yield the desired output for SEDA applications under dynamic loading environment and its behavior is similar to the above theoretical arguments in the experiment. Despite some oscillations in the behavior, the controller is very effective at keeping the performance near the target. The oscillation here might be resulted by Java's garbage collection.

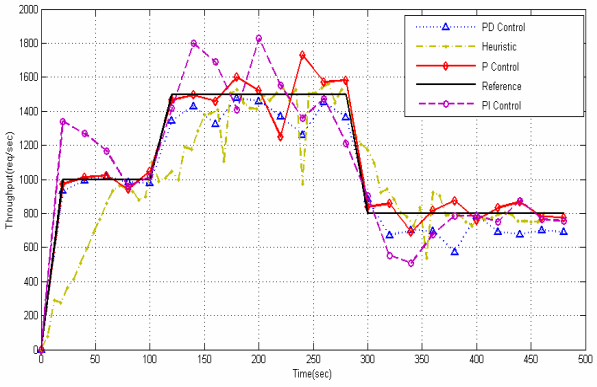

Figure 6. the Performance on SEDA-based Web Server

[1] We used SUN's NIO ([1] [14]) to implement non-blocking I/O

Among these four control approaches, the heuristic controller used in original SEDA shows the slowest adaptation rate, and the convergent time is strongly affected by the change of the reference signals. In theory, PI control is a sound way in improving the system performance by removing the steady-state error. However, in this case, the controlled system mathematical model limits the positions of the zeros and poles in the control function and thus shrinks the scope of and for selections. In terms of the model constraints, we therefore can obtain the minimum overshoot at 30% in this experiment as shown in Figure 6. With regards to the PD control, it demonstrates a promising performance in this test with greater errors shown in Figure 6. Contrasting to the above control approaches, P control exhibits the best performance. It not only has the fastest adaptation rate and the least errors, the design process of P control is also the simplest. Moreover, the performance of P control is independent to the reference signals. In general, steady-state error is viewed as a significant weakness existing in P control. In this research, we apply both mathematical proof and experiment results to demonstrate that our design is good at remedying this problem.

Although in other working environments, PI and PD controller may perform better than these tests, however, either PI or PD design process is much more complicated than P control and the steady-state errors and convergent rate achieved by PI/PD cannot be better than P. Based on these facts, we argue that proportional control could be the best selection for this autonomous control system design.

5 RELATED WORK

Applying control in autonomous computing has attracted a lot of research efforts. The control approaches generally can be cataloged into two main streams: one is the admission control such as queuing control [3, 19], and the other is the feedback-based control [4, 5, 8, 10, 11]. Besides the two main streams, some other approaches like neural-fuzzy control [6], resource containment and service degradation etc [19] also attract some attention.

In general, the admission control is simpler, but has limitations like heavy manual costs, slow convergence and fixed policies etc. The original SEDA is a typical example of admission control. In contrast to the admission control mechanism, our approach shows better adaptation performance in a dynamic loading environment, and demonstrates that it can reduce manual work in parameters setting and tuning for system design and management.

Feedback-based control approaches exhibit significant advantages in high quality self-correcting and self-stabilizing. As a result, recently both software industry [4, 5] and academic community [8] put a lot of efforts in this area. In [8], Lu and Abdelzaher used PI control to adaptively adjust the thread pool size and demonstrated that feedback control is a useful set of tools for managing resources utilization and QoS. In [4], based on the auto-tune agent design, IBM took a further step by drawing another argument that LQR can perform very well for MIMO systems. Our design is also a kind of the autonomous computing system in this catalog. Compared with the previous work, our model exhibits the following advantages. First, our design is a global control strategy, rather than a single thread pool model-based control approach. Based on the SEDA pattern, every stage in SEDA works as a thread-based concurrency model. On some degrees, the single thread pool model like Apache can be regarded as a special case

of the SEDA architecture that processes all requests in one stage. This implies that our approach is also available to fully support the thread pool models if required. Second, we take the service rate as the performance metric, which gives our control approach potential to improve other performances. In addition, we exploited the classical control modules (P, PI, PD) in our auto-tune design. Compared with other complicated adaptive controls on autonomous computing, our approach is simple and able to offer an alternative way to efficiently self-tune the system working in dynamic workload environments.

As a note, our design currently focuses on web servers running in single machine. However, our design and approach can be potentially used to support multiple machines for a complicated distributed system.

6 CONCLUSIONS

In this paper, we presented and evaluated an auto tune design for SEDA-based application performance management. Our contributions include a design of adaptive control model based on feedback control, as well as developing a practical approach to optimize the control parameters. A SEDA-based web server was used to validate our design. Three well-known control models (P, PI and PD) were evaluated using the SPEC web benchmark against the web servier. The experimental results show that our auto-performance controller can effectively optimize the resources in SEDA-based applications and significantly reduce manual configurations. Our work demonstrates that, instead of applying complicated control theory and algorithms, P control based pre-compensation model is good enough to control multiple-stage software systems.This makes it feasible to build our approach into SEDA middleware and apply it for a large range of applications. In the future, more complicated SEDA-based applications will be used to further improve and extend our approach.

7 REFERENCES

[1] Beltran V., Carrera D., et al.: Evaluation the scalability of java event-driven web servers. Proc. of International Conference on Parallel Processing (ICPP'04), IEEE. (2004)

[2] Benjamin C. Kuo and Golnaraghi F.: Automatic Control Systems (8th edition), Wiley, ISBN: 0471134767. (2002)

[3] Chen H. and Mohapatra P.: Session-based overload control in QoS-aware Webservers. Proc. of IEEE INFOCOM2002, pages 516-524. (2002)

[4] Diao Y., Hellerstein J.L., et al.: Managing web server performance with autotune agents. IBM System journal Vol 42, No 1, pages 136-149. (2003)

[5] Diao Y., Gandhi N., et al.: Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache Web server. Proc. of the Network Operations and Management Symposium, Florence, Italy.(2002)

[6] Diao Y., Hellerstein J.L., et al.: Optimizing quality of service using fuzzy control. Proc. of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management. Springer-Verlag, pages 42--53. (2002)

[7] Liu X., Lui Sha, et al.: Online Response Time Optimization of Apache Web Server. Proc. of the 11th International Workshop on Quality of Service (IWQoS 2003), pages 461-478. (2003)

[8] Lu C., Abdelzaher T.F., et al.: A feedback control architecture and design methodology for service delay guarantees in web servers. Technical Report CS-2001-06, University of Virginia, Department of Computer Science. (2001)

[9] Menasce D.A. and F.Almeida V.A.: Capacity Planning for Web Services Metrics, Models, and Methods, Prentice Hall PTR Upper Saddle River, N.J.07458, ISBN: 0-13-065903-7. (2002)

[10] Hellerstein J.L., Diao Y., et al.: IBM Research Report: applying control theory to computing systems. Proc. of Computer Science RC23459 (W0412-008). (2004)

[11] Hellerstein J.L., Diao Y., et al.: Feedback control of computing systems, IEEE Press Wiley-Interscience, ISBN:0-471-26637-X. (2004)

[12] Ljung, L.: System identification: theory for the user 2nd ed. Upper Saddle River, N.J: Prentice Hall, ISBN: 0136566952. (1999)

[13] the MathWorks Inc: System Identification Toolbox

[14] SUN Microsystems INC. New I/O APIs. http://java.sun.com/j2se/1.4.2/docs/guide/nio. (2002)

[15] Oppenheim A.V., Schafer R.W., et al.: Discrete-time signal processing, Prentice Hall Signal Processing Series, ISBN: 0137549202. (1999).

[16] Vivek S. Pai, Peter Druschel, et al. Flash: An efficient and portable Web server. Proc. of the USENIX 1999 Annual Technical Conference. (1999).

[17] Welsh M.: NBIO: Nonblocking I/O for Java http://www.eecs.harvard.edu/~mdw/proj/java-nbio/

[18] Welsh M.: An architecture for highly concurrent, well-conditioned internet services (Thesis). Computer Science, University of California at Berkeley. (2002)

[19] Welsh M. and Culler D.: Adaptive Overload Control for Busy Internet Servers. Proc. of the Fifth USENIX Symposium on Internet Technologies and Systems (2003)

[20] Welsh M. and Culler D.: Virtualization considered harmful: OS design directions for well-conditioned Services. Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS VIII). (2001)

[21] Welsh M., Culler D., et al.: SEDA:An architecture for well-conditioned scalable internet services. Proc. of the 18th ACM Symposium on Operating Systems Principles, anff, Canada. (2001)

[22] SPECweb99 Benchmark, Copyright © 1995 - 2006 Standard Performance Evaluation Corporation http://www.spec.org/web99/