The Cluster-Based Time-Aware Web System

Krzysztof Zatwarnicki and Anna Zatwarnicka

Department of Electrical, Control and Computer Engineering,
Opole University of Technology, Opole, Poland
{k.zatwarnicki,anna.zatwarnicka}@gmail.com

Abstract. The problem of providing fixed Quality of Web Services (QoWS) is now crucial for further development and application in new areas of internet services. In this paper, we present the MLF (Most Loaded First) adaptive and intelligent cluster-based Web system which provides quality of service on a fixed level. The proposed system keeps the page response time within established boundaries in such a way that with a heavy workload, the page response times, for both small and complex pages, would not exceed the imposed time limit. We show, in experiments conducted with the use of a cluster of real Web servers, that the system is efficient and more effective than other examined systems.

Keywords: quality of web services, guaranteeing web page response time, HTTP request scheduling, request distribution.

1 Introduction

Over the past few years, the Internet has become the most innovative source for information and data. It has evolved from a medium for only privileged users into a medium we could not imagine living without. The rapid development of systems using WWW technology has given rise to the need for research on the effectiveness of the whole system in delivering the necessary content to the user.

Through the many years of the Web development different aspects of QoWS have been noted as the most important. In the early nineties, the most important thing was to properly service HTTP requests, sent by the Web client and deliver the Web page in a response. At the beginning of the 21st century, the Internet began to become more popular and widely used. During that period many popular Web services had problems with simultaneous service of numerous of clients. One of the most prominent ways to evaluate Web systems then was to measure their throughput (number of requests serviced in a time unit). Nowadays, in order to provide appropriate throughput of the system, Web clusters are involved, and the problem of gaining adequate capacity is not a crucial. The challenge now is to service HTTP requests and deliver Web pages to the client within an acceptable amount of time [1].

The problem of servicing HTTP requests, taking the response time in to account was already discussed in our previous work. We have proposed systems minimizing response times in a locally distributed Web cluster system [2], a globally distributed Web cluster system with a broker [3] and a globally distributed

Web cluster system without a broker [4]. In our later work, we dealt with the problem of providing fixed quality of service in Web systems consisting of one Web server [5], a locally distributed cluster of servers [6], and a globally distributed Web cluster with broker [7].

In this paper, we come back to the problem of providing fixed quality of service in a locally distributed cluster of servers and present the MLF system. The system was initially described in [6] together with the results of simulation experiments. This article presents the results of experiments conducted for the MLF system with the use of real Web servers constituting the Web cluster. We analyze these results to determine if the system is equally or more efficient than other systems already known from literature or used in the Internet.

There are many papers on how to provide and guarantee Web service quality. Most of them concern maintaining the quality of the service for individual HTTP requests [8–13]. Very few papers have been dedicated to Web systems guaranteeing to service Web pages within a limited amount of time. In those papers [14, 15], the proposed solution maintains the quality of the Web service only for a limited group of users. Guaranteeing quality of service or providing fixed quality almost always involves rejecting requests of users not belonging to a privileged group. The MLF proposed in this article system not only provides fixed quality of service, keeping the page response time within established boundaries, but also treats all users equally. It should be also noticed that MLF system can keep the quality of service only for the acceptable amount of incoming traffic above which the quality becomes lower than expected.

The paper is divided into four sections. Section 2 presents the MLF system and the methods to schedule and distribute HTTP requests. Section 3 describes the conducted experiments and the results. Finally, Section 4, presents concluding remarks.

2 MLF System

The MLF system consists of: clients sending HTTP requests, MLF Web switch queuing and distributing HTTP requests, Web servers servicing the requests and database servers delivering data to the Web servers (Fig. 1a). The Web switch, the database and the Web server together form the Web cluster. The manner of operation of the Web cluster determines the MLF method. According to the method, clients send HTTP requests to the Web cluster, where the Web switch receives all of the requests, queues them and distributes them among Web servers. The requests are serviced by the Web servers with the assistance of database servers. Responses are delivered back to the Web switch and immediately sent to the clients.

The Web switch controls the operations of the cluster. It especially takes care to deliver the entire Web page to the client within the fixed time $t_{\max}$. To achieve this, the Web switch calculates the term in which each of the incoming requests have to be serviced on the Web server. The requests are queued in the switch according to these calculated terms. When each request leaves the queue, the Web

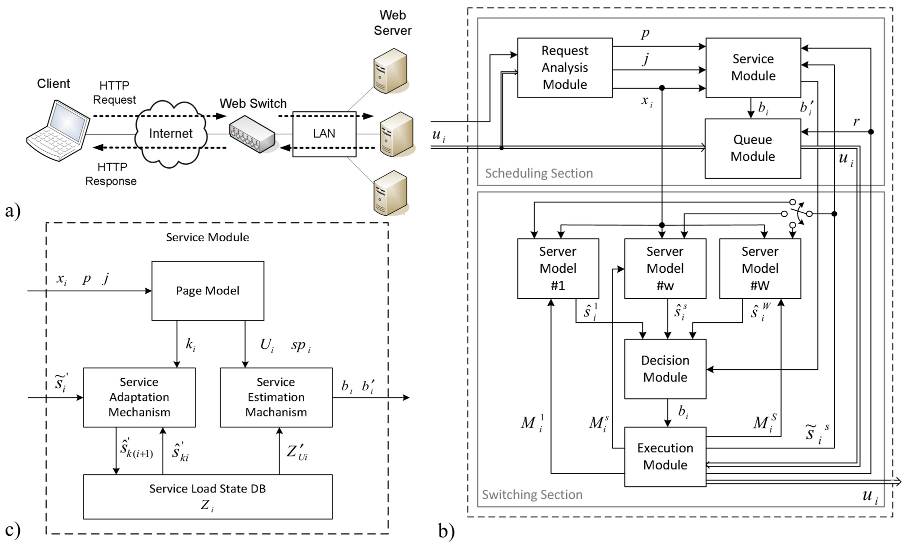

Fig. 1. MLF system: a) Overall view, b) Web switch, c) Service module

switch determines the Web server able to service the request within required time. In the end the request is sent to the chosen Web server.

The Web switch is logically divided into two separate sections (Fig. 1b). The first one is a scheduling section, which is responsible for scheduling requests upon their entrance to the Web cluster. The second section is a switching section, which distributes requests among Web servers.

After receiving the incoming HTTP request u_i, where i is the request index, the Web switch passes it to the Request Analysis Module (RAM). The RAM fetches from the request the following information: address x_i of requested object, identifier j of the client sending the request, and identifier p of the page to which the requested object belongs. The p and j identifiers are included in a *cookie* section of the HTTP request. Upon receiving information from the RAM, a Service Module (SM) computes deadline b_i, indicating the moment the service of the HTTP request has to be started, and term b_i' in which the request service has to be finished. After calculating deadline b_i, the request is passed to the Queue Module (QM) in which the requests are queued according to Earliest Deadline First policy (requests with the shortest deadlines are placed at the beginning of the queue). Not all of the requests are placed in the QM. When the number r of requests serviced concurrently by all of the Web servers in the cluster is smaller than the value $r_{\max}$ then the request u_i is passed directly to the switching section. The value $r_{\max}$ is the lowest number of requests serviced by web servers in the cluster, for which the service reaches the maximal throughput. Requests placed in the QM leave the queue and are passed to the switching section only when $r < r_{\max}$.

The main element of the scheduling section is the SM, those structure is complex (Fig. 1c). The SM is composed of page model (PM), service load state database (SLSDB), service estimation mechanism (SEM) and service adaptation mechanism (SAM). The PM contains information about the structure of Web pages served by the Web cluster, the classes of the objects belonging to the pages and the clients downloading HTTP objects. On the base of x_i, p and j the PM determines time sp_i and vector U_i. Time sp_i is measured from the moment the client requests the first object belonging to the page p to the moment the request u_i arrives. Vector $U_i = [k_i^1, \ldots, k_i^l, \ldots, k_i^L]$ contains information about the classes of objects belonging to page p and not being downloaded yet by the j-th client. The class k_i^l of the object, where $k_i^l \in \{1, \ldots, K\}$, $l = 1, \ldots, L$, and $k_i^1 = k_i$, is determined on the basis of the object's size for static objects (files), whereas every dynamic object (created at the request arrival) has its own individual class. Objects belonging to the same class have similar service times.

The SLSDB stores information $Z_i' = [\hat{s}_{1i}', \ldots, \hat{s}_{ki}', \ldots, \hat{s}_{Ki}']$ about service times for different classes, where $\hat{s}_{ki}'$ is the estimated time to service, by the Web cluster, the request belonging to the k-th class.

Information stored in SLSDB are used by the SEM to compute b_i and b_i'. The SEM takes from SLSDB the information $Z_{Ui}' = [\hat{s}_{k^1 i}', \ldots, \hat{s}_{k^l i}', \ldots, \hat{s}_{k^L i}']$ about service times of objects pointed in the U_i vector. The deadline b_i is calculated in the following way $b_i = \tau_i^{(1)} + \Delta b_i - \hat{s}_{1i}$, where $\tau_i^{(1)}$ is the time of the i-th request's arrival, and $\Delta b_i - \hat{s}_{k^1 i}'(t_{\max} - p_i)/(\lambda \sum_{l=1}^{L} \hat{s}_{k^l i}')$ is the period of time which the request can spend being queued and serviced by the Web server. The λ is a concurrency factor, and it depends on the number of Web objects being downloaded concurrently for given the Web page. According to [16], the value of this factor for average Web pages can be set to $\lambda = 0.267$, or if the WWW pages are not typical, the value can be designated experimentally. The term b' is calculated as follows: $b_i' = \tau_i^{(1)} + \Delta b_i$, and that is the term by which the service of the request has to be finished by the Web server.

The SAM module is responsible for adapting the information Z_i'. After each previously queued request is serviced, the SAM updates the service time for the class the request belonged to. The modification is conducted in the following way $\hat{s}_{k(i+1)}' = \hat{s}_{ki}' + \hat{\eta}(\hat{s}_i - \hat{s}_{ki}')$, where $\hat{s}$ is a measured value of the service time, and $\hat{\eta}$ is an adaptation factor.

After the request u_i leaves the QM it is directed to the switching section of the Web switch. The switching section is composed of: server model modules (SMM), a decision module (DM), and an execution module (EM).

Every SMM is assigned to one Web server working in the cluster. The SMM estimates the service time $\hat{s}_i^w$ of the request for the Web server the module is assigned to, where $w \in \{1, \ldots, W\}$ is the server index, and W is the number of Web servers in the cluster. The service time is computed on the base of information of the requested object class k_i and the load $M_i^w = [e_i^2, f_i^2]$ of the server the SMM is assigned to, where e_i^w is the number of requests being concurrently serviced by the w-th server and f_i^w is the number of dynamic request concurrently serviced.

The DM chooses the Web server to service the request. The decision is made in the following way:

$$z_i = \begin{cases} \min\{w : w \in \{1,\ldots,W\}\} & \text{if } r = r_{\max} \text{ and } \exists_{w\in\{1,\ldots,W\}} \hat{s}_i^w \geq \Delta b_i' \\ w_{\min} : \hat{s}_i^{w_{min}} = \min\{\hat{s}_i^w : w \in \{1,\ldots,W\}\} & \text{in other case} \end{cases} \quad (1)$$

where $\Delta b_i' = \tau_i^{(2)} - b_i'$, and $\tau_i^{(2)}$ is the moment the request u_i leaves the QM. According to the Formula (1) the Web server with the lowest index, which is able to service the request in a time not longer than $\Delta b_i'$, is chosen. If there is no such server, the server offering the shortest service time is chosen. Thanks to this solution, Web servers with the lowest indexes are the most loaded but, still able to service requests, and the servers with higher indexes are unloaded and able to service requests very quickly, when necessary.

When the decision z_i is taken, the EM sends the request u_i to the chosen server. The module also measures the service time $\tilde{s}_i$, and collects the information e_i^s and f_i^s.

The most complex module of the switching section is the SMM. To estimate the service time $\hat{s}_i^w$ of the request, for the given Web server, a neuro-fuzzy model is used (Fig. 2a). Because there are many classes $k_i = 1,\ldots,K$ of requested objects, the same neuro-fuzzy model with different parameters (weights) for each class is used. It can even be said that each of the SMM has K separated neuro-fuzzy networks. All parameters for different classes and networks are stored in the parameter database $Z_i = [Z_{1i},\ldots,Z_{ki},\ldots,Z_{Ki}]$, where $Z_{ki} = [C_{ki}, D_{ki}, S_{ki}]$, $C_{ki} = [c_{1ki},\ldots,c_{lki},\ldots,c_{(L-1)ki}]$ and $D_{ki} = [d_{1ki},\ldots,d_{mki},\ldots,s_{Jki}]$ are parameters of input fuzzy set functions, and $S_{ki} = [s_{1ki},\ldots,s_{jki},\ldots,s_{Jki}]$ are parameters of output fuzzy set functions. Input fuzzy set functions are triangular (Fig. 2b) and are denoted as $\mu_{F_{el}}(s)$, $\mu_{F_{fm}}(f_i)$, $l = 1,\ldots,L$, $m = 1,\ldots,M$, whereas ointentfirstutput fuzzy sets functions $\mu_{Sj}(s)$ are singletons (Fig. 2c). The values L and M were chosen experimentally and set to 5, and $J = L \cdot M$.

The service time is calculated in the following way: $\hat{s}_i = \sum_{j=1}^{J} s_{jki}\mu_{R_j}(e_i, f_i)$, while $\mu_{R_j}(e_i, f_i) = \mu_{F_{el}} \cdot \mu_{F_{fm}}(f_i)$. The values of parameters C_{ki}, D_{ki}, S_{ki} are modified in an adaptation process using the Back Propagation Method each time the service of the request on a given server finishes. The parameters of output fuzzy sets are modified as follow: $s_{jk(i+1)} = s_{jki} + \eta_s \cdot (\tilde{s}_i - \hat{s}_i) \cdot \mu_{R_j}(e_i, f_i)$, whereas parameters of input fuzzy sets are computed in following way
$c_{\phi k(i+1)} = c_{\phi ki} + \eta_c(\tilde{s}_i - \hat{s}_i) \sum_{m=1}^{M}(\mu_{F_{fm}}(f_i) \sum_{l=1}^{L}(s_{((m-1)\cdot L+l)ki} \partial\mu_{F_{el}}(e_i)/\partial c_{\phi ki}))$
and
$d_{\gamma k(i+1)} = d_{\gamma ki} + \eta_d(\tilde{s}_i - \hat{s}_i) \sum_{l=1}^{L}(\mu_{F_{el}}(e_i) \sum_{m=1}^{M}(s_{((l-1)\cdot M+m)ki} \partial\mu_{F_{fm}}(f_i)/\partial d_{\gamma ki}))$
where η_s, η_c, η_d are adaptation ratios, $\phi = 1,\ldots,L-1$, $\gamma = 1,\ldots,M-1$.

3 Testbed and Results of Experiments

In order to evaluate the MLF system simulation experiments were conducted in our previous research [5]. Results of the experiments show that the MLF system can provide higher quality of service than other reference and well known

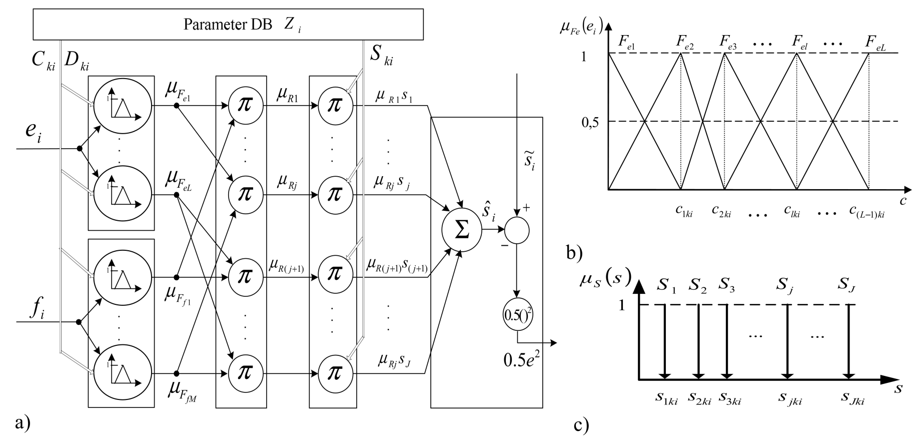

Fig. 2. Server model: a) neuro-fuzzy model, b) input fuzzy sets functions, c) output fuzzy sets functions

distribution methods. The next step to evaluate the system is to conduct experiments with the use of real Web servers and the MLF Web switch. The results of the experiments presented in this article should enable answering the question of whether the proposed system is equally or more effective than other systems already known of from literature or used in real Web switches.

The experiments were conducted using four Web servers, one computer acting as a Web switch and one computer simulating the behavior of Web clients. The first computer with an Intel Core i5-3470 3.3 GHz processor, and Ubuntu 13.04 Desktop operating system, was acting as a generator of the HTTP requests. The second computer, with an Intel Core i7-2670 2.2 GHz processor and Fedora 18 operating system, hosted the Web switch server software. The computers chosen for the Web, and database servers, had the lowest computational power (Intel Celeron 1.7 GHz, Ubuntu 13.04 Server), so it was easy to reach the maximal capacity of the Web servers without overloading other elements of the system. All of the computers were connected through a gigabyte Repotec RP-G3224V network switch.

The Web server hosted five different Web pages. Table 1 presents the structure of the pages. The pages were static and dynamic. All of the pages contained from 10 to 30 embedded objects from 1 to 100 KB in size. Dynamic pages used PHP as the script language generating the content of the page. One of the pages also used SQL requests to the MySQL database, containing 3 related tables of 10 000 rows in size each.

The Web switch server software was written in the C++ language with the use of the *libsoup* [17] and *boost* [18] libraries supporting the supervision of the HTTP requests. The *gcc* compiler was used to create the executable file.

Table 1. Web pages used in the experiments

Name	Type and size of the frame object of the page	Embedded objects number and sizes	MySQL Database
Static 10	Static 1 KB	10, size 1–100 KB, sum 477 KB	—
Static 30	Static 1 KB	30, size 1–100 KB, sum 1.39 MB	—
Dynamic 10	Dynamic, PHP	10, size 1–100 KB, sum 477 MB	—
Dynamic 30	Dynamic, PHP	30, size 1–100 KB, sum 1.39 MB	—
Dynamic MySQL 30	Dynamic, PHP	30, size 1–100 KB, sum 1.39 MB	requests to database containing 3 tables, 10 000 rows each

In order to compare the MLF method with other well known and often applied distribution methods the Web switch had implemented four different scheduling policies:

– MLF,
– LARD (Locality-Aware Request Distribution): one of the best distribution methods taking into account localization of the previously requested object,
– CAP (Content Aware Policy): an algorithm uniformly distributing HTTP requests of different types,
– RR (Round-Robin): an algorithm distributing uniformly all incoming requests.

The software of the HTTP request generator was written in the C++ language with the use of the *LibcURL* library, which enables the creation of HTTP requests and the supervision of the process of sending requests and receiving responses. The request generator generated requests in a similar way to modern Web browsers. It created a given number of virtual clients. Each client, at first, opened a TCP connection to send the requests concerning the frame of the page. After that, it sent in concurrent TCP connections requests concerning objects pointed out in the header of the HTML document. After receiving the frame as a whole, the client opened up to 6 TCP connections to download the embedded objects. Immediately after finishing downloading the Web page as a whole, the client started to download the next page.

The software of the generator was not only generating HTTP request but also collecting information concerning the mean value of request response time and satisfaction. Nowadays the response time is one of the most important measures of the effectiveness of the Web system. Most of the results illustrating the effectiveness of the MLF system in comparison to other systems show the response time in a load function.

The satisfaction allows for determining whether the MLF system is operating as expected and provides the page response time to be no longer then $t_{\max}$ even when the load is high. The satisfaction is often used to evaluate real-time soft systems. It is equal to 1 when the page response time is shorter than $t^s_{\max}$, and decreases to 0 when the time is longer than $t^h_{\max}$ (Fig. 3). In all of the experiments, it has been adopted that $t^s_{\max} = t_{\max} = 2000\,\mathrm{ms}$ and $t^h_{\max} = 2t^s_{\max}$. Experiments were conducted for various increasing numbers of simulated clients [19].

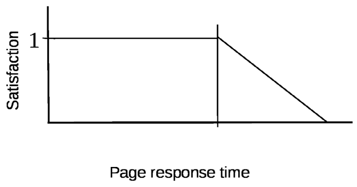

Fig. 3. Satisfaction function

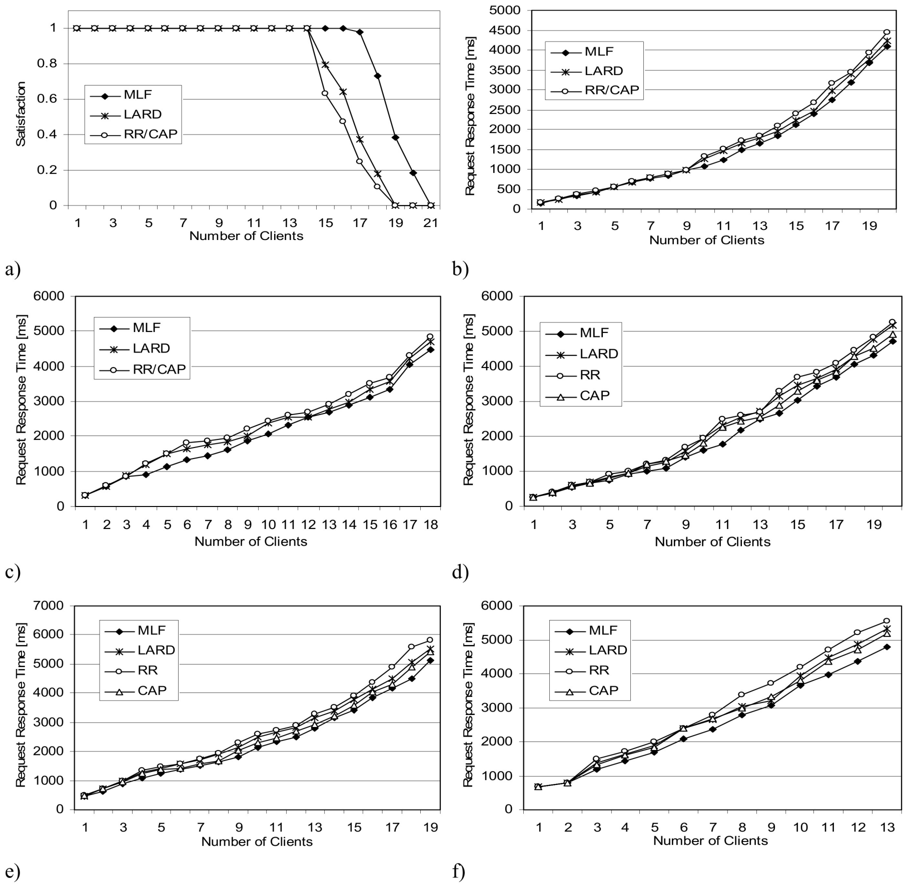

Fig. 4. Results of experiments: a) Satisfaction vs. load for the Web page Static 10; Mean request response time vs. load for Web pages: b) Static 10, c) Static 30, d) Dynamic 10, e) Dynamic 30, f) Dynamic MySQL 30

The Figure 4a presents a diagram of satisfaction vs. number of clients generating HTTP requests. It can be noticed, that when the load is low (the number of clients is between 1 and 14), the satisfaction for all the strategies is very high and equal to 1. This means that almost every Web page was downloaded by the clients with a time no longer than $t_{\max}s$. When the load is higher (the number of clients is greater than 15), the satisfaction for LARD and RR/CAP (for static Web pages the RR and CAP algorithms operate in the same way) algorithms significantly decreased. For the MLF method, the satisfaction begins to decrease when the load is very high and equals to 18. That demonstrates that MLF system is able to provide a page response time no longer than $t_{\max}$ even when the load is high.

Diagrams b to f in Fig. 4 present the mean request response time vs. the load for different Web pages. In every experiment the mean request response time was lowest for the MLF method. When the load is low, the differences between distribution methods are negligible. However, when the load increases, the request response time for the MLF method also increases but is considerably lower than for other distribution methods. It can be also noticed that the difference between MLF method and the others methods are greater for the dynamic Web pages. The presented results confirm results obtained during simulations [6] therefore it can be concluded that under control of the MLF method the cluster-based Web system is more effective than for other examined methods.

4 Summary

In this paper the HTTP request scheduling and distribution cluster-based Web system providing service at a fixed level, was presented. The proposed MLF system can deliver Web pages to the clients in a time not longer than the Web provider demands. Decision algorithms, used in the MLF method, applies adaptive algorithms using neuro-fuzzy models in its construction. Experiments conducted with the use of real Web servers confirm results obtained during simulations and presented in other articles. A high level of quality can be provided even when the system is heavily loaded. The request response times for individual HTTP requests are considerably lower for the MLF method than for other distribution methods. The results of experiment show that the Web system working under control of the MLF method is more effective than for other examined methods.

References

1. McCabe, D.: Network analysis, architecture, and design. Morgan Kaufmann, Boston (2007)
2. Zatwarnicki, K.: Adaptive control of cluster-based web systems using neuro-fuzzy models. International Journal of Applied Mathematics and Computer Science (AMCS) 22(2), 365–377 (2012)

3. Zatwarnicka, A., Zatwarnicki, K.: Adaptive HTTP request distribution in time-varying environment of globally distributed cluster-based Web system. In: König, A., Dengel, A., Hinkelmann, K., Kise, K., Howlett, R.J., Jain, L.C. (eds.) KES 2011, Part I. LNCS, vol. 6881, pp. 141–150. Springer, Heidelberg (2011)
4. Zatwarnicki, K.: Neuro-Fuzzy Models in Global HTTP Request Distribution. In: Pan, J.-S., Chen, S.-M., Nguyen, N.T. (eds.) ICCCI 2010, Part I. LNCS (LNAI), vol. 6421, pp. 1–10. Springer, Heidelberg (2010)
5. Zatwarnicki, K.: Adaptive Scheduling System Guaranteeing Web Page Response Times. In: Nguyen, N.-T., Hoang, K., Jędrzejowicz, P. (eds.) ICCCI 2012, Part II. LNCS (LNAI), vol. 7654, pp. 273–282. Springer, Heidelberg (2012)
6. Zatwarnicki, K.: Operation of Cluster-Based Web System Guaranteeing Web Page Response Time. In: Bădică, C., Nguyen, N.T., Brezovan, M. (eds.) ICCCI 2013. LNCS (LNAI), vol. 8083, pp. 477–486. Springer, Heidelberg (2013)
7. Zatwarnicki, K.: Guaranteeing quality of service in globally distributed Web system with brokers. In: Jędrzejowicz, P., Nguyen, N.T., Hoang, K. (eds.) ICCCI 2011, Part II. LNCS, vol. 6923, pp. 374–384. Springer, Heidelberg (2011)
8. Abdelzaher, T.F., Shin, K.G., Bhatti, N.: Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach. IEEE Trans. Parallel and Distributed Systems 13(1), 80–96 (2002)
9. Blanquer, J.M., Batchelli, A., Schauser, K., Wolski, R.: Quorum: Flexible Quality of Service for Internet Services. In: Proc. Symp. Networked Systems Design and Implementation (2005)
10. Harchol-Balter, M., Schroeder, B., Bansal, N., Agrawal, M.: Size-based scheduling to improve web performance. ACM Trans. Comput. Syst. 21(2), 207–233 (2003)
11. Kamra, A., Misra, V., Nahum, E.M.: A Self Tuning Controller for Managing the Performance of 3-Tiered Websites. In: Proc. Workshop Quality of Service, pp. 47–56 (2004)
12. Schroeder, B., Harchol-Balter, M.: Web servers under overload: How scheduling can help. In: 18th International Teletraffic Congress, Berlin, Germany (2003)
13. Olejnik, R.: An Impact of the Nanoscale Network-on-Chip Topology on the Transmission Delay. In: Kwiecień, A., Gaj, P., Stera, P. (eds.) CN 2011. CCIS, vol. 160, pp. 19–26. Springer, Heidelberg (2011)
14. Suchacka, G.z., Borzemski, L.: Simulation-based performance study of e-commerce Web server system – results for FIFO scheduling. In: Zgrzywa, A., Choroś, K., Siemiński, A. (eds.) Multimedia and Internet Systems: Theory and Practice. AISC, vol. 183, pp. 249–259. Springer, Heidelberg (2013)
15. Wie, J., Xue, C.Z.: QoS: Provisioning of client-perceived end-to-end QoS guarantees in Web servers. IEEE Trans. on Computers 55(12) (2006)
16. Zatwarnicki, K., Zatwarnicka, A.: Estimation of Web Page Download Time. In: Kwiecień, A., Gaj, P., Stera, P. (eds.) CN 2012. CCIS, vol. 291, pp. 144–152. Springer, Heidelberg (2012)
17. Libsoup library description, `http://developer.gnome.org/libsoup/stable/` (access September 10, 2013)
18. Boost C++ libraries, `http://www.boost.org/` (access September 10, 2013)
19. Platek, M.: Guaranteeing quality of service in cluster-base Web system. M.S. thesis, Department of Electroengineering, Automatic Control and Computer Science, Opole University of Technology, Opole, Poland (2013)