2010 International Conference on Recent Trends in Information, Telecommunication and Computing

Simulation Environment for Minimizing Response Time in an Autonomic Computing System using Fuzzy Control

Harish S. Venkatarama
Reader, CS&E Department,
Manipal Institute of Technology,
Manipal – 576 104, India
harish.sv@manipal.edu

Kandasamy Chandra Sekaran
Professor, Dept. of Computer Engg.
National Institute of Technology,
Srinivasnagar-Surathkal, India
kch@nitk.ac.in

***Abstract*—eCommerce is an area where an Autonomic Computing system could be very effectively deployed. eCommerce has created demand for high quality information technology services and businesses seek ways to improve the quality of service in a cost-effective way. Properly adjusting tuning parameters for best values is time-consuming and skills-intensive. This paper describes a simulation environment to implement an approach to automate the tuning of MaxClients parameter of Apache web server using a fuzzy controller and knowledge of the affect of the parameter on quality of service. This is an illustration of the self-optimizing characteristic of an autonomic computing system.**

Keywords-autonomic computing, fuzzy control

I. INTRODUCTION

The advent and evolution of networks and Internet, which has delivered ubiquitous service with extensive scalability and flexibility, continues to make computing environments more complex [1]. Along with this, systems are becoming much more software-intensive, adding to the complexity. There is the complexity of business domains to be analyzed, and the complexity of designing, implementing, maintaining and managing the target system. I/T organizations face severe challenges in managing complexity due to cost, time and relying on human experts.

All these issues have necessitated the investigation of a new paradigm, Autonomic computing [1], to design, develop, deploy and manage systems by taking inspiration from strategies used by biological systems. eCommerce is one area where an Autonomic Computing system could be very effectively deployed. eCommerce has created demand for high quality information technology (IT) services and businesses seek ways to improve the quality of service (QoS) in a cost-effective way. As an example, performance of an Apache web server [24] is heavily influenced by the MaxClients parameter, but the optimum value of the parameter depends on system capacity and workload. Properly adjusting tuning parameters for best values is time-consuming and skills-intensive. This paper describes a simulation environment to implement an approach to automate the tuning of MaxClients parameter of Apache web server using a fuzzy controller.

From [2], we see that the autonomic computing architecture provides a blue print for developing feedback control loops for self-managing systems. This observation suggests that control theory will be of help in the construction of autonomic managers.

II. RELATED WORK

Control theory has been applied to many computing systems, such as networks, operating systems, database management systems, etc. The authors in [3] propose to control web server load via content adaptation. The authors in [5] extend the scheme in [3] to provide performance isolation, service differentiation, excess capability sharing and QoS guarantees. In [4][8] the authors propose a relative differentiated caching services model that achieves differentiation of cache hit rates between different classes. The same objective is achieved in [6], which demonstrates an adaptive control methodology for constructing a QoS-aware proxy cache. The authors in [7] present the design and implementation of an adaptive architecture to provide relative delay guarantees for different service classes on web servers.

Real-time scheduling theory makes response-time guarantees possible, if server utilization is maintained below a pre-computed bound. Feedback control is used in [9] to maintain the utilization around the bound. The authors in [10][11] demonstrate the power of a control theoretic analysis on a controller for doing admission control of a Lotus Notes workgroup server.

MIMO techniques are used in [12][13] to control the CPU and memory utilization in web servers. Queuing theory is used in [14] for computing the service rate necessary to achieve a specified average delay given the currently observed average request arrival rate. Same approach is used to solve the problem of meeting relative delay guarantees in [15].

The authors in [16] present a framework that monitors client perceived service quality in real-time with considerations of both network transfer time and server-side queuing delays and processing time. The authors in [17], present a fuzzy controller to guarantee absolute delays.

The authors in [18] present a Linear-Parameter-Varying approach to the modeling & design of admission control for Internet web servers. The authors in [19] [20] study the performance/power management of a server system.

The authors in [21] propose an approach to automated enforcement of service level agreements (SLAs) by

978-0-7695-3975-1/10 $25.00 © 2010 IEEE
DOI 10.1109/ITC.2010.87

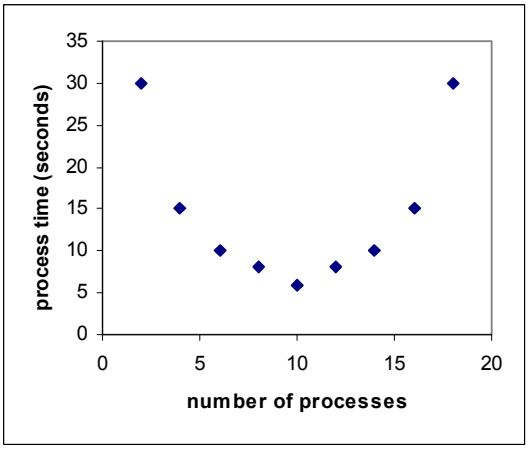

Figure 1. Process time curve

constructing information technology (IT) level feedback loops that achieve business objectives, especially maximizing SLA profits (the difference between revenue and costs). Similarly, the authors in [22] propose a profit-oriented feedback control system that automates the admission control decisions in a way that balances the loss of revenue due to rejected work against the penalties incurred if admitted work has excessive response times. The authors in [23] describe an approach to automate parameter tuning using a fuzzy controller that employs rules incorporating qualitative knowledge of the effect of tuning parameters.

III. SYSTEM BACKGROUND

The system studied here is the Apache web server. In Apache version 2.2 (configured to use Multi-Processing Module prefork), there are a number of worker processes monitored and controlled by a master process [24]. The worker processes are responsible for handling the communications with the web clients. A worker process handles at most one connection at a time, and it continues to handle only that connection until the connection is terminated.

A parameter termed MaxClients limits the size of this worker pool, thereby providing a kind of admission control in which pending requests are kept in the queue. MaxClients should be large enough so that more clients can be served simultaneously, but not so large that resource contention occurs. The optimal value depends on server capacity and the nature of the workload. If MaxClients is too small, there is a long delay due to waits in the queue. If it is too large resources become over utilized which degrades performance as well. The combined effect is that the response time is a concave upward function of MaxClients.

Fig. 1 shows a typical curve to model the response type behavior of a typical Apache server. Here process time denotes the time taken by a process to run to completion. As shown in Fig. 1, if there are only 2 processes running, it takes about 30 seconds for each of them to complete, if there are 4 processes, each of them take 15 seconds, and so on. The process time is minimum (about 6 seconds) when 10 processes are running. This corresponds to the optimum value of MaxClients in an Apache server. When number of processes is increased beyond 10, the process time starts to increase. This corresponds to MaxClients being too large.

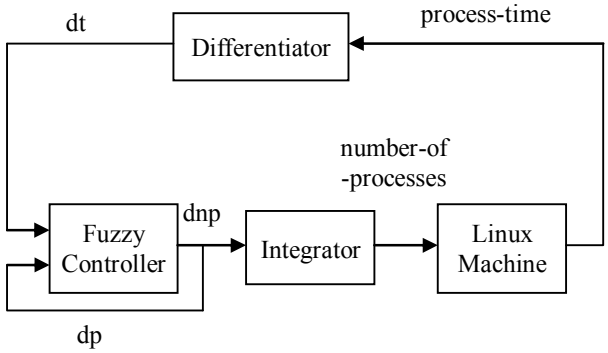

Figure 2. Block diagram of the fuzzy control system

IV. DESIGN OF FUZZY CONTROLLER

The block diagram of the fuzzy control system is shown in Fig. 2. The system being controlled is a linux machine. A number of processes will be running on the machine, the exact number depends on a parameter *number-of-processes*. The time taken by the processes are measured and input to a differentiator whose output is the *change-in-process-time* (dt) between current and previous intervals. The fuzzy controller has two inputs: change-in-process-time (dt) and *change-in-number-of-processes* (dp) between intervals. The controller's output is *next-change-in-number-of-processes* (dnp), whose value is taken as the value of change-in-number-of-processes for the next interval. An integrator converts this value into number-of-processes.

Any fuzzy control system [25] involves three main steps

1) Fuzzification
2) Inference mechanism
3) Defuzzification

The heart of the fuzzy controller involves a set of IF-THEN rules stored in a rule base. The rules are expressed using linguistic variables and linguistic values. Linguistic variables exist in one-to-one correspondence with numeric variables and take on a "degree of truth" for each possible linguistic value. Converting the input numeric variables into linguistic values of linguistic variables is known as fuzzification. Membership functions are used for the conversion. Next the inference mechanism invokes each appropriate rule, generates a result for each, then combines the results of all the rules. Defuzzification involves converting the combined result back into a specific numeric output value.

Fig. 3 shows the triangular membership functions used for the fuzzification of the inputs and defuzzification of the output. The measured numeric values will be multiplied by factors known as the normalized gains, denoted by gdp and gdt. That is why the x-axis shows -1 and 1 for all the membership functions. The output value obtained will be denormalized by dividing by the normalized gain, gnp, to

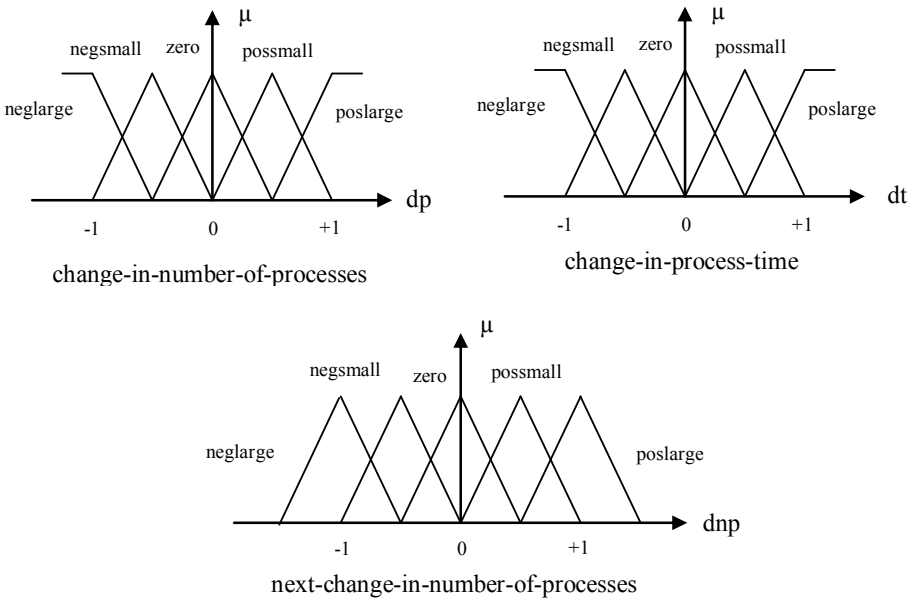

Figure 3. Membership functions and fuzzy inference

obtain the actual output value. Fig. 1 illustrates that process time is a concave upward function of the number of processes. Hence, a gradient descent procedure is used to minimize process times. This is described using fuzzy rules shown in Table I.

TABLE I. FUZZY RULE BASE

Rule	IF			THEN
	change-in-number-of-processes	***AND***	***change-in-process-time***	***next-change-in-number-of-processes***
1	neglarge	AND	neglarge	neglarge
2	negsmall	AND	neglarge	negsmall
3	zero	AND	neglarge	zero
4	possmall	AND	neglarge	possmall
5	poslarge	AND	neglarge	poslarge
6	neglarge	AND	negsmall	neglarge
7	negsmall	AND	negsmall	negsmall
8	zero	AND	negsmall	zero
9	possmall	AND	negsmall	possmall
10	poslarge	AND	negsmall	poslarge
11	neglarge	AND	zero	zero
12	negsmall	AND	zero	zero
13	zero	AND	zero	zero
14	possmall	AND	zero	zero
15	poslarge	AND	zero	zero
16	neglarge	AND	possmall	poslarge
17	negsmall	AND	possmall	possmall
18	zero	AND	possmall	zero
19	possmall	AND	possmall	negsmall
20	poslarge	AND	possmall	neglarge
21	neglarge	AND	poslarge	poslarge
22	negsmall	AND	poslarge	possmall
23	zero	AND	poslarge	zero
24	possmall	AND	poslarge	negsmall
25	poslarge	AND	poslarge	neglarge

Since the value of number of processes that minimizes the process time is not known, these rules are described in terms of changes to number of processes and process times values. As an example, consider rule 5. It means that the number of processes have been increased by a large amount (in the beginning of the current measurement interval) and it is observed that the process time has decreased by a large amount by the end of the interval. This means in the process time curve, the operating point is to the left of the minimum, and proceeding in the correct direction. Hence, it is continued to be moved in the same direction. That is, for the next interval, the number of processes is increased further. Thus, rules 1 through 10 take care of the correct situations where as rules 16 through 25 handle the incorrect situations. In rules 16 through 25 the previous action caused the process time to increase, so the direction has to be "reversed". Later the consequents from all the activated rules are weighted using the centre of gravity method to obtain the (normalized) output value.

V. IMPLEMENTATION DETAILS

Ubuntu running on a 1.8 GHz Pentium IV desktop is used as the platform for running the simulations. The simulation environment consists of

- A load program to create processes
- A differentiator routine, which finds the difference between process times
- A fuzzy controller program, which finds the optimum value of the number of processes and

1. set process-count to 0
2. get value of number-of-processes from the integrator
3. do steps i. to iii. continuously
   i. wait for small duration
   ii. if at the beginning of a measurement interval, get updated value of number-of-processes from the integrator
   iii. if (process-count < number-of-processes) do steps a and b
      a) create new child process
      b) increment process-count

Figure 4. Pseudo code for load program

– An integrator routine, which obtains the value of required number of processes from change in number of processes.

Simulation readings are recorded after every interval, called *measurement* interval.

The output of the integrator is the input to the load program. The load program reads it's input at the beginning of every measurement interval. The parent process in the load program creates and maintains that many child processes. Creation of each child process corresponds to the arrival of a client in Apache server. Hence, before creating a child process, a parent waits for a small duration. The time taken by the Apache server to service a client is simulated by means of a delay routine. This delay routine is invoked within each child process and the quantum of delay depends on number-of-processes so that the relation between the latter and process time is as shown in Fig. 1. Thus if there are two child processes running, it takes about 30 seconds for each of them to complete. If there are four processes, each of them take 15 seconds and so on. From Fig. 1, the optimum value of the number-of-processes is 10 and the process time corresponding to this situation is 6 seconds. Each child process, just before terminating sends the time taken to the differentiator.

The pseudo code for the load program is shown in Fig. 4. The quantity *process-count* contains the actual number of child processes running while number-of-processes contains the value sent by the integrator, which is the required number of child processes. Initially process-count is set to 0 and value for number-of-processes is obtained from the integrator. Next it enters an infinite loop where it's main job is to create the required number of child processes so that the value of process-count always matches the value of number-of-processes. Before creating a new process, it waits for a random duration. Also at the beginning of every measurement interval, the integrator sends an updated value of number-of-processes which is read by the load program.

The pseudo code for the child process is shown in Fig. 5. It first calls a delay, the duration of which depends upon number-of-processes and a parameter called *loadfactor*. Loadfactor takes a value 1, 2 or 3 to simulate low load, medium load and high load. For a fixed number-of-processes, delay is proportional to the loadfactor. It then sends the time taken to execute the delay, to the differentiator. Just before terminating, a signal is sent to the parent process, which will decrement process-count. A loadfactor value of 2 was used to get the curve of Fig. 1.

1. call delay (duration of delay depends upon required-no-of-processes and loadfactor)
2. send time taken to differentiator
3. send signal to parent so that it can decrement process-count

Figure 5. Pseudo code for child process

The fuzzy controller program takes 2 inputs, change-in-number-of-processes and change-in-process-time. The value of next-change-in-number-of-processes obtained in the previous measurement interval is taken as the value of change-in-number-of-processes for the current measurement interval. The value of change-in-process-time is obtained from the differentiator. The controller calculates the adjustment required for the number of processes for the next measurement interval. This output is sent to the integrator and also taken as one of the inputs for the next measurement interval.

The measurement interval should be large enough to reduce the effect of transients and also small enough so that the controller is able to quickly respond to changes. A measurement interval of 3 minutes was used. After waiting 2 minutes for the transients to reduce, readings of process time of processes that exited in the last minute are taken. The median of these values are used to further reduce the effect of the transients. For the normalizing gains, large values increase the speed of the controller, but too large values will cause the system to oscillate. After experimenting with a few values, the values selected were gdp = gdnp = 1/2 and gdt = 1/5. This means a change of 2 in the number of processes or a change of 5 seconds in process time is considered to be large.

VI. RESULTS

Table II shows the values of the input and output variables for loadfactor equal to 1. The minimum delay obtained is 4 seconds. Table III shows the results for loadfactor equal to 2. The minimum delay obtained is 7 seconds. This value is larger because of the higher loadfactor. Finally, table IV shows the results for loadfactor equal to 3. The minimum delay obtained is 10 seconds. Here, it is seen that irrespective of the load, controller always adjusts the number of processes for minimum process time.

VII. CONCLUSIONS

This paper describes a simulation environment to illustrate the self-optimizing characteristic of an autonomic computing system. Here quality of service is optimized by using fuzzy control. The simulation environment is easy to implement and reproduce using minimal computer resources.

TABLE II. FOR LOADFACTOR = 1

Normalized			next change in no. of processes	no. of processes	time taken
change in no. of processes	change in process time	next change in no. of processes			
-	-	-	-	2	15
-	-	-	-	4	9
1	-1.2	1	2	6	6
1	-0.6	1	2	8	4
1	-0.4	0.7	1.4	9	4
0.7	0.0	0.0	0.0	9	4

It also provides a framework to experiment with enhancements and modifications to the basic autonomic computing system used here.

As seen from table I, if there is no change in anyone or both of the inputs, there will be no change in the output. Once the controller reaches a stable state, it will stop responding to further changes in inputs. Future work includes modification to the rules, to make it more sensitive to changes in input.

REFERENCES

[1] M. Salehie and L. Tahvildari, "Autonomic Computing: Emerging trends and open problems," in Proceedings of the Workshop on the Design and Evolution of Autonomic Application Software, 2005.

[2] Y. Diao, J.L. Hellerstein, S. Parekh, R. Griffith, G. E. Kaiser and D. Phung, "A control theory foundation for self-managing computing systems," IEEE Journal on Selected Areas in Communications, Vol. 23, No. 12, December 2005.

[3] T. F. Abdelzaher and N. Bhatti, "Web server Quality of Service management by adaptive content delivery," International Workshop on Quality of Service, June 1999.

[4] Y. Lu, A. Saxena and T. F. Abdelzaher, "Differentiated caching services - A control-theoretical approach," Proceedings of the International Conference on Distributed Computing Systems, April 2001.

[5] T. F. Abdelzaher, K. G. Shin and N. Bhatti, "Performance guarantees for web server end-systems : A control-theoretical approach," IEEE Transactions on Parallel and Distributed Systems, Vol. 13, No. 1, January 2002.

[6] Y. Lu, T. F. Abdelzaher, C. Lu and G. Tao, "An adaptive control framework for QoS guarantees and it's application to differentiated caching services," Proceedings of the International Conference on Quality of Service, May 2002.

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

[8] Y. Lu, T. F. Abdelzaher and A. Saxena, "Design, implementation and evaluation of differentiated caching services," IEEE Transactions on Parallel and Distributed Systems, Vol. 15, No. 5, May 2004.

[9] T. F. Abdelzaher and C. Lu, "Modeling and performance control of internet servers," IEEE Conference on Decision and Control, December 2000.

[10] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram and J. Bigus, "Using control theory to achieve service level objectives in performance management," IFIP/IEEE International Symposium on Integrated Network Management, May 2001.

[11] N. Gandhi, D. M. Tilbury, S. Parekh and J. Hellerstein, "Feedback control of a lotus notes server : Modeling and control design," Proceedings of the American Control Conference, June 2001.

[12] Y. Diao, N. Gandhi, J. L. Hellerstein, S. Parekh and D. M. Tilbury, "Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache web server," Proceedings of the IEEE/IFIP Network Operations and Management, April 2002.

[13] N. Gandhi, D. M. Tilbury, Y. Diao, J. Hellerstein and S. Parekh, "MIMO control of an Apache web server : Modeling and controller design," Proceedings of the American Control Conference, May 2002.

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

[15] Y. Lu, T. Abdelzaher, C. Lu, L. Sha and X. Liu, "Feedback control with queuing-theoretic prediction for relative delay guarantees in web servers," Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium, 2003.

[16] J. Wei and C. Xu, "Feedback control approaches for Quality of Service guarantees in web servers," Fuzzy Information Processing Society, 2005.

[17] Y. Wei, C. Lin, T. Voigt and F. Ren, "Fuzzy control for guaranteeing absolute delays in web servers," Proceedings of the 2nd International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks, August 2005.

[18] W. Qin and Q. Wang, "Feedback performance control for computer systems : an LPV approach," Proceedings of the American Control Conference, June 2005.

[19] W. Qin, Q. Wang, Y. Chen and N. Gautham, "A first-principles based LPV modeling and design for performance management of Internet web servers," Proceedings of the American Control Conference, June 2006.

[20] W. Qin and Q. Wang, "Modeling and control design for performance management of web servers via an LPV approach," IEEE Transactions on Control Systems Technology, Vol. 15, No. 2, March 2007.

[21] Y. Diao, J. L. Hellerstein and S. Parekh, "A business-oriented approach to the design of feedback loops for performance management," Proceedings of the 12th IEEE International Workshop on Distributed Systems : Operations and Management, October 2001.

[22] Y. Diao, J. L. Hellerstein and S. Parekh, "Using fuzzy control to maximize profits in service level management," IBM Systems Journal, Vol. 41, No. 3, 2002.

[23] Y. Diao, J. L. Hellerstein and S. Parekh, "Optimizing Quality of Service using fuzzy control," Proceedings of Distributed Systems Operations and Management, 2002 - Springer

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

[25] Notes on fuzzy control system from Wikipedia. URL= http://en.wikipedia.org/wiki/Fuzzy_control_system

TABLE III. FOR LOADFACTOR = 2

Normalized			next change in no. of processes	no. of processes	time taken
change in no. of processes	change in process time	next change in no. of processes			
-	-	-	-	2	30
-	-	-	-	4	16
1	-2.8	1	2	6	11
1	-1	1	2	8	7
1	-0.8	1	2	10	6
1	-0.2	0.4	0.8	11	7
0.4	0.2	-0.2	-0.4	11	7
-0.2	0.0	0.0	0.0	11	7

TABLE IV. FOR LOADFACTOR = 3

Normalized			next change in no. of processes	no. of processes	time taken
change in no. of processes	change in process time	next change in no. of processes			
-	-	-	-	2	44
-	-	-	-	4	23
1	-4.2	1	2	6	16
1	-1.4	1	2	8	12
1	-0.8	1	2	10	10
1	-0.4	0.7	1.4	11	11
0.7	0.2	-0.4	-0.8	10	10
-0.4	-0.2	-0.2	-0.4	10	10