2012 IEEE 36th International Conference on Computer Software and Applications Workshops

Learning-Based Multi-Controller Coordination for Self-Optimization

Richang Lin, Bihuan Chen, Yi Xie, Xin Peng, Wenyun Zhao
School of Computer Science, Fudan University, Shanghai 201203, China
{*11210240018, 11110240008, 11210240033, pengxin, wyzhao*}*@fudan.edu.cn*

***Abstract*—A complex software system may have hundreds of tuning parameters with both itself and its runtime environments. To perform optimally at runtime, a self-adaptive system often needs to employ a series of controllers to tune different parameters. In most cases, these parameters are not independent but interact with each other, thus demanding effective coordination among different controllers. Due to system complexity and the uncertain and changing environments, there are usually no explicit relationships among relevant controlled parameters. In this paper, we propose a learning-based approach to achieve effective coordination among multiple controllers for self-optimization. Based on specific controllers for different controlled parameters, our approach uses an additional coordinator to adaptively switch among different controllers. A learning-based algorithm is adopted to continually evaluate the effectiveness of each controller, providing decision basis for the coordination. The results of our experimental study with a Web-based system show that our approach can significantly improve the effectiveness of self-optimization with coordination among multiple controllers.**

***Keywords*-Self-Adaptive System; Learning; Coordination; Self-Optimization.**

I. Introduction

A self-optimizing system can continually seek opportunities to improve its own performance and efficiency [1]. This self-optimization can usually be achieved by introducing control theory into the runtime management of software systems and employing various controllers for different tuning parameters. Following the general MAPE-K (Monitor, Analyze, Plan, Execute and Knowledge) control loop [1], self-optimization controllers dynamically tune relevant parameters periodically or after detecting deviation of a predefined optimization objective. In this area, feedback controllers ([2], [3], [4], [5], [6]) and fuzzy controllers ([7], [8], [9]) are widely used for different kinds of parameters.

Usually, a complex software system may have hundreds of tuning parameters with both itself and its runtime environments. Therefore, a self-optimizing system often needs to employ a series of controllers to tune different parameters. For example, for a Web-based system, the maximum number of client connections, the length of client session timeout, and the size of database connection pool are typical tuning parameters that influence the system performance. In most cases, these parameters are not independent but interact with each other. For example, with the tuning of some parameters, some processing units of a system may become the performance bottleneck, and after parameters for these units are tuned, other processing units may become the new bottleneck. Due to system complexity and the uncertain and changing environments, there are usually no explicit relationships among relevant controlled parameters. Therefore, effective coordination among different controllers is essential for achieving self-optimization in a highly changing and uncertain environment.

Currently, most researches in this area ([2], [3], [6], [7], [8], [9]) focus on specific controllers for single tuning parameters. Only a few of them ([4], [5]) consider the control of multiple parameters for a system, but rely on the assumptions of linear relationships between relevant input and output variables or stable environments.

In this paper, we propose a learning-based approach to achieve effective coordination among multiple controllers for self-optimization. Based on specific controllers for different controlled parameters, our approach uses an additional coordinator to adaptively switch among different controllers. A learning-based algorithm is adopted to continually evaluate the effectiveness of each controller, providing decision basis for the coordination.

To evaluate the effectiveness of our approach, we conduct an experimental study with an online shopping system deployed on Nginx [10] Web server. In the experiment, three server configuration parameters are tuned for the optimization of system throughput. The study analyzes and compares the throughput of our approach with other control approaches like individual controller only, using all controllers without coordination, randomly selected controller. The results show that our approach is more effective and stable than other approaches for throughput optimization.

The rest of the paper is structured as follows. Section II illustrates the problem of multi-controller coordination with a running example. Section III describes the proposed approach. Section IV presents the experimental study. Section V introduces some related work, and finally Section VI concludes the paper.

II. A Running Example

In this section, an online shopping system deployed on Nginx Web server is used as a running example. It is also used as the subject system of our experimental study (see Section IV).

The system provides basic online shopping functionalities for customers like login service, searching service, products

978-0-7695-4758-9/12 $26.00 © 2012 IEEE
DOI 10.1109/COMPSACW.2012.39

IEEE computer society

viewing service, shopping cart service and payment service. The optimization objective of the system is to achieve a high throughput, which can be measured by the successfully processed orders in a predefined time interval (e.g., one minute). Three key configuration parameters of Nginx are identified as the controlled parameters, i.e., *max_clients*, *keepalive_timeout* and *worker_cpu_affinity*. These parameters have significant but different influence on the throughput as explained below.

max_clients means the maximum number of concurrent client connections the server can handle. A lower value to this parameter will affect throughput negatively due to limiting the server capacity and thus wasting the idle resources. Vice verse, a higher value will also influence the throughput negatively because of losing the server capacity and thus jamming the current requests. Hence, when the server load is heavy, this parameter should be decreased to release the resources to improve the throughput; when the server load is light, it should be increased to make use of the idle resources to ensure a high throughput.

keepalive_timeout means the number of seconds the server will keep the connection alive and wait for another request before closing the connection. A lower timeout will lead to frequent re-connections to a same client, thus increasing the response time of a request and reducing the throughput. On the other hand, a higher timeout will cause a performance problem in a heavily loaded server because more server processes will be kept occupied waiting for subsequent requests from idle clients. Hence, this parameter should be tuned appropriately to achieve a high throughput.

worker_cpu_affinity is set to bind each worker process to specific CPU(s) for the purpose of load-balancing. For example, there are two worker processes and four CPUs, and the *worker_cpu_affinity* is set to 0001 1010, which means the first worker process is bound to CPU0 while the second one is bound to CPU1 and CPU3. When CPU0 is found to be heavily loaded, *worker_cpu_affinity* could be tuned to 0101 1010 to bind the first worker process to CPU0 and CPU2 in order to balance the load to CPU0 and CPU2. Hence, this parameter should be tuned properly to balance the load in order to optimize the throughput.

Usually, the configuration parameters of Nginx are initially set by system administrators. To achieve self-optimization, the system introduces a specific controller for each configuration parameter. It is obvious that these controlled parameters are not independent of each other. For example, a long *keepalive_timeout* may aggravate the resource consumption by idle client connections and limit the number of concurrent client connections that can supported by the system even if *max_clients* has been raised. Therefore, the controllers for these parameters should be effectively coordinated to ensure effective throughput optimization.

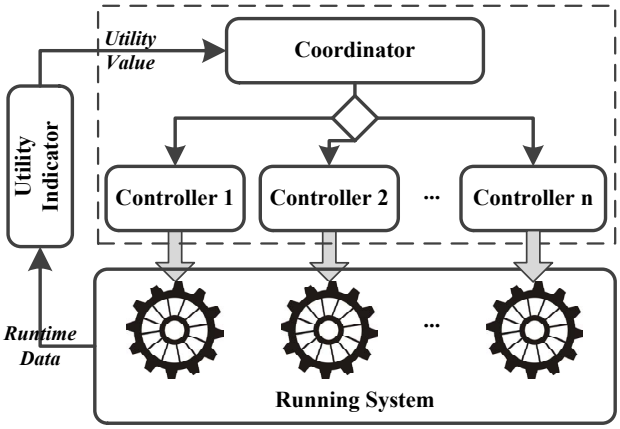

Figure 1. Overview of Our Approach.

III. Our Approach

In this section, we will first present an overview of our approach, and then detail the steps and algorithms involved.

A. Overview

An overview of our approach is shown in Figure 1. For each system or environment parameter relevant to the optimization objective, a controller is designed according to its specific characteristic of control.

Rather than simply tuning all the relevant parameters simultaneously, our approach activates only one controller at the same time. An additional coordinator is introduced to achieve effective coordination among different controllers. The coordinator is itself a feedback controller, which takes runtime utility measurement produced by the utility indicator as feedback and produces a coordination choice as output. Based on the utility feedback and a learning-based algorithm, the coordinator continually monitors and estimates the effectiveness of all the controllers. On the other hand, it periodically evaluates the utility feedback to decide whether the system utility is effectively optimized by the current controller. If not, the coordinator may switch from the current controller to another selected controller based on the effectiveness estimation. After controller switching, the new activated controller takes over the system optimization by tuning corresponding parameter.

It should be emphasized that our coordination mechanism can ensure that after switching a deactivated controller may get the chance of being activated again (see Section III-C).

B. Parameter-Specific Controller Design

The designs of parameter-specific controllers are based on their specific characteristics, and are independent with each other. In practice, there are many widely-used control techniques such as PI (Proportional and Integral) controller (e.g., [3], [4]), PID (Proportional, Integral and Differential)

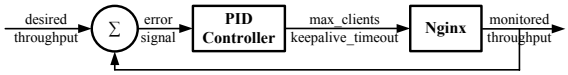

Figure 2. Structure of Feedback Control.

controller (e.g., [2], [5], [6]), fuzzy controller (e.g., [7], [8], [9]), heuristics-based controller (e.g., [7]), etc.

In this paper, we focus on the effective coordination rather than the design of these parameter-specific controllers. Hence, whether a feedback controller or a fuzzy controller should be used to tune a specific parameter is out of the scope of this paper. Here, we illustrate the design with the controllers of the three parameters introduced in Section II.

For *max_clients* and *keepalive_timeout*, we employ a PID controller in consideration of its stability and responsiveness. Figure 2 shows the structure of feedback control. The *monitored throughput* is the actual value of the throughput of online shopping system in the last interval (e.g., one minute), and is fed back to the PID controller. The *desired throughput* is the expected value of the throughput, and is updated at runtime as the average of the last 5 monitored throughputs to reflect the runtime environment because the desired throughput could be different at different environments. Based on the *error signal* which is computed as the percentage of monitored throughput's difference from the desired throughput, *max_clients* and *keepalive_timeout* are tuned such that the monitored throughput will become closer to the desired throughput. Equation 1 is the formulation of PID control, where *e(t)* is the error signal at time *t*, and *u(t)* is the control variable at time *t*, representing the percentage of increment/decrement of *max_clients* and *keepalive_timeout*. *Kp*, *Ki* and *Kd* are user-defined parameters for proportional, integral and differential control, and can be tuned by the method in [11]. For *max_clients*, these control parameters are set to 0.35, 0.1 and 0.25. For *keepalive_timeout*, these control parameters are set to 0.3, 0.1 and 0.2. More details about PID controller can be found in [12].

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

For *worker_cpu_affinity*, we employ a heuristics-based controller due to its discrete characteristic. The following two heuristic rules are used:

- **H1:** If there is a CPU whose usage is over the upper threshold α, and there exist another CPU whose usage is below the lower threshold β, then one worker process will be moved from the heavily-loaded CPU to the lightly-loaded one.
- **H2:** If there is a CPU whose usage is over the upper threshold α, and there is no CPU whose usage is below the lower threshold β, then the worker process which is bound to multiple CPUs will be removed from the heavily-loaded CPU.

In our study, the upper threshold α is set to 70% and the the lower threshold β is set to 50%. For example, if the usage of CPU0 is 80% that is larger than the upper threshold and the usage of CPU1 is 40% that is smaller than the lower threshold, then we can move one of the worker processes from CPU0 to CPU1 to balance the CPU load in order to optimize the throughput.

C. Learning-based Coordination

To achieve optimal coordination among different controllers, the coordinator needs to continually evaluate the effectiveness of every controller on the one hand and make switching decisions on the other hand. Due to the nature of controller switching, an activated controller (especially feedback controller) may work with incomplete knowledge about the current system initially after its activation. To avoid oscillation caused by frequent controller switching and allow enough time for each activated controller to take effect, we employ a mechanism of timing delay for each activated controller.

Algorithm 1 Coordination Procedure

```
1:  procedure COORDINATING(monUtl)
2:      if delay > 0 then
3:          delay ← delay − 1
4:      else
5:          delta ← (monUtl − desUtl)/desUtl
6:          if delta < α then
7:              switch to the best controller
8:          end if
9:      end if
10:     update desUtl
11: end procedure
```

Algorithm 1 gives the procedure of our controller coordination. It takes the monitored utility value (*monUtl*) as input and choose the best controller. If the chosen controller is the same as the current one, then no controller switching will occur; otherwise a new controller will be activated to replace the current one. The procedure first checks whether the current controller is still within its timing delay. If during this delay, the current controller is still in charge of system optimization by tuning the corresponding parameter (Line 2-3). Otherwise, the algorithm checks whether the monitored utility value is below the desired one (*desUtl*) by certain degree. If not, meaning that the current controller is effective to system optimization, the current controller will continue to tune its corresponding parameter. Otherwise, the algorithm evaluates the effectiveness of every controllers (i.e., updates the *effect score* of every controller), makes switching decisions (i.e., chooses the controller with the highest score) and resets the delay to the chosen controller's

timing delay (Line 6-8). Finally, the algorithm updates the desired utility value as the average of the last 5 monitored utility value (Line 10).

In order to evaluate the effectiveness of every controller, we measure an effect score for every controller, which is calculated as the weighted sum of the historical performance of each controller. We define the performance of a controller as the delta between the monitored utility value when the controller is activated and when the controller is deactivated. For example, controller 1 is activated and deactivated at time t_1 and t_2, and the monitored throughput at time t_1 and t_2 is 600 and 500, then the performance of controller 1 at time t_2 is -100. Equation 2 is the formulation of score computation, which is computed when the current controller is deactivated by a new one. For the current controller, its new performance is given a higher weight than the old ones because it reflects the effectiveness more precisely than the old ones. Similarly, other controllers' scores are also less-weighted.

$$score(c) = \begin{cases} 0.5 \times score(c) & : \quad c \neq current \\ 0.5 \times score(c) + perf(c) & : \quad c = current \end{cases} \tag{2}$$

IV. Experimental Study

A. Experimental Settings

In the experiments, we simulated a continuous running of about 100 minutes of the online shopping system deployed on Nginx 1.0.12 in order to demonstrate the effectiveness of our approach. The stress testing tool JMeter 2.6 was used to simulate real-life concurrent accesses from about 600 users to the system. The experiments were conducted on a Dell PowerEdge R210 server with 4 Intel Quad 2.4 GHz processors and 4GB RAM, running Ubuntu 11.10.

Note that the time interval was set to one minute, and the timing delay was set to 10 minutes for all the controllers. The CPU usage was directly read from Ubuntu per interval, and throughput was obtained through analyzing the server log files per interval.

We conducted the experiments with five approaches using the same experimental setting:

- **Static**: this is the traditional approach in which the three parameters of Nginx are statically set by the systems administrators and will not be tuned at runtime.
- **All**: in this approach, all the three parameters are tuned simultaneously using their specific controllers.
- **Random**: this is the approach in which a controller is randomly chosen if the current controller is ineffective.
- **Single**: in this approach, only one individual controller is employed to tune its corresponding parameter. Hence, we conducted three set of experiments for the three controllers.
- **Coordination**: this is the proposed approach in which the best controller is chosen if the current controller is ineffective.

B. Results

Figure 3 (a) shows the average throughput of the five approaches. It can be observed that our coordination approach outperforms all the other approaches in the long run. The coordination approach may be worse than the random and single approach in the earlier stage because of the lack of data. The random and single approach are almost the same and outperform the static and random approach. And the all approach only outperforms the static approach. Numerically, our approach has a gain of 4.23% over the static approach, a gain of 3.72% over the all approach, and a gain of about 2.21% over the random and single approach. This demonstrates the effectiveness of our approach in terms of throughput.

Figure 3 (b) shows the standard deviation of throughput of the five approaches. It shows that our approach has the smallest standard deviation (25.6), which is smaller than the all approach (83.2), static approach (53.3), and random and single approach (31.6) by about 225%, 108%, and 23%. This demonstrates that our approach is more stable than the others.

Based on the previous two observations, we can find that the approach employing adaptive control is better than the approach without any control; the approach employing control coordination is better than the approach without coordination.

C. Evaluation

In order to further evaluate the effectiveness of the proposed approach, we investigate the process of our learning-based coordination in details. Figure 4 records the coordination of our approach. The controller for *max_clients* is identified as the red line, the controller for *worker_cpu_affinity* is identified as the green line, and the controller for *keepalive_timeout* is identified as the purple line. Note that the first 5 minutes (the blue line) is used to learn the desired throughput and thus no controller is employed.

Initially, the controller for *max_clients* was employed to optimize the throughput between time 6 and time 17. At time 18, the throughput was below the desired one, and the controller for *worker_cpu_affinity* was chosen and activated. Although the throughput was still below the desired one at the beginning, the controller for *worker_cpu_affinity* was still employed because the controller was in its timing delay. The delay was reached at time 28, and the controller worked well until time 50. This shows that the timing delay is necessary for the controller to take effect before it is replaced by another one.

At time 50, the controller for *keepalive_timeout* was chosen, and replaced by the controller for *worker_cpu_affinity* when its timing delay was reached (at time 61). At time 80, although the throughput was below the desired one and thus a better controller was required, the controller for *worker_cpu_affinity* was still employed because it was

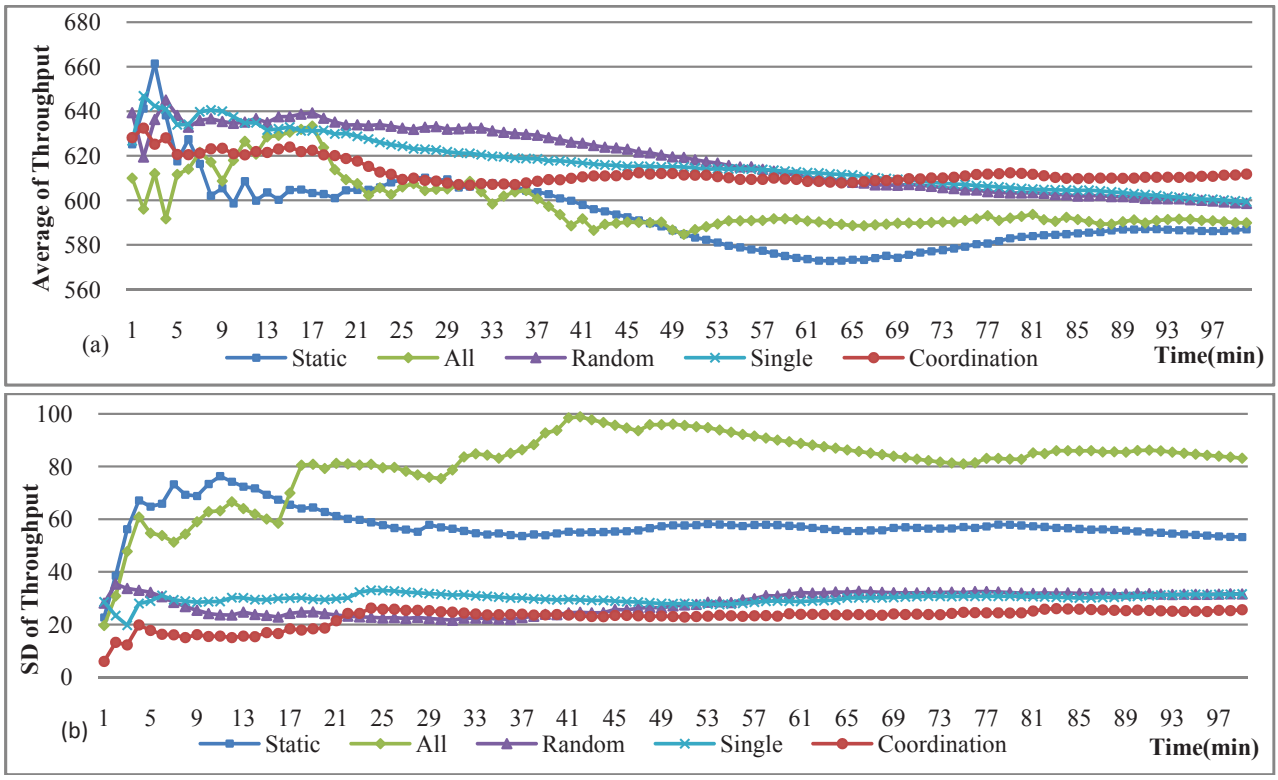

Figure 3. Experimental Results with the Average and Standard Deviation of the Throughput.

already the best one. Until time 83, it was replaced by the controller for *max_clients*.

V. Related Work

Steere et al. [2] proposed a schema of allocating CPU to threads based on proportion and period, in which a PID controller is used to assign automatically both proportion and period. Chen et al. [6] presented a double-layer control model, combining PID controller and heuristics-based controller, to optimize system throughput in a changing environment. The PID controller in [6] is used to dynamically tune the maximum online users. Abdelzaher et al. [3] proposed a utilization control loop, using the PI controller, to achieve overload protection, performance guarantees and service differentiation in the presence of load unpredictability.

Liu et al. [7] explored three approaches to the optimization of response time through tuning *MaxClients* which is a configuration parameter of Apache web server. The first is based on traditional Newton's Method, the second is based on fuzzy control theory and the third is a heuristics-based technique. Gmach et al. [8] proposed AutoGlobe, an architecture for dynamic resource management, in which a fuzzy controller is used to generate actions to exceptional situations such as overload, failures, imminent QoS violations and idle situations. Similar to [7], a heuristics-based controller is also employed and compared to the fuzzy controller. Yang et al. [9] proposed a fuzzy control-based approach for handling uncertainty in software.

Complementary to our approach, this work concentrates on the design of parameter-specific controllers based on feedback control theory, fuzzy logic, or heuristic rules, and thus deals with single-input and single-output (SISO) systems in which there is a single parameter to be tuned and a single objective to be optimized. And our approach pays more attention on the coordination of multiple controllers, and thus deals with multiple-input and multiple-output (MIMO) systems in which there are multiple parameters to be tuned and multiple objectives to be optimized (by synthesizing multiple objectives into one utility measurement).

Diao et al. [4] presented a MIMO (multiple-input and multiple-output) feedback control to enforce policies for interrelated metrics with application to the Apache Web server, in which system identification technique [13] is used to model the relationships between two server parameters (*MaxClients* and *KeekAlive*) and two objectives (CPU and Memory utilization). Bayan et al. [5] presented a way to achieve automatic stress and load testing, in which system identification technique is used to identify the inputs (e.g., total clients, percentage of regular and VIP clients) that affect the resource of interest (e.g., response time), and a PID controller is used to automatically tune the inputs to

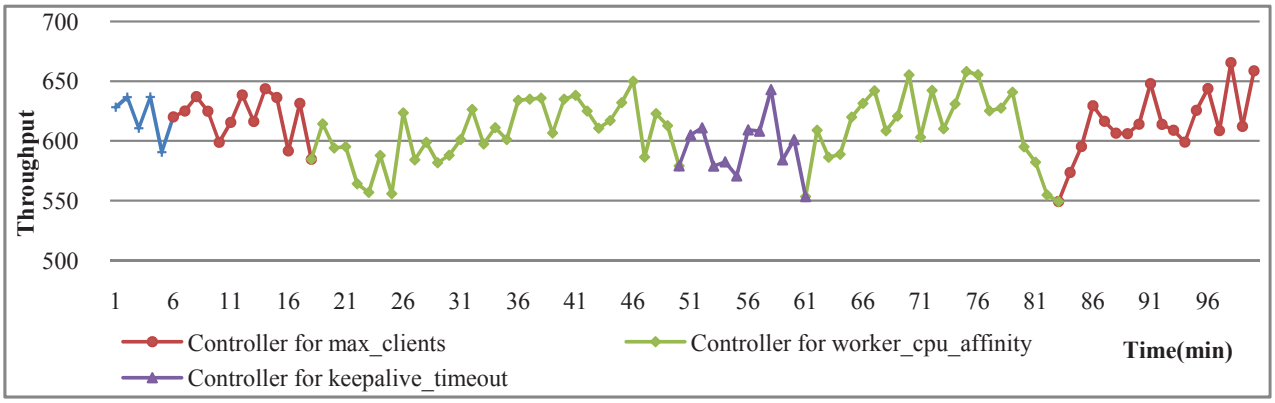

Figure 4. The Process of Our Learn-Based Coordination.

achieve regulation of the resource.

These approaches use system identification technique to model the relationships between the tuning parameters and objectives of a system under the assumption that their relationships are linear which is not always hold in practice. Compared to these approaches, our approach does not rely on such linear relationships.

VI. Conclusions

In this paper, we have proposed a learning-based multi-controller coordination approach for self-optimizing systems with multiple parameters. Based on specific controllers for different controlled parameters, our approach continually evaluates the control effect of each controller, and makes switching decisions among different controllers with a feedback-based coordinator. Our experimental study with a Web-base system has shown that our approach can significantly improve the effectiveness of self-optimization with multiple controllers.

In the future, we hope to perform wider experiments on real-life systems to further evaluate the effectiveness of our approach. Besides, we intend to explore more advanced learning algorithms to improve our coordination mechanism.

Acknowledgment

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. 2012AA011202.

References

[1] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[2] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A feedback-driven proportion allocator for real-rate scheduling," in *Proceedings of the third symposium on Operating systems design and implementation*, 1999, pp. 145–158.

[3] T. F. Abdelzaher, K. G. Shin, and N. Bhatti, "Performance guarantees for web server end-systems: A control-theoretical approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 1, pp. 80–96, 2002.

[4] Y. Diao, N. G, J. L. Hellerstein, S. Parekh, and D. M. Tilbury, "Using mimo feedback control to enforce policies for inter-related metrics with application to the apache web server," in *Proceedings of the Network Operations and Management Symposium*, 2002, pp. 219–234.

[5] M. Bayan and J. a. W. Cangussu, "Automatic feedback, control-based, stress and load testing," in *Proceedings of the ACM symposium on Applied computing*, 2008, pp. 661–666.

[6] B. Chen, X. Peng, and W. Zhao, "Towards runtime optimization of software quality based on feedback control theory," in *Proceedings of the First Asia-Pacific Symposium on Internetware*, 2009, pp. 10:1–10:8.

[7] X. Liu, L. Sha, Y. Diao, S. Froehlich, J. L. Hellerstein, and S. Parekh, "Online response time optimization of apache web server," in *Proceedings of the 11th International Conference on Quality of Service*, 2003, pp. 461–478.

[8] D. Gmach, S. Krompass, A. Scholz, M. Wimmer, and A. Kemper, "Adaptive quality of service management for enterprise services," *ACM Trans. Web*, vol. 2, no. 1, pp. 8:1–8:46, 2008.

[9] Q. Yang, J. Lü, J. Xing, X. Tao, H. Hu, and Y. Zou, "Fuzzy control-based software self-adaptation: A case study in mission critical systems," in *Workshop Proceedings of the 35th Annual IEEE International Computer Software and Applications Conference*, 2011, pp. 13 –18.

[10] Nginx. [Online]. Available: http://www.nginx.org/

[11] J. A. Shaw. Pid algorithm and tuning methods. [Online]. Available: http://www.jashaw.com/pid/tutorial/

[12] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, 5th ed. Prentice Hall, 2006.

[13] J.-N. Juang, *Applied System Identification*, 1st ed. Prentice Hall, Englewood Cliffs, New Jersey, 1993.