Zatwarnicki *Journal of Cloud Computing: Advances, Systems and Applications* (2020) 9:30
https://doi.org/10.1186/s13677-020-00179-6

Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH **Open Access**

Two-level fuzzy-neural load distribution strategy in cloud-based web system

Krzysztof Zatwarnicki

Abstract

Cloud computing Web systems are today the most important part of the Web. Many companies transfer their services to the cloud in order to avoid infrastructure aging and thus preventing less efficient computing. Distribution of the load is a crucial problem in cloud computing systems. Due to the specifics of network traffic, providing an acceptable time of access to the Web content is not trivial. The utilization of the load distribution with adaptive intelligent distribution strategies can deliver the highest quality of service, short service time and reduce the costs. In the article, a new, two-level, intelligent HTTP request distribution strategy is presented. In the process of designing the architecture of the proposed solution, the results of earlier studies and experiments were taken into account. The proposed decision system contains fuzzy-neural models yielding minimal service times in the Web cloud. The article contains a description of the new solution and the test-bed. In the end, the results of the experiments are discussed and conclusions and presented.

Keywords: Cloud computing, Fuzzy-neural modeling, Fuzzy-neural network, HTTP request distribution, Intelligent system, Load balancing, Web cloud system, Web systems simulation

Introduction

Today cloud computing is essential in running most online businesses and plays a very important role in IT. It has opened new opportunities for providing large-scale computing resources [1]. Cloud computing systems enable their clients to access a shared pool of computing resources like servers, storage, applications and services that can be dynamically configured and delivered on-demand [2, 3]. This kind of organization of resources improves the general performance, utilization of resources, energy consumption management and helps to avoid SLA (Service Level Agreement) violation [4]. To achieve those kinds of goals load balancing mechanisms are implemented. Those techniques are known and used for decades in distributed systems like cluster and grid-based systems. However, the proper (effective) distribution of the load is still an open problem in cloud computing that needs new architecture structures and algorithms to meet new customer demands. Little comprehensive research about load balancing in the field of cloud computing has been done.

Mainly in load balancing approaches, we can distinguish very simple techniques in which decisions are taken very fast but the quality of decision is low. More sophisticated algorithms taking in to account the state of the controlled system and intelligent approaches offer high quality of services.

The intelligent approaches can significantly improve the utilization of resources and let to achieve aims required by administrators of the systems.

In the previous work [5–8] a research on intelligent fuzzy-neural distribution methods minimizing service time in cluster-based and cloud-based Web systems has been presented. In the field of cloud systems, an effective HTTP request distribution system using two-layer architecture, in which decisions were made on two independent levels by web switches learning mutual behavior has been proposed [9, 10]. In this article, a novel HTTP request distribution method using one-layer architecture is presented. The decision mechanism, being a key element

Correspondence: k.zatwarnicki@gmail.com
Institute of Computer Science, Opole University of Technology, Opole, Poland

of the method, uses fuzzy-neuro techniques to make decisions on two levels. On the first level, an algorithm is choosing a group of web servers, while in the second level, the next algorithm selects the specific server, out of the chosen group, to service the HTTP request. The presented solution is heuristic, uses intelligent and adaptive decision algorithms and is designed to minimize the service time of HTTP requests in cloud-based Web systems.

The rest of the article is composed as follows. In section two the related work is presented with a description of the previous, selected works constituting the basis for a new solution. Section three contains a description of the new HTTP distribution method. In section four the test-bed and the results of experiments are discussed. Section five summarizes the article.

Related work

Cloud computing, similarly to many other computer technologies uses distributed systems to meet the very high demand for computational power. To distribute tasks and load among nodes in the system load balancing methods are used [11]. Those methods help to improve utilization of the resources, reduce the response times and enable elastic scalability, which is an essential part of the cloud computing [12, 13].

Load balancing mechanisms used in Web clouds can be divided into three main categories [11, 14–17]: static, dynamic and adaptive. In static load balancing assignments are conducted with the use of deterministic or probabilistic algorithms that do not take in to account the current state of the system. Simple strategies have been popular from the beginning of cluster-based Web systems and they are still being improved. An example of such a strategy is Round Robin, assigning incoming HTTP requests to the subsequent servers. A slightly more intelligent version taking into account the Web server load was created by Xu Zongyu and Wang Xingxuan [18]. Stochastic strategies are also still developed and become more and more sophisticated [4, 19, 20].

In a dynamic approach, the decisions are made on the basis of the current state of the system. The most popular in AWS (Amazon Web Services) [21] dynamic load balancing algorithm is Least Load, assigning HTTP requests to the nodes with the least value of chosen load measure.

The adaptive approach is the most complex one and the decisions are not only made on the basis of the state of the system but also the strategy can change when the state of the system is changing [22]. Most of the adaptive strategies are intelligent approaches. Many of the specialists claim that only this kind of strategy can effectively provide an acceptable time of access to the Web content in the conditions of typical Web traffic, characterized by self-similarity and burstiness [23–26]. Among many artificial techniques used in intelligent strategies, we can distinguish Ant Colony Optimization (ACO) algorithm proposed by Kumar Nishant et al. [27], in which generated ants traverse the width and length of the cloud network in the way that they know about the location of both the under-loaded and over-loaded nodes. Another interesting HTTP request distribution method using the natural phenomena-based strategies is called Artificial Bee Colony (ABC) [25]. It uses a decision mechanism imitating the behavior of a bee colony. Particle Swarm Optimization (PSO) strategy is another solution based on heuristic algorithms and is designed to schedule requests to individual components of the cloud [28].

Artificial neural networks have been also used in adaptive load distribution systems [29–31]. A good example of a solution taking into account the energy consumption is presented in [32].

Another group of intelligent adaptive approaches, using fuzzy-neural models, was proposed in the articles of the author and the research group. Those are the solutions enabling global distributions among server rooms located in different geographical locations e.g. GARD [6] and GARDIB [8] and local approaches like FARD [5] and FNRD strategies [7]. All the systems work in this way to minimize the response time for each HTTP request separately. In the latest work [9, 10], a proposal of a two-layer architecture of the Web cloud was made. In such a solution, the devices called Web switches are making independent decisions on two layers. On the first layer, a Web switch is distributing HTTP requests among the availability zones - groups of servers located in the same geographical location (a region) (Fig. 1.a). On the second layer Web, switches are distributing requests inside availability zones (simply zones). The obtained results showed a much better performance for the intelligent fuzzy-neural web switches than for non-intelligent strategies. We compared also results of the work of our intelligent FNRD Web switch in one-layer architecture (Fig. 1.b) and two-layer architecture. In one-layer architecture one Web switch was distributing requests among all servers in the zones.

Results for two-layer architecture were surprisingly much better than for one layer architecture.

Taking into account the results of the latest research, a new distribution method and design of a new Web switch for one-layer architecture are proposed in this article. In the new method, all of the Web servers are divided into groups, a separate mechanism in the Web switch is choosing a group of servers and another mechanism chooses a server to service the request in the group. The new strategy is called Two-Level Fuzzy-Neural Request Distribution, or simply TLFNRD.

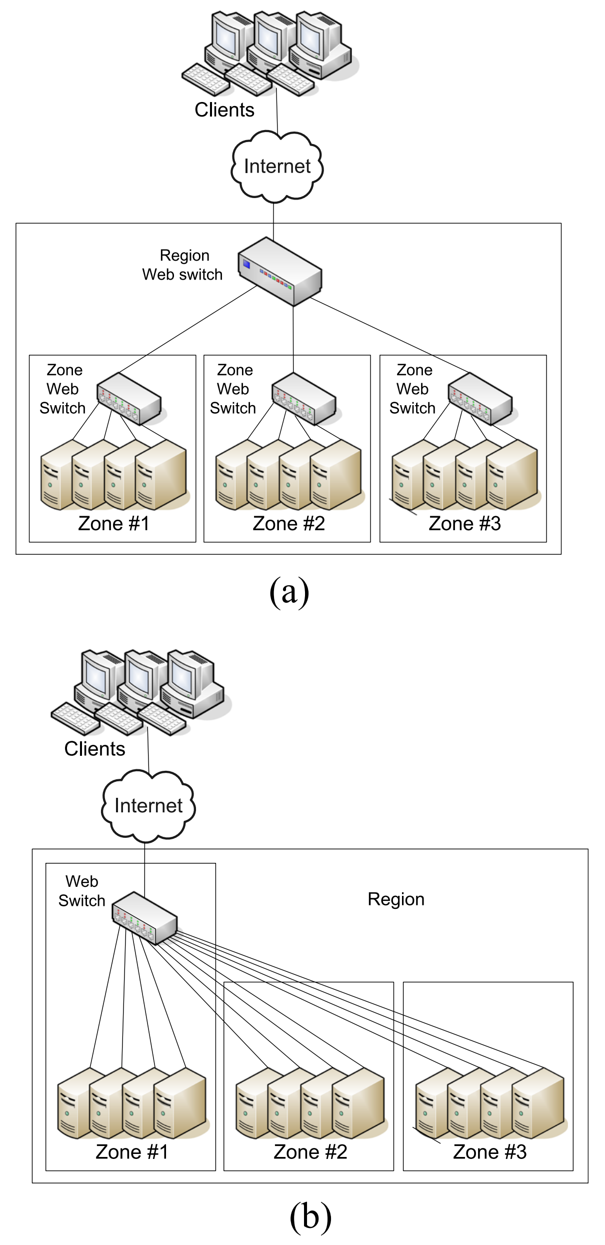

Fig. 1 Web cloud systems: **a** two-layer architecture, **b** one layer architecture

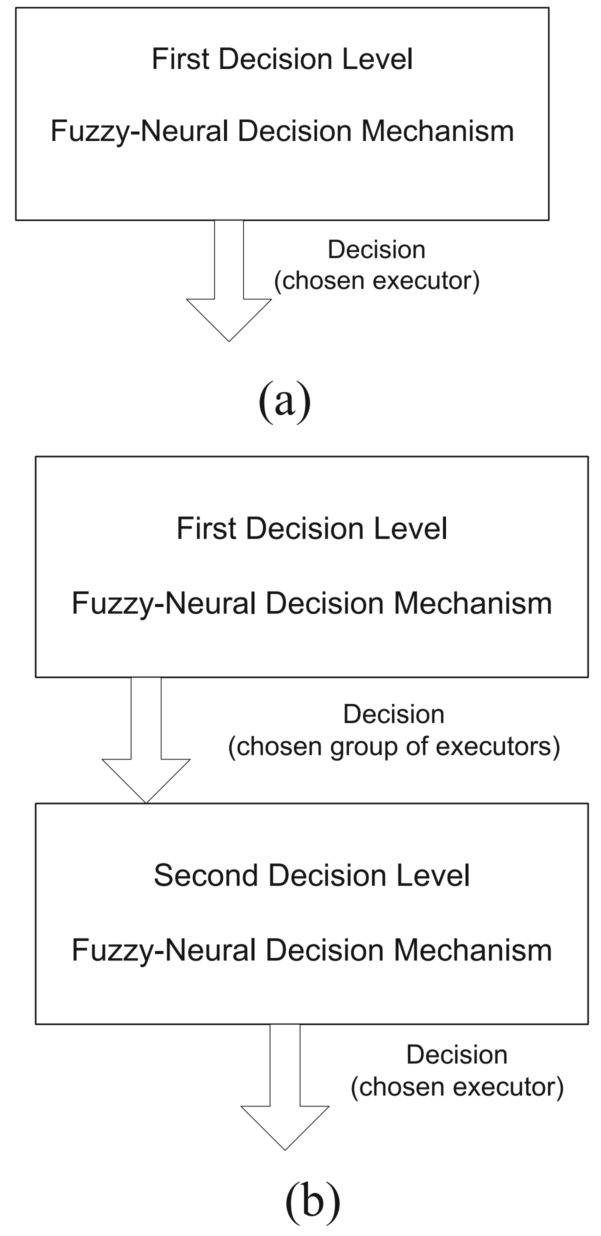

Fig. 2 Decision mechanisms: **a** one-level decision mechanism, **b** two-level decision mechanism

The main difference between methods and algorithms presented in the works [5–10] and the TLFNRD strategy is that the previously proposed Web switches and brokers make the distribution decision only on one level (Fig. 2a). Those switches and brokers are estimating service times using the fuzzy-neural mechanism for all executors servicing HTTP requests (e.g. Web servers or availability zones). In the TLFNRD strategy, Web switch makes decisions on two levels (Fig. 2b). On each level, separate fuzzy-neural mechanism estimates service times for logical groups of servers (on the first level) and servers itself (on the second level). In the proposed solution, neuro-fuzzy mechanisms on different levels can, in some way, cooperate and learn mutual behaviors.

This article should help to answer if a two-level fuzzy-neural distribution strategy TLFNRD is better then a simple one-level FNRD approach.

The detailed description of the new TLFNRD solution is presented in the next section.

Two-level fuzzy-neural web switch

The main aim of the proposed Web switch is to distribute HTTP requests to minimize service time for each request. Service times are measured from the moment the Web switch sends a request to the chosen server to the moment the Web switch receives the response. The Web switch is processing requests in the order in which they are received. Requests are not queued or scheduled. In addition, all Web servers working in the cloud can service all of the HTTP requests.

It should be noticed here that the design of the presented switch does not include all of the features of Web switches used in practical applications. Due to the clarity of the presentation, it lacks solutions connected with security, resistance to cyber-attacks, and failover system used when the Web server or the Web switch fails itself. However, the presented method of request distribution fully enables the implementation of the indicated mechanisms, and in particular, the design of the request distribution algorithm supports the failover mechanism.

As mentioned above, the Web switch makes its decisions on two logical levels. On the first level, a group of servers is chosen. On the second level, a single server is selected from the group.

The overall construction of the Web switch is presented in Fig. 3. The Web switch consists of the following modules: the request analysis, group, server group, redirection, and measurement modules.

The HTTP request is redirected in the following way in the presented Web switch. At first, the request is classified. Request belonging to the same class have similar service times. The group module chooses a group of servers that can service the HTTP request in the shortest time. The server group module, for which group was chosen, selects the Web server for which the estimated service time is the shortest. The redirection module sends the HTTP request to the chosen server. After servicing the request, the HTTP response is sent back by the Web server to the Web switch and the Web switch passes it to the client (due to clarity reasons this process is not presented in Fig. 3). The measurement module measures the real service time and sends it to the group module and to the server group module, which was previously chosen. Both of the modules update information about processing time in the cloud.

In the following subsections, the decision-making process is described in detail.

Classification of http requests

The incoming HTTP request r_i (where i is the index of request and $i = 1, \ldots, I$) is at the beginning assigned to a class k_i ($k_i \in \{1, \ldots, K\}$) in the request analysis module. The classification is made in this way to make requests having a similar response time to belong to the same class. Serviced HTTP requests should be constructed properly according to the HTTP protocol. In the system, there are distinguished two types of HTTP requests: static and dynamic. Static requests have responses delivered from the files (like HTML files, jpg, png, and other picture files) placed on the Web server, and are classified by their sizes. Dynamic requests have content generated by the Web serve after the request arrival (by executing scripts on the server like PHP, Python, Java or .Net Core, e.t.c.), and are classified separately by its address. The effectiveness of this method of classification has been confirmed both in simulation experiments as well as in research on a real cluster system [7, 33].

The FIRST decision level

The group module makes the decision on the first level. It chooses a group of servers that will deliver the

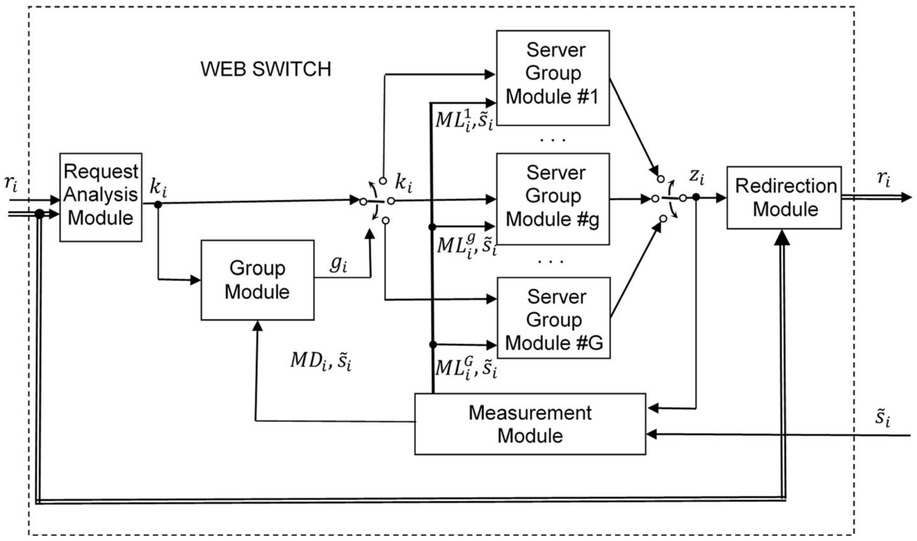

Fig. 3 Design of the Web switch

response. Dividing the servers into groups is not physical but only logical.

The group module chooses this group of servers g_i (where $g_i \in \{1, \ldots, G\}$), for which the estimated service time is the shortest. The decision is made by taking into account a load of groups of servers $MD_i = [MG_i^1, \ldots, MG_i^g, \ldots, MG_i^G]$, where $MG_i^g = [e_i^g, f_i^g]$, e_i^g and f_i^g are measures of the load on a server group g_i at the moment of arrival of i th request. Those measures were chosen according to the experiments in [5, 7, 33]. The e_i^g is the overall number of requests being currently serviced by the group of servers, and f_i^g is the number of dynamic requests serviced. The group module also adapts to the changing environment by taking in to account a measured service time $\tilde{s}_i$ after the request service.

The information MD_i and $\tilde{s}_i$ are delivered by the measurement module.

The construction of the group module is complex and is similar to the construction of the server group module. A detailed description of both of the modules and the overall working functionality is presented in section 3.4.

The second decision level

There are as many server group modules as the number G of groups of servers. Those modules constitute the second decision level. The server group module chooses the server z_i to service the i th request. It is assumed that all servers are general-purpose Web servers using the same hardware and being able to service each HTTP request within the Web service.

In the Web switch only the g_i th server group module, chosen by the group module for the given request, is making the decision. The construction and the action of the server group module are exactly the same, as the group module, and is described in section 3.4. The inputs for the module are a load of servers in the group $ML_i^g = [MS_i^1, \ldots, MS_i^z, \ldots, MS_i^Z]$, where $MS_i^z = [e_i^z, f_i^z]$, Z is the number of servers in the group, e_i^z and f_i^z are measures of the load on a server z and the meaning of them is the same as for the group module. Also, the module is adapting to the environment by taking into account the measured service time $\tilde{s}_i$, but only in the case when, one of the Web servers of the group was servicing the i th request.

The redirection module, with the use of TCP/IP protocol stack in the operating system, redirects the r_i *th* request to the chosen Web server z_i. The Web switch also receives the response from the server and sends it to that client, which sent the request (Fig. 1b). This process is not included in Fig. 3 because it is not important for our considerations.

The measurement module collects information MD_i, $ML_i^1, \ldots, ML_i^g, \ldots, ML_i^G$ and $\tilde{s}_i$ necessary for other modules to make decisions. Because the Web switch sent the HTTP requests to Web servers and receives HTTP responses it can measure the service time $\tilde{s}_i$ and the number of static and dynamic HTTP requests being serviced by individual Web servers. Importantly, this module acquires information available on the Web switch and does not need to use other data sources.

The process of choosing an executor to service the request

The group module and the server group module choose the group of servers or a single server to service the HTTP request. Both of the decision elements act in a similar way and do not require information about the internal structure of the part of the system for which the decision is taken. For this reason, the group module and the server group are called in this section selection module. Consequently, a group of servers or a single server are called an executor.

Figure 4 presents the overall structure of the selection module. The selection module contains a decision module and as many executors models as the number of executors (a group of servers or Web servers) belonging to the group.

Each of the executor models estimates service time for executor it corresponds to and for the i th request belonging to the class k_i. The estimation is done every time a new HTTP request arrives before making the distribution decision on the basis of the information of the load of the executor $M_i^w = [e_i^w, f_i^w]$ (which is equivalent to the load MG_i^g and MS_i^z from the previous sections), where w is an index of the executor, and $w = 1, \ldots, W$. The executor model updates its information about the executor, by taking into account the measured service time $\tilde{s}_i$, only if the executor model corresponds to the element of the system that serviced the i th request.

The decision module chooses the executor d_i for which the estimated service time $\hat{s}_i^w$ is the shortest, according to

$$d_i = \min_w \left\{ \hat{s}_i^w : w \in \{1, 2, \ldots, W\} \right\}. \quad (1)$$

The key element of the system is the executor module which estimates the service time for the given executor and can adapt to the changing environment. It owes its capabilities to the use of a fuzzy-neural mechanism. This kind of construction of the executor was introduced and described in detail in [7].

The fuzzy structure of the system is based on the Mamdani [34] model, while the neural approach permits to change the parameters of input and output fuzzy sets.

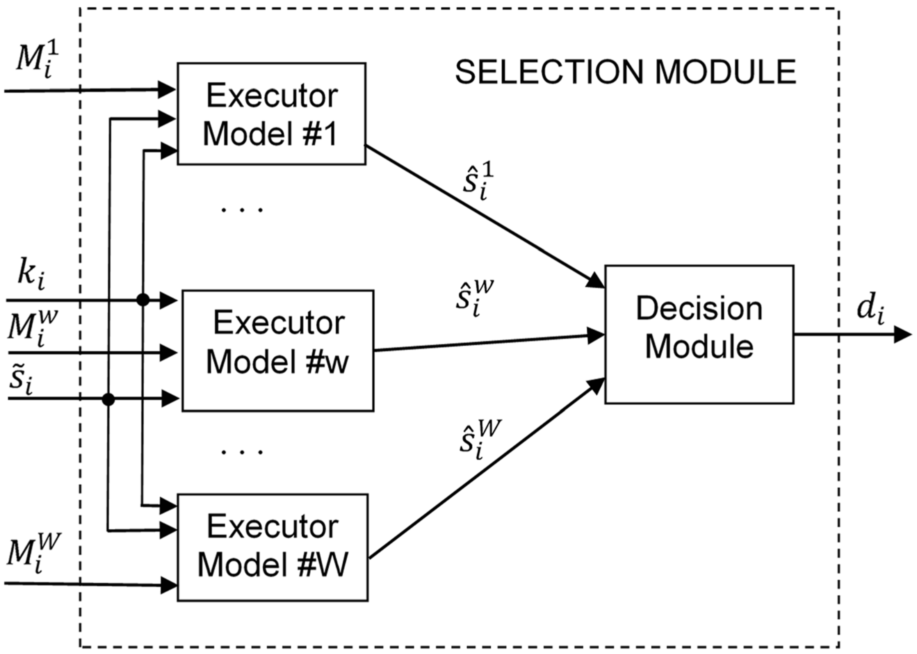

Fig. 4 Design of the selection module

The overall structure of the executor module fuzzy-neural network is presented in Fig. 5.a. The superscripts indicating the executor's number (w) have been omitted in the figure as well as in the remainder of the article.

Inputs for the fuzzy-neural network are the parameters of the load e_i, f_i of the executor, while the output is the estimated service time $\hat{s}_i$. Parameters of the model are stored in the parameter database $Z_i = [Z_{1i}, \ldots, Z_{ki}, \ldots, Z_{Ki}]$, where $Z_{ki} = [C_{ki}, D_{ki}, S_{ki}]$, $C_{ki} = [c_{1ki}, \ldots, c_{lki}, \ldots, c_{Lki}]$ and $D_{ki} = [d_{1ki}, \ldots, d_{mki}, \ldots, d_{Mki}]$ are the input fuzzy set function parameters and $S_{ki} = [s_{1ki}, \ldots, s_{jki}, \ldots, s_{Jki}]$ are the output fuzzy set function parameters. For each class k_i of the HTTP request, a different set of parameters is used in the fuzzy system. In this way, there are K different sets of parameters, and therefore, it can be said that there are K different fuzzy-neural models.

The fuzzy set functions for inputs $\mu_{F_{el}}(e_i)$, $\mu_{F_{fm}}(f_i)$, $l = 1, \ldots, L$; $m = 1, \ldots, M$, are triangular and the meaning of their parameters $c_{1ki}, \ldots, c_{lki}, \ldots, c_{Lki}$ is presented on the Fig. 5.b. Similar meaning have the parameters $d_{1ki}, \ldots, d_{mki}, \ldots, d_{Mki}$. The output fuzzy set functions $\mu_{Sj}(s)$ are singletons, where the parameters $s_{1ki}, \ldots, s_{jki}, \ldots, s_{Jki}$ denote the positions of each singleton (Fig. 5.c).

The estimated service time is calculated in the following way

$$\hat{s}_i = \sum\nolimits_{j=1}^{J} s_{jki} \cdot \mu_{R_j}(e_i, f_i), \tag{2}$$

where $\mu_{R_j}(e_i, f_i) = \mu_{F_{el}}(e_i) \cdot \mu_{F_{fm}}(f_i)$.

The process of adaptation is conducted every time the executor, corresponding to the executor model, services the request. Both the input and output fuzzy set parameters are tuned with the use of the Back Propagation Method [35] taking into account the measured service time $\tilde{s}_i$. Modification of the parameters is conducted in the following way:

$$s_{jk(i+1)} = s_{jki} + \eta_s \cdot (\tilde{s}_i - \hat{s}_i) \cdot \mu_{R_j}(e_i, f_i), \tag{3}$$

$$c_{\phi k(i+1)} = c_{\phi ki} + \eta_c (\tilde{s}_i - \hat{s}_i) \cdot \sum\nolimits_{m=1}^{M} \left(\mu_{F_{fm}}(f_i) \sum\nolimits_{l=1}^{L} \left(s_{((m-1)\cdot L + l)ki} \partial \mu_{F_{el}}(e_i) / \partial c_{\phi ki} \right) \right), \tag{4}$$

$$d_{\gamma k(i+1)} = d_{\gamma ki} + \eta_d (\tilde{s}_i - \hat{s}_i) \cdot \sum\nolimits_{l=1}^{L} \left(\mu_{F_{el}}(e_i) \sum\nolimits_{m=1}^{M} \left(s_{((l-1)\cdot M + m)ki} \partial \mu_{F_{fm}}(f_i) / \partial d_{\gamma ki} \right) \right), \tag{5}$$

where η_s, η_c, η_d are adaptation ratios, $\phi = 1, \ldots, L - 1$, $\gamma = 1, \ldots, M - 1$ [7].

In the beginning, the values of the input parameters $c_{1ki}, \ldots, c_{lki}, \ldots, c_{Lki}$ and $d_{1ki}, \ldots, d_{mki}, \ldots, d_{Mki}$ are evenly distributed over the space of executor operation, while the output parameters are set to zero. In this way, the estimated service times are always close to zero for those executors for which the decision system is not yet learned. Over time, the Web switch adapts and the estimated service times become longer and closer to the real service times [7].

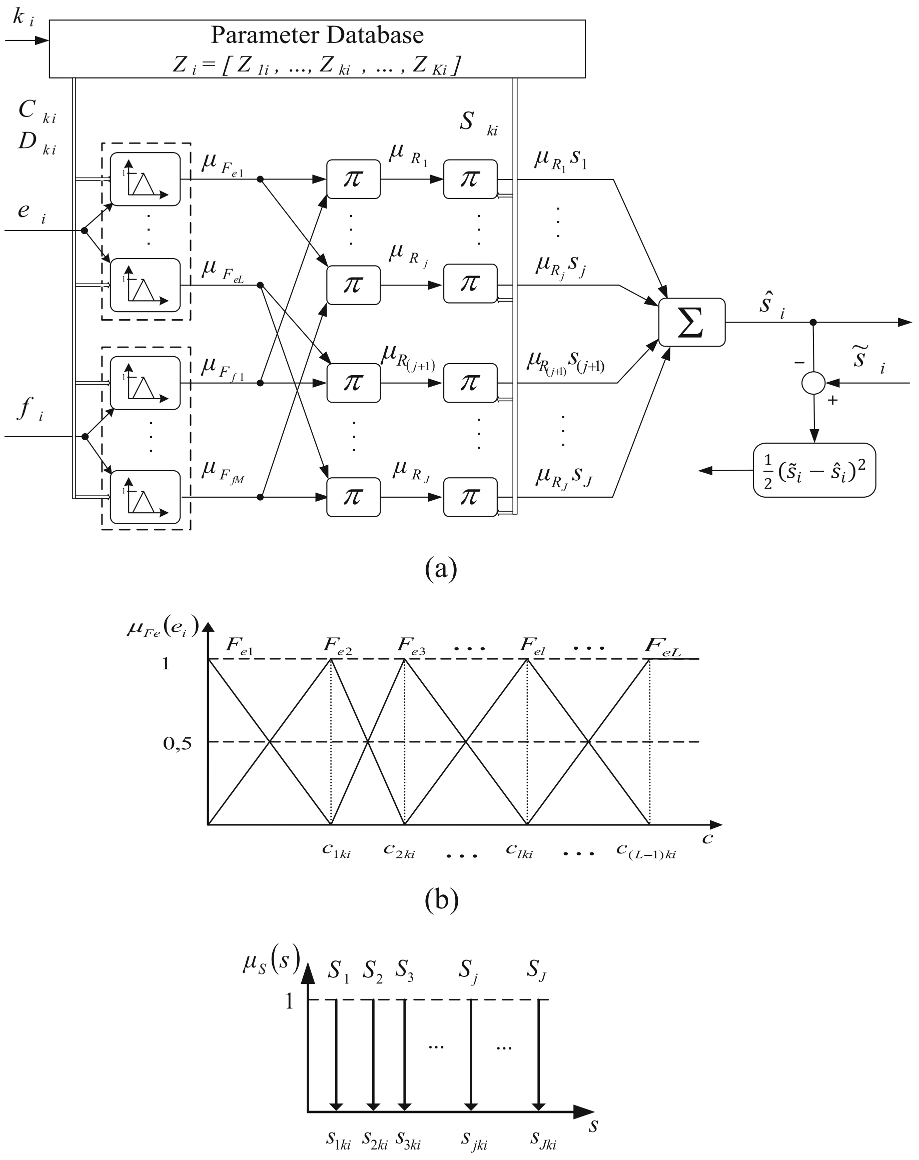

Fig. 5 Neuro-fuzzy model: (**a**) overall view, (**b**) input fuzzy sets functions, (**c**) output fuzzy sets functions

In the preliminary experiments, the optimal value of the number of input fuzzy sets has been determined as $L = M = 10$, and the number of output fuzzy sets is equal to $J = \mathrm{L} \cdot \mathrm{M}$.

Experiments and discussion

Research and experiments which were conducted for Web switches working in two-layer architecture showed that the cooperation of intelligent switches can significantly reduce the service time. In this section, it will be determined whether the use of a single-layer architecture with the two-level decision-making strategy TLFNRD is advantageous and better than the one-level intelligent FNRD decision strategy.

To evaluate the proposed system, simulation experiments have been conducted. The simulation program was written in the OMNeT++ environment. The OMNeT ++ provides appropriate libraries as well as the environment for conducting simulation and is the most popular system for evaluating networking systems [36].

The simulation program was divided into independent modules that imitate the behavior of different parts of the real system, namely: HTTP request generator, Web switch, Web servers, and database server. The scheme of the simulator is presented in Fig. 6.

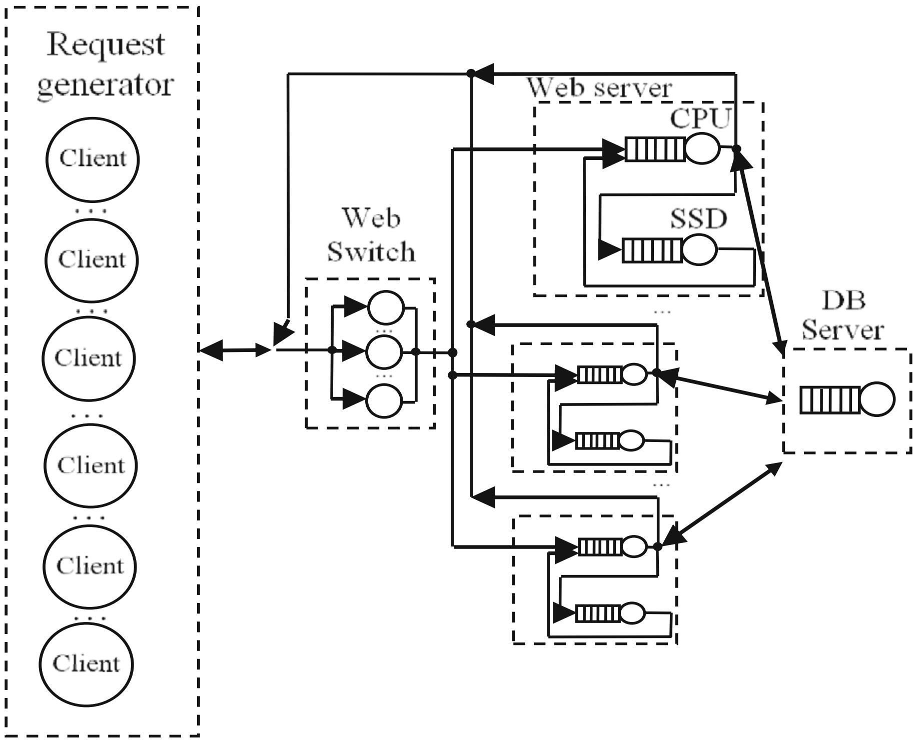

Fig. 6 The simulation model

To determine the values of the real system parameters, which can be used in the simulation, preliminary experiments have been conducted in a manner similar to that in [37]. The experiments were conducted for Web server with a computer equipped with Intel Core i7 7800X CPU, a Samsung SSD 850 EVO driver and 32 GB of RAM. The Apache Web server was running WordPress, the most popular CMS system in the world which is used on more than 25% of the world's websites [38]. Thank that it was possible to simulate the behavior of many real, business-oriented Websites.

The module of the request generator in the simulator contained many submodules of clients, each of which behaved like a real Web browser. To download a Web page, they were downloading the first document with HTML content, and then opening up to 6 TCP connections to fetch other elements of the page like css, js, pictures and other files. The number of Web pages downloaded by a single client during one session was modeled according to the behavior of human beings with the use of the Inverse Gaussian distribution ($\mu = 3.86$, $\lambda = 9.46$). The time between the opening of subsequent pages (the user think time) was modeled according to the Pareto distribution ($\alpha = 1.4$, $k = 1$) [39]. Each client after finishing its session was deleted and a new one was invoked.

Each of the clients was downloading a simulated Web site whose parameters (type and size of HTML and nested objects) were exactly the same as those in the very popular site https://www.sonymusic.com [40] running also on WordPress.

The Web switch in the simulator was able to distribute HTTP requests with the use of strategies popular in Amazon AWS [21] Web switches and with the use of intelligent strategies, namely:

- Round Robin (RR);
- Least Load (LL) – assigns HTTP requests to the nodes with the lowest number of serviced HTTP requests;
- Partitioning (P) – assigns requests to servers previously chosen for this kind of request. In the experiments, a modification of the P algorithm was used. It was more adaptive and behaving like LARD algorithm [41], in which, if the server is overloaded than the service of a given type of requests is moved to the least loaded server;
- Fuzzy-Neural Request Distribution (FNRD) – intelligent strategy using fuzzy-neural approach and single-level decision algorithm [7];
- Two-Level Fuzzy-Neural Request Distribution (TLFNRD) – the new approach.

In order to properly evaluate the TLFNRD and FNRD strategy, the simulator should have implemented other intelligent strategies known from the literature.

However, the implementation of those complex strategies can be very time-consuming. Moreover, the strategies, in most cases, are not described in detail in the articles and their implementation would not have fully reflected the way of working and the intentions of the authors. Therefore, it is almost impossible to compare the best solutions today. In this article the work of fuzzy-neural strategies is compared with the dynamic LL strategy that is very good in practical solutions and is a kind of the reference line.

The Web switch was modeled in the simulation as a single queue. The service times were measured on a real server with Intel Xeon E5–2640 v3 processor and were as follow: LL 0.0103 μs, RR 0.00625 μs, P 0.0101 μs, FNRD 0.2061 μs, TLFNRD 0.2033 μs.

Each of the Web server modules contained a separate queue modeling processor and SSD drive. Service times were acquired for the server described above. The RAM memory acted as the cache memory for the file system.

The database server was modeled as a single queue. The service times were measured for the same server as the Web server was running.

The experiments have been conducted for four different cloud systems containing: 8, 10, 12 and 14 Web servers, and a single database server. The chosen number of servers was not the smallest one that can be found in real solutions. Because of the nature of the two-level decision system, it would be recommended to use bigger Web clusters.

During the experiments, the mean service time was measured and in each experiment different load, measured as the number of generated clients, was used. Also 40 million of HTTP requests were served in every experiment, the warming phase was taking about 10 million requests and for 30 million the service time was measured.

Before starting the experiments, it was necessary to decide how to logically divide the web servers into groups in the TLFNRD strategy. Research presented in [9] for two-layer architecture indicated that there should be two servers in each group. Therefore, experiments have been carried out for such settings.

The results of the experiment are presented in Fig. 7. Each of the four diagrams presents the results for the cloud system containing a different number of Web servers.

In the real world, the end-users expect to get the content of the Web page as soon as it is possible. A significant part of the time of delivering the content to the user is the time of servicing HTTP requests. So it is crucial to service the request quickly to keep the high quality of service even the load of the cloud system is high.

The best results with short service times, for non-intelligent solutions, have been achieved for the LL

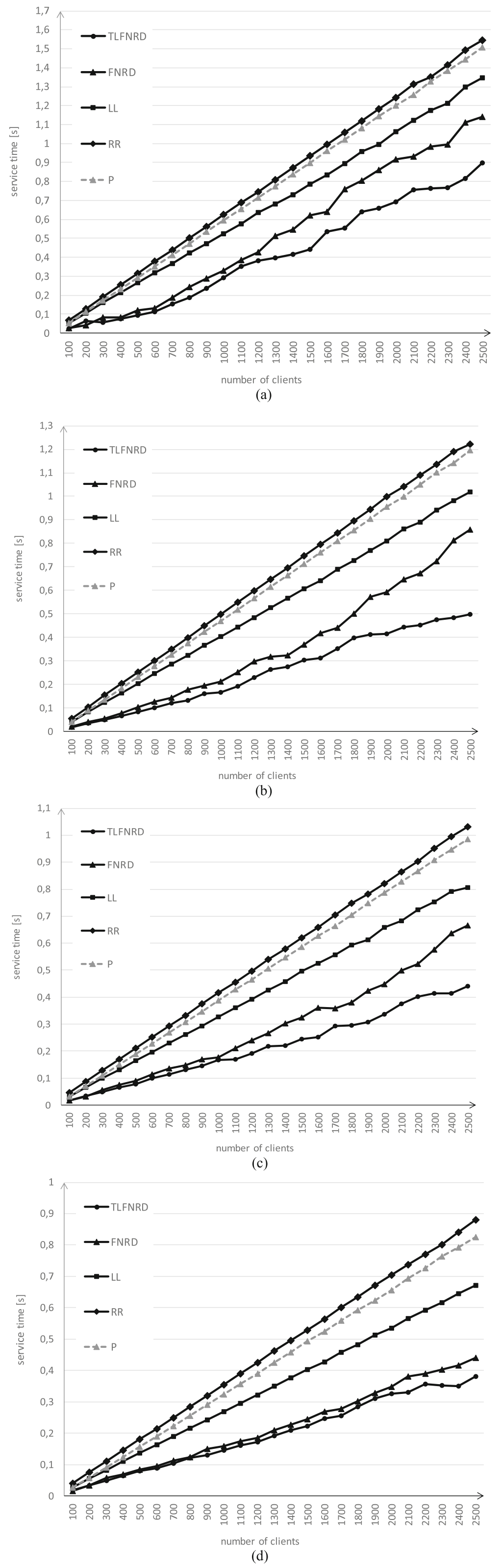

Fig. 7 Mean service time in load function (number of clients) in Web cluster containing: (**a**) 8 Web servers, (**b**) 10 Web servers, (**c**) 12 Web servers, (**d**) 14 Web servers

strategy. This strategy is simple, adaptive and very effective, especially in practical applications. Service times for this strategy are significantly lower than for RR and P approaches.

However, the results for both intelligent strategies were much better than for non-intelligent ones. In all of the experiments, service times for the new TLFNRD strategy were shorter than for FNRD and, when the load was increasing, the distance between the results was becoming significantly bigger.

In many of the experiments, service times for TLFNRD were two times lower than for LL strategy. In consequence, to keep the same quality of service for the TLFNRD strategy as for LL, a lower number of servers is necessary, and the maintenance costs are much lower.

It is also worth mentioning that in the simulation, the time of making a decision was taken into account. Although the decision times for the FNRD and the TLFNRD strategies are almost two orders of magnitude higher than for other strategies, the Web switch was not a bottleneck for the system in the experiments and obtained overall service times were much shorter for intelligent strategies.

Since the TLFNRD strategy uses information available within the Web switch in the decision-making process, the implementation of the proposed strategy would be possible in presently used Web switches in the Web cloud. The strategy can be used both in software and hardware Web switches. It should be noticed, however, that making decisions in the TLFNRD strategy requires more computing power than simple strategies like RR or LL need. Therefore the TLFNRD Web switch should have adequate computing power.

Summing up the research results, it is clearly beneficial to use the two-level intelligent, fuzzy-neuro, decision-making strategy. Results for the one-level, fuzzy-neuro strategy are worse. The proposed new method increases the quality in the Web cloud systems and lowers the costs. Further research on the new solution can deliver information on its uses and features.

Summary

In this article, a new HTTP request distribution strategy for cloud-based Web systems was presented. The proposed TLFNRD strategy is a new quality in the field of load balancing strategies using the fuzzy-neuro approach. The new strategy uses a two-level decision system in which, on the first level, a group of servers is chosen to service the request, and on the second level, a single Web server is selected. On each level, the fuzzy-neural model estimates the service times for chosen elements.

To evaluate the TLFNRD strategy a simulation environment was designed and implemented. The simulator was able to imitate correctly the behavior of Web clients, as well as the work of Web switch and both the Web and the database servers. In all of the experiments, intelligent strategies get much better results than the non-intelligent strategies used mostly in popular Web cloud systems. The two-level TLFNRD strategy resulted in shorter service times than the one-level intelligent FNRD strategy, especially when the load was high.

The research results indicate that the new solution is important and further research into two-level decision-making systems should be continued.

Abbreviations

ABC: Artificial Bee Colony; ACO: Ant Colony Optimization; AWS: Amazon Web Services; GARD: Global Request Distribution; GARDIB: Global Request Distribution with Broker; FARD: Fuzzy Adaptive Request Distribution; FNRD: Fuzzy- Neural Request Distribution; HTTP: Hypertext Transfer Protocol; LARD: Locality-Aware Request Distribution; LL: Least Load; P: Partitioning; RAM: Random-Access Memory; RR: Round Robin; PSO: Particle Swarm Optimization; SSD: Solid-State Drive; TLFNRD: Two-Level Fuzzy- Neural Request Distribution

Acknowledgements

Not applicable.

Author's contributions

KZ played the most important role in this paper. The author(s) read and approved the final manuscript.

Authors' information

Krzysztof Zatwarnicki is with the Institute of Computer Science at the Opole University of Technology, Opole, Poland. He received Ph.D. in computer science from Wrocław University of Science and Technology in 2003 and D.Sc. degrees in the same field from West Pomeranian University of Technology Szczecin, Poland in 2013. Since 2013 serves as Head of the research group at the Opole University of Technology. He has published over 70 journal and conference papers in the areas of artificial intelligence, computer networks, application of artificial intelligence in computer networks, load distribution and sharing in a cluster and cloud-based Web systems.

Funding

Not applicable.

Availability of data and materials

Because a code involves our interests, we're sorry but we can't publish source code at present. However, they are available from the author on a reasonable request.

Competing interests

The author declares that he has no conflict of interest.

Received: 24 February 2020 Accepted: 1 June 2020
Published online: 11 June 2020

References

1. Costello K, Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17.5 Percent in 2019, 2019. https://www.gartner.com/en/newsroom/press-releases/2019-04-02-gartner-forecasts-worldwide-public-cloud-revenue-to-g, Accessed 09.06.2019
2. Lee B T G, Patt R, JeffVoas C. DRAFT Cloud Computing Synopsis and Recommendations, 2011. http://csrc.nist.gov/publications/nistpubs/800-146/sp800-146.pdf. Accessed 09.01.2020
3. Puthal D (2015) Cloud computing features, issues, and challenges: a big picture, international conference on computational intelligence and networks (CINE). IEEE Press, Bhubaneshwar, India, pp 116–123

4. Patiniotakis I, Verginadis Y, Mentzas G (2015) PuLSaR: preference-based cloud service selection for cloud service brokers. J Internet Serv Appl 6. https://doi.org/10.1186/s13174-015-0042-4
5. Borzemski L, Zatwarnicki K (2003) A Fuzzy Adaptive Request Distribution algorithm for cluster-based Web systems. Proceeding of 11th Euromicro Conference on Parallel Distributed and Network based Processing. IEEE Press, Genua
6. Borzemski L, Zatwarnicki K, Zatwarnicka A (2007) Adaptive and intelligent request distribution for content delivery networks. Cybern Syst 38(8):837–857
7. Zatwarnicki K (2012) Adaptive control of cluster-based web systems using neuro-fuzzy models. Int J Appl Math Comput Sci 22(2):365–377
8. Zatwarnicki K (2011) Guaranteeing quality of Service in Globally Distributed web System with brokers, proceedings of computational collective intelligence technologies and applications: third international conference. ICCCI 2011, vol 6923. Springer-Verlag, Gdynia, pp 374–384
9. Zatwarnicki K, Zatwarnicka (2019) A Cooperation of Neuro-Fuzzy and Standard Cloud Web Brokers. In: Borzemski L, Świątek J, Wilimowska Z (eds) Information Systems Architecture and Technology: Proceedings of 40th Anniversary International Conference on Information Systems Architecture and Technology – ISAT 2019. ISAT 2019. Advances in Intelligent Systems and Computing, vol 201. Springer, Cham, p 1050 243–254
10. Zatwarnicki K, Zatwarnicka A (2019) A Comparison of Request Distribution Strategies Used in One and Two Layer Architectures of Web Cloud Systems. In: proc. Computer Networks. CN 2019. Communications in Computer and Information Science, vol 1039. Springer, Cham, pp 178–190
11. Alakeel A (2010) A guide to dynamic load balancing in distributed computer systems. Int J Comput Sci Info Secur 10(6):153–160
12. Rimal BP (2011) Architectural requirements for cloud computing systems: an enterprise cloud approach. J Grid Comput 9(1):3–26
13. Ponce LM, Santos W, Meira W (2019) et al, Upgrading a high performance computing environment for massive data processing. J Internet Serv Appl 10. https://doi.org/10.1186/s13174-019-0118-7
14. Nuaimi KA (2012) A Survey of Load Balancing in Cloud Computing: Challenges and Algorithms. Network Cloud Computing and Applications (NCCA), 2012 Second symposium on. IEEE,London, UK.
15. Zenon C, Venkatesh M, Shahrzad A (2011) Availability and load balancing in cloud computing. Int Confer Comput Soft Model IPCSI 14:134-140
16. Campelo RA, Casanova MA, Guedes DO et al (2020) A brief survey on replica consistency in cloud environments. J Internet Serv Appl 11. https://doi.org/10.1186/s13174-020-0122-y
17. Afzal S, Kavitha G (2019) Load balancing in cloud computing – a hierarchical taxonomical classification. J Cloud Comp 8:22. https://doi.org/10.1186/s13677-019-0146-7
18. Xu Z, Xingxuan W (2015) A predictive modified round robin scheduling algorithm for web server clusters. Proceedings of 34th Chinese Control Conference. IEEE, Hang-Zhou
19. Brototi M, Dasgupta K, Dutta P (2012) Load Balancing in Cloud Computing using Stochastic Hill Climbing-A Soft Computing Approach. In: Proc. 2nd International Conference on Computer, Communication. Control and Information Technology (C3IT)
20. Walczak M, Marszalek W, Sadecki J (2019) Using the 0-1 test for chaos in nonlinear continuous systems with two varying parameters: parallel computations. IEEE Access 7:154375–154385
21. Amazon, How Elastic Load Balancing Works. 2019. https://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/how-elastic-load-balancing-works.html . Accessed 23 Jan 2020
22. Rafique A, Van Landuyt D, Truyen E et al (2019) SCOPE: self-adaptive and policy-based data management middleware for federated clouds. J Internet Serv Appl 10. https://doi.org/10.1186/s13174-018-0101-8
23. Crovella M, Bestavros A (1997) Self-similarity in world wide web traffic: evidence and possible causes. IEEE/ACM Trans Networking 5(6):835–846
24. Domańska J, Domański A, Czachórski T (2005) The Influence of Traffic Self-Similarity on QoS Mechanisms. Proceedings of SAINT 2005 Workshop, Trento
25. Remesh Babu KR, Samuel P (2016) Enhanced bee Colony algorithm for efficient load balancing and scheduling in cloud. In innovations in bio-inspired computing and applications. In: Advances in Intelligent Systems and Computing, vol 424. Springer, Cham, pp 67–78
26. Suchacka G, Dembczak A (2017) Verification of web traffic Burstiness and self-similarity for multiple online stores. Adv Intell Syst Comput 655:305–314
27. Nishant K Load Balancing of Nodes in Cloud Using Ant Colony Optimization, Computer Modelling and Simulation (UKSim). 2012 UKSim 14th InternationalConference on, vol 2012. IEEE, Cambridge, UK
28. Suraj P, Wu L, Guru MS, Buyya R (2010) A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments. Proceedings of 24th IEEE International Conference on Advanced Information Networking and Applications, Perth
29. Sharifian S, Akbari MK, Motamedi SA (2005) An Intelligence Layer-7 Switch for Web Server Clusters. In: International Conference: Sciences of Electronic, Technologies of Information and Telecommunications SETIT, 3rd edn, pp 8–24
30. Boutaba R, Salahuddin MA, Limam N et al (2018) A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. J Internet Serv Appl 9. https://doi.org/10.1186/s13174-018-0087-2
31. Seok-Pil L, Nahm E-S (2012) Development of an optimal load balancing algorithm based on ANFIS modeling for the clustering web-server. Communications in Computer and Information Science, vol 310, pp 783–790
32. Sallami NMA, Daoud AA, Alousi SA (2013) Load balancing with neural network. Int J Adv Comput Sci Appl 4(10):138–145
33. Zatwarnicki K, Platek M, Zatwarnicka A (2015) A cluster-based quality aware web system. Information systems architecture and technology, vol 430. Springer, Cham, pp 15–24
34. Mamdani EH (1977) Application of fuzzy logic to approximate reasoning using linguistic synthesis, vol C-26. IEEE Transactions on Computer, Washington, DC, p 1182 - 1191
35. Rumelhart D, Hinton G, Williams R (1986) Learning representations by back-propagating errors. Nature. 323:533–536
36. OMNeT++ Discrete Event Simulator. 2020. https://www.omnetpp.org/. Accessed 02 Feb 2020.
37. Zatwarnicki K (2008) Determination of parameters of parameters of web server simulation model. Information Systems Architecture and Technology: Web Information Systems, Models, Concepts & Challenges. Springer, Cham, pp 25–36
38. Munford M. How WordPress Ate The Internet in 2016… And The World in 2017.2017. https://www.forbes.com/sites/montymunford/2016/12/22/how-wordpress-ate-the-internet-in-2016-and-the-world-in-2017/. Accessed 02 Feb 2020.
39. Cao J, Cleveland SW, Gao Y, Jeffay K, Smith FD, Weigle MC (2004) Stochastic models for generating synthetic HTTP source traffic. Proceedings of Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies. INFOCOM, Hong-Kong, pp 1547–1558
40. Sony music, Main Page. 2019. https://www.sonymusic.com/ . Accessed 09 Mar 2019.
41. Pai VS, Aron M, Banga G, Svendsen M, Druschel P, Zwaenepoel W, Nahum E (1998) Locality-aware request distribution in cluster-based network servers. Proceedings of the 8th international conference on architectural support for programming languages and operating systems. San Jos;. California; USA. ACM SIGOPS Operating Syst Rev 32(5):205–216

Publisher's Note