Towards Runtime Optimization of Software Quality Based on Feedback Control Theory

Bihuan Chen, Xin Peng, Wenyun Zhao
School of Computer Science, Fudan University, Shanghai 200433, China
{0572227, pengxin, wyzhao}@fudan.edu.cn

ABSTRACT
The increasingly complex environments in which software systems are running today have made runtime software quality unstable and hardly in an optimal state, especially for those systems in open and dynamic environments, e.g. Internetware. In this paper, we explore the effectiveness of software cybernetics and feedback control theory in runtime software quality optimization. We propose a method of runtime quality optimization by using feedback control theory. Specially, we consider the problem of runtime optimization for a specific quality attribute, namely throughput, for Web-based systems. We design a double-layer feedback control model for the problem and implement the runtime optimization control method. In the method, runtime feedbacks are collected and used by the control model to adjust related control parameters. The experimental study has demonstrated the effectiveness of software cybernetics and feedback control theory in runtime quality optimization.

Categories and Subject Descriptors
D.3.4 [**Software Engineering**]: Processors – *optimization, run-time environment.*
D.4.8 [**Software Engineering**]: Performance – *monitors.*

General Terms
Control, Software Quality, Self-Adaptation

Keywords
Feedback Control, Quality Optimization, Throughput, Runtime Reconfiguration.

1. INTRODUCTION
In traditional software development, the architects need to make design decisions at design time according to the quality requirements and predicted runtime environments and resource constraints of the target system. This kind of static design decisions can make applicable system designs if the runtime environments and resource constraints, e.g. load, memory, and bandwidth, can be properly predicted. However, runtime environments are difficult to be predicted exactly, especially for Internetware, which keeps evolving after deployment due to the open, dynamic, and ever-changing Internet, as well as the various user preferences [1]. This problem often makes the runtime quality of a system unacceptable or unstable. In order to provide the capability of runtime optimization, some systems involve dynamic configurations to enable the system administrators to tune quality factors at run time. However, the manual optimization has several deficiencies, such as additional administrator efforts, slow reactions, and hard to make precise tuning decisions. Therefore, the traditional static design-time decisions and manual tunings at run time can not ensure an optimal runtime quality in an uncertain and changing environment.

Recently, self-adaptation techniques have been proposed as a solution to runtime quality optimization. Being one of the basic features of Internetware, self-adaptation means that the software system can monitor its runtime state and behavior and adjust them when necessary according to pre-defined policies [2]. Self-adaptive systems can configure and reconfigure themselves at run time to handle such things as changing user needs, system intrusions or faults, a changing operational environment, and resource variability [3]. Therefore, self-adaptive systems can optimize their runtime quality by context-aware adaptations. In this paper, we focus on runtime adaptation to the changing environments with the purpose of quality optimization like response time, throughput, etc.

In general, self-adaptive systems can be characterized by the three core functionalities [4]: (1) monitor (sensing) the environment to recognize "problems", (2) take decisions on which behavior to exhibit, and (3) realize the behavior change by adaptation. For the problem of runtime quality optimization, self-adaptive decision, that is to decide the right optimization action according to the runtime environment, is a key point. Recently, software cybernetics [5-6] has been introduced to implement optimization control based on runtime feedback. Software cybernetics explores the interplay between software and control and is motivated by the fundamental question whether or not and how software behavior can be controlled [6]. Feedback control theory deals with the control of the behavior of dynamical systems. It monitors the behavior of the controlled object and uses it in an online feedback control strategy to achieve the desired behavior. Once a deviation from the desired behavior is detected, a set of alternative solutions is computed to correct for such derivation [6]. In other words, feedback control can endure the sudden disturbance in uncertain environments. The relationship between quality attributes and runtime environments is complex, fuzzy, and hardly described precisely. Therefore, it is an effective

Internetware'09, October 17–18, 2009, Beijing, China.

method to achieve continuous quality optimization by using the control theory based on runtime feedbacks.

To this end, we propose a method of runtime quality optimization by using feedback control theory. Specially, we consider the problem of runtime optimization for one single quality attribute, namely throughput, for Web-based systems. We design a double-layer feedback control model for the problem and implement the runtime optimization control method. Our experiments on a Web-based system have shown that the throughput can be remarkably and continuously improved. Our work has demonstrated the effectiveness of software cybernetics and feedback control theory in runtime quality optimization, and founded the basis for further research of a general runtime optimization method for multiple quality attributes.

The remainder of this paper is organized as follows. Section 2 presents the background of the feedback control theory and the optimization control problem. Section 3 describes our method in detail. Section 4 repots an experimental study with analysis and discussion. Section 5 introduces some related work. And finally, Section 6 concludes the paper and plans the future work.

2. BACKGROUND

This section first presents the theory foundation, the feedback control theory in the field of automatic control, and then introduces the optimization control problem of throughput in Web-based systems.

2.1 Feedback Control Theory

In control theory, a control model can be classified into closed-loop control or open-loop control by whether or not the output of a system influences the control process. Closed-loop control is also called feedback control. Although feedback control is complex and probably causes stability issues, it has the following advantages over open-loop control [7]: guaranteed performance, distinctly reduced system error, improved control precision, disturbance rejection, and reduced sensitivity to parameter variations.

The basic structure of feedback control is presented in Figure 1. The *output* of a system which is monitored is fed back to the *Controller*. The *Controller* then takes the *error* (delta) between *set point* and *output* to control the *Process*, the controlled object, and drive the *output* to be close to *set point*.

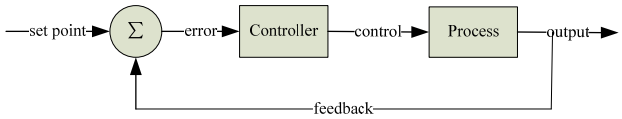

Figure 1: Basic Structure of Feedback Control

2.2 The Optimization Control Problem of Throughput in Web-based systems

In this paper, we consider the case of throughput optimization for Web-based systems under high-load concurrent accesses. In this situation, usually the system designers will limit the maximum online users to guarantee the service quality and prevent the system to crash. Figure 2 shows the relationship among the average response time, the throughput and the number of online users with the change curves captured in our experiment. It can be seen that the throughput firstly increases with the increment of online users, but after the optimal point (the white triangle on the curve) is reached it decreases. We can see response time is the negative factor for throughput: it keeps increasing along with the number of the online users. Moreover, sometimes the server may even crash if the load keeps increases greatly.

Usually the system uses a parameter *limit* to control the maximum online users: if the number of online users is greater than *limit*, then the system will refuse more users to logon. Therefore, the control parameter *limit* largely affects the performance and throughput of the system. If the *limit* of the number of online users is too high, e.g. the white square on the curve, the server's performance will drop dramatically. If the *limit* is too low, e.g. the white circle on the curve, the server's resources will be wasted. The appropriate *limit* is closely related to server performance, availability of runtime resources, access load, etc. So it's usually not a reasonable way to set the *limit* to a constant.

In traditional software development, usually, the parameter *limit* will be set before running according to some kind of experiential estimation, and be manually adjusted by the system administrator at run time according to the system status. A critical problem of this kind of subjective and experiential control is that the system is often running in a non-optimal state, i.e. overloaded or underloaded. For example, the *limit* is fixed to 100 before running. Due to the changing runtime environment (CPU utility, memory available, etc.), sometimes it can serve 110, 120, or even more concurrent user accesses, but the throughput can not be further improved when the upper limit of concurrent user accesses has been reached (100). Then the system is in an underloaded state, and the server resources are not adequately utilized. Sometimes the server can only serve 90, 80 or even less concurrent user accesses, but the system may allow more concurrent users. Then the system is in an overloaded state, and the average respond time will increase greatly, making the throughput decline. Therefore, the corresponding optimization control problem here can be concluded as: how to dynamically adapt the control parameter *limit* to make the system always run at a fully loaded state (non-overloaded and non-underloaded). An direct index for the optimization is the system throughout, which can be measured by the requests successfully handled by the system in a time unit, e.g. the number of new order records in an online shopping system.

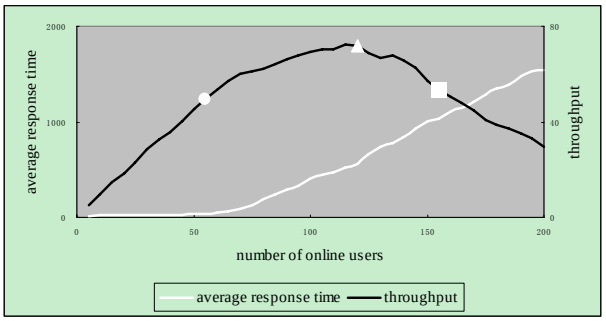

Figure 2: Average Response Time, Throughput VS Number of Online Users

3. RUNTIME OPTIMIZATION METHOD BASED ON FEEDBACK CONTROL THEORY

In this section, we first present an overview to the proposed optimization method, and then introduce the PID control model used in our method and our double-layer Feedback controller for throughput optimization.

3.1 Method Overview

The framework of our method is presented in Figure 3. The Web-based system uses a control parameter *limit* to control the maximum online users as most Web-based systems. What is different in our method is that the parameter is not fixed or updated manually by the administrator, but dynamically adapted by the feedback control module according to the feedbacks reflecting the changing environments. Thus, the system can continuously run with a proper number of online users that is well adapted to the changing environments, and the system throughput is then largely optimized.

The framework covers the three core functions in self-adaptive systems: monitoring, decision making and adaptation. Runtime monitoring is responsible for collecting and analyzing runtime information such as the number of requests, average response time, throughput, etc. Based on the monitored information, the feedback control module decides the adaptation to the control parameter *limit* by using our double-layer feedback control model. Then the control parameter is reconfigured according to the decision and the system will use the new parameter until it is updated in the next adaptation. The optimization is an iterative process that can be conducted at a fixed frequency, e.g. every 1 minute.

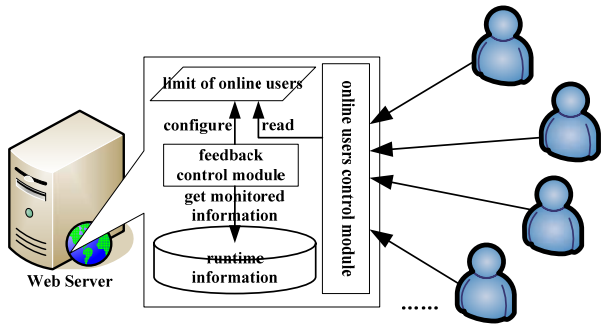

Figure 3: Method Overview

3.2 The PID Control Model

The core of the feedback control is the design of the *Controller* in Figure 1. There are many types of control techniques available such as P controller, PI controller, PD controller, and PID controller. The classical PID controller has been adopted here in consideration of the precision, stability, and responsiveness. The PID controller drives the *output* to the *set point* (the desired behavior) by using the feedback information. Equation 1 is the formulation of the PID control algorithm where *u(t)* is the control variable of the controller at time *t*, and *e(t)* (*set point* - *output*) is the error signal at time *t*.

$$u(t) = Kp * e(t) + Ki * \int_0^t e(\tau)d\tau + Kd * \frac{de(t)}{dt} \quad .....(1)$$

As shown in Equation 1, PID controller is composed of three parts: the proportional control, the integral control, and the derivative control. *Kp* is the parameter of the proportional control which deals with the current behavior of the *process* (reaction to the current error); *Ki* is the parameter of the integral control which deals with the past behavior of the *process* (reaction based on the sum of the recent errors); *Kd* is the parameter of the derivative control which deals with the future behavior of the *process* by predicting (reaction based on the rate at which the error has been changing) [7] [8]. *Kp*, *Ki*, and *Kd* are user-defined constants which vary from one control system to another. By tuning the three constants at design-time, the PID controller can provide control action designed for specific process requirements such as high precision, minimum oscillations, quick response, and low overshoots.

3.3 The Double-Layer Feedback Controller for Throughput Optimization

After analyzing the control problem presented in section 2.2, we design the *Controller* as a double-layer controller (the inner PID controller and the outer simple feedback controller), as show in Figure 4.

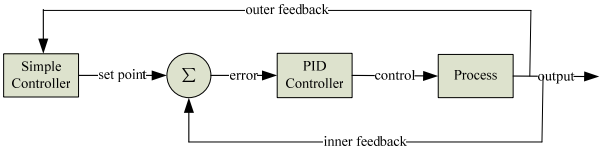

Figure 4: Structure of the Double-layer Controller

3.3.1 The Inner Controller

The inner feedback control adopts the PID controller. Its *set point* is the desired value of the average response time of the Web server. The difference between the *set point* and the *output* (the monitored average response time) is *e(t)*. *u(t)* is calculated by the PID control algorithm. Then the *limit* is increased/decreased by *u(t)/set point * limit* according to whether *u(t)* is positive or not. As will be discussed in section 3.3.3, the inner controller serves as the bridge to optimize the overall quality (throughput). The objective of the inner controller is to reconfigure the *limit* to stabilize the average response time to the desired value.

$$u(t) = Kp * e(t) + Ki * \sum e(t) + Kd * (e(t) - e(t-1)) \quad .....(2)$$

$$u(t) = u(t-1) + Kp * (e(t) - e(t-1)) + Ki * e(t) + Kd * (e(t) - 2 * e(t-1) + e(t-2)) \quad .....(3)$$

The formulation of the PID control algorithm is Equation 1, which can be transformed into Equation 2 for simplicity based on numerical calculation theory. However, Equation 2 is related to all the past states of the system (the sum of the past errors). To eliminate the accumulation of errors, Equation 2 can be further transformed into Equation 3 (the incremental formulation) which is only related to the past three states: *e(t)*, *e(t-1)*, and *e(t-2)*. As

mentioned in section 3.2, *Kp*, *Ki* and *Kd* are user-defined constants which are decided at design-time. Here, we tune them manually by the method in [9] by the following steps: (1) initially set *Kp* to a small value and set *Ki*, *Kd* to zero, then increase *Kp* by step until the *output* oscillates; (2) set *Kp* to half of its value, then increase *Ki* until any offset is correct in sufficient time for the *Process*; (3) finally increase *Kd* until the *output* is acceptably quick to reach the *set point*.

3.3.2 The Outer Controller

The outer feedback control adopts the simple feedback controller. It dynamically controls the *set point* of the inner PID controller based on the monitored throughput of the Web server. The simple feedback controller just accepts the heuristic strategy: if the throughput increases, the *set point* of the inner PID controller will be increased by some constant; if the throughput decreases, the *set point* will be decreased by some constant. To avoid oscillations, the increment and the decrement should be different. The objective of the outer controller is to optimize the throughput under the high-load environment together with the inner PID controller through reconfiguring the *set point*.

3.3.3 Why Use Double-layer Controller

There are three main reasons for choosing the double-layer controller over single-layer controller. Firstly, the throughput is not linear with the number of online users while the average response time is approximately linear with the number of online users. As shown in Figure 2, the throughput initially increases with the increment of online users; gradually, the server trends to be fully loaded; after reaching the optimal point, it begins to decrease. However, the average response time keeps increasing along with the number of online users in the rough. On the other hand, the PID controller is only applied to linear relationships, which means we can't directly use the PID controller to control the throughput.

Secondly, the single-layer controller which uses the heuristic strategy has low precision due to the fixed increment or decrement to the *limit*, which is unacceptable to our optimization control problem.

Finally, our overall control objective is to continuously maintain the throughput in the optimized state against the runtime environment, which keeps changing at run time. Therefore, the desired throughput can not be set to a fixed value. However, what PID controller can do is to drive the *output* to the *set point* (desired value). Therefore, only single-layer controller can't satisfy our objective.

Hence, we adopt the high-precision PID controller as the inner controller to precisely reconfigure the *limit* to stabilize the average response time to the desired value, and the low-precision simple feedback controller as the outer controller to dynamically reconfigure the set point of the inner controller to achieve optimal throughput performance.

4. EXPERIMENTAL STUDY AND DISCUSSION

In this section, we demonstrate and validate the proposed method using a Web-based system. We first introduce the implementation of the experiment, and then compare the results using different control models and discuss related issues.

4.1 Implementation

In our method, runtime monitoring can be different from one platform to another and from one application to another as long as it can get the essential runtime information. Here, it is realized through online analysis of the access log of the Web server which we choose Tomcat in our experiment. Tomcat's default configuration file (server.xml) should be modified to generate the access log, which generates one line of information for each request processed by the server in a standard/custom format. In our experiment, the format is customized to "%h %t %r %s %D", in which %h stands for remote IP address, %t stands for date and time, %r stands for method and request URI, %s stands for HTTP status code of the response, and %D stands for time taken to process the request (in millisecond). The access log is analyzed every other minute to get the essential information such as total number of requests, the total time taken to process the requests and the total number of operations finished after logon. On the basis of the three values, the monitored average response time and throughput is calculated.

Decision making is realized by the double-layer control model designed in section 3.3. The inner controller performs the PID control using the monitored average response time while the outer controller performs the simple feedback control using the throughput. The three constants, *Kp*, *Ki*, and *Kd*, of the inner PID controller are manually tuned and set to 0.3, 0.2, 0.3 respectively.

Adaptation is relatively simple. It dynamically reconfigures the control parameter *limit* according to the control result (the desired *limit*) of the inner controller.

The whole feedback control process (monitoring, decision making and adaptation) is timed and automatic. We use the Interface ServletContextListener and the Class TimerTask in JAVA to implement it. After the Web server starts, the process is executed immediately.

In our experiment, the stress testing tool JMeter was used to provide continuous concurrent accesses of 300 users to stimulate the actual runtime environment of the Web-based system. In detail, there were always 300 users to try to logon to the Web-based system. If the actual number of online users was smaller than the *limit*, logon was allowed, and the user performed a series of operations including logout. Otherwise, logon was rejected. Meanwhile, the access log was analyzed every other minute to get the average response time and the throughput.

The experiment was separated into 4 steps: (1) traditional static design, (2) single-layer simple feedback control, (3) single-layer PID control, (4) double-layer control. Every step ran for 2 hours. As will be seen, the throughput increases by degrees from (1) to (4).

4.2 Results and Comparisons

In our experiment, the throughput curves of the system under the four kinds of optimization designs, namely traditional static design, single-layer simple feedback control, single-layer PID control and double-layer control, are given and compared.

4.2.1 Traditional Static Design

Traditional static design usually sets the *limit* to a constant which makes the Web-based system achieve a high throughput under a specific environment (design/test). The *limit* is set at design time, or at installation time, or before running, which

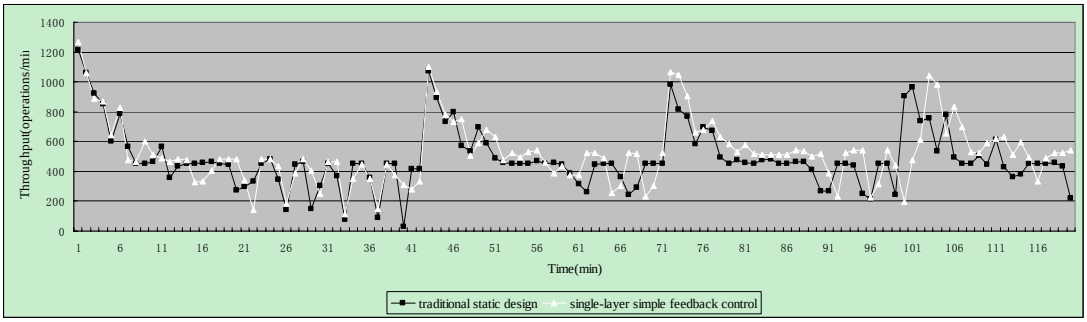

Figure 5: Traditional Static Design VS Single-layer Simple Feedback Control

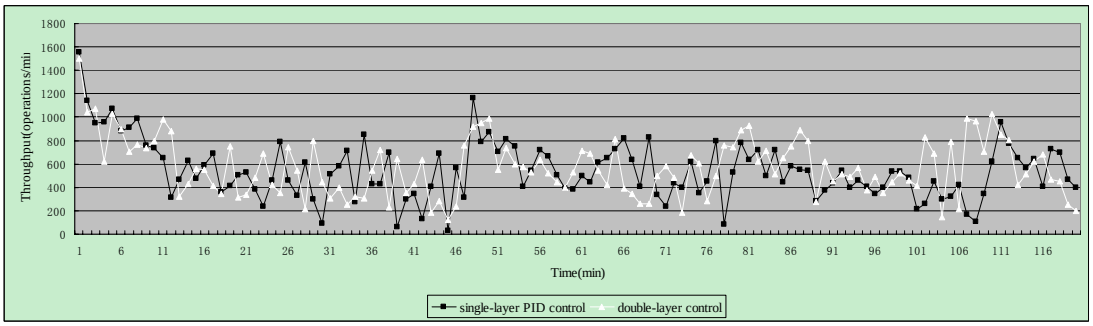

Figure 6: Single-layer PID Control VS Double-layer Control

won't change at run time. Whether the limit is appropriate at run time is also unknown. As a result, such a system can't adapt to the changing situations under the uncertain environment.

Here, the *limit* was set to 150. The black curve of Figure 5 shows the relationship between throughput and time. The throughput is relatively stable on the rough. However, throughput itself is a nondeterministic nonfunctional attribute, and the environment such as memory utility, CPU utility, and etc. is changing, which causes throughput unstable at times as shown in Figure 5. In this step, the average throughput is 488 operations per minute.

Here, the initial limit was also set to 150. The white curve of Figure 5 shows the relationship between throughput and time. In Figure 5, the white curve is basically similar with the black curve except that it is a littler higher, which means a higher throughput. In this step, the average throughput is 529 operations per minute. Compared with the traditional static design, it is increased by 8.4%. Meanwhile, the average number of online users is 155. This shows the fact that the improvement of the throughput is obvious although only simple feedback control is adopted.

4.2.3 Single-layer PID Control

The single-layer PID controller refers to the inner controller mentioned in section 3.3.1 which controls the average response time but not the throughput. The *limit* is increased/decreased applying the PID control algorithm using the monitored average response time and the desired average response time (600ms). The configuration is made every other minute. It is the practical foundation for using double-layer controller to observe the throughput through the control of average response time.

Here, the initial *limit* was also set to 150. The black curve of Figure 6 shows the relationship between throughput and time. The

4.2.2 Single-layer Simple Feedback Control

The single-layer simple feedback controller refers to the controller similar to the outer controller in section 3.3.2. The controller also accepts the heuristic strategy: In the latest 10 analyses of the access log from the previous 15 analyses, if the average throughput increases, the limit is increased by some constant; if the average throughput decreases, the *limit* is decreased by some constant. To avoid oscillations, the increment and the decrement should be different. The configuration is made every 16 minutes. Due to the fixed increment or decrement to the *limit*, the control lacks of precision.

throughput fluctuates more severely than in Figure 5, which is the true description of the runtime changing environment. In this step, the average throughput is 544 operations per minute, which is increased by 11.5% compared with the traditional static design, and by 2.8% compared with single-layer simple feedback control. Meanwhile, the average number of online users is 180.

4.2.4 Double-layer Control

The double-layer controller is introduced in section 3.3, which employs a simple feedback controller to the *set point* (desired value) of the single-layer PID controller for the purpose of dynamically adjusting the desired value according to the throughput. The outer controller makes the configuration every 16 minutes while the inner controller makes the configuration every other minute.

Here, the initial *limit* was also set to 150, and the initial desired average response time of the inner controller was set to 600ms. The white curve of Figure 6 shows the relationship between throughput and time. The throughput also fluctuates more severely than in Figure 5, but is similar with the black curve in Figure 6 because double-layer controller is evolved from the

single-layer PID controller. In this step, the average throughput is 574 operations per minute, which is increased by 17.6% compared with the traditional static design, and by 5.5% compared with single-layer PID control. Meanwhile, the average number of online users is 207.

Altogether, the experiment results of the four kinds of designs are concluded and compared in a more obvious way in Figure 7. As can be seen, the average throughput increases by degrees from step (1) to (4). Especially, the difference between the traditional static design and our ultimate double-layer control design shows the remarkable improvement in throughput and demonstrates the effectiveness of software cybernetics and feedback control theory in runtime quality optimization.

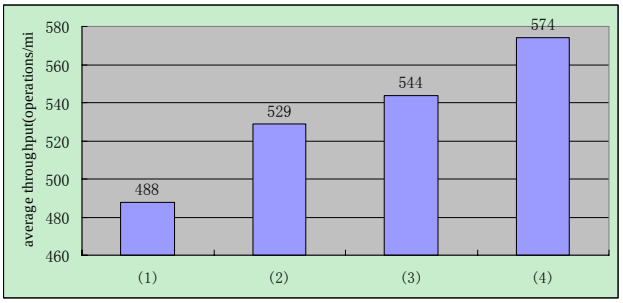

Figure 7: Average Throughput for Four Kinds of Designs

4.3 Discussion

From Figure 5 and Figure 6, we can see the common interesting phenomena: the initial throughput is high, and then it decreases dramatically. The process is durative for about 5 minutes. One possible reason is that the available computer resources are rich when the Web-based system and JMeter begin to run. After a while, due to the untimely release of resources, the throughput decreases and finally be fluctuant.

In this experiment, only single quality attribute (throughput) is considered (Although the average response time is considered in the inner controller, the ultimate goal is the throughput. As long as the throughput could be increased, the actual value of the average response time is not considered.). However, it is the common situation to consider multiple quality attributes which usually conflict with each other. Thus, how to make tradeoff between conflicting quality attributes will be improved in the future.

Besides, the three constants, *Kp*, *Ki*, and *Kd*, of the inner PID controller are manually tuned at design-time. However, the three constants vary from one control system to another and from one platform to another. On the other hand, too much *Ki* will cause instability, and too much *Kd* will cause excessive response and overshoot [7]. Thus, how to do automatic PID tuning to get an appropriate set of *Kp*, *Ki* and *Kd* in different platforms based on the collected information to ensure portability is also included in the future work.

Furthermore, feedback control with prediction will be considered in the future. Its purpose is to predict and prevent the dramatic decrement of throughput such as at time t=28, t=43, t=45, t=73 and t=104 in Figure 6. In fact, the derivative control in the inner PID controller has some kind of prediction, but it's far from enough. The feedback control together with system identification techniques [10] can make it.

5. RELATED WORK

Hong Mei et al. [2] present a software architecture (SA) centric approach for Internetware's self-adaptation, focusing on the three fundamental issues of self-adaptation including the scope, operability and trustworthiness. Internetware is capable of perceiving the dynamic changes of its environment and adjust itself to meet its functionality, performance, dependability, etc. when it is published [1]. In the SA centric approach, all of the self-adaptive actions are performed based on SA: runtime state and behavior of Internetware are represented and changed in the form of runtime soft-ware architecture; the knowledge for self-adaptation is captured, organized and reasoned in the form of SA so that automatic analysis and decision-making are achieved [2]. Compared with the abstractive and general architecture-level self-adaptation, the self-adaptation in our method is concrete and focuses on runtime optimization for quality attributes. Moreover, our work has founded the basis for further research of a general runtime optimization method for multiple conflicting quality attributes.

Jie Yang et al. [11] present an approach to the tradeoff between quality attribute regarding runtime information. Their approach consists of three phases. Firstly, the SA (software architecture) for the target system is analyzed to locate the potential tradeoff points for various quality attributes; secondly, candidate strategies for the tradeoff are designed and recorded in the SA model; finally, the decision of which strategy should be applied is postponed until runtime when there is enough runtime information to determine the most appropriate strategy [11]. The candidate strategies are predefined at design-time, and whether they are suitable for the complex runtime situations can only be evaluated by experts. At run time, the following situation probably happens: none of the predefined candidate strategies is appropriate, but one of them has to be selected. As a result, the adaptive strategy may not improve the desired quality attributes. In our method, such situation won't happen because the strategy is produced at run time.

Kaiyuan Cai et al. [5] propose that the application of software cybernetics is the development of self-adaptive systems. They propose a robust supervisory control (RSC) approach based on software cybernetics to software fault-tolerance. The proposed approach requires only a single version of software and is based on a theoretically rigorous foundation. João W. Cangussu et al. [12] propose the State Model Adaptive Run Time (SMART) framework for dynamic adaptable systems. The proposed framework is based on the mathematics of feedback control theory and system identification techniques. It supports the continuous monitoring of the system (mainly the resources) in a dynamic changing environment, and the applying of feedback control to predict and regulate (select alternative design choices) the behavior of the system.

Furthermore, feedback control theory is also applied successfully to process scheduling in real-time operating system [13] and automatic stress and load testing tools [8] [14]. [8] [13-14] adopt the self-adaptive approach based on single-layer PID controller for the purpose of maintaining the system at the desired system behavior which is fixed at run time. For instance, if the desired average response time in stress testing is 1200ms, the number of clients of a web system will automatically increased until the average response time reaches the desired value of

1200ms. However, for the problem of throughput optimization of Web-based systems, our purpose is to maintain the throughput in the optimized state against the runtime environment. Therefore, the single-layer PID controller can't directly control the throughput. As a result, we propose the double-layer control model to realize optimization control, which adds a simple feedback controller to the input of the PID controller to reconfigure the desired value according to the changing environment.

Xue Liu et al. [15] explore approaches to online optimization of the MaxClients which is a configuration parameter of Apache web server and controls the maximum number of workers. They investigate two optimizers that are based on Newton's Method and 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 of MaxClients. While [15] focuses on the quality attribute of response time, our method focuses on the quality attribute of throughput which is more meaningful to Web-based systems. Besides, [15] and our method deal with optimization problems but not regulation problems as [16-17] do.

Tarek F. Abdelzaher et al. [18] describe overload protection, performance guarantees, and service differentiation in the presence of load unpredictability of a Web server using classical feedback control theory. Compared with our double-layer controller aiming at optimization, performance guarantees in [18] refers to achieving the desired response time and throughput using PI controller which is similar with our inner PID controller.

6. CONCLUSION

In this paper, we propose a runtime quality optimization method for the case of throughput optimization in Web-based systems based on feedback control theory. We design a double-layer control model, combining the PID controller with the simple feedback controller, to realize the optimization of single quality attribute in the uncertain and changing environments. An experimental study with a Web-based system has been conducted to validate our method and the control model. The results have proven the effectiveness of software cybernetics and feedback control theory in runtime quality optimization.

The work conducted in this paper only considers the optimization for single quality attribute. However, stable and optimal overall quality is much more important for the stakeholders. Therefore, overall optimization for multiple quality attributes should be considered in quality-oriented self-adaptive systems. In this kind of quality optimization, dynamic tradeoff among several conflicting is essential. In our future work, we will further explore the problem of runtime quality optimization for multiple attributes with dynamic tradeoff. Moreover, automatic PID tuning and predictive feedback control will be considered as well.

7. ACKNOWLEDGMENTS

This work is supported by National Natural Science Foundation of China under Grant No. 90818009, National High Technology Development 863 Program of China under Grant No. 2007AA01Z125 and 2007AA01Z179, and Shanghai Leading Academic Discipline Project under Grant No. B114.

8. REFERENCES

[1] Hong Mei, Gang Huang, Hanyan Zhao, Wenpin Jiao. A Software Architecture Centric Engineering Approach for Internetware. Science in China Series F: Information Sciences, 2006, 49(6): 702－730.

[2] Hong Mei, Gang Huang, Ling Lan, Junguo Li. A Software Architecture Centric Self-adaptation Approach for Internetware. Science in China Series F : Information Sciences, 2008, 51(6): 722-742.

[3] Betty H.C. Cheng, Rogério de Lemos, Stephen Fickas, etc. SEAMS 2007: Software Engineering for Adaptive and Self-Managing Systems. International Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2007.

[4] Mirko Morandini, Loris Penserini, Anna Perini. Towards Goal-Oriented Development of Self-Adaptive Systems. International Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2008.

[5] Kaiyuan Cai, Xiangyun Wang. Towards a Control-Theoretical Approach to Software Fault-Tolerance. Fourth International Conference on Quality Software, QSIC 2004.

[6] Kaiyuan Cai, João W. Cangussu, Raymond A. DeCarlo, Aditya P. Mathur. An Overview of Software Cybernetics. Eleventh International Workshop on Software Technology and Engineering Practice, STEP 2004.

[7] Gene F. Franklin, J. David Powell, Abbas Emami-Naeini. Feedback Control of Dynamic Systems. Prentice-Hall, 5th Edition, 2006.

[8] Mohamad Bayan, João W. Cangussu. Automatic Feedback, Control-Based, Stress and Load Testing. 23rd Annual ACM Symposium on Applied Computing, SAC 2008.

[9] John A. Shaw. Pid algorithms and tuning methods. http://www.jashaw.com/pid/tutorial/.

[10] L. Ljung. System Identification: Theory for the user. Prentice-Hall, 1987.

[11] Jie Yang, Gang Huang, Wenhui Zhu, Xiaofeng Cui, Hong Mei. Quality Attribute Tradeoff Through Adaptive Architectures at Runtime. Journal of Systems and Software, 2009, 82(2): 319-332.

[12] João W. Cangussu, Kendra Cooper, Changcheng Li. A Control Theory Based Framework for Dynamic Adaptable Systems. ACM Symposium on Applied Computing, SAC 2004.

[13] David C. Steere, Ashvin Goel, Joshua Gruenberg, Dylan McNamee, Calton Pu, Jonathan Walpole. A Feedback-driven Proportion Allocator for Real-Rate Scheduling. In Operating Systems Design and Implementation, pages 145-158, 1999.

[14] Mohamad S. Bayan, João W. Cangussu. Automatic Stress and Load Testing for Embedded Systems. 3rd International Workshop on Software Cybernetics. In the 30th Annual IEEE International Computer Software and Applications Conference, COMPSAC 2006.

[15] Xue Liu, Lui Sha, Yixin Diao, Steven Froehlich, Joseph L. Hellerstein, Sujay Parekh. Online Response Time Optimization of Apache Web Server. International Workshop on Quality of Service, IWQoS 2003, pages 461-478.

[16] Yixin Diao, Neha Gandhi, Joseph L. Hellerstein, Sujay Parekh, Dawn 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.

[17] Lui Sha, Xue Liu, Ying Lu, Tarek Abdelzaher. Queuing Model Based Network Server Performance Control. In Proceedings of the IEEE Real-Time Systems Symposium, 2002.

[18] Tarek F. Abdelzaher, Kang G. Shin, Nina Bhatti. Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach. IEEE Transactions on Parallel and Distributed Systems, 2002, 13(1).