Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2020) 28: 2718 – 2734

doi:10.3906/elk-2001-33

A fuzzy neural network for web service selection aimed at dynamic software rejuvenation

Kimia REZAEI KALANTARI[1], **Ali EBRAHIMNEJAD**[2,*], **Homayun MOTAMENI**[3]
[1]Department of Computer Engineering, Babol Branch, Islamic Azad University, Babol, Iran
[2]Department of Mathematics, Qaemshahr Branch, Islamic Azad University, Qaemshahr, Iran
[3]Department of Computer Engineering, Sari Branch, Islamic Azad University, Sari, Iran

Received: 06.01.2020 • **Accepted/Published Online:** 04.05.2020 • **Final Version:** 25.09.2020

Abstract: Software rejuvenation is an effective technique to counteract software aging in continuously running applications such as web service-based systems. In these systems, web services are allocated based on the requirements of receivers and the facilities of servers. One of the challenges while assigning web services is how to select appropriate server to reduce faults. In this paper, we propose dynamic software rejuvenation as a proactive fault-tolerance technique based on the neural fuzzy system. While considering a threshold for the rejuvenation of each web service, we completed the training based on the features of the service providers as well as the requirements of the receivers. The results of simulations revealed that our strategy can decrease the failure rate in comparison with state-of-the-art strategies and improve system availability in web services.

Key words: Software aging, software rejuvenation, fuzzy neural network, web service

1. Introduction

With the rapid growth of Internet technology, the reliability of web applications has become the main concern. The important issue in providing reliability involves the performance and availability of web applications or servers. Software aging can be defined as a growing weakening of the internal state of the software. When software applications are run continuously for long periods, the corresponding software processes lead to aging. The reasons for processing aging are lack of memory and nonrelease file locks, data corruption in the operating environment, and so on. Aging will affect the performance of a device and ultimately cause the application to fail. Software rejuvenation is one of the dynamic fault tolerance techniques used to solve the software aging problem.

Software rejuvenation does not solve the root cause of software aging. Therefore, software rejuvenation will be run cyclically at a predetermined or scheduled time to maintain software system firmness.

The two main approaches adopted to determine the timing for software rejuvenation are time-based and inspection-based methods. Time-based approaches determine the optimal rejuvenation time by analyzing the relationship between the state of the software system and the assumption of the distribution of system faults. The inspection-based policy is usually pursued through continuous monitoring at runtime and fault behavior using statistical analysis of data in an effort to estimate the intervals of software rejuvenation. However, the relatively complex computing process generally results in greater resource fatigue in the software system. Therefore, frequent inspections at runtime in a software system cause the system to suffer certain amounts of

*Correspondence: a.ebrahimnejad@qaemiau.ac.ir

overheads in terms of faults and costs. This should be considered together with the fault and cost of failure in an attempt to obtain maximum efficiency [1].

Web services are fundamental software artifacts for building service-oriented applications [2]. The web service architecture standard, as shown in Figure 1, is made up of three parts, namely web service providers, web service brokers, and web service requesters. In the standard web service architecture, the web service provider is used to host web services and applications. The web service broker typically makes the web service publicly available, while the web service requester plays a role in linking the web service operation with the application environment in this architecture [3, 4].

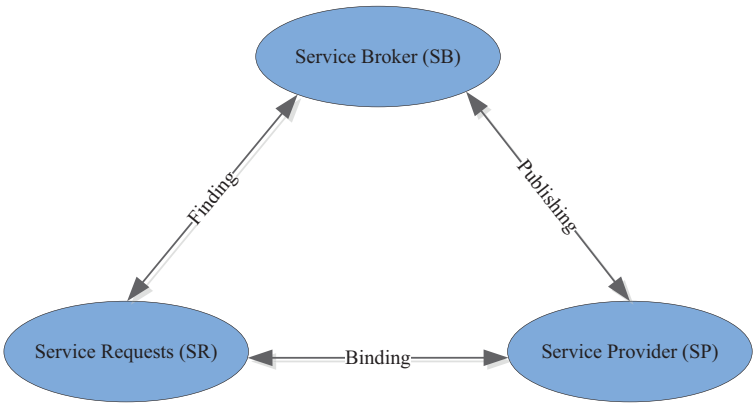

Figure 1. Web service architecture [4].

Since software rejuvenation depends on different criteria that are often qualitative and vary under different circumstances, we used a fuzzy neural network in the present paper. A fuzzy neural network is used because the proportion of the impact of these criteria should be determined by several stages of training. Furthermore, the system idle time during the rejuvenation routine was minimized by two or more versions instead of using one version per service. Of course, this was achieved by determining the number of optimal versions per service such that, firstly, the overhead of using multiple versions is minimized, and, secondly, the system idle time for each service is minimized.

The remainder of this paper is organized as follows. Section 2 describes previous studies on rejuvenation. Then our proposed method is given in Section 3. In Section 4, we describe the experiments and tests. Finally, Section 5 outlines the results.

2. Literature review

Software aging can be defined as a gradual weakening of the internal state of the software. Software rejuvenation is one of the new, dynamic techniques of fault tolerance used to solve the software aging problem. Old techniques of software fault tolerance are reactive; that is, after an occurrence of a fault event, they engage in repairing it and give rise to a great deal of repair overhead. While software rejuvenation is an active technique preventing fault before its occurrence or slowing down its occurrence, the cost of such overhead is less than that of repairing the fault.

Software rejuvenation can be found in a wide variety of software systems, leading to gradual degradation of performance in terms of time/load and ultimately system failures. In order to minimize the effects of aging and prevent substantial losses incurred by system failure, software rejuvenation can be carried out vigorously to restore the system's performance in web service-based systems.

Levitin et al. [5] aimed to optimize a state-based rejuvenation policy in an attempt to maximize the probability of task completion (PTC) for software systems inspected on a regular basis. Their new method facilitated the optimal selection of the state-based rejuvenation decision function under any specific inspection interval. In [6], the optimal service selection was extremely intricate, seriously challenging the smart cloud storage management. Fuzzy logic can help achieve the selection of the most ideal response to customer requirements in terms of quality of service (QoS) and costs.

It will be useful to predict the number of software faults over the course of development. Liu et al. [7] proposed a new hybrid aging prediction model that incorporates well the autoregressive integrated moving average (ARIMA) and long short-term memory (LSTM) models for more appropriate fitting of the linear pattern as well as mining the nonlinear relationship in the time series of computing resource usage data for cloud services. Their model was not, however, perfect, since it assumed that the linear and nonlinear components of the sequence are merely additive. Rathore and Kumar [8] set out a technique to compare the capabilities of six fault prediction methods for predicting the potential number of faults.

Kalantari et al. [1] investigated an inspection-based method for preventive maintenance in a software operating system that was periodically inspected at specific intervals using Markov's model. The optimum inspection interval has been obtained to minimize faults and costs. The function of this method is acceptable in simple software; however, it needs to be improved in complex software. Kalantari et al. [9] studied dynamic software rejuvenation based on a combination of ant colony optimization (ACO) and gravitational emulation local search (GELS). Despite its efficiency, the ACOGELS-based approach is not good for dealing with large-scale combinatorial problems because of its time complexity.

Reliability is an essential requirement for web service-based systems, which must provide services with high availability, high fault tolerance, and dynamic deployment capabilities [10, 11]. Dantas et al. [12] presented availability models for private cloud architectures based on the Eucalyptus platform. Cloud computing systems basically provide access to large data sets and computing resources. However, one of the main reasons behind the sluggish growth of cloud computing is security [13]. Araujo et al. [14] proposed a software framework mainly used as a service for private cloud computing. Their research focused on the effectiveness of software within the framework of Eucalyptus in terms of system workloads and high demands for remote storage and virtual models. They also proposed an approach that offers time series analysis for rejuvenation scheduling in an attempt to reduce the fault time by predicting the appropriate time for rejuvenation. Meenakshi et al. [15] suggested another approach that allocated resources with minimal waste in cloud computing based on sending requests to the request tuner.

With the advent of peer-to-peer systems, there has been more emphasis on the need to discover and exploit diverse groups, multiple features, and distributed and dynamic resources. Bai et al. [16] quantitatively analyzed software rejuvenation techniques from service provider and user views in a virtualized system deploying virtual machine monitor (VMM) reboot and live virtual machine migration techniques for rejuvenation. They constructed an analytical model by using a semi-Markov process (SMP) and derive formulas for calculating application service (AS) availability and job completion time.

Guo et al. [17] used discrete web services to measure software aging failures and software rejuvenation strategies based on the belief that discrete web services provide a loose connection feature to the server. In their research, the method used several linear regression methods to calculate the aging of individual web services.

Mooij et al. [18] showed that semiautomatic software rejuvenation in industrial operations is feasible and cost effective. They used specific domain models involving abstract implementation details while applying a practical combination of manual and automated techniques. In Levitin et al. [19], software rejuvenation coupled with check-pointing occasionally saved the system state on a reliable storage so that the mission task could be resumed from the last save checkpoint. The limitation of that work was the assumption of full state independent rejuvenation, leading to a degraded system.

In general, most of the previous methods proposing to optimize software rejuvenation in various software systems have shortcomings. In the present study, we intended to overcome most of these shortcomings, a few of which are summarized below:

1. Most of the methods proposed for optimizing the software rejuvenation in software systems are one-dimensional; that is, only one aspect of software rejuvenation has been considered. For example, Meng et al. [20] considered their nonsequential inspection intervals to optimize software rejuvenation. They did not, however, consider other effective dimensions in software rejuvenation. Machida and Miyoshi [21] considered an optimal stopping problem for software rejuvenation. However, they did not investigate sequences with different time intervals.

2. Most of the methods proposed for software rejuvenation have static management in software systems; that is, they consider a static structure for software rejuvenation. However, they did not adopt different methods for software rejuvenation at the time of running the software under different circumstances [22, 23].

3. Many software rejuvenation methods stop the software system at the time of software rejuvenation; that is, user requests are not answered at the time of rejuvenation. This policy of the implementation mechanism simplifies the procedure, but reduces the system's availability [24, 25].

4. Some software rejuvenation methods include a version for the software. In addition, the fault tolerance is low in the systems. If an error occurs at the time of rejuvenation of this single version of the software, the system availability is reduced [26, 27].

The fault tolerance for web services is considered an important research topic in choosing a web service. The existing fault tolerance web service models have certain limitations. For example, various important sources of fault tolerance do not integrate, or do not focus on various attributes of service quality such as performance and accessibility so as to satisfy user requirements.

In an effort to overcome these constraints, Nguyen et al. [28] proposed a Bayesian fault tolerance model for web services that includes several criteria such as reliability, availability, and safety during the evaluation of fault tolerance of web services. Moreover, their method has largely eliminated the problems of user preferences and fault tolerance based on several parameters of service quality. Wu et al. [29] employed replication strategies to achieve the trade-off between fault tolerance effect and fault tolerance loss. They proposed an A*-based heuristic alternative path searching (AHPS) algorithm to find suitable replacement schemes for composite services.

Liu et al. [4] introduced a fault tolerance evaluation model with reasonable correction logic. The method improved the SOA framework by incorporating a trust management module. Then the trust evaluation model based on amendatory subjective logic was proposed by viewing trust relationships of service entities in web service networks as a small-world network. In order to overcome internal software rejuvenation, Torquato et al.

[30] presented availability models based on stochastic Petri nets to evaluate two VM live migration approaches. The method was based on the redundancy schemes called warm-standby and cold-standby. Wahab et al. [31] explored the categorization of different types of web services including single, mixed, and associative web services in their review article. They further outlined the effective criteria for evaluating fault tolerance in each of the aforementioned web services.

3. The proposed method

This section presents a new method for solving the shortcomings of previous works mentioned in Section 2.

For the first problem in software rejuvenation, we covered server reliability and time threshold based on requested web services. The proposed method has dynamic rejuvenation by measuring the server's reliability and time threshold factors in the middle of software running to solve the second shortcoming.

In the proposed method, the systems requiring rejuvenation do not receive any requests from users but some alternative systems are available to respond to users' requests. This in turn prevents the system from stopping. In the suggested method for the last shortcoming in previous related works, several web service providers were considered for responding to receivers. In this case, when one server stops, the whole system does not halt, thus increasing accessibility in rejuvenation time. In the present study, we adopted single web services, for which we set criteria such as runtime, threshold of reliability, and maximum deadline on the receiver side, which requests the web service. On the server side, we considered criteria such as reliability, failure rate, and rejuvenation rate. We calculated the mean time to failure (MTTF) of all web services in order to calculate the inspection periods for rejuvenation. We also set the inspection period to be the lowest MTTF for all web services.

In the present research, the servers provide web services based on receivers' requests for them under two conditions. Firstly, they need to have minimal reliability requirements for those services. Secondly, there should as far as possible be maximum matching between the web service requested and the web service allocated. The reliability of web services in the present research was considered exponentially according to Eq. (1), where λ is the rate of failure.

$$R(t) = e^{-\lambda t} \tag{1}$$

If allocation happens exactly based on the request for web services, one of the problems in fault-tolerance systems is the decreasing availability for some infeasible requested services. In order to solve this shortcoming, the new method used fuzzy allocation to provide flexible implementation facility between the requested and allocated web services. The main problem in using the fuzzy method alone, however, is how to assign thresholds between several fuzzy sets in fuzzy variable matched request web service and service allocated. In order to achieve maximum allocation and resolve the deficiencies in the fuzzy method, the proposed method used a fuzzy neural network in which the system at training level can determine the threshold between several fuzzy sets for implementation.

The general structure of the new algorithm is displayed in Table 1. In this structure, the first line defines the number of stages in a scenario that includes a set of requested web services. At each stage of the run, the appropriate server must be found for each of the requested web services. RWS_Num denotes the total number of web services requested. For each web service requested at any stage, if the web service has failed or expired, it will no longer need to be reviewed (lines 3 to 5). In line 6, we find the appropriate server for the current web service using a fuzzy system with a structure described later.

Table 1. The general structure of the newly proposed algorithm.

```
1. For step=1:MaxExT(RWSA)
2.    for i=1: RWSNum
3.      if (RWSA(i).Fail)‖(RWSA(i).ExT==0)
4.       continue with next RWS
5.      end
6.      find nero fuzzy suitable web service(RWSA(i), WSA);
7.      if (not find )
8.       consider change attributes for RWSA(i)
9.     else
10.        Run by Best find suitable web service
11.        consider change attributes for RWSA(i) and Best find suitable web service
12.      end
13.   end
14.   for i=1:WSNum
15.     if Status of WSA(i) is rejuvenation
16.       run rejuvenation for WSA(i)
17.     end
18.     if Reliability of WSA(i) < threshold
19.       perform WSA(i) in rejuvenation Status
20.     end
21.     consider change attributes for WSA(i) for next step
22.   end
23. end
```

If a suitable server is not found for the current web service, the necessary changes should be made to the current web service. In fact, one should be subtracted from the time period, and providing it does not have any time left, it should fail. If suitable servers were found in the subsequent levels and the allocation happened, the slight service could not run in the acceptable time (lines 8 and 9). Upon finding appropriate servers, in order to run the current web service, this service will be run with the most appropriate server. In addition, changes will be made to the features of the current web service (lines 10 and 11).

At each stage, after making decisions for all web services, changes should be made to the receivers for the next stages (lines 14 to 22). If the current condition is server rejuvenation, the rejuvenation procedure should continue for it. Upon completion of the rejuvenation process, it should be switched to the ready mode (lines 15 to 17). The threshold value depends on the structure of the web service provider and is based on the value of the server's reliability that provides the desired level of service in this scenario. The threshold in cloud computing for data centers is usually defined by the manufacturers. Different thresholds were considered in the suggested method based on the server's structures. If the reliability of a server is lower than the threshold, the rejuvenation procedure should be performed on it for the next stages. As a result, its mode should be changed to rejuvenation (lines 18 to 20). Finally, at the end of each stage, the last changes should be applied to each service for later stages (line 21).

In the structure of the newly proposed algorithm in Table 1, the most important step is choosing appropriate servers. In the present article, a fuzzy neural network was used since considering a 100% web service compliance with the receiver's request may bring about numerous limitations and may result in many failures in addressing the requests. Figure 2 shows the structure of the neural fuzzy system to find the appropriate server.

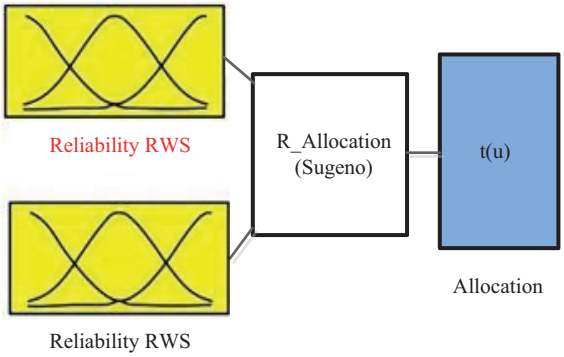

Figure 2. The general structure of the fuzzy system finding the appropriate server.

As can be seen in Figure 3, we use the Sugeno fuzzy system to find the appropriate server because the fuzzy system must be Sugeno type before being used in the neurofuzzy system.

Figure 3 shows the fuzzy sets of input fuzzy variable ''Reliability RWS''. The fuzzy sets of input fuzzy variable ''Reliability WS'' are exactly the same as those of the input fuzzy variable ''Reliability RWS''. During simulations in neurofuzzy systems, we tested the sets of input fuzzy variables with 3, 4, and 5 fuzzy sets. The results of this test can be seen in Section 5.

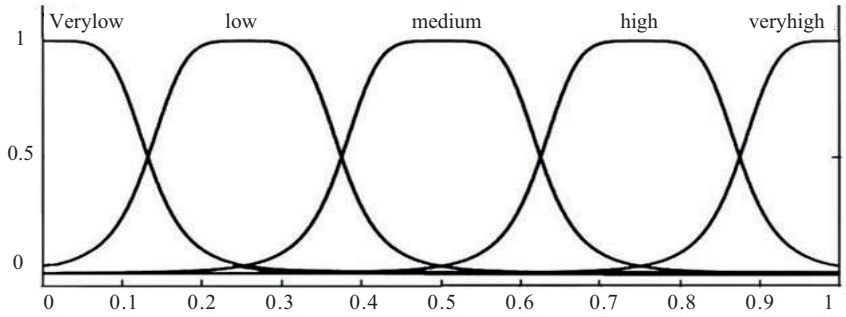

Figure 3. Fuzzy sets of input fuzzy variable ''Reliability RWS''.

For fuzzy input and output variables, we consider the parabolic, triangular, trapezoidal, and Gaussian fuzzy functions. Since we used the Sugeno fuzzy system, the output variable is considered linear and the number of its values is determined by the product of the number of fuzzy sets of each fuzzy input variable. Basically, if, for example, each of the fuzzy input variables has 3 fuzzy sets, different values in the output variable will equal 9. The basis of the output variable is the degree of similarity of the request with the server based on fuzzy rules, according to which the server can be considered as the allocation candidate.

The number of fuzzy rules considered in the Sugeno fuzzy system is also dependent on the product of the number of fuzzy sets of each fuzzy input variable. At the beginning, the weighting coefficient of all rules is set to be 1. After the training of the neural fuzzy system, the rules of these coefficients are optimized and changed. For example, Table 2 shows the fuzzy rules of the proposed system for each fuzzy set of fuzzy input variables.

Table 2. Rules of the neural fuzzy system to find the appropriate server.

1. If (ReliabilityRWSis low) and (ReliabilityWS is low) then (Allocation is out1mf1) (1)
2. If (ReliabilityRWSis low) and (ReliabilityWS is medium) then (Allocation is out1mf2) (1)
3. If (ReliabilityRWSis low) and (ReliabilityWS is high) then (Allocation is out1mf3) (1)
4. If (ReliabilityRWSis medium) and (ReliabilityWS is low) then (Allocation is out1mf4) (1)
5. If (ReliabilityRWSis medium) and (ReliabilityWS is medium) then (Allocation is out1mf5) (1)
6. If (ReliabilityRWSis medium) and (ReliabilityWS is high) then (Allocation is out1mf6) (1)
7. If (ReliabilityRWSis high) and (ReliabilityWS is low) then (Allocation is out1mf7) (1)
8. If (ReliabilityRWSis high) and (ReliabilityWS is medium) then (Allocation is out1mf8) (1)
9. If (ReliabilityRWSis high) and (ReliabilityWS is high) then (Allocation is out1mf9) (1)

4. Simulations and tests

4.1. Data sets and neural fuzzy systems

In the present study, we conducted simulations on Windows 7 and a Corei5 system. Meanwhile, we simulated fault tolerance features for servers using MATLAB. Then, in this simulation, we made evaluation possible based on scenarios of web service requests that allow the scenarios to be created randomly as well as implementing the scenarios that exist in the previous data set.

In the simulations conducted, we tested receivers' request scenarios based on the selection of appropriate web services with or without fuzzy systems. See Table 3 for several scenarios considered to run in simulations.

Table 3. Specifications of different scenarios to run in simulations.

Data set name	Different types of web services	Number of different web services	Number of receivers
Data1	10	50	100
Data2	5	50	100
Data3	10	50	80
Data4	5	50	80
Data5	10	40	60
Data6	5	40	60
Data7	10	40	50
Data8	5	40	50

In the simulations conducted, the appropriate fuzzy system was determined through 32 different fuzzy systems based on different fuzzy functions and different deductions and defuzzification according to Table 4. As can be seen in Table 4, we used different fuzzy functions for fuzzy input variables. The trapmf fuzzy function is of trapezoidal type derived from Equation (2).

$$Trapmf(x,a,b,c,d) = \begin{cases} 0 & x \le a \\ \frac{x-a}{b-a} & a \le x \le b \\ 1 & b \le x \le c \\ \frac{d-x}{d-c} & c \le x \le d \\ 0 & d \le x \end{cases} \tag{2}$$

In Equation (2), the parameters a, b, c, and d define the domain of the trapezoidal function and x is the input variable of this function. The trimf fuzzy function is of triangular type derived from Equation (3).

$$Trapmf(x,a,b,c) = \begin{cases} 0 & x \le a \\ \frac{x-a}{b-a} & a \le x \le b \\ \frac{c-x}{c-b} & b \le x \le c \\ 0 & c \le x \end{cases} \tag{3}$$

In Equation (3), the parameters a, b, c, and d denote the domain of the triangular function domain and x is the input variable of this function. The fuzzy function $pimf$ is a type of parabolic function in accordance with Equation (4).

$$pimf(x,a,b,c,d) = \left\{ \begin{array}{ll} 0 & x \le a \\ 2\left(\frac{x-a}{b-a}\right)^2 & a \le x \le \frac{a+b}{2} \\ 1-2\left(\frac{x-a}{b-a}\right)^2 & \frac{a+b}{2} \le x \le b \\ 1 & b \le x \le c \\ 1-2\left(\frac{x-c}{d-c}\right)^2 & c \le x \le \frac{c+d}{2} \\ 2\left(\frac{x-c}{d-c}\right)^2 & \frac{c+d}{2} \le x \le d \\ 0 & x > d \end{array} \right\} \tag{4}$$

Here x is the input variable of the function; a, b, c, and d are constant values that specify the domain of this function; and gaussmf is a type of Gaussian function based on Equation (5).

$$gaussmf(x,\sigma,c) = e^{\frac{-(x-c)^2}{2\sigma^2}} \tag{5}$$

As can be seen in Table 2, we used min, max, prod, and probor methods for fuzzy arguments. All of these methods take two sets of inputs with the same number of elements, creating an output set. Their functionality on the elements of sets is peer-to-peer (P2P). The general structure of these fuzzy argumentation methods is based on Equations (6) to (9).

$$\min(a,b) = \left\{ \begin{array}{ll} a & a \le b \\ b & a > b \end{array} \right\} \tag{6}$$

$$\max(a,b) = \left\{ \begin{array}{ll} b & a \le b \\ a & a > b \end{array} \right\} \tag{7}$$

$$prod(a,b) = a * b \tag{8}$$

$$probor(a,b) = a + b - (a * b) \tag{9}$$

As can be seen in Table 4, we used two methods, namely wtaver and wtsum, for defuzzification of the implemented neural fuzzy systems. Since the neural fuzzy systems in the present study are for dynamic

rejuvenation of web services, they were shortly named NNFDWSR. In the tests conducted to evaluate different fuzzy systems, the nonfuzzy details in the proposed method were simulated as the traditional method (TWSR). The runtime and number of failed web services were also considered. Moreover, we simulated and compared the method described in Jia et al. [32], which is a regression-based method for web service rejuvenation (RBWSR).

Table 4. Different neural fuzzy systems in the newly proposed method.

FIS NAME	FIS TYPE	Mf Type	And Method	Or Method	Defuzzification method
NFDWSR1	sugeno	pimf	min	max	wtaver
NFDWSR2	sugeno	pimf	prod	max	wtaver
NFDWSR3	sugeno	pimf	min	probor	wtaver
NFDWSR4	sugeno	pimf	prod	probor	wtaver
NFDWSR5	sugeno	pimf	min	max	wtaver
NFDWSR6	sugeno	pimf	prod	max	wtaver
NFDWSR7	sugeno	pimf	min	probor	wtaver
NFDWSR8	sugeno	pimf	prod	probor	wtaver
NFDWSR9	sugeno	pimf	min	max	wtaver
NFDWSR10	sugeno	pimf	prod	max	wtaver
NFDWSR11	sugeno	pimf	min	probor	wtaver
NFDWSR12	sugeno	pimf	prod	probor	wtaver
NFDWSR13	sugeno	pimf	min	max	wtaver
NFDWSR14	sugeno	pimf	prod	max	wtaver
NFDWSR15	sugeno	pimf	min	probor	wtaver
NFDWSR16	sugeno	pimf	prod	probor	wtaver
NFDWSR17	sugeno	pimf	min	max	wtaver
NFDWSR18	sugeno	pimf	prod	max	wtaver
NFDWSR19	sugeno	pimf	min	probor	wtaver
NFDWSR20	sugeno	pimf	prod	probor	wtaver
NFDWSR21	sugeno	pimf	min	max	wtaver
NFDWSR22	sugeno	pimf	prod	max	wtaver
NFDWSR23	sugeno	pimf	min	probor	wtaver
NFDWSR24	sugeno	pimf	prod	probor	wtaver
NFDWSR25	sugeno	pimf	min	max	wtaver
NFDWSR26	sugeno	pimf	prod	max	wtaver
NFDWSR27	sugeno	pimf	min	probor	wtaver
NFDWSR28	sugeno	pimf	prod	probor	wtaver
NFDWSR29	sugeno	pimf	min	max	wtaver
NFDWSR30	sugeno	pimf	prod	max	wtaver
NFDWSR31	sugeno	pimf	min	probor	wtaver
NFDWSR32	sugeno	pimf	prod	probor	wtaver

Figure 4 illustrates the structure and rules of the simulated system. The model contains 9 rules according to two inputs together with three linguistic variables for each. The tolerance parameter defines a threshold on

the error acceptable between the actual output and the desired value. The number of epochs is specified to be 1000. The training process halts whenever the error threshold is met (around the 50th epoch in our work). In our new method, root mean square error (RMSE) defines the checking of error against the number of epochs.

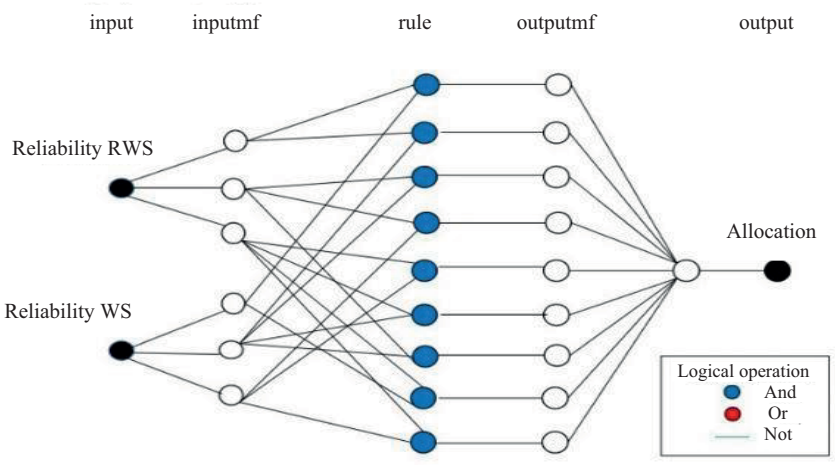

Figure 4. The structure of the fuzzy neural network.

In the proposed method, a back propagation neural network was considered. The neuron numbers in the hidden layers were determined based on considered rules in fuzzy systems. In the present study, we trained all neural fuzzy systems presented in Table 4 with 3, 4, and 5 fuzzy sets for input variables on the data sets up to 1000 steps. Then they were tested on 20% of the data, which were randomly selected.

Table 5 shows the number of failed web services for scenarios described in Table 3 using different neural fuzzy methods.

Tables 5 and 6 illustrate which neural fuzzy systems could be considered as a desirable option for our newly proposed method. For that purpose, we initially explored all systems while identifying those with minimal failed web services. Finally, the optimum system was detected for our programming system, it was compared in terms of runtime on various data sets, and then neural fuzzy system 4 was selected as the most ideal option.

4.2. Discussion

The majority of methods previously presented for optimization of software rejuvenation come with certain deficiencies. That explains why we are striving to contribute to this field. A few key drawbacks were summarized at the end of Section 2. At this stage, we specified what efforts have been made to resolve each problem. Neurofuzzy system structures are adopted to identify a suitable server for two reasons: 1) considering 100% of the adoption of web services based on receivers' requests might give rise to substantial restriction and fail for many requests, 2) the great response of fuzzy logic to uncertainly and its logical reasoning.

In the present study, we employed NFDWSR in order to predict the overall QoS values based on the quality parameter and mitigated the number of failures through rejuvenation. The combination of neural network and fuzzy logic enables us to design an adaptive fuzzy inference system for QoS evaluation. In this respect, the membership functions of the input quality parameter were trained and optimized over the course of the learning mechanism, where the overall QoS can be estimated with the minimum number of errors. The proposed FIS for finding a suitable service provider consists of two inputs: the first input identifies the reliability of the requested

Table 5. Number of failed web services using different fuzzy neural techniques.

Data set / system	Data1	Data2	Data3	Data4	Data5	Data6	Data7	Data8	The best number of fuzzy set
NFDWSR1	3	4	0	0	5	0	5	8	3
NFDWSR2	4	4	1	0	7	0	5	9	4
NFDWSR3	4	3	1	0	7	0	6	9	4
NFDWSR4	3	3	0	0	6	0	5	8	4
NFDWSR5	3	3	0	0	5	0	4	7	3
NFDWSR6	3	4	1	0	6	0	5	9	3
NFDWSR7	4	4	1	0	7	0	5	9	4
NFDWSR8	4	3	0	0	6	0	4	8	4
NFDWSR9	5	4	1	1	7	0	7	10	4
NFDWSR10	4	4	1	2	7	0	7	9	3
NFDWSR11	5	4	1	2	7	0	6	8	4
NFDWSR12	4	5	1	2	8	0	7	10	4
NFDWSR13	5	4	1	0	6	0	5	9	3
NFDWSR14	4	4	1	0	7	0	5	9	4
NFDWSR15	5	5	1	0	7	0	5	9	4
NFDWSR16	5	5	1	0	8	0	6	9	4
NFDWSR17	6	5	2	1	7	1	7	12	3
NFDWSR18	5	4	1	2	7	0	6	11	3
NFDWSR19	8	8	4	2	7	1	6	13	4
NFDWSR20	5	4	1	2	7	0	6	11	4
NFDWSR21	5	5	3	0	7	0	5	13	4
NFDWSR22	4	4	1	0	7	0	5	10	4
NFDWSR23	5	5	1	0	7	0	5	11	3
NFDWSR24	5	3	0	0	8	0	5	10	4
NFDWSR25	4	4	0	0	6	0	6	9	4
NFDWSR26	4	4	0	0	6	0	6	9	4
NFDWSR27	4	5	1	0	7	1	7	10	4
NFDWSR28	4	4	0	0	6	0	6	10	4
NFDWSR29	5	5	1	0	6	0	6	10	4
NFDWSR30	5	5	1	0	6	0	6	10	4
NFDWSR31	5	4	1	0	7	0	6	10	4
NFDWSR32	5	4	1	0	6	0	6	10	4

web service and consists of five membership functions with “very low”, “low”, “medium”, ”high”, and “very high” labels. The thresholds were defined based on the distribution of the parameters in the data set. The second input of the fuzzifier identifies the reliability of the web service provider, where the membership functions and numerical intervals are identical in order to truly check the accordance of the two input parameters. In the presented system, the match rate of the user’s requested web service reliability and the web service providers’

Table 6. Runtime based on different scenarios using different neurofuzzy methods.

Data set / system	Data1	Data2	Data3	Data4	Data5	Data6	Data7	Data8	Means result
NFDWSR1	6.13743	7.9763	7.7501	8.65965	10.5713	9.2444	6.47067	7.8675	8.084661
NFDWSR2	9.6247	7.30689	6.32315	7.57969	10.4893	6.629	5.95961	4.7557	7.3335061
NFDWSR3	10.4041	7.85078	7.75212	8.6199	6.76828	9.5689	4.16972	4.3215	7.4319061
NFDWSR4	8.34742	5.38692	9.18167	10.179	9.75098	7.1744	6.63842	6.3337	7.8740557
NFDWSR5	10.3784	7.59597	9.10149	8.54209	8.44576	10.214	4.33388	4.5993	7.9013439
NFDWSR6	6.78025	6.56375	9.24121	10.1151	6.77212	8.2967	6.1075	5.8756	7.4690287
NFDWSR7	8.95587	7.51189	6.81393	8.44243	6.56969	10.928	4.66867	4.478	7.2960934
NFDWSR8	7.67584	5.79247	7.70359	8.02772	10.7823	10.641	4.21071	7.3204	7.7693166
NFDWSR9	7.21104	6.69134	7.96542	10.7423	8.37985	10.924	5.20582	7.1549	8.034308
NFDWSR10	8.99852	7.15651	8.64147	9.49938	7.3016	7.0761	7.99632	4.77	7.6799885
NFDWSR11	6.1467	7.2448	9.4684	9.51129	7.35695	8.1601	5.8429	8.4174	7.7685675
NFDWSR12	6.70382	8.42209	8.40144	8.19322	7.35916	8.4271	5.92809	4.5428	7.2472143
NFDWSR13	8.65278	5.90475	7.23079	9.12344	7.63313	7.807	6.46836	5.1938	7.2517497
NFDWSR14	9.70969	8.93065	8.78612	8.04745	9.12831	6.985	7.62523	7.9584	8.3963575
NFDWSR15	9.67992	6.04291	8.1746	6.60131	8.41367	7.9072	4.64594	4.8044	7.0337541
NFDWSR16	7.90299	5.37692	8.19336	8.61916	9.63177	9.6495	6.55412	4.1512	7.5098783
NFDWSR17	6.30963	6.2784	7.88889	9.44501	8.33429	10.19	6.87344	8.3589	7.9598101
NFDWSR18	8.391	6.30058	5.97533	9.24931	10.0046	8.4055	4.36329	5.1991	7.2360992
NFDWSR19	6.69146	6.12402	7.48038	8.87214	8.55841	10.439	6.07221	8.2463	7.8105117
NFDWSR20	8.86969	8.83078	6.58318	9.54255	7.80079	9.5231	6.78056	4.306	7.7795819
NFDWSR21	7.14656	5.89616	8.50525	10.2998	8.05008	10.012	6.70133	4.0302	7.5802118
NFDWSR22	8.70977	6.54708	9.62196	6.50518	8.58102	8.4096	5.84367	7.4657	7.7104961
NFDWSR23	7.45112	8.13896	7.62111	6.66093	7.29143	9.7479	5.89394	4.6872	7.1865819
NFDWSR24	7.53506	7.42956	6.36285	9.82292	7.59282	10.628	5.07625	7.4448	7.7365776
NFDWSR25	6.84898	6.14999	5.91001	9.09294	9.57513	8.9597	5.70292	6.9	7.3924545
NFDWSR26	8.91428	7.71607	8.36104	10.7533	7.44021	9.6918	4.94492	4.5373	7.7948563
NFDWSR27	8.73287	6.80055	7.56426	9.47875	9.96628	8.076	6.64804	5.8727	7.8924316
NFDWSR28	9.78868	8.33167	6.65398	9.26057	9.12012	8.9333	7.47976	5.1915	8.094953
NFDWSR29	7.43133	5.47686	9.72923	9.40498	8.65758	9.3769	6.17886	6.9129	7.8960856
NFDWSR30	8.44749	7.88419	7.85123	10.9717	7.48404	6.9761	4.43879	4.2862	7.2924576
NFDWSR31	7.82061	6.79349	7.14617	9.93577	9.32553	9.9739	7.73141	8.3773	8.3880299
NFDWSR32	6.86413	5.5555	8.6332	6.92219	8.86432	8.8865	7.44456	6.1818	7.4190351

reliability were calculated. In fact, it determines the system conformity in terms of whether a service provider can execute a service to a suitable extent. Fuzzy rules are then defined to specify the appropriate logic mapping. We have considered nine fuzzy rules that determined all possible allocation of the web services via a service provider in our method.

To obtain the best neural fuzzy system, several systems were considered with different membership functions and defuzzification methods. Meanwhile, the best fuzzy system with minimum number of failure and runtime was selected during the work.

The results in Figure 5 demonstrate that the proposed fuzzy neural network rejuvenation method (NFDWSR4) has a lower failed web service rate in comparison with the nonfuzzy rejuvenation method (TDWSR) and the regression-based regenerative model (RBWSR). The reason for this improvement over the nonfuzzy approach is the restriction of considering 100% of combination between web service receivers' request and web service providers' reliability in the nonfuzzy method. The basic disadvantage of the regression-based method compared to our method is its reliance on stopping the software system at the time of software rejuvenation. On the other hand, according to Figure 6, the proposed method achieves the highest execution time due to the deeper time complexity of the neural network compared to other methods.

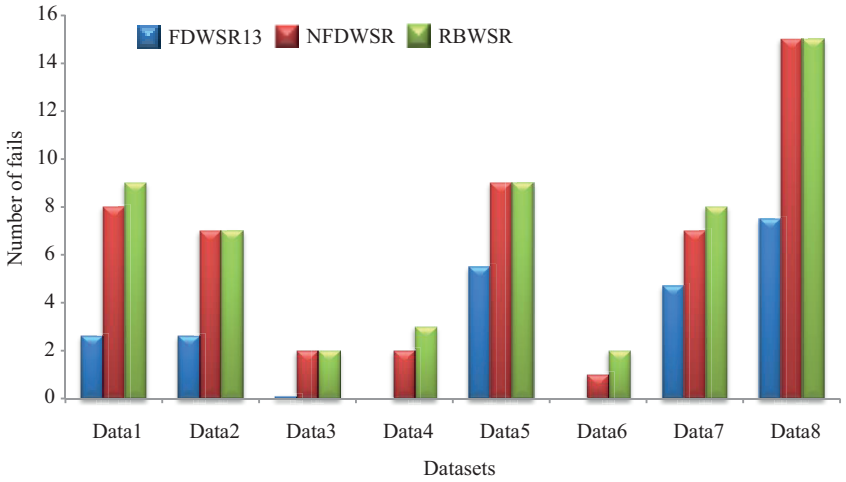

Figure 5. The number of failed web services in the neural, traditional, and regression fuzzy methods.

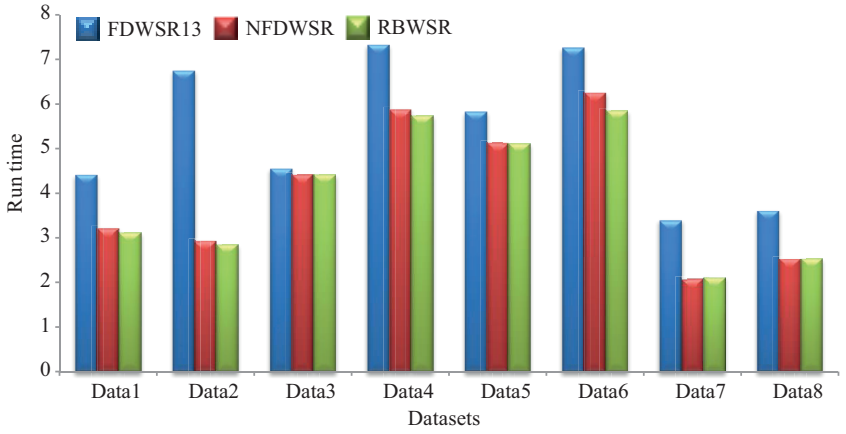

Figure 6. Runtime scenarios using traditional, regression, and neural fuzzy methods.

5. Conclusions

In web service-based applications, evaluation of QoS values is an important issue but is difficult due to uncertain, poorly available information about quality parameters as well as their nonlinear relations. In the present research, the servers provide web services based on receiver requests in two scenarios. Firstly, they are expected

to have minimal reliability requirements for those services. Secondly, there should be maximum matching between the requested web service and the allocated web service as far as possible. If allocation takes place exactly based on the request in web services, a major problem in fault-tolerance systems is the curtailed availability for specific infeasible requested services.

In an effort to tackle this deficiency, the new method employed a type of fuzzy allocation that facilitates flexible implementation between the requested and allocated web services. The main disadvantage in adoption of the fuzzy method in isolation, however, is how to assign thresholds between multiple fuzzy sets in fuzzy variable matched request web service and allocated service. In order to obtain maximum allocation and fix the drawbacks of the fuzzy technique, the novel method employed a fuzzy neural network in which the system at training level can specify the threshold between multiple fuzzy sets for implementation. The new neural fuzzy system, dubbed NFDWSR, can adjust the membership functions initially defined by experts. In particular, this neural fuzzy system is used to tune QoS parameters adaptively according to a given data set. During the learning process, the difference between predicted and desired quality is mitigated. The trained system can be utilized in the decision-making process for QoS-aware adaptation.

The experimental results in Section 4 illustrate the capability of the proposed approach in finding a relationship between the quality variables and predicting the overall QoS values, while decreasing the failure rate by rejuvenation of unsuitable services.

The obtained results confirm the research strategy has on average 51% and 54.5% less failed web services than TDWSR and RBWSR, respectively. Moreover, the runtime of the proposed method is 74% and 77.5% more than TDWSR and RBWSR. Of course, such increase in runtime, in line with the 50% reduction in the number of failed web services, shows the high fault tolerance of the proposed method, which increases the availability of web service providers. This neural fuzzy system in the proposed method can be used for environments that require high fault tolerance. It should be noted that the runtime of the proposed method is slightly high for online environments and requires more hardware overhead.

For future research, other criteria such as reliability and validity can be considered to improve the accuracy of the proposed method and choose the right service providers of web services. Moreover, this method should be considered along with clustering methods for real-world environments where the numbers of web services and requesters are far higher.

References

[1] Kalantari KR, Ebrahimnejad A, Motameni H. Dynamic software rejuvenation in web services: a whale optimization algorithm based approach. Turkish Journal of Electrical Engineering & Computer Sciences 2019; 28: 890-903.

[2] Beron M, Bernardis H, Miranda E, Riesco D, Pereira M, Henriques P. Measuring the understandability of WSDL specification. web service understanding degree approach and system. Computer Science and Information systems 2016; 13 (3): 779-807.

[3] Gupta R, Kamal R, Suman U. A QoS-supported approach using fault detection and tolerance for achieving reliability in dynamic orchestration of web services. Journal of Information Technology 2018; 10 (1): 71-81.

[4] Liu M, Wang L, Gao L, Li H, Zhao H, Men M. A Web Service trust evaluation model based on small-world networks. Knowledge-Based Systems 2014; 57: 161-167.

[5] Levitin G, Xing L, Xiang Y. Optimizing software rejuvenation policy for tasks with periodic inspections and time limitation. Reliability Engineering & System Safety 2020; 197; 1-31.

[6] Esposito C, Ficco M, Palmieri F, Castiglione A. Smart cloud storage service selection based on fuzzy logic, theory of evidence and game theory. IEEE Transactions on Computers 2016; 65 (8): 2348-2362.

[7] Liu J, Tan X, Wang Y. CSSAP: Software Aging Prediction for Cloud Services Based on ARIMA-LSTM Hybrid Model. In: IEEE International Conference on Web Services; Milan, Italy; 2019. pp. 283-290.

[8] Rathore SS, Kumar S. An empirical study of some software fault prediction techniques for the number of faults prediction. Soft Computing 2017; 21 (24); 7417-7434.

[9] Kalantari KR, Ebrahimnejad A, Motameni M. An efficient improved ant colony optimization algorithm for dynamic software rejuvenation in web services. IET Software 2019. doi: 10.1049/iet-sen.2019.0018

[10] Torres E, Callou G, Andrade E. A hierarchical approach for availability and performance analysis of private cloud storage services. Computing 2018; 100 (6): 621-644.

[11] Takeshi Endo A, Simao A. Event tree algorithms to generate test sequence for composite web services. Software: Testing, Verification and Reliability 2019; 29 (3): 1-25.

[12] Dantas J, Matos R, Araujo J, Maciel P. Eucalyptus-based private clouds: availability modeling and comparison to the cost of a public cloud. Computing 2015; 97 (11): 1121-1140.

[13] Xiuguo Wu. A security-aware data replica placement strategy based on fuzzy evaluation in cloud. International Journal of Intelligent & Fuzzy Systems 2018; 35(1): 243-255.

[14] Araujo J, Matos R, Alves V, Maciel P, De Souza FV et al. Software aging in the eucalyptus cloud computing infrastructure: characterization and rejuvenation. ACM Journal on Emerging Technologies in Computing Systems 2014; 10 (1): 1-22.

[15] Meenakshi A, Sirmathi H, Anitha J. Cloud computing based resource provisioning using K-mean clustering and GWO prioritization. Soft Computing 2019; 1-11.

[16] Bai J, Chang X, Machida F, Trivedi KS, Han Z. Analyzing software rejuvenation techniques in a virtualized system: service provider and user views. IEEE Access 2020; 9: 6448-6459.

[17] Guo J, Song X, Wang Y, Zhang B, Li X. The Measurement of software aging damage and rejuvenation strategy for discrete web services. Advanced Materials Research 2012; 2: 432-437.

[18] Mooij A, Eggen G, Hooman J, Wezep H. Cost-Effective Industrial Software Rejuvenation Using Domain-Specific Models. In: Proceedings of 8th Conference on Theory and Practice of Model Transformations; Berlin, Germany; 2015; pp. 66-81.

[19] Levitin G, Xing L, Huang H. Optimization of partial software rejuvenation policy. Reliability Engineering & System Safety 2019; 182: 63-72.

[20] Meng H, Liu J, Hei X. Modeling and optimizing periodically inspected software rejuvenation policy based on geometric sequences. Reliability Engineering & System Safety 2015; 133: 184-191.

[21] Machida F, Miyoshi N. An Optimal Stopping Problem for Software Rejuvenation in a Job Processing System. In: Workshop on Software Reliability Engineering; NW Washington, DC, USA; 2015; 3; pp. 139-143.

[22] Okamura H, Dohi T. Dynamic software rejuvenation policies in a transaction-based system under Markovian arrival processes. Performance Evaluation 2013; 70; 197-211.

[23] Ning G, Zhao J, Lou Y, Alonso J, Matias R et al. Optimization of two-granularity software rejuvenation policy based on the Markov Regenerative Process. IEEE Transactions on Reliability 2016; 65 (4): 1630-1646.

[24] Dohi D, Okamura H, Trivedi KS. Optimizing software rejuvenation policies under Interval Reliability Criteria. In: Proceeding of 9th Conference on Ubiquitous Intelligence and Computing; Fukuoka, Japan; 2012. pp. 478-485.

[25] Cotroneo D, Iannillo A K, Natella R, Pietrantuono R, Russo S. The Software Aging and Rejuvenation Repository. IEEE Transaction on Software Reliability Engineering 2015; 2: 108-113.

[26] Agepati R, Gundala N, Amari S. Optimal Software Rejuvenation Policies. Reliability and Maintainability Symposium 2013: 341-347.

[27] Guo J, Song X, Wang Y, Zhang B, Li X. The Measurement of Software Aging Damage and Rejuvenation Strategy for Discrete Web Services. Advanced Materials Research 2012; 2: 432-437.

[28] Nguyen HT, Zhao W, Yang J. A trust and reputation model based on bayesian network for web services. In: Proceeding of 8th Conference on Web Services; Miami, FL, USA; 2010. pp. 251-258.

[29] Wu Y, Peng G, Wang H, Zhang H. A two-stage fault tolerance method for large-scale manufacturing network. IEEE Access 2020; 7: 81574-81592.

[30] Torquato M, Umesh I M, Maciel P. Models for availability and power consumption evaluation of a private cloud with VMM rejuvenation enabled by VM live Migration. Journal of Supercomputing 2018; 74 (9): 4817-4841.

[31] Wahab O.R, Bentahar J, Otrok H, Mourad A. A survey on trust and reputation models for web services. Decision Support Systems 2015; 74: 121-134.

[32] Jia Sh, Hou Ch, Wang J. Software aging analysis and prediction in a web server based on multiple linear regression algorithm. In: 9^{th} IEEE International Conference on Communication Software and Networks; Guangzhou, China; 2017. pp. 1452-1456.