12

An Application of Fuzzy Controllers: Autonomic Computing Systems

Harish S. V., Reader, Dept. of CS&E,[1] and
Chandra Sekaran K., Professor, Dept. of CS&E,[2]
[1]*Manipal Institute of Technology, Manipal - 576 104,*
[2]*National Institute of Technology – Karnataka, Surathkal, Mangalore – 575 025, India*

1. Introduction

The difficulty of managing today's computing systems goes well beyond the administration of individual software environments. The need to integrate several heterogeneous environments into corporate-wide computing systems, and to extend that beyond company boundaries into the Internet, introduces new levels of complexity. Relying solely on further innovations in programming methods will not get us through the present complexity crisis. The only option remaining is Autonomic Computing – computing systems that can manage themselves given high level objectives from administrators.

An autonomic system has four major characteristics: self-configure, self-heal, self-optimize and self-protect (Salehie & Tahvildari, 2005).

Self-configuring is the capability of adapting automatically and dynamically to environmental changes. This characteristic has two aspects

1. installing, (re-)configuring and integrating large, complex network intensive systems
2. adaptability in architecture or component level to re-configure the system for achieving the desired quality factors.

Self-healing is the capability of discovery, diagnosing and reacting to disruptions. Such a system must be able to recover by detecting a failed component, taking it off-line to be fixed, and replacing the fixed component into the system without any apparent disruption.

Self-optimizing is the capability to efficiently maximize resource allocation and utilization for satisfying requirements of different users. While, in a short term, self-optimizing can address the complexity of managing system performance, in a long run its components will automatically and proactively seek ways to tune their operations and make themselves more cost efficient.

Self-protecting is the capability of reliably establishing trust, and anticipating, detecting and recovering from the effects of attacks with two aspects

1. defending the system against correlated problems arising from malicious attacks or cascading failures that remain uncorrected by self-healing measures
2. anticipating problems based on early reports from sensors and taking steps to avoid or mitigate them.

The autonomic computing architecture (explained later) provides a blue print for developing feedback control loops for self-managing systems. This observation suggests

that control theory might provide guidance as to the structure of and requirements for autonomic managers.

Intelligent control emerged as a viable alternative to conventional model-based control schemes because issues such as uncertainty or unknown variations in plant parameters and structure can be dealt with more effectively. This improves the robustness of the control system. One of the ways of developing an intelligent control system is through Fuzzy control. Fuzzy logic offers the important concept of fuzzy set theory, fuzzy if-then rules and approximate reasoning which deals with imprecision and information granularity.

E-commerce is an area where an Autonomic Computing system could be very effectively deployed. E-commerce 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. Properly adjusting tuning parameters for best values is time-consuming and skills-intensive.

The objectives of this chapter are to minimize response time by maximizing system utilization and also to maximize the profit of an e-commerce system by maximizing system utilization. The outline of the chapter is as follows. Initially the basic concepts of Autonomic Computing, Fuzzy Control and applications of Fuzzy Control to e-commerce system are explained. Then the contributions made in these areas are clearly explained focussing on the methods used.

1.1 Concepts of autonomic computing system

Figure 1.1 depicts the components and key interactions for a single autonomic manager and a single resource. The resource, sometimes called a managed element, is what is being made more self-managing. This could be a single system (or even an application within a system), or it may be a collection of many logically related systems. Sensors provide a way to obtain measurement data from resources, and effectors provide a means to change the behavior of the resource. Autonomic managers read sensor data and manipulate effectors to make resources more self-managing. The autonomic manager contains components for monitoring, analysis, planning, and execution. Common to all of these is knowledge of the computing environment, service level agreements, and other related considerations. The monitoring component filters and correlates sensor data. The analysis component processes these refined data to do forecasting and problem determination, among other activities. Planning constructs workflows that specify a partial order of actions to accomplish a goal specified by the analysis component. The execute component controls the execution of such workflows and provides coordination if there are multiple concurrent workflows. (The term "execute" may be broadened to "enactment" to include manual actions as well.) Scaling is achieved by having a single autonomic manager control multiple resources and by applying the architecture recursively so that lower level managers are treated as resources by higher level managers. In essence, the autonomic computing architecture provides a blue print for developing feedback control loops for self-managing systems. This observation suggests that control theory might provide guidance as to the structure of and requirements for autonomic managers.

1.2 Fuzzy logic concepts

Fuzzy logic refers to a logical system that generalizes classical two valued logic for reasoning under uncertainty. In a broad sense fuzzy logic refers to all the theories that employ fuzzy sets which are classes with boundaries that are not sharply defined.

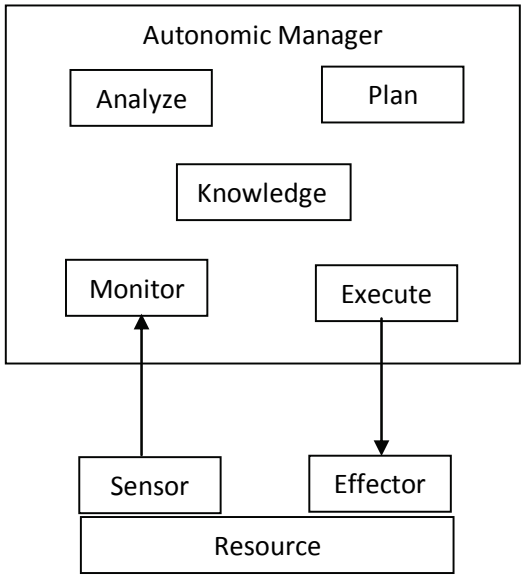


Fig. 1.1. Autonomic Computing architecture

Fuzzy logic is a technology for developing intelligent control. It achieves machine intelligence by offering a way for representing and reasoning about human knowledge that is imprecise by nature. Even though fuzzy logic is not the only technique for developing AI systems, it is unique in its approach for explicit representation of the impreciseness in human knowledge and problem solving techniques. Fuzzy logic offers a practical way for designing nonlinear control systems. It achieves nonlinearity through piece wise linear approximation. The basic building block of a fuzzy logic control system is a set of fuzzy if then rules that approximates a functional mapping.

Fuzzy logic can be used for controlling a process that is too nonlinear or too ill understood to use conventional control designs. It also enables control engineers to easily implement control strategies used by human operators. Briefly fuzzy logic is mainly to deal with complex systems and also for the ease of describing human knowledge.

Fuzzy logic has emerged as a viable alternative to conventional model-based control schemes because issues such as uncertainty or unknown variations in plant parameters and structure can be dealt with more effectively. This improves the robustness of the control system. Fuzzy logic offers the important concept of fuzzy set theory, fuzzy if-then rules and approximate reasoning which deals with imprecision and information granularity.

The three main steps that are part of any fuzzy control system (Yen & Langari, 2005) –

i) Fuzzification ii) Inference mechanism iii) 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. For example, "IF *temperature* IS *high* THEN *speed* IS *high*". This means, increase the speed of the fan if temperature is high. The terms *temperature* and *speed* are linguistic variables, while *high* is a linguistic value. Linguistic variables exist in one-to-one correspondence with numeric variables. Linguistic variables take on linguistic values that correspond to the values of the corresponding numeric variables. For example, *temperature* can take on values high, medium

or low corresponding to the numeric variable for temperature. 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.

1.3 Application of fuzzy control to e-commerce – an overview

E-commerce is one area where an Autonomic Computing system could be very effectively deployed. E-commerce has created demand for high quality information technology (IT) services. For example, a "buy" transaction that takes more than a few seconds may cause the customer to abandon the purchase. As a result, businesses are seeking quality of service (QoS) guarantees from their service providers. (Diao et al., 2002a). These guarantees are expressed as part of service level agreements (SLAs). SLA is a part of a service contract where the level of service is formally defined. It is a contract that exists between customers and their SP, client or between SPs. Many SLAs include specifications (Diao et al., 2001) of:

- revenue that is accrued to the SP for services delivered and
- costs that are incurred by the SP in the form of rebates to customers if previously agreed constraints are violated or the service is unavailable.

An SLA is characterized by a profit model. Consider a profit model described by 3 parameters

1. r, the revenue received for each completed transaction;
2. W, the response time constraint; and
3. c, the cost incurred if a transaction's response time exceeds W (offending transaction)

Thus, Profit = Revenue – Cost, where

Revenue = r * (number of completed transactions)

Cost = c * (number of offending transactions)

Since demand for services is often unpredictable, providers must sometimes make tradeoffs between losing revenue and incurring penalties. Making such choices is skill intensive and time consuming, and the decisions must be made in real time.

An ecommerce system is basically a client server system. The server being the most important part, it is very advantageous if autonomic computing concepts are incorporated into the server. 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. The worker processes are responsible for handling the communications with the web clients, including the work required to generate the responses. A worker process handles at most one connection at a time, and it continues to handle only that connection until the connection is terminated. Thus, the worker is idle between consecutive requests from its connected client.

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 (Diao et al., 2002a).

The setting of MaxClients can also be carried out by looking at the profits (Diao et al., 2002b). Consider an e-commerce system, in which revenues accrue if the admitted requests are processed within the specified deadline and costs are incurred otherwise. If MaxClients is too small, the number of requests that can be processed in a given interval is small. Though the number of violations and hence, costs will be small (mostly zero), profits will be less because of decreased revenue. As MaxClients increases, revenue increases proportionately till the point where the server gets saturated. Thereafter there will be no further increase in revenue but there will be an increase in costs because of increased violations. The combined effect is that profits are concave downwards in the parameter, MaxClients.

2. Minimizing response time

2.1 Simulation using M/M/1 queue and processes

Here the client server architecture is simulated using an M/M/1 queue and processes. Parameter MaxClients is simulated by max-requests. The response time is minimum for an optimum value of max-requests. In the next subsection, the simulation environment used is described. Later, the design and implementation of a fuzzy controller for optimizing the value of max-requests is presented. This ensures that the response time is minimized.

2.1.1 Simulation environment

A workload generator is used to simulate the generation of requests from many clients. The workload generator generates requests such that the time between generations of consecutive requests is exponentially distributed. The processing of these requests by the server is simulated by a program, in which the parent process creates a child process every time a request is received. Each child process sleeps for a time which is exponentially distributed before exiting. Thus, the client server architecture is simulated here as an M/M/1 queue.

2.1.2 Design and Implementation of Fuzzy Controller

The block diagram of the fuzzy control system is shown in **Figure 2.1**. The fuzzy controller has two inputs: change-in-response-time (dr) and change-in-max-requests (dm) between

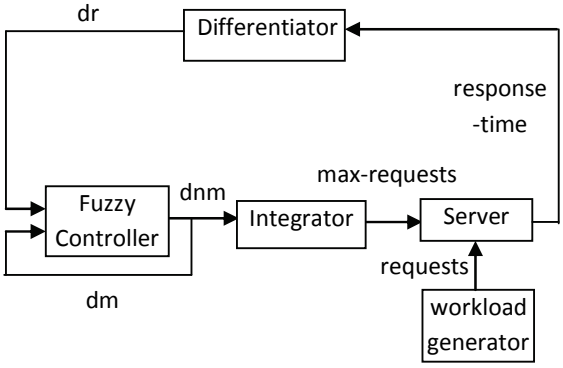


Fig. 2.1. Fuzzy control system – minimizing response time

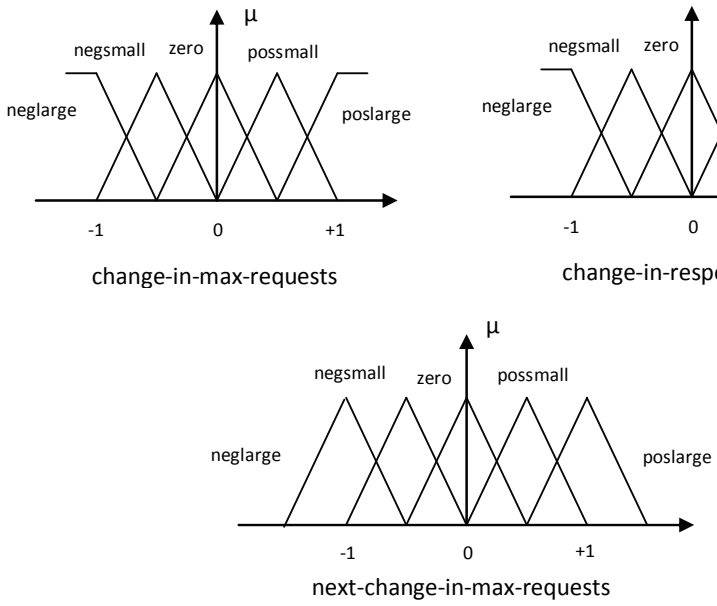


Fig. 2.2. Membership functions – minimizing response time

intervals. The controller's output is next-change-in-max-requests (dnm). An integrator converts this value into max-requests. Next-change-in-max-requests of this interval is taken as change-in-max-requests for the next interval. The value of change-in-response-time is obtained from the differentiator.

The triangular membership functions used for the fuzzification of the inputs and defuzzification of the output are shown in **Figure 2.2**. In each case, the parameter is divided into 5 intervals called neglarge, negsmall, zero, possmall and poslarge. The measured numeric values are multiplied by normalized gains. This is the reason why the x-axis shows -1 and 1 for all the membership functions. Inputs change-in-response-time (dr) and change-in-max-requests (dm) are multiplied by ng-dr and ng-dm respectively. Output is denormalized by multiplying by ng-dnm to obtain next-change-in-max-requests (dnm). Response time is a concave upward function of max-requests. Hence, a gradient descent procedure is used to minimize response times. This is described using fuzzy rules shown in **Table 2.1**.

Since the value of max-requests that minimizes the response-time is not known, these rules are described in terms of changes to max-requests and reponse-times. As an example, consider rule 5. It means that max-requests has been increased by a large amount (in the beginning of the current measurement interval) and it is observed that the response-time has decreased by a large amount by the end of the interval. This means the change to max-requests is in the correct direction. Hence, it is continued to be changed in the same direction. That is, for the next interval, max-requests 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 response-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.

Rule	IF			THEN
	change-in-max-requests	AND	change-in-response-time	next-change-in-max-requests
1	neglarge	AND	neglarge	neglarge
2	negsmall	AND	neglarge	negsmall
3	zero	AND	neglarge	possmall
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	negsmall
12	negsmall	AND	zero	zero
13	zero	AND	zero	zero
14	possmall	AND	zero	zero
15	poslarge	AND	zero	possmall
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	negsmall
24	possmall	AND	poslarge	negsmall
25	poslarge	AND	poslarge	neglarge

Table 2.1. Fuzzy rules – minimizing response time

Normally, when one or both inputs are zero, the output is set to zero. But in the rules 3, 11, 15, and 23, the output is set to a small value. This helps the controller to converge faster. As an example, let us consider rule 23. Without any change in max-requests, there is a large increase in response-time. This means that the incoming requests need larger service times and the number of requests admitted should be decreased. Hence, max-requests is decreased by a small value.

The set-up for the simulation consists of

- a workload generator program to generate requests,
- a server program to service the requests,
- a differentiator routine,
- a fuzzy controller program and
- an integrator routine.

The incoming request from the workload generator is first put into a queue in the server. When the server becomes free, the first request in the queue is dequeued. Workload generator is set to generate requests such that the time between arrivals of consecutive

requests on an average (mean inter-arrival) is 0.2 second. That is 300 requests per minute on an average. Mean service time is set to 60 seconds.
Simulation readings are recorded after every measurement interval. At the end of every measurement interval, response time of that interval is sent to the differentiator whose output is the change-in-response-time (dr) between current and previous intervals. The number of requests accepted by the server, is limited by the parameter max-requests, which is updated by the integrator at the beginning of every measurement interval. The parameter max-requests correspond to MaxClients in an Apache web server.
A measurement interval of 3 minutes is used. To ensure that transients do not affect the readings, readings are taken for the last 1 minute of the 3 minute interval. Response time values of the requests which entered service in the last 1 minute are noted and the average is calculated. 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 ng-dr = ng-dm = 1/10 and ng-dnm = 10. This means a change of 10 in response-time or in max-requests is considered to be large.

2.2 Results

Here to minimize the response time the client server architecture is simulated as an M/M/1 queue and processes. That is, the time between generations of consecutive requests is exponentially distributed. Also processing of each request is simulated by a process which runs for a time which is exponentially distributed. Parameter MaxClients is simulated by max-requests. The response time is minimum for an optimum value of max-requests. The controller minimizes the response time by finding an optimum value for max-requests.
The variation of response time with respect to max-requests is plotted in **Figures 2.3, 2.4 and 2.5.** The mean of the distribution of the inter-arrival times between consecutive requests is kept constant at 0.2 second. This facilitates easy comparison among the three sets of results.
Figure 2.3 shows the result for the case where mean of the service time distribution is 40 seconds. One can see that there is some oscillation. Parameter max-requests increases to 100, before settling to a value around 80. The minimum response time obtained is about 49 seconds. Initially change-in-max-requests is positive, while change-in-response-time is negative. This means the value of max-requests is increasing towards the optimum value. However there is an overshoot and so the controller decreases max-requests towards the optimum.
Figure 2.4 shows the result for the case where mean of the service time distribution is 30 seconds. Once again there is some oscillation, but it is reduced. Parameter max-requests increases to about 104, before settling to a value around 98. The minimum response time obtained is about 30 seconds. The response time is smaller because of the reduced service time. As before, there is an overshoot before the controller decreases max-requests towards the optimum.
Figure 2.5 shows the result for the case where mean of the service time distribution is 20 seconds. There is almost no oscillation. Parameter max-requests settles to a value of about 103. Since the service time is smaller than the previous two cases, the response time obtained of around 20 seconds is also lesser than that obtained previously.
Thus, it is seen that the controller always adjusts the value of max-requests for minimizing response-time.

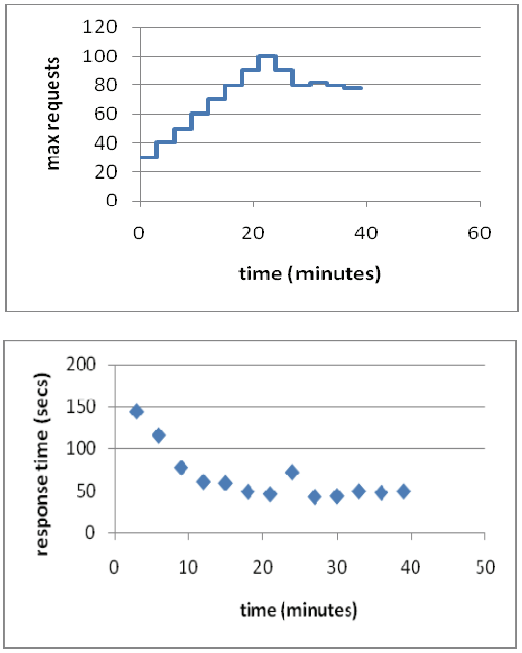


Fig. 2.3. With mean of the service time distribution = 40 secs

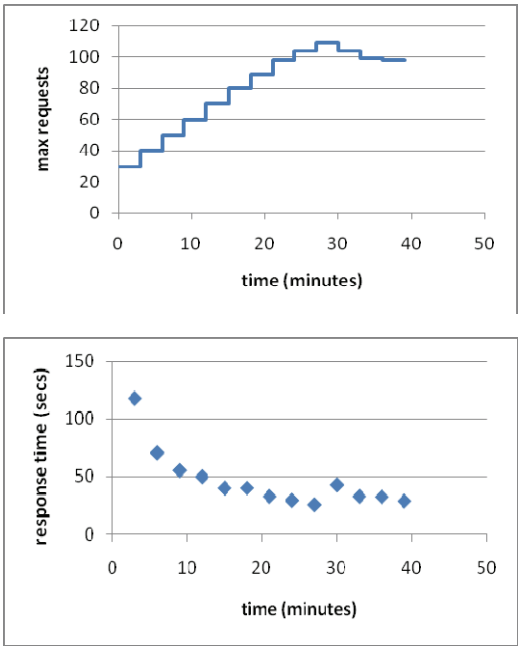


Fig. 2.4. With mean of the service time distribution = 30 secs

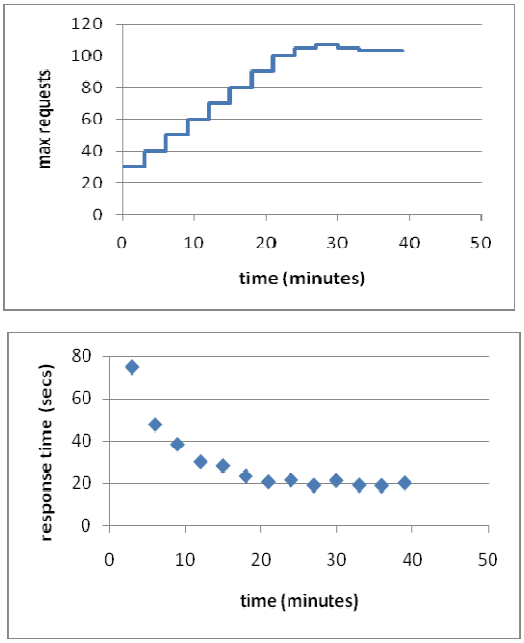


Fig. 2.5. With mean of the service time distribution = 20 secs

3. Maximizing profit of an e-commerce system

3.1 Simulation using M/M/1 queue and processes

Here also the client server architecture is simulated using an M/M/1 queue and processes. As before, parameter MaxClients is simulated by max-requests. The profit is maximum for an optimum value of max-requests. In the next subsection, the simulation environment used is described. This is followed by the design and implementation of a fuzzy controller for optimizing the value of max-requests. This ensures that the profit is maximized.

3.1.1 Simulation environment

A workload generator is used to simulate the generation of requests from many clients. The workload generator generates requests such that the time between generations of consecutive requests is exponentially distributed. The processing of these requests by the server is simulated by a program, in which the parent process creates a child process every time a request is received. Each child process sleeps for a time which is exponentially distributed before exiting. Thus, the client server architecture is simulated here as an M/M/1 queue.

3.1.2 Design and Implementation of Controller

The block diagram of the fuzzy control system is shown in **Figure 3.1**. The client server architecture is simulated here as an M/M/1 queue. The number of requests accepted by the server is limited by the parameter max-requests, which is updated by the integrator at the beginning of every measurement interval. The parameter max-requests corresponds to

MaxClients in an Apache web server. The number of child processes which are able to run to completion are called completed transactions, while those which are unable to run to completion are called violating transactions. These two values are sent to the profit module for calculating profit. This value of profit is input to a differentiator whose output is the change-in-profit (dft) between current and previous intervals. The fuzzy controller has two inputs: change-in-profit (dft) and change-in-max-requests (dm) between intervals. The controller's output is next-change-in-max-requests (dnm), whose value is taken as the change-in-max-requests for the next interval. An integrator converts this value into max-requests.

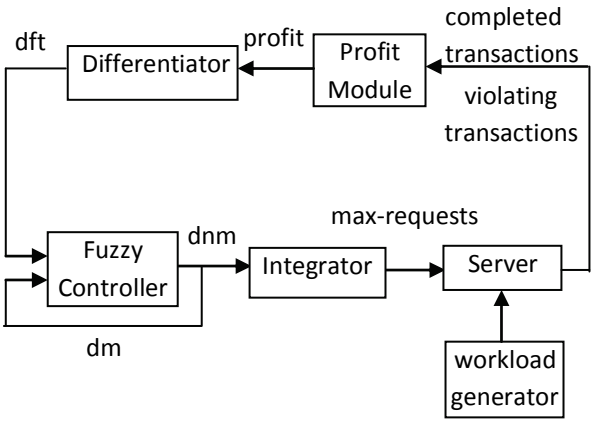


Fig. 3.1. Fuzzy control system – maximizing profit

The triangular membership functions used for the fuzzification of the inputs and defuzzification of the output are shown in **Figure 3.2**. In each case, the parameter is divided into 5 intervals called neglarge, negsmall, zero, possmall and poslarge. The measured numeric values are multiplied by the normalized gains. Value change-in-profit (dft) is multiplied by ng-dft, while change-in-max-requests (dm) is multiplied by ng-dm. The output value next-change-in-max-requests (dnm) is denormalized by multiplying by the normalized gain, ng-dnm, to obtain the actual output value. It is previously noted that profit is a concave downward function of max-requests. Hence, a hill climbing procedure is used to maximize profit. This is described using fuzzy rules shown in **Table 3.1.**

Since the value of max-requests that maximizes the profit is not known, these rules are described in terms of changes to max-requests and profit. As an example, consider rule 25. It means that max-requests has been increased by a large amount (in the beginning of the current measurement interval) and it is observed that the profit has increased by a large amount by the end of the interval. This means the change to max-requests is in the correct direction. Hence, it is continued to be changed in the same direction. That is, for the next interval, max-requests is increased further. Thus, rules 16 through 25 take care of the correct situations where as rules 1 through 10 handle the incorrect situations. In rules 1 through 10 the previous action caused the profit to decrease, 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.

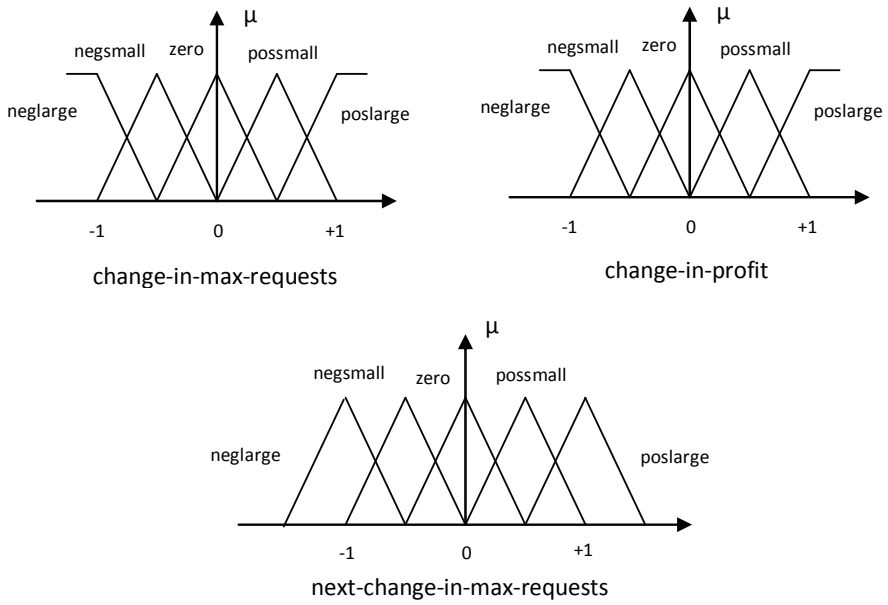


Fig. 3.2. Membership functions – maximizing profit

Normally, when one or both inputs are zero, the output is set to zero. But in the rules 3, 11, 15, and 23, the output is set to a small value. This helps the controller to converge faster. As an example, let us consider rule 23. Without any change in max-requests, there is a large increase in profit. This means that the incoming requests need smaller service times and and more such requests can be admitted. Hence, max-requests is increased by a small value.

The simulation environment consists of

- a workload generator program to generate requests,
- a server program to service the requests,
- a profit module for calculating profit values,
- a differentiator routine,
- a fuzzy controller program and
- an integrator routine.

Simulation readings are recorded after every measurement interval. A measurement interval of 60 seconds was used.

The profit module contains the profit model which is characterized by r, the revenue per completed transaction and c, the cost per violating transaction. Three profit models are defined. P1: r = c, that is, equal weight is assigned to completed and violating transactions; P2: r = k*c, more weight is assigned to completed transactions; P3: r = c/k, more weight is assigned to offending transactions. The constant k should be specified in the SLA. In this work, value for k is taken as 5.

Too large normalizing gains result in the controller oscillating, while too small ones result in a slow performance. For better performance, different values of normalizing gains were selected for different profit models. For profit model P1, ng-dft = ng-dm = 1/5 and ng-dnm = 5. For P2, ng-dft = 1/25, ng-dm = 1/5 and ng-dnm = 5. For P3, ng-dft = 1/10, ng-dm = 1/5 and ng-dnm = 5.

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

Table 3.1. Fuzzy rules – maximizing profit

3.2 Results

Here to maximize the profit, the client server architecture is simulated as an M/M/1 queue and processes. That is, the time between generations of consecutive requests is exponentially distributed. Also processing of each request is simulated by a process which runs for a time which is exponentially distributed. Parameter MaxClients is simulated by max-requests. Parameter max-requests is the upper limit of the number of requests accepted by the server in the given interval. The number of requests which are able to run to completion are called processed-requests. These contribute to the revenue, while those which are not able to run to completion, called, violating-requests contribute to the cost. The contributions of processed-requests and violating-requests towards the profit are decided by the profit model.

Let 'r' be the revenue per processed-requests, 'c' the cost per violating-requests and 'k' be a constant. For profit model P1, r = c, that is, equal weight is assigned to processed-requests and violating-requests. For profit model P2, r = c * k, that is, more weight is assigned to processed-requests. For profit model P3, r = c / k, that is, more weight is assigned to violating-requests. Irrespective of the profit model, profit is maximum for an optimum value

of max-requests. The controller maximizes the profit by finding an optimum value for max-requests.

The variation of profit with respect to max-requests for various profit models are plotted in **Figures 3.3, 3.4 and 3.5**. In this simulation, values selected are r = 1, c = 1 and k = 5.

The results for profit model P1 are shown in **Figure 3.3**. As mentioned before, equal weight is assigned to processed-requests as well as violating-requests. It can be seen that the controller sets max-requests to moderate values. The profit is also moderate.

The results obtained for profit model P3 are shown in **Figure 3.4**. In this case, more weight is assigned to the violating-requests. In an attempt to reduce the number of violating-requests, the controller tries to be more conservative and sets max-requests to comparatively smaller values. The profit is smaller, but it will reduce further if the controller increases the value of max-requests. Thus the controller has maximized the profit, even in the presence of constraints.

The results obtained for profit model P2 are shown in **Figure 3.5**. Since more weight is assigned to processed-requests, the controller is more aggressive and sets max-requests to comparatively larger values. This can be seen when these results are compared with that of **Figure 3.3**. Larger values of max-requests combined with a more favorable profit model leads to a high value of profit.

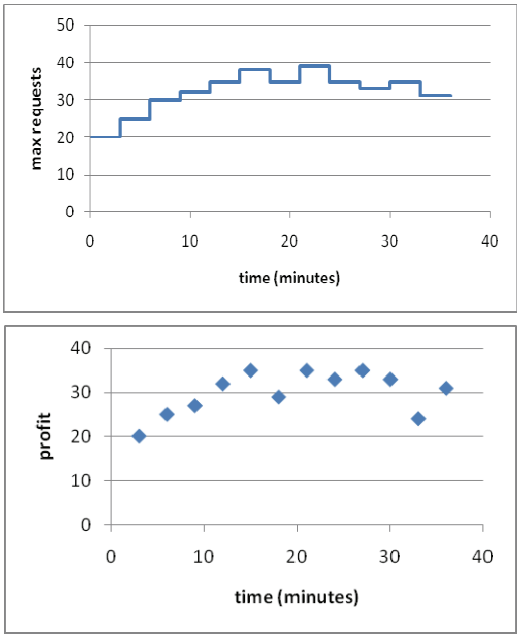


Fig. 3.3. For profit model P1

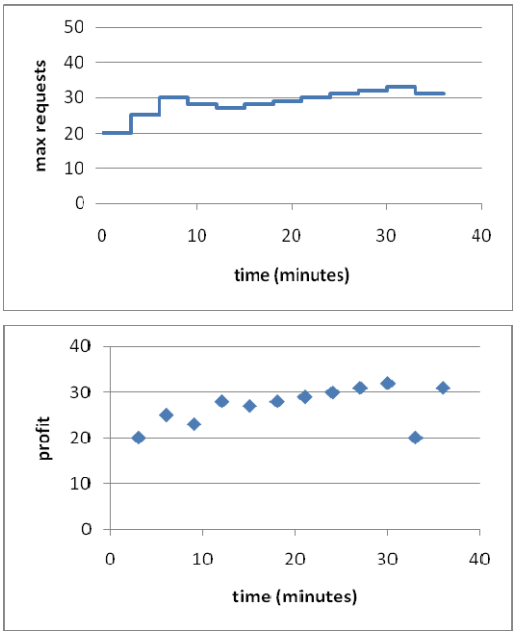


Fig. 3.4. For profit model P3

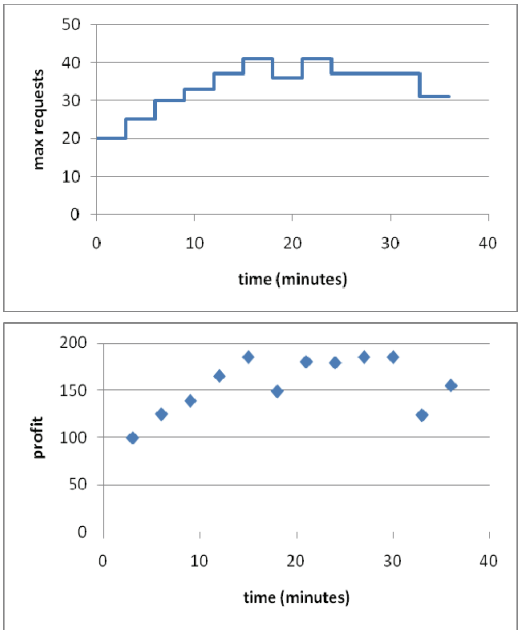


Fig. 3.5. For profit model P2

4. Conclusions

This chapter focuses on two objectives: i) Minimize the response time, and ii) Maximize the profit of an e-commerce system. The client server architecture is simulated using an M/M/1 queue and processes. In case the server is busy, the incoming requests wait in a queue. The average time spent by requests in the queue is the response-time. Here, MaxClients is simulated by max-requests. A fuzzy controller is designed and implemented for minimizing the response-time by optimizing the value of max-requests. The results obtained are also presented. It is seen that the fuzzy controller was successful in minimizing response-time.
To meet the second objective, the client server architecture is again simulated using an M/M/1 queue. Here also, MaxClients is simulated by max-requests. A fuzzy controller is designed and implemented for maximizing the profit by optimizing the value of max-requests. For these simulations, it is seen that the fuzzy controller is able to maximize profit. Thus it can be concluded that fuzzy controllers play a vital role in the area of autonomic computing systems.

5. References

Diao, Y., Hellerstein, J. L. and Parekh, S. (2001). "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.*

Diao, Y., Hellerstein, J. L. and Parekh, S. (2002a). "Optimizing Quality of Service using fuzzy control," *Proceedings of Distributed Systems Operations and Management,* Vol: 2506 Springer, 42-53.

Diao, Y., Hellerstein, J. L. and Parekh, S. (2002b). "Using fuzzy control to maximize profits in service level management," *IBM Systems Journal,* Vol. 41, No. 3, 403-420.

Harish S. V., and Chandra Sekaran, K. (2009). "Maximizing Profit in an Autonomic Computing System: A Fuzzy Control approach," *International Journal of Recent Trends in Engineering*, Vol. 2, No. 1, Association of Computer, Electronics and Electrical Engineers and Academy Publishers, Finland.

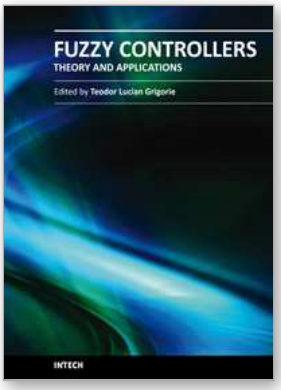
Harish S. V., and Chandra Sekaran, K. (2010). "Minimizing Response Time in an Autonomic Computing System: A Fuzzy Control approach," *International Journal of Computer Science and Systems,* Vol. 1, No. 3, Universal Society of Applied Research (USAR), Prague, Czech Republic.

Kephart, J. O., Chess, D. M. (2003). "The vision of Autonomic Computing," *IEEE Computer Society,* 41-50.

Liu, X., Sha, L., Diao, Y., Froehlich, S., Hellerstein, J. L. and Parekh, S. (2003). "Online Response Time Optimization of Apache Web Server," *Springer-Verlag, Berlin*, 461-478.

Salehie, M. & Tahvildari, L. (2005). Autonomic Computing: Emerging Trends and Open Problems, *Proceedings of the Workshop on the Design and Evolution of Autonomic Application Software*, 2005.

Yen, J. and Langari, R. (2005). *Fuzzy Logic: Intelligence, Control and Information*, Pearson Education, India.



Fuzzy Controllers, Theory and Applications
Edited by Dr. Lucian Grigorie

ISBN 978-953-307-543-3
Hard cover, 368 pages
Publisher InTech
Published online 28, February, 2011
Published in print edition February, 2011

Trying to meet the requirements in the field, present book treats different fuzzy control architectures both in terms of the theoretical design and in terms of comparative validation studies in various applications, numerically simulated or experimentally developed. Through the subject matter and through the inter and multidisciplinary content, this book is addressed mainly to the researchers, doctoral students and students interested in developing new applications of intelligent control, but also to the people who want to become familiar with the control concepts based on fuzzy techniques. Bibliographic resources used to perform the work includes books and articles of present interest in the field, published in prestigious journals and publishing houses, and websites dedicated to various applications of fuzzy control. Its structure and the presented studies include the book in the category of those who make a direct connection between theoretical developments and practical applications, thereby constituting a real support for the specialists in artificial intelligence, modelling and control fields.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821