applied sciences

MDPI

Article

Providing Predictable Quality of Service in a Cloud-Based Web System

Krzysztof Zatwarnicki

Department of Computer Science, Opole University of Technology, Proszkowska 76, 45-758 Opole, Poland; k.zatwarnicki@po.edu.pl

Abstract: Cloud-computing web systems and services revolutionized the web. Nowadays, they are the most important part of the Internet. Cloud-computing systems provide the opportunity for businesses to undergo digital transformation in order to improve efficiency and reduce costs. The sudden shutdown of schools and offices during the pandemic of Covid 19 significantly increased the demand for cloud solutions. Load balancing and sharing mechanisms are implemented in order to reduce the costs and increase the quality of web service. The usage of those methods with adaptive intelligent algorithms can deliver the highest and a predictable quality of service. In this article, a new HTTP request-distribution method in a two-layer architecture of a cluster-based web system is presented. This method allows for the provision of efficient processing and predictable quality by servicing requests in adopted time constraints. The proposed decision algorithms utilize fuzzy-neural models allowing service times to be estimated. This article provides a description of this new solution. It also contains the results of experiments in which the proposed method is compared with other intelligent approaches such as Fuzzy-Neural Request Distribution, and distribution methods often used in production systems.

Keywords: web cloud system; cloud computing; web systems simulation; fuzzy-neural network; fuzzy-neural modeling; intelligent system; HTTP request distribution

Citation: Zatwarnicki, K. Providing Predictable Quality of Service in a Cloud-Based Web System. *Appl. Sci.* **2021**, *11*, 2896. https://doi.org/10.3390/app11072896

Academic Editor: Eui-Nam Huh

Received: 3 March 2021
Accepted: 22 March 2021
Published: 24 March 2021

1. Introduction

Nowadays, one of the most complex systems built by humans is the Internet network which is composed of numerous resources, web content, access devices, and a large number of users. Over the last decade, the number of active Internet users has increased to 4.66 billion. Almost 91% of World Wide Web (WWW) users use mobile devices as well as mobile phone networks and this percentage is expected to increase in the future [1].

To ensure effective and fast access to web content, many companies use cloud-computing solutions. This has opened new opportunities for providing large-scale computing resources [2]. The global cloud-computing market size is expected to grow from USD 371.4 billion in 2020 to USD 831 billion by 2025. The sudden shutdown of offices, enterprises, and schools during the Covid 19 pandemic has increased the demand for cloud solutions and services. It seems that many solutions developed during the pandemic will remain, especially in the field of the home office. However, this requires further continuous transfer of solutions known from everyday life to the Internet platform [3].

Cloud-computing systems enable access to a shared pool of computing resources such as servers, storage, applications, and services delivered on-demand [4,5]. The organization of resources improves the performance, utilization of resources, and energy consumption management and helps to avoid SLA (Service-Level Agreement) violations [6].

The solutions used in cloud computing mean that their application costs are lower compared to building server infrastructures in enterprises. Nevertheless, cloud-computing systems do not provide access to unlimited resources from the client's point of view, mostly due to the costs of the use. In order to reduce the costs and increase the quality of service

(QoS), or especially the quality of web service (QoWS), load balancing and load sharing mechanisms are implemented. The proper and effective distribution of the load is still an open problem in cloud computing. Little comprehensive research in the field of cloud computing has been done. New architecture structures and algorithms meeting clients' demands need to be developed.

The nodes in the cloud-based web system are distributed. There are two types of load sharing techniques related to the centralization of decision units [7]—centralized and distributed. In centralized systems, the distribution decisions are made by a single node. This node stores knowledge of the whole cloud system and uses load sharing algorithms. Distributed and hierarchical load sharing involves different layers of the cloud. There is no single node fully responsible for making distribution decisions.

The complexity of cloud infrastructures has created the need for innovative monitoring approaches [8]. The complex nature, at many different levels, of this kind of system is the underlying cause of many problems. This has an effect on all aspects of cloud operation and needs to be handled properly in order to provide high performance, dependability, and quality of service etc. [9].

Most cloud systems are physically divided into regions (server rooms) placed in different geographical locations. Regions are built of availability zones—parts of the regions with independent network infrastructure [10].

Web switches distribute the load in the cloud systems. Taking into account the architecture of a typical web cloud using only one region, a centralized cloud contains one web switch distributing the load all over the region. The switch is placed in one of the zones. It is called one-layer architecture (Figure 1a). A distributed hierarchical web cloud contains many web switches making decisions on different layers. Figure 1b presents a two-layer architecture with one web switch in the region and separate web switches in zones.

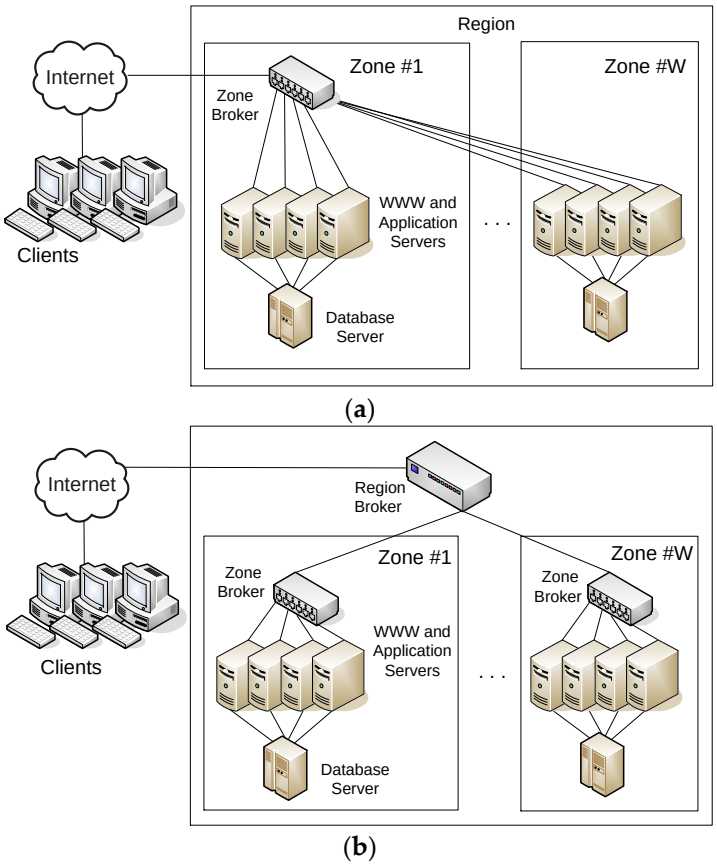


Figure 1. Cloud-base web systems: (**a**) centralized one-layer architecture and (**b**) distributed two-layer architecture.

The quality of the web services can be assessed by the end-users in many different ways. An interesting content of the website ensures an increase in the number of users. However, it is crucial to deliver the content immediately after receiving a request. The users will consider the service to be of low quality when the time required to get the page is too long. Therefore, the quality of the web service is often evaluated based on the time required to receive the content. Time delays concerning fetching web content are related to the time of sending data via the Internet and the time of servicing HTTP requests by the web services, especially cloud-based web systems.

Based on predictability, the web services can be categorized as best-effort, predictable, and guaranteed [11]. Best-effort services provide no control over how the service will satisfy the user requirements. It means that the rest of the system (users and applications) will need to adapt to the service's state and the service will be unpredictable. In most cases, best-effort services do their best to service requests as fast as possible, but they cannot guarantee anything. Guaranteed service is the opposite of best-effort service. Guaranteed service is predictable and reliable. When service is not available, the provider must account for the loss of service. There are also predictable services. They are something between best-effort and guaranteed services. They provide some degree of predictability but do not require the accountability of guaranteed service.

The HTTP request distribution method in a two-layer architecture of a cluster-based web system is presented in the article. The method allows predictable QoWS to be provided by keeping the service times within adopted constraints and at the same time processing HTTP requests very efficiently. The method is called Web Cloud Earliest Deadline First (WCEDF). The main element designed to be used in the method is the WCEDF web switch. It allows the incoming requests to be queued by letting the time-consuming requests be serviced first. It additionally distributes requests in a way that keeps selected resources of the cloud idle and ready to service time-consuming requests faster. The presented method is another step towards the development of algorithms and methods of load distribution and sharing in cloud-computing systems.

The rest of the article is composed as follows: In Section 2, the related work is presented with a description of the previous works that constitute the basis for a new solution. Section 3 provides a description of the WCEDF method. In Section 4, the testbed and results of experiments and discussion are presented. Section 5 presents conclusions and the directions of future research. Section 6 summarizes the article.

2. Related Work and Motivations

Cloud computing uses distributed systems to achieve the very high demand for computational power and quality of service. Load balancing and sharing methods are used to distribute tasks and requests among nodes in the system [12]. Those methods help to reduce utilization of the resources, improve the response times, and enable scalability [13,14].

Web-based cloud systems that are publicly available for users usually use simple solutions as far as of load distribution is concerned. For example, Amazon Web Service (AWS)—the largest player on the cloud-computing market, provides the following algorithms in the web switches available for use [10]: Round-Robin, which is a carousel algorithm forwarding requests to subsequent servers, Last-Loaded that assigns HTTP requests to the nodes with the least loaded servers, and Path-based routing in which requested resources are assigned to given web servers.

Distribution strategies for cluster-based systems have been developed for a long time and nowadays, they are adapted to new structures and solutions of cloud-based systems. Load balancing and sharing strategies can be divided into three main categories [12,15–18]—static, dynamic, and adaptive. In static load balancing, decisions are made using deterministic or probabilistic algorithms that do not take into account the current state of the system. Round-Robin and Path-based routing are a good example of such strategies.

In a dynamic approach, load balancing assignments are conducted based on the current state of the system. The most popular dynamic load balancing algorithm is Least Load.

The adaptive approach includes the most complex solutions. Decisions are made on the basis of the state of the system, however the strategy can change when the state of the system is changing [19]. The majority of the adaptive strategies are intelligent approaches. Some specialists [20] claim that only those approaches can effectively provide the acceptable service time in web traffic conditions, typically characterized by self-similarity and burstiness [21–23].

Many studies have been conducted on the application of artificial intelligence mechanisms in distribution strategies. Some of the algorithms use natural-phenomena-based strategies. A good example of such a strategy is the Artificial Bee Colony (ABC) whose decision mechanism imitates the behavior of bees [18]. Another strategy is Particle Swarm Optimization (PSO) which is based on an algorithm taking into account the costs of service and transmission [24,25]. Artificial neural networks also have been used in adaptive load distribution systems [26,27].

The problem of providing guaranteed services in web systems has been the topic of much recent research [28–33]. Most of the works focus on providing differentiated services for classes of clients using priority-based scheduling. However, proposed solutions allow the quality of the web services to be retained only for a specific group of users, and it is connected with rejecting requests of other users. There have only been a few works devoted to the problem of providing predictable services [34–36].

Many previously designed solutions providing guaranteed services for web systems can be applied in cloud-computing web systems. Unfortunately, most of them do not fit well to the architecture and the structure of the cloud. There have not been many works devoted to the problem of designing solutions for guaranteed and predictable services provided in cloud-based web systems [37–40].

The author's latest works were devoted to the problem of adapting solutions designed for cluster-based web systems to the specificity of work in the web cloud. Intelligent FNRD (Fuzzy-Neural Request-Distribution) web switches working in two layers of the cloud—the region and zones, have been used [41–43]. The web switches were using a neuro-fuzzy decision system estimating the HTTP request service times. The results of the conducted experiments revealed that the intelligent web brokers in the two-layer architecture cooperated and learned each other's behavior achieving very high efficiency of cloud-based web system in this way [44]. Taking into account the results and experience in designing predictable services in cluster-based systems [34,36] a new distribution method was proposed by the author. The proposed WCEDF method uses intelligent web switches and provides predictable services in a two-layer architecture of a cluster-based web system.

3. Web Cloud Earliest Deadline First Method and Web Switch

Servicing HTTP requests on a high level of quality in cloud-based web systems is crucial for many website owners. Requirements regarding the quality of web services can be different depending on the expectations of the users of the website. Presented in this article, the WCEDF method allows HTTP requests to be distributed in this way to keep the service time within the expected time limits.

The web cloud system using the WCEDF method is devoted to work in a single region of the cloud and is composed of the following elements (Figure 1b):

- WCEDF web switch distributing HTTP requests among availability zones in the region;
- FNRD web switches distributing HTTP requests among web servers in availability zones;
- Frontend WWW servers and backend database servers, together servicing HTTP requests.

The most important element in the system is the WCEDF web switch receiving HTTP requests from clients (web browsers) and distributing them among availability zones in one region of the cloud system. The WCEDF web switch uses techniques of artificial intelligence to adapt to the changing environment and to learn the behavior of availability

zones. Inside the zones, the intelligent FNRD web switches distribute HTTP requests among web servers.

As was mentioned before, previous research showed that FNRD web switches working in different layers cooperate with each other achieving very good results. In the WCEDF method, it is also desired to get the effect of cooperation of web switches and to additionally achieve a new goal in decision-making. In the proposed method, the WCEDF switch working on the region should cooperate with FNRD switches working on a zone layer.

The main aim of the proposed WCEDF switch is to make the cloud system service the request in a time that is no longer than the one adopted for a specific website. The adopted time boundaries to service the request have to be reasonable and possible to achieve even for time-consuming requests requiring more resources. It is also assumed that the system will not reject HTTP requests to guarantee the service time when the load of the web system is too high. In this case, the WCEDF web switch will do its best to make the time as short as possible.

It should be noticed that the design of the WCEDF switch does not include all features of productive web switches used in practical applications. It lacks solutions associated with security, resistance to cyber-attacks, and a failover system used when the web server or the web switch fails.

The proposed WCEDF web switch is composed of two main sections (Figure 2). The first one, Scheduling Section, queues the incoming requests letting the time-consuming request to be serviced first. The second, Switching Section, distributes requests in a way that keeps selected availability zones idle, ready to service time-consuming requests faster. Both sections act independently, trying to achieve a similar goal, which is servicing the request in a time shorter than the adopted value t_{max}. This value determines the level of quality of service which should be achieved for a specific web system.

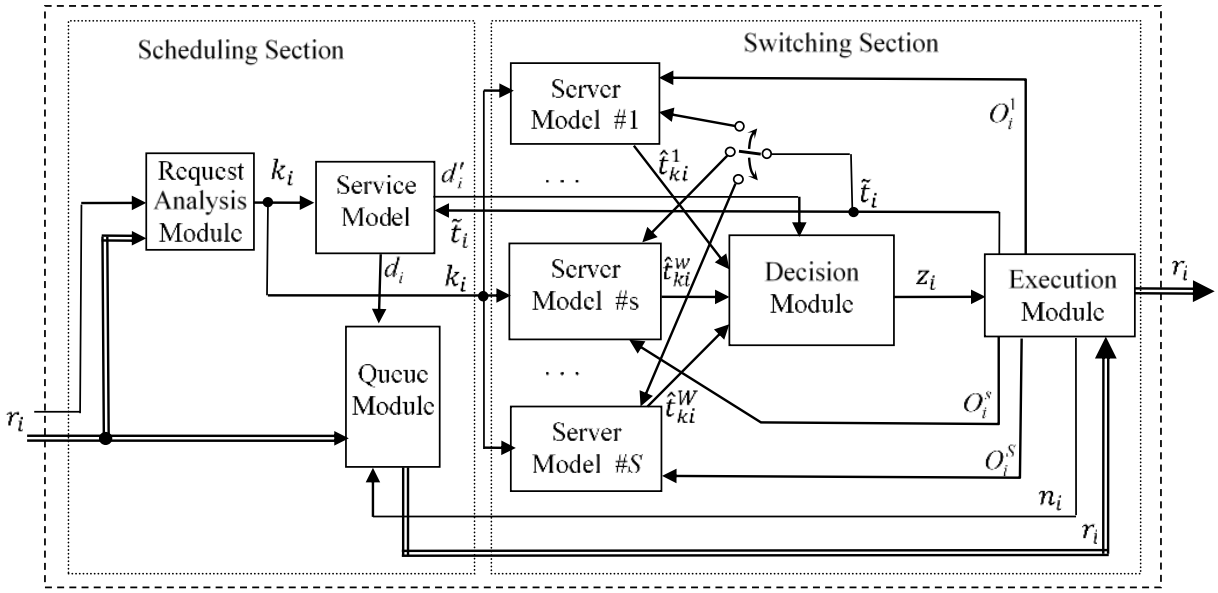


Figure 2. Web Cloud Earliest Deadline First (WCEDF) web switch design.

3.1. Scheduling Section

The Scheduling Section queues and organizes the order of incoming requests in a way that lets the requests to be serviced in a time shorter than the expected time t_{max}. The time should be achievable for a not-overloaded web system, even for time-consuming requests.

The scheduling section is composed of three modules—a request-analysis module, a service module, and a queue module.

The incoming HTTP request, r_i (where i is the index of request and $i = 1, \ldots, I$) is, on one hand, a physical HTTP request containing expressions compatible with HTTP protocol, and on the other hand, it is the set of information. In Figure 2, the request r_i is marked by a double line, representing the physical flow of the request in the switch, and a single line representing carried information.

The request analysis module classifies incoming HTTP request r_i to single class k_i ($k_i \in \{1, \ldots, K\}$), which is a number representing requests having similar service times. Requests can be divided into two groups. The first consists of requests for static objects such as images, html, style css, or javascript files. The second group of requests (called dynamic requests) is connected with the dynamic content of responses produced at the moment of arrival of the request to the WWW server (by executing scripts on the server like PHP, Python, Java, or Net Core). As far as static requests are concerned, the classification is conducted on the basis of the requested files' size. In the case of dynamic requests, they are classified separately for each requested object, for example, taking into account the address of the object.

The queue module stores requests and puts them into the queue, Q_i. The policy of the queue is EDF (Earliest Deadline First) and the requests are organized in accordance with deadlines $d_{i-j}, \ldots, d_i$ calculated in the service model for each request put into the queue. The deadline d_i determines when the service of the request r_i should begin on web servers. The request r_i can leave the queue Q_i if it is first in the queue and the number n_i of the requests being serviced in the web system is not greater than n_{max}. The value n_{max} is a maximum number of requests serviced at the same time in the web system. If $n_i < n_{max}$ then the request is not placed in the queue and is passed to the switching section and serviced in the web system.

Queueing requests at the front of the system has many advantages, but it also has some disadvantages. When we queue requests, we have the opportunity to reorder them in a way that lets the problematic request to be serviced first. Unfortunately, keeping requests in the queue prevents them from being serviced on the web servers. Therefore, it is very important to find the balance between the length of the queue and the number of requests being serviced in the rest of the system. The value n_{max} determines both queue length and number of serviced requests. It is calculated for the web system as follows

$$n_{max} = \begin{cases} n_{min}, \; if \; \overline{N} < n_{min} \\ w \cdot \overline{N}, \; otherwise \end{cases}, \tag{1}$$

where $\overline{N}$ is the mean number of requests in the web system (the sum of queued and serviced requests) calculated periodically after servicing determined, large number, or requests. n_{min} is a minimum value that can be assigned to the n_{max}. In the experiments described in Section 4, it was obtained that $n_{max} = 500$ and the mean value $\overline{N}$ and n_{max} were recalculated after servicing each 10,000 requests.

The service model calculates the deadline d_i and a term d'_i. As mentioned earlier, the deadline d_i points the time that the request r_i should start to be serviced. The term d'_i is a moment when the service should be finished and the response should be sent to the client. It is calculated for every request as follows:

$$d'_i = \tau_i^{(1)} + t_{max} \tag{2}$$

where $\tau_i^{(1)}$ is the moment of arrival of r_ith request.

The deadline d_i is calculated only for requests being put into the queue module (not sent directly to the switching section)

$$d_i = \tau_i^{(1)} + t_{max} - t'_{ki} \tag{3}$$

In order to calculate the deadlines, the service module stores information about service times in $U'_i = [t'_{1i}, \ldots, t'_{ki}, \ldots, t'_{Ki}]$, where t'_{ki} is an estimated service time of the request

belonging to the kth class and is the most actual information at the moment of arrival of r_ith request. The time t'_{ki} is always updated when a service of request belonging to kth class is finished and the previous request is stored in the queue module. The new time is calculated as follows:

$$t'_{k(i+1)} = t'_{ki} + \eta'(\widetilde{t}_i - t'_{ki}) \quad (4)$$

where $\widetilde{t}_i$ is a service time of r_ith request and η' is adaptation ratio determining the speed of changes, $\eta' \epsilon (0, 1)$. The time $\widetilde{t}_i$ is measured by the switching section from the moment when the request is sent to the rest of the system to the moment when the response to the request arrives to the WCDF web switch. It should be noticed that time t'_{ki} is calculated for an almost constant load of the system equal to n_{max}. However, because n_{max} can change periodically, the adaptation ratio η' should be big enough to let the vector U'_i adapt quickly to the new conditions. Preliminary experiments showed that η' should be equal to 0.6.

3.2. Switching Section

After leaving the scheduling section, the request r_i is passed to the switching section. This section distributes HTTP requests among web switches in the availability zones. The request is processed in this section as follows—first, the service time is estimated for each availability zone in the system, then the section makes a decision as to which zone should service the request. In most cases, the most loaded zone is chosen only if it is able to service the request within the time constraints. In the next step, the request is sent to the chosen zone. After finishing the service, the response is sent back to the switching section. The section measures the service time and load of the system. It also updates information about service times. In the end, the response is sent to the client.

The switching section contains three types of modules—zone model, decision, and execution modules. The number of zone model modules is equal to the number of availability zones in the cloud. Each zone model corresponds to one zone.

The zone model is responsible for estimating service time $\hat{t}^w_{ki}$ of r_ith request for corresponding wth zone, where $w \epsilon [1, \ldots, W]$. The estimation is done based on the information of the load of the zone $M^w_i = [e^w_i, f^w_i]$, where e^w_i is the overall number of requests being currently serviced by the zone, and f^w_i is the number of serviced dynamic requests. The load M^w_i is collected by the executor module.

The zone model adapts to the changing environment, after the request's service, by taking into account the measured service time $\widetilde{t}_i$, only if the corresponding zone was chosen to service the request. The model owes its adaptation capabilities to the use of a fuzzy-neural mechanism. The module's fuzzy logic structure is based on the Mamdani model [45], and it allows the calculation of the service time on the basis of the load, while the neural nature of the module permits it to tune its parameters. The structure of the presented model is described in detail in [41] and it is used in the WCEDF switch due to its advantages, including good quality of time estimation and the possibility of adaptation to the changing environment.

The structure of the neuro-fuzzy network used in the zone model is presented in Figure 3a. Superscripts indicating the executor's number (w) have been omitted in the figure as well as in the other markings in the rest of the description of the zone model.

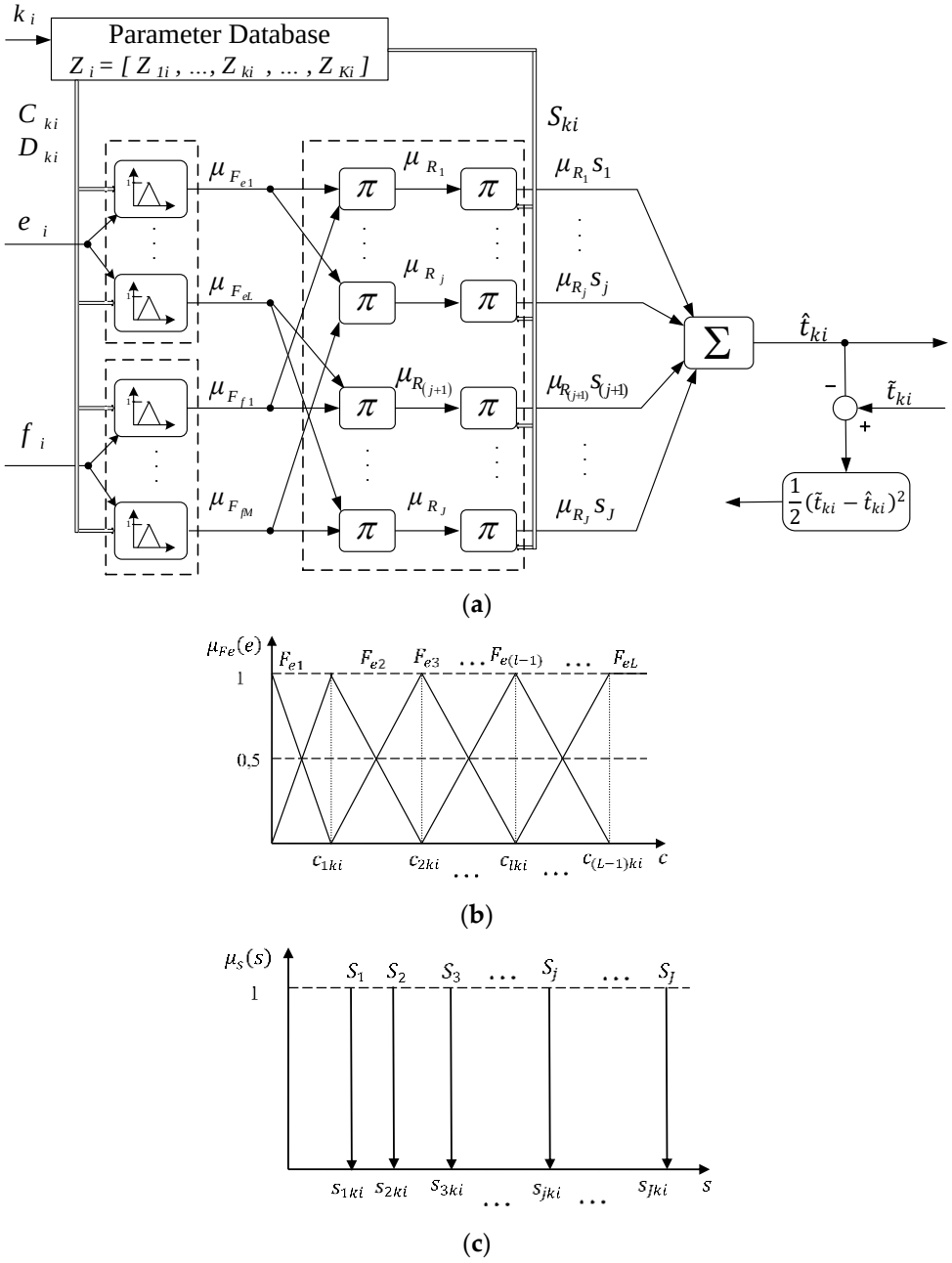


Figure 3. Neuro-fuzzy model: (**a**) overall view, (**b**) input fuzzy set functions, and (**c**) output fuzzy sets functions.

Inputs for the neuro-fuzzy network are the load of the zone e_i, f_i, and the class k_i of the request, while the output is the estimated service time $\hat{t}_{ki}$. The model parameters are stored in the 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 input fuzzy set function parameters and $S_{ki} = \left[s_{1ki}, \ldots, s_{jki}, \ldots, s_{Jki}\right]$ are output fuzzy set function parameters. The parameters can change in time and the index i in the database indicates the state at the moment of arrival of the r_ith request. A different set of parameters is used in the fuzzy system for each class k, $k = 1, \ldots, K$. Therefore, there are K different sets and it can be considered as K different fuzzy-neural models in each zone model.

The fuzzy set functions for input e_i are $\mu_{F_{el}}(e_i)$, $l = 1, \ldots, L$, and for input f_i are $\mu_{F_{fm}}(f_i)$, $m = 1, \ldots, M$. Figure 3b presents the triangular shape of functions $\mu_{F_{el}}(e_i)$ and the meaning of parameters $c_{1ki}, \ldots, c_{lki}, \ldots, c_{Lki}$. Functions $\mu_{F_{fm}}(f_i)$ have a similar shape and meaning to parameters $d_{1ki}, \ldots, d_{mki}, \ldots, d_{Mki}$. The output fuzzy set functions $\mu_{Sj}(s)$

are singletons (Figure 3c) and their parameters $s_{1ki}, \ldots, s_{jki}, \ldots, s_{Jki}$ point to the positions of each singleton.

The estimated service time is calculated as follows:

$$\hat{t}_{ki} = \sum_{j=1}^{J} s_{jki} \cdot \mu_{R_j}(e_i, f_i) \tag{5}$$

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

The presented method of calculation of the service time in the zone is typical for fuzzy logic systems. The adaptation abilities are, however, connected with the neural nature of the zone module. The Back Propagation Method [46] is used to tune both the input and output fuzzy set parameters. The adaptation is processed each time the zone, corresponding to the zone model, finishes the request's service and the measured service time $\widetilde{t}_i$ is passed from the execution module to the zone model.

The parameters in the database Z_i are tuned in the following way:

$$s_{jk(i+1)} = s_{jki} + \eta_s \cdot \left(\widetilde{t}_i - \hat{t}_{ki}\right) \cdot \mu_{R_j}(e_i, f_i) \tag{6}$$

$$c_{\varphi k(i+1)} = c_{\varphi ki} + \eta_c \left(\widetilde{t}_i - \hat{t}_{ki}\right) \cdot \sum_{m=1}^{M} \left(\mu_{F_{fm}}(f_i) \sum_{l=1}^{L} \left(s_{((m-1)\cdot L+l)ki} \partial \mu_{F_{el}}(e_i) / \partial c_{\varphi ki} \right) \right) \tag{7}$$

$$d_{\gamma k(i+1)} = d_{\gamma ki} + \eta_d \left(\widetilde{t}_i - \hat{t}_{ki}\right) \cdot \sum_{l=1}^{L} \left(\mu_{F_{el}}(e_i) \sum_{m=1}^{M} \left(s_{((l-1)\cdot M+m)ki} \partial \mu_{F_{fm}}(f_i) / \partial d_{\gamma ki} \right) \right) \tag{8}$$

where η_s, η_c, η_d are adaptation ratios, $\varphi = 1, \ldots, L-1, \gamma = 1, \ldots, M-1$ [41].

The initial values of input fuzzy set function parameters C_{ki} and D_{ki} are evenly distributed over the space of executor operation. The output fuzzy set function parameters S_{ki} are initially set to zero. In this way, the estimated service times are close to zero at the beginning of the operation of the web switch. When servicing subsequent requests, the zone model tunes the parameters and the estimated service times become longer and closer to the real service times. The rate of change depends on the adaptation ratios. The preliminary experiments allow their optimal values, $\eta_s = 0.01$ and$\eta_c = \eta_d = 0.4$, to be determined. Furthermore, during the experiments, the optimal number of input fuzzy sets has been determined as $L = M = 10$, and the number of output fuzzy sets is equal to $J = L \cdot M$ [41].

The decision module is the key element of the switching section. It chooses the availability zone to service the request according to the algorithm designed especially for systems providing predictable service. The decision is made on the basis of estimated service times $\hat{t}^1_{ki}, \ldots, \hat{t}^w_{ki}, \ldots \hat{t}^W_{ki}$ in the following way:

$$z_i = \begin{cases} \min\{w : w \in \{1, \ldots, \mathrm{W}\} \wedge \Delta d'_i \leq \hat{t}^s_i\} \text{ if } n_i = n_{\max} \text{ and } \exists\limits_{w \in \{1,\ldots,S\}} \hat{t}^s_i \geq \Delta d'_i \\ w_{\min} : \hat{t}^{w_{\min}}_i = \min\{\hat{t}^w_i : w \in \{1, \ldots, W\}\} \text{ in other cases} \end{cases} \tag{9}$$

where $\Delta d'_i$ is time remaining to service the request before reaching the deadline d'_i (calculated in the switching section's service model), and $\Delta d'_i = d'_i - \tau^{(2)}_i$, $\tau^{(2)}_i$ is the moment the request enters the scheduling section. According to the proposed formula, we chose the zone with the lowest index w, which offers service time that allows the request to be serviced before the deadline d'_i. In this way, in most cases, the zone with the lowest index $w = 1$ is chosen to service the request and the zone becomes the most loaded one. Each zone with the greater index w is less loaded, and it is also ready to service more demanding requests. If the zones are overloaded and none of them is able to offer satisfying service time, then we use a classical solution known from the FNRD method [41] and the zone offering the shortest service time is chosen.

The execution module is responsible for physically sending the requests through the network to availability zones. This module also receives responses and sends them back

to clients. The second duty of the module is to collect information regarding requests being serviced in zones. The module measures service times $\widetilde{t}_i$, the number of serviced concurrently requests n_i, and a load of zones $M_i^1, \ldots, M_i^w, \ldots, M_i^W$, $i = 1, \ldots, I$. The measured values are passed to other modules in the web switch.

It is worth noticing that all of the information necessary to make decisions needed to queue requests and distribute them is available within the web switch and does not need to be fetched from zones. This is very convenient for maintenance reasons.

4. Experiments and Discussion

Research and experiments conducted up until now on web switches and brokers using presented neuro-fuzzy models showed that the devices could make high-quality decisions and cooperate well with each other in two-layer architecture. The solution proposed in this article also uses neuro-fuzzy models together with a queueing module and a new decision algorithm. The results of experiments presented in this section allow the proposed new distribution method and the design of the WCEDF web switch to be evaluated.

4.1. Testbed

The experiments were conducted in a simulation environment for web switches working in two-layer web cloud architecture. The simulation program was written in the OMNeT++ programming tool, which provides appropriate libraries and environment for conducting networking systems simulations [47].

The program was divided into the following logical modules: HTTP request generator, WCEDF web switch, FNRD web switch, web servers, and database servers. The scheme of the simulation program is presented in Figure 4.

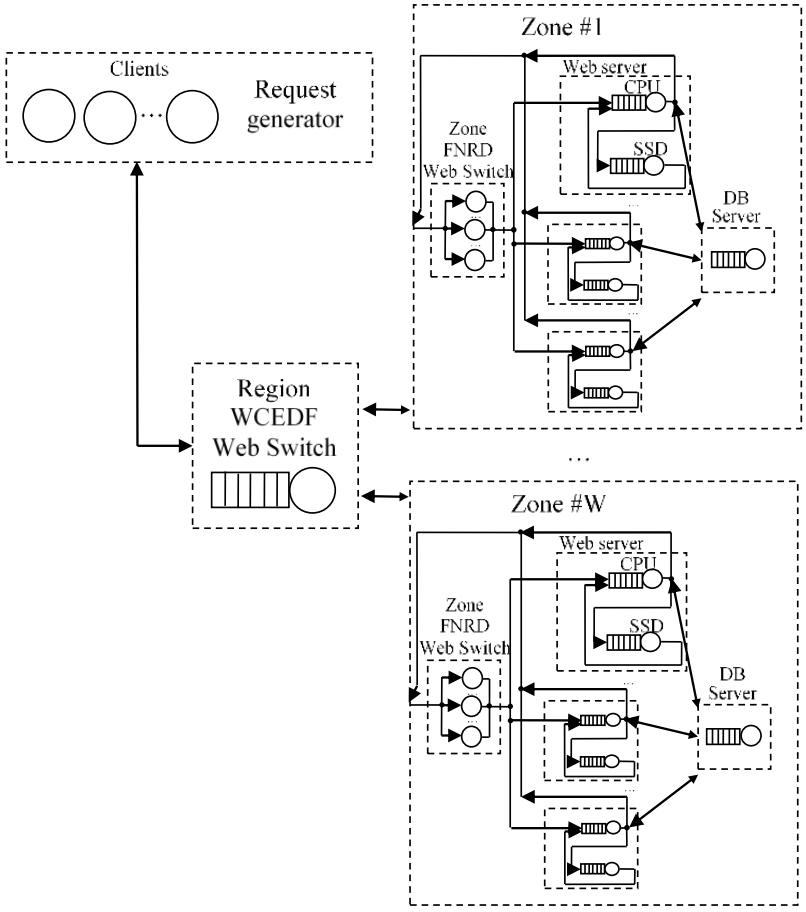


Figure 4. Simulation model.

The request generator in the simulation program was responsible for generating HTTP requests. It contains submodules simulating the behavior of web clients, which are web browsers controlled by human users. To get the web page, the simulated web client first downloads the HTML document and then opens up to six TCP (Transmission Control Protocol) connections in order to retrieve other elements such as pictures, js, or css files. The number of web pages downloaded by clients was modeled according to inverse Gaussian distribution ($\mu = 3.86$, $\lambda = 9.46$), whose parameters were designated on the basis of the research on real users' behavior [48]. The period of time between downloading subsequent pages was modeled according to the Pareto distribution ($\alpha = 1.4$, $k = 1$). After finishing downloading the web page, the client's process was deleted and a new one was invoked.

Clients were simulating downloading web pages. Parameters of the web pages (size of downloaded objects and their types and number on the page) were the same as those in the very popular site https://www.sonymusic.com (accessed on 23 March 2021) [49] running on WordPress.

The WCEDF web switch module was distributing HTTP requests among zones in the simulator. There are four request distribution strategies implemented in the module. The first three strategies are used and popular in AWS [10] web switches, while the last two can be classified as intelligent:

- Round Robin (RR)—assigns HTTP requests to subsequent zones;
- Least Load (LL)—assigns incoming HTTP requests to the zone with the lowest number of serviced HTTP requests;
- Path-based routing (P)—requested resources are assigned to designated zone/web server. In the experiments, a modification of the algorithm was used. It adapted to changing load and behaved like the LARD algorithm [50]. According to this, when the server is overloaded, the service of a given type of requests is moved to the least-loaded server;
- Web Cloud Earliest Deadline First (WCEDF)—strategy presented in this article,
- Fuzzy-Neural Request Distribution (FNRD)—the FNRD strategy is in some way similar to the WCEDF strategy. It also uses neuro-fuzzy models to estimate service time. However, the FNRD strategy does not queue the requests, and the distribution algorithm chooses a web server offering the shortest service time.

The FNRD web switch module was responsible for distributing requests inside zones among web servers. The switch used the following strategies: Round Robin, Least Load, Path-based routing, and Fuzzy-Neural Request Distribution (FNRD).

The WCEDF and FNRD web switches were modeled in the simulation as a single queue. The distribution decision service times were measured on a real server with an Intel Xeon E5-2640 v3 processor and were as follows: LL 0.0103 µs, RR 0.00625 µs, P 0.0101 µs, FNRD 0.2061 µs, and WCEDF 0.2085 µs.

The web servers in the simulation program were composed of elements causing the most significant delays in servicing requests, namely processor and SSD drive. Both of the resources were modeled as a single queue. Also, the main memory was modeled. It acted as cache memory for the file system and used the least-recently-used policy. The service times for both processor and SSD drive were obtained in experiments with a website using WordPress and running on a server with an Intel Core i7 7800X CPU, a Samsung SSD 850 EVO driver, and 32 GB RAM.

The database server was modeled as a single queue. The service times for the server were obtained for the same hardware as the web server was using.

The experiments were conducted for 12 web servers working in three zones (four web servers in each zone). This configuration is the most efficient for web cloud system with two-layer architecture controlled by cooperating FNRD web switches [43]. As was mentioned earlier, the WCEDF method uses the WCEDF web switch in the region and FNRD web switches in availability zones. In the presented experiments, the WCEDF method was compared with the web cloud-based system which uses on both layers the same distribution strategies—FNRD (configuration marked FNFN), Least Load (configuration marked LL),

Round Robin (configuration marked RR), and Path-based routing (configuration marked P). Previously published results of experiments revealed that the configuration FNFN is very efficient and the obtained service times can be even two times shorter than for other popular distribution strategies used in two-layer configurations [42,43].

4.2. Experiments

To evaluate the experiments' results, different metrics were used—mean service time, 95 percentile, and 98 percentile of service time. One of the WCEDF method's main aims is to service the requests in a time shorter than the required value t_{max}. The new proposed approach requires the application of an adequate quality factor, allowing evaluation of the most important features of the proposed method. The mean value of satisfaction was chosen. It is often used to evaluate the effect of the work of real-time soft systems.

The value of satisfaction was calculated as follows:

$$Satisfaction(\bar{t}) = \begin{cases} 1 \text{ if } \widetilde{t}_i < t^s_{\max} \\ 0 \text{ if } \widetilde{t}_i > t^h_{\max} \\ 1 - \frac{t^h_{\max} - \widetilde{t}_i}{t^h_{\max} - t^s_{\max}} \text{ in other cases} \end{cases} \quad (10)$$

where t^s_{max} is the "soft time" after which satisfaction starts decreasing to 0 and t^h_{max} is the "hard time" after which the user will leave the page. Figure 5 presents the satisfaction in the service time function. In experiments, it was assumed that $t^s_{max} = t_{max}$ and $t^h_{max} = 2 \cdot t_{max}$.

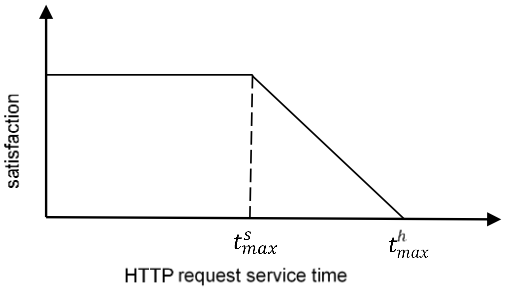
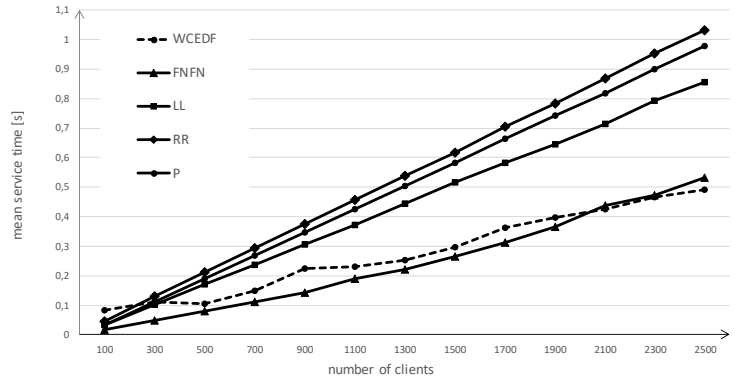


Figure 5. Satisfaction function.

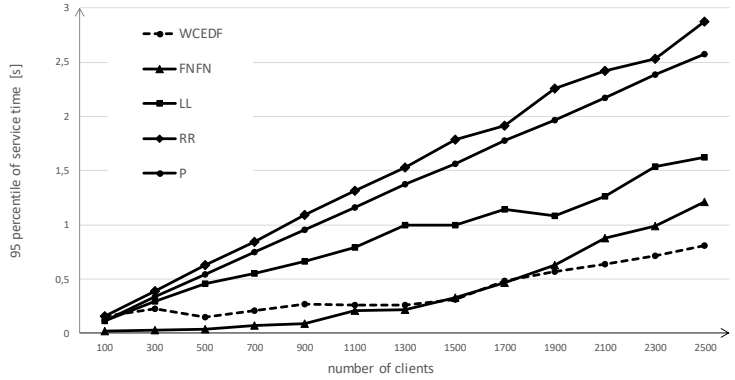
Each experiment was conducted for different loads, measured as the number of simulated clients. The number of clients varied from 100 to 2500. Also, during a single experiment, 40 million HTTP requests were served. The warming phase was taking 10 million requests, and for 30 million, the service time was measured.

The experiments were conducted for four adopted values of t_{max}: 0.5 s, 0.75 s, 1 s, and 2 s. Figure 6 presents the results of experiments for $t_{max} = 0.5$ s .

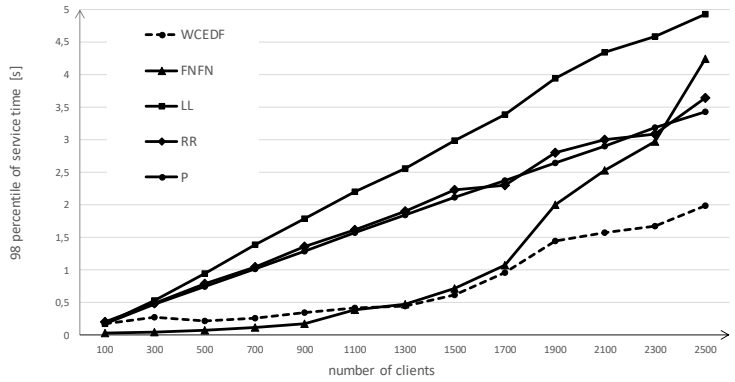
As it can be seen in Figure 6, when the load of the system was low, the mean response time, the 95 percentile, and the 98 percentile were the lowest for the FNFN configuration. The FNFN configuration was very effective in minimizing HTTP request service times. When the load increased, the 95 and the 98 percentile became lower for the WCEDF strategy. This was especially noticeable in the case of the 98 percentile. The WCEDF strategy was able to keep the service time within constraints ($\widetilde{t}_i < t_{max}$) when the load was low and keep the time as low as possible when the load increased. The FNFN configuration in heavier loads had lower mean service times. However, there are also requests with long service times, which is noticeable on the 98 percentile graph. The same phenomenon can be seen on the cumulative distribution of the service time graph (Figure 6d), picturing results for the load of 2300 clients. About 80% of requests were serviced by the FNFN cloud in a time shorter than for the WCEDF method. However, the longest 20% of service times also belonged to the FNFN configuration, which indicates that the standard deviation was greater for the FNFN than for the WCEDF. The WCEDF method, in most cases, offered a longer mean service time, but the flow of requests was much more predictable, structured and in the end, there are not requests being serviced for a longer time.



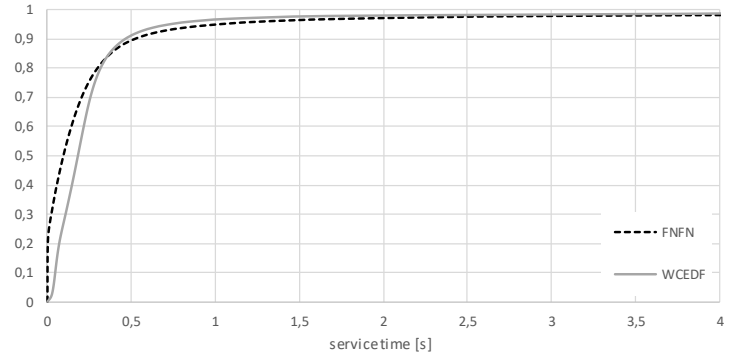
(**a**)



(**b**)



(**c**)

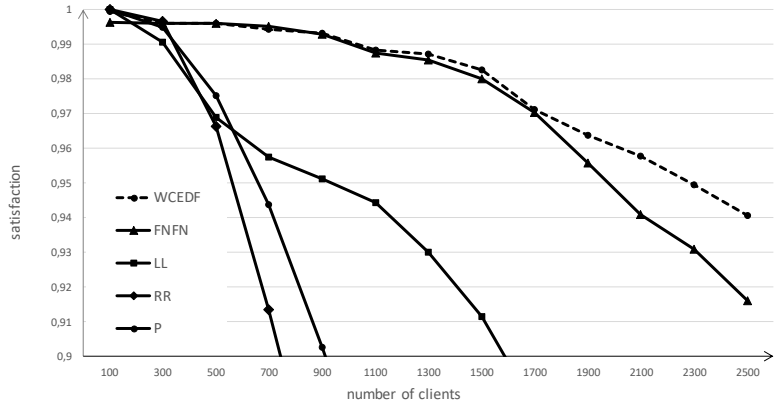


(**d**)

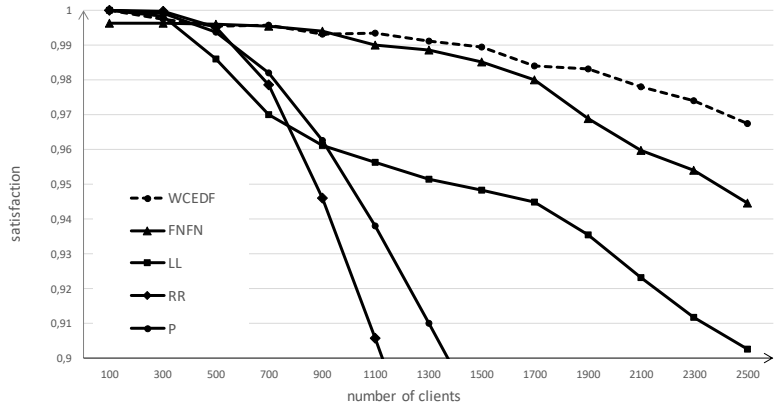
Figure 6. Results of experiments for $t_{max} = 0.5$ s. (**a**) Mean service time in load function (number of clients), (**b**) the 95 percentile of service time in load function, (**c**) the 98 percentile of service time in load function, and (**d**) the cumulative distribution of service time for a number of clients equal to 2300.

Results obtained for classical configurations (LL, RR, and P) were much worse and differed significantly from intelligent solutions.

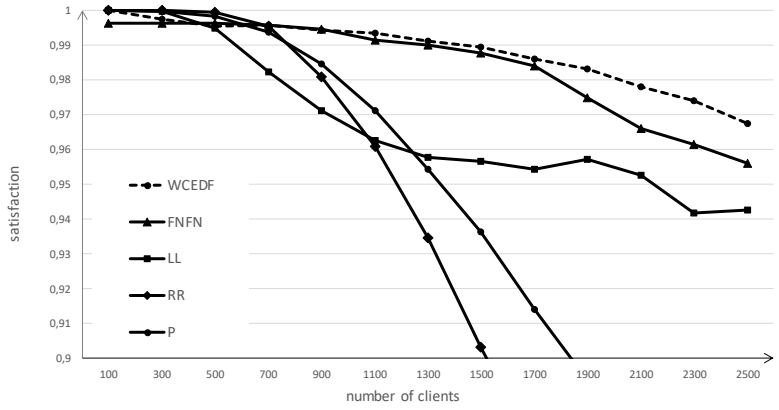
Figure 7 presents graphs of satisfaction for different obtained times t_{max}. As can be seen, the satisfaction was the highest for the WCEDF method, especially when the load was heavy. When the load was low, the measured satisfaction was high and close to 1 for each of the distribution strategies and methods in each experiment. The strategies LL, RR, and P, obtained poor results for a heavier load, especially for shorter t_{max} times equal to 0.5 s and 0.75 s. The satisfaction for the WCEDF method was close to other distribution strategies in experiments with $t_{max} = 2$ s. In this case, obtained results were good because the constraints were not narrow and easy to achieve even for not-intelligent, classical strategies.



(**a**)

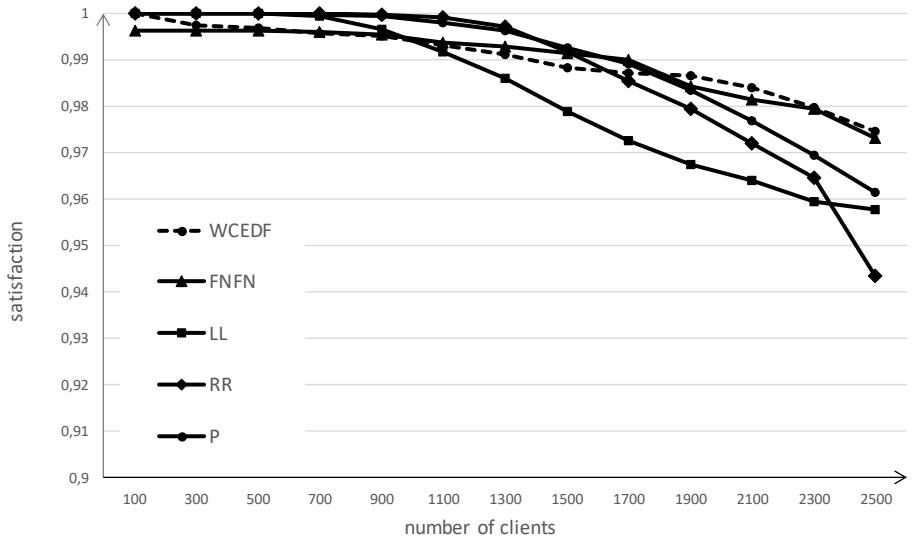


(**b**)



(**c**)

Figure 7. *Cont.*



(**d**)

Figure 7. Satisfaction in load function for different t_{max} times. (**a**) $t^s_{max} = 0.5$ s, $t^h_{max} = 1$ s; (**b**) $t^s_{max} = 0.75$ s, $t^h_{max} = 1.5$ s; (**c**) $t^s_{max} = 1$ s, $t^h_{max} = 2$ s; and (**d**) $t^s_{max} = 2$ s, $t^h_{max} = 4$ s.

Summing up the results, the WCEDF method meets the expectations that were set during its design. The web cloud controlled by the WCEDF method was highly efficient, and it obtained high satisfaction even when the constraints were narrow and the load was heavy. The flow of requests was much more predictable and structured in the case of the WCEDF than for the FNFN configuration.

5. Conclusions and Directions for Future Research

The WCEDF request distribution method used in cloud-based web systems comprises two main elements—the WCEDF web switch and the FNRD web switches. The WCEDF web switch allows to the quality of service to be maintained and makes the flow of HTTP requests inside the cloud much more structured and shaped. The FNRD web switches allow requests to be distributed inside the availability zones almost in a time-optimal way [42].

The novelty of the method is a combination of:

- application of the queue of requests in the front of the web system with the method of determining the deadline service times;
- distribution algorithm, designed especially for the proposed solution;
- application of FNRD web switches in the second layer of the decision system.

The application of neuro-fuzzy models used to estimate the service times proved to be accurate.

Promising results obtained for the WCEDF method indicate that the WCEDF switch most likely cooperates with FNRD switches, similarly to results obtained for the FNRD switches located in the region and zones [43]. If this statement is true, it will be confirmed in further publications.

The WCEDF method makes the service of the web cloud very efficient and, at the same time, more predictable than is the case with other methods providing the best-effort quality of service. Using this method, however, does not guarantee that the web system can maintain the required quality. The useful features of the WCEDF method encourage the development of a new version of the method that would guarantee the quality of service in cloud-based web systems.

6. Summary

A new HTTP request distribution method for cloud-based web systems was presented in the article. The proposed WCEDF method is applicable in two-layer architectures of cloud-computing web systems. It allows HTTP requests to be serviced in time constraints within the established quality of the web service. The main element used in the method is the WCEDF web switch. It queues incoming requests taking into account the time constraints and distributes the requests in a way that keeps selected parts of the cloud idle and ready to service time-consuming requests faster.

To evaluate the WCEDF method, a simulation environment was implemented. The simulator imitated web clients' behavior, as well as the work of a web switches and both the web and the database servers.

The results of the experiments show that the WCEDF method met the expectations. The web cloud controlled by the WCEDF method was efficient and able to service requests within time limits, even when the constraints were narrow and the load heavy. The value of satisfaction for the new method was highest in each experiment than any other method tested. The flow of requests inside the cloud was much more predictable and structured in the case of the WCEDF than for other distribution strategies. Obtained results indicate that the WCEDF web switch most likely co-operates with intelligent FNRD switches.

The research results indicate that the new solution is important and that further research is needed and development should be continued.

Funding: This research received no external funding.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Global Digital Population as of January 2021 (in Millions). Available online: https://www.statista.com/statistics/617136/digital-population-worldwide/ (accessed on 18 February 2021).
2. Costello, K.; Rimol, M. Gartner Forecasts Worldwide Public Cloud End-User Spending to Grow 18% in 2021. Gartner. Available online: https://www.gartner.com/en/newsroom/press-releases/2020-11-17-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-grow-18-percent-in-2021 (accessed on 18 February 2021).
3. Research and Markets, Raport, Global Forecast to 2025. Available online: https://www.researchandmarkets.com/reports/5136796/cloud-computing-market-by-service-model (accessed on 18 February 2021).
4. Lee, B.T.G.; Patt, R.; JeffVoas, C. DRAFT Cloud Computing Synopsis and Recommendations. 2011. Available online: http://csrc.nist.gov/publications/nistpubs/800-146/sp800-146.pdf (accessed on 18 February 2021).
5. Puthal, D. Cloud Computing Features, Issues, and Challenges: A Big Picture. In Proceedings of the 2015 International Conference on Computational Intelligence and Networks (CINE), Odisha, India, 12–13 January 2015; pp. 116–123.
6. Patiniotakis, I.; Verginadis, Y.; Mentzas, G. PuLSaR: Preference-based cloud service selection for cloud service brokers. *J. Internet Serv. Appl.* **2015**. [CrossRef]
7. Katyal, M.; Mishra, A. A Comparative Study of Load Balancing Algorithms in Cloud Computing Environment. *Int. J. Distrib. Cloud Comput.* **2013**, *1*, 806–810.
8. Montes, J.; Sanchez, A.; Memishi, B.; Pérez, M.S.; Antoniou, G. GMonE: A complete approach to cloud monitoring. *Future Gener. Comput. Syst.* **2013**, *29*, 2026–2040. [CrossRef]
9. Montes, J.; Sánchez, A.; Pérez, M.S. Riding Out the Storm: How to Deal with the Complexity of Grid and Cloud Management. *J. Grid Comput.* **2012**, *10*, 349–366. [CrossRef]
10. AWS documentation, How Elastic Load Balancing Works. Available online: https://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/how-elastic-load-balancing-works.html (accessed on 18 February 2021).
11. McCabe, D.J. *Network Analysis, Architecture, and Design*; Morgan Kaufmann: Boston, MA, USA, 2007.
12. Alakeel, A. A guide to dynamic load balancing in distributed computer systems. *Int. J. Comput. Sci. Inf. Secur.* **2010**, *10*, 153–160.
13. Rimal, B.P. Architectural requirements for cloud computing systems: An enterprise cloud approach. *J. Grid Comput.* **2011**, *9*, 3–26. [CrossRef]
14. Ponce, L.M.; dos Santos, W.; Meira, W.; Guedes, D.; Lezzi, D.; Badia, R.M. Upgrading a high performance computing environment for massive data processing. *J. Internet Serv. Appl.* **2019**, *10*. [CrossRef]
15. Nuaimi, K.A. A Survey of Load Balancing in Cloud Computing: Challenges and Algorithms. In Proceedings of the 2012 Second Symposium on Network Cloud Computing and Applications (NCCA), London, UK, 3–4 December 2012. [CrossRef]
16. Zenon, C.; Venkatesh, M.; Shahrzad, A. Availability and Load Balancing in Cloud Computing. In Proceedings of the International Conference on Computer and Soft modeling IPCSI, Singapore, 23 February 2011.

17. Campelo, R.A.; Casanova, M.A.; Guedes, D.O. A brief survey on replica consistency in cloud environments. *J. Internet Serv. Appl.* **2020**, *11*. [CrossRef]
18. Afzal, S.; Kavitha, G. Load balancing in cloud computing—A hierarchical taxonomical classification. *J. Cloud Comp.* **2019**, *8*. [CrossRef]
19. Rafique, A.; Van Landuyt, D.; Truyen, E. SCOPE: Self-adaptive and policy-based data management middleware for federated clouds. *J. Internet Serv. Appl.* **2019**, *10*, 1–19. [CrossRef]
20. Remesh, B.K.R.; Samuel, P. *Enhanced Bee Colony Algorithm for Efficient Load Balancing and Scheduling in Cloud. Advances in Intelligent Systems and Computing*; Springer: Cham, Switzerland, 2016; pp. 67–78.
21. Crovella, M.; Bestavros, A. Self-similarity in World Wide Web traffic: Evidence and possible causes. *IEEE/ACM Trans. Netw.* **1997**, *5*, 835–846. [CrossRef]
22. Domańska, J.; Domański, A.; Czachórski, T. The Influence of Traffic Self-Similarity on QoS Mechanisms. In Proceedings of the SAINT 2005 Workshop, Trento, Italy, 20 February 2005.
23. Suchacka, G.; Dembczak, A. *Verification of Web Traffic Burstiness and Self-Similarity for Multiple Online Stores. Advances in Intelligent Systems and Computing*; Springer International Publishing: Heidelberg, 2017; Volume 655, pp. 305–314.
24. Suraj, P.; Wu, L.; Mayura Guru, S.; Buyya, R. A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments. In Proceedings of the 24th IEEE International Conference on Advanced In-formation Networking and Applications, Perth, WA, Australia, 20–23 April 2010. [CrossRef]
25. Farid, M.; Latip, R.; Hussin, M.; Abdul Hamid, N.A.W. A Survey on QoS Requirements Based on Particle Swarm Optimization Scheduling Techniques for Workflow Scheduling in Cloud Computing. *Symmetry* **2020**, *12*, 551. [CrossRef]
26. Sharifian, S.; Akbari, M.K.; Motamedi, S.A. An Intelligence Layer-7 Switch for Web Server Clusters. In Proceedings of the 3rd International Conference: Sciences of Electronic, Technologies of Information and Telecommunications SETIT, Susa, Tunisia, 27–31 March 2005; pp. 5–20.
27. Bryniarska, A. The n-Pythagorean Fuzzy Sets. *Symmetry* **2020**, *12*, 1772. [CrossRef]
28. Abdelzaher, T.F.; Shin, K.G.; Bhatti, N. Performance Guarantees for Web Server End-Systems. A Control-Theoretical Approach. *IEEE Trans. Parallel Distrib. Syst.* **2002**, *13*, 80–96. [CrossRef]
29. Blanquer, J.M.; Batchelli, A.; Schauser, K.; Wolski, R. Quorum: Flexible quality of service for internet services. In *Proceedings. of the 2nd Conference on Symposium on Networked Systems Design & Implementation*; USENIX Association: Berkeley, CA, USA, 2005; Volume 2, pp. 159–174.
30. Borzemski, L.; Zatwarnicki, K. CDNs with Global Adaptive Request Distribution. In Proceedings of the 12th International Conference on Knowledge-Based & Intelligent Information & Engineering Systems, Zagreb, Croatia, 3–5 September 2008; Springer: Heidelberg, Germany, 2008; pp. 117–124.
31. Harchol-Balter, M.; Schroeder, B.; Bansal, N.; Agrawal, M. Size-based scheduling to improve web performance. *ACM Trans. Comput. Syst.* **2003**, *21*, 207–233. [CrossRef]
32. Kamra, A.; Misra, V.; Nahum, E. A Self Tubing Controller for Managing the Performance of 3-Tiered Websites. In Proceedings of the 12th International Workshop Quality of Service, Montreal, QC, Canada, 7–9 June 2004; pp. 47–56.
33. Wei, J.; Xu, C.-Z. QoS: Provisioning of client-perceived end-to-end QoS guarantees in Web servers. *IEEE Trans. Comput.* **2006**, *55*, 1543–1556.
34. Zatwarnicki, K. Adaptive Scheduling System Guaranteeing Web Page Response Times. Computational Collective Intelligence; Lecture Notes in Artificial Intelligence. In Proceedings of the 4th International Conference, ICCI, Ho Chi Mihn, Vietnam, 28–30 November 2012; Springer: Berlin/Heidelberg, Germany, 2012.
35. Zatwarnicki, K.; Płatek, M.; Zatwarnicka, A. A Cluster-Based Quality Aware Web System. In *ISAT 2015–Part II. Advances in Intelligent Systems and Computing*; Springer: Cham, Switzerland, 2015; Volume 430.
36. Zatwarnicki, K. Operation of Cluster-Based Web System Guaranteeing Web Page Response Time. In *Computational Collective Intelligence. Technologies and Applications*; Lecture Notes in Computer Science (ICCCI 2013); Bădică, C., Nguyen, N.T., Brezovan, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; Volume 8083.
37. Serrano, D. Towards QoS-Oriented SLA Guarantees for Online Cloud Services. In Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, Delft, The Netherlands, 13–16 May 2013; pp. 50–57.
38. Chhetri, M.B.; Vo, Q.; Kowalczyk, R. Policy-Based Automation of SLA Establishment for Cloud Computing Services. In Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Ottawa, ON, Canada, 13–16 May 2012; pp. 164–171.
39. Rao, J.; Wei, Y.; Gong, J.; Xu, C. QoS Guarantees and Service Differentiation for Dynamic Cloud Applications. *IEEE Trans. Netw.Serv. Manag.* **2013**, *10*, 43–55.
40. Jing, W.; Zhao, C.; Miao, Q.; Song, H.; Chen, G. QoS-DPSO: QoS-aware Task Scheduling for Cloud Computing System. *J. Netw. Syst. Manag.* **2021**, *29*. [CrossRef]
41. Zatwarnicki, K. Adaptive control of cluster-based web systems using neuro-fuzzy models. *Int. J. Appl. Math. Comput. Sci.* **2012**, *22*, 365–377. [CrossRef]
42. Zatwarnicki, K.; Zatwarnicka, A. Application of an Intelligent Request Distribution Broker in Two-Layer Cloud-Based Web System. In *Computational Collective Intelligence*; Lecture Notes in Computer Science (ICCCI 2019); Springer: Cham, Switzerland, 2019; Volume 11684.

43. Zatwarnicki, K.; Zatwarnicka, A. An Architecture of a Two-Layer Cloud-Based Web System Using a Fuzzy-Neural Request Distribution. *Vietnam J. Comput. Sci.* **2020**, *7*. [CrossRef]
44. Zatwarnicki, K. Two-level fuzzy-neural load distribution strategy in cloud-based web system. *J. Cloud Comput.* **2020**, *9*, 1–11. [CrossRef]
45. Mamdani, E.H. Application of fuzzy logic to approximate reasoning using linguistic synthesis. *IEEE Trans. Comput.* **1977**, *C-26*, 1182–1191. [CrossRef]
46. Rumelhart, D.; Hinton, G.; Williams, R. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536. [CrossRef]
47. OMNeT++ Discrete Event Simulator. Available online: https://www.omnetpp.org/ (accessed on 18 February 2021).
48. Cao, J.; Cleveland, S.W.; Gao, Y.; Jeffay, K.; Smith, F.D.; Weigle, M.C. Stochastic models for generating synthetic HTTP source traffic. In Proceedings of the Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies. INFOCOM, Hong-Kong, China, 7–11 March 2004; pp. 1547–1558.
49. Sony Music, Main Page. Available online: https://www.sonymusic.com/ (accessed on 9 March 2020).
50. Pai, V.S.; Aron, M.; Banga, G.; Svendsen, M.; Druschel, P.; Zwaenepoel, W.; Nahum, E. Locality-aware request distribution in cluster-based network servers. In Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, USA, 10 October 1998; pp. 205–216.