1-1-2015

Using neural networks to forecast available system resources: an approach and empirical investigation

Yun-Fei Jia
Civil Aviation University of China

Zhi Quan Zhou
University of Wollongong, zhiquan@uow.edu.au

Ke-Xian Xue
Institute of NBC Defence of the PLA

Lei Zhao
Beijing Institute of Control Engineering

Kai-Yuan Cai
Beijing University of Aeronautics and Astronautics, kycai@buaa.edu.cn

Using neural networks to forecast available system resources: an approach and empirical investigation

Abstract

Software aging refers to the phenomenon that software systems show progressive performance degradation or a sudden crash after longtime execution. It has been reported that this phenomenon is closely related to the exhaustion of system resources. This paper quantitatively studies available system resources under the real-world situation where workload changes dynamically over time. We propose a neural network approach to first investigate the relationship between available system resources and system workload and then to forecast future available system resources. Experimental results on data sets collected from real-world computer systems demonstrate that the proposed approach is effective.

Publication Details

Jia, Y., Zhou, Z. Quan., Xue, K., Zhao, L. & Cai, K. (2015). Using neural networks to forecast available system resources: an approach and empirical investigation. International Journal of Software Engineering and Knowledge Engineering, 25 (4), 781-802.

Using Neural Networks to Forecast Available System Resources: An Approach and Empirical Investigation

Yun-Fei Jia, Zhi Quan Zhou, Ke-Xian Xue, Lei Zhao, and Kai-Yuan Cai

Abstract

Software aging refers to the phenomenon that software systems show progressive performance degradation or a sudden crash after longtime execution. It has been reported that this phenomenon closely relates to the exhaustion of system resources. This paper quantitatively studies available system resources under the real-world situation where workload changes dynamically over time. We propose a neural network approach to first investigate the relationship between available system resources and system workload and then to forecast future available system resources. Experimental results on data sets collected from real-world computer systems demonstrate that the proposed approach is effective.

Keywords: forecasting; neural networks; software aging; software reliability; system availability; system resources; system workload.

I. INTRODUCTION

Reliability and availability are key qualities of computer systems. More often than not, system failures are attributed to software than hardware [12,32]. When an application server runs continuously for a long period of time, many error conditions in its process space or kernel space

This work was supported in part by the National Key Technology R&D Program (Grant No. 2011BAH24B12) and a linkage grant of the Australian Research Council (Project ID: LP100200208).

Yun-Fei Jia is with the Tianjin Key Laboratory for Advanced Signal Processing, Civil Aviation University of China, Tianjin 300300, China.

Zhi Quan Zhou is with the School of Computer Science and Software Engineering, University of Wollongong, Wollongong, NSW 2522, Australia. All correspondence should be addressed to Zhi Quan Zhou, e-mail: zhiquan@uow.edu.au, phone: (61-2) 4221-5399.

Ke-Xian Xue is with the Institute of NBC Defence of the PLA, China.

Lei Zhao is with the Beijing Institute of Control Engineering, Beijing 100080, China.

Kai-Yuan Cai is with the Department of Automatic Control, Beijing University of Aeronautics and Astronautics, Beijing, 100191, China.

can be accumulated. Examples of these error conditions are memory leak, numerical error accumulation, out-of-order processes or threads, unreleased file tables, and data corruption. These error conditions will eventually become critical, such as exhaustion of computing resources of the system, paroxysmal crash, increased response time and degraded performance. This phenomenon is called *software aging*, which will result in many disastrous consequences. A real-world example is the loss of life owing to software aging in the safety-critical weapon-control system of Patriot in 1991 [25]. Possible root causes of software aging include residual defects in the software [13,17].

Although researchers have proposed many assumptions about the causes and evolvement of software aging [6,15,17,18], its influencing factors are still not well identified or quantified. This fundamental question can only be answered by experimental research. However, only a very limited number of such experimental studies on software aging have been reported in major software and reliability journals [13,14,29]. This contrasts unfavorably with the growing awareness and widely accepted importance of experiment-based studies [7,29].

The exhaustion of system resources is considered to be a primary cause of software aging [13]. Grottke et al. proposed to forecast the usage of computing resources with an autoregressive model [13], which assumes an even workload over time. Under even workload, aging is the only factor affecting available resources, thus AR model is sufficient to forecast this aging trend. Unfortunately, however, the workload of real-world systems is often uneven. When workload increases, the available resources of a computer system will decrease quickly. A question that naturally arises is: Can we forecast the available system resources based on workload information of the system? Vaidyanathan and Trivedi incorporated the effect of workload in their software aging model [30]. They grouped workload into eight clusters, and calculate the exhaustion rate of computing resources with respect to each workload cluster. They found that the exhaustion rate

was faster with higher system activity. Nevertheless, their findings were only for modeling purpose rather than for forecasting the exhaustion of resources with varying workload. Moreover, in many of the workload states, the dynamics of the resources demonstrates very high variance, resulting in very broad confidence intervals of the exhaustion rate. The highly irregular and oscillatory behavior of the data makes most trend models insufficient. Further, the models of Vaidyanathan and Trivedi [30] are offline models and hence cannot forecast available resources online, where workload patterns differ at different times. These issues are addressed in the present paper.

In the present paper, we study available system resources for real-world computing systems where workload dynamically changes over time. The proposed method can be used in various areas – for instance, for designing countermeasures (such as software rejuvenation) against software aging, or for providing a basis for deciding how many connections to a Web server should be cut off to avoid the exhaustion of computing resources in accordance with admission control [31]. The proposed method is based on neural networks. In this paper, computing resources refer to the resources of the operating system, such as Real Memory Free, CPU usage rate, Used Swap Space, I/O usage, and so on. We focus on the Real Memory Free and Used Swap Space since they are considered to be leading indicators of aging [30].

The rest of this paper is organized as follows: Section II provides background information and reviews related studies. Section III describes the data sets used in our experiments and introduces some basic concepts of our approach. Section IV investigates the relationship between system workload and available system resources. Section V presents our approach and experimental results for forecasting available system resources. Section VI concludes the paper and points out future research topics.

II. BACKGROUND

The phenomenon of software aging was first reported by Marshall [25]. The aging problem was in the Patriot missile system, and it was solved by resetting the weapon-control system every eight hours. A few years later, Huang et al. [17] proposed a model of software aging together with a counteraction, namely software rejuvenation. The studies on software aging and control can roughly be classified into four parts: mechanism, metrics, modeling and control. Research on software aging mechanisms focuses on causes and effects, influencing factors and evolution of software aging. It provides a basis for extracting a metric of software aging and provides observations for modeling research. The objective of research on software aging metrics is to detect and estimate the severity of software aging. It provides both quantitative metrics for the research in mechanism and measurable objective for the research in control. Software aging modeling can formulate the aging process based on observations from experiments, and can determine the effectiveness of software control. It can also provide a model for the research in control. Software aging control is the ultimate objective of software aging research. It aims to detect and estimate the severity of software aging, and select optimal rejuvenation policy to heal the aged software.

Matias et al. used *design of experiment* (DOE) and accelerated degradation test (ADT) techniques to characterize the aging phenomenon [24]. They found that the “page size” and “page type” factors were responsible for over 99% of memory size variation in httpd processes. Zhao et al. injected memory leaks to a test bed to expedite aging. They used the experimental results to estimate some parameters of the Weibull distribution, the lifetime distribution of the running software [36]. Jia et al. analyzed the evolution of software aging in Apache httpd and reported that the aging process is chaotic and can only be forecasted in limited ahead time [18]. Shereshevsky et

al. [28] monitored the Hölder exponent (a measure of the local rate of fractality) of the system parameters and found that system crashes were often preceded by the second abrupt increase in this measure.

Because software aging involves many complicated and interrelated factors, it is not easy to propose a metric that reflects all the factors to measure the degree of software aging. Software aging is characterized, for instance, by consistent throughput loss in [8], by increased response time in [13], and by exhaustion of computing resources in [14]. Grottke et al. proposed an *estimated time to exhaustion* metric to predict the approximate time of depletion of system resources [13]. A comprehensive evaluation function was proposed in [19] to measure the aging speed of the Apache server.

Modeling, on the other hand, is a mainstream of software aging research. Modeling begins by making assumptions about the mechanism of aging (including its causes and effects), and constructs mathematical models to describe the aging process. It helps with the validation of the effectiveness of software rejuvenation and the optimal schedules for software rejuvenation. Vaidyanathan and Trivedi [30] monitored operating system activities, and described workload based on four important parameters, namely *cpuContextSwitch*, *sysCall*, *pageIn*, and *pageOut*, using a clustering method. The effect of workload on resource depletion was quantified by means of slopes, but the issue of real time values of each resource parameter under varying workload was not addressed. Huang et al. [17] proposed a three-state stochastic model, including a robust state, a failure-prone state and a failure state. This model was extended and studied in detail by other researchers to answer similar questions [9,10]. Chen et al. [8] introduced a threshold to judge the current pattern of software aging and to describe its nonlinearity. The above models are all Markov models. Jia et al. [20] introduced a nonlinear model to describe the evolution of software aging.

El-Shishiny et al. [11] exploited neural networks to mine the patterns of the usage of computing resources. But their work neglected the influence of system workload. Hong et al. [15] proposed an idea of closed-loop design of software rejuvenation to reset the computer system based on feedback information. Their objective was to determine the optimal rejuvenation time with the feedback information. On the other hand, Jia and Cai introduced control theory to software rejuvenation, including how to apply system identification, controller design and evaluation to software rejuvenation [21]. Zhao et al. described the workload of an HTTP server using a queuing model. Further, using a distributed rejuvenation algorithm, they found the optimal rejuvenation time [35].

This paper falls into the area of software aging modeling. Our aim is to investigate the relationship between workload and available computing resources, and to quantitatively predict future available computing resources.

III. PRELIMINARIES

A. The Data Sets

To study the usage of resources in computer systems requires the collection of real-world data. For this purpose, we used the data sets reported in [30]. The data collection process is briefly described below. A monitoring tool was used to collect operating system resource usage data (such as physical/virtual memory usage and file/process table usage) and system activity data (such as paging activity and CPU utilization) from nine heterogeneous UNIX workstations. These workstations were connected by an Ethernet LAN at the Duke Department of Electrical and Computer Engineering. These workstations provided various services, and the inputs from clients were unknown. From these workstations, more than one hundred parameters were monitored at regular intervals (10 minutes) for more than three months. These parameters “include those that

describe the state of the operating system resources, state of the processes running, information on the /tmp file system, availability and usage of network related resources, and information on terminal and disk I/O activity". In this study, we used the data sets collected from the workstations named Rossby and Jefferson. The data set of Jefferson was not illustrated or studied in [30].

We analyzed the system resources data (represented by Real Memory Free and Used Swap Space as explained previously) and system workload data. To measure the latter, it is natural to think of the HTTP connection rate, that is, the number of HTTP requests coming from clients per unit time. It is, however, not a good measure of workload. This is because, first, an HTTP request may be CPU intensive or I/O intensive. For example, a static HTML page request is more often I/O intensive since it involves little computation [22]. On the other hand, a dynamic request such as a database query will involve much CPU usage. These types of requests will result in different bottleneck of the system. Secondly, even for the same type of HTTP request, say I/O intensive requests, those requesting large files will cost more resources than those requesting small files. Finally, requests from clients may not necessarily be HTTP requests.

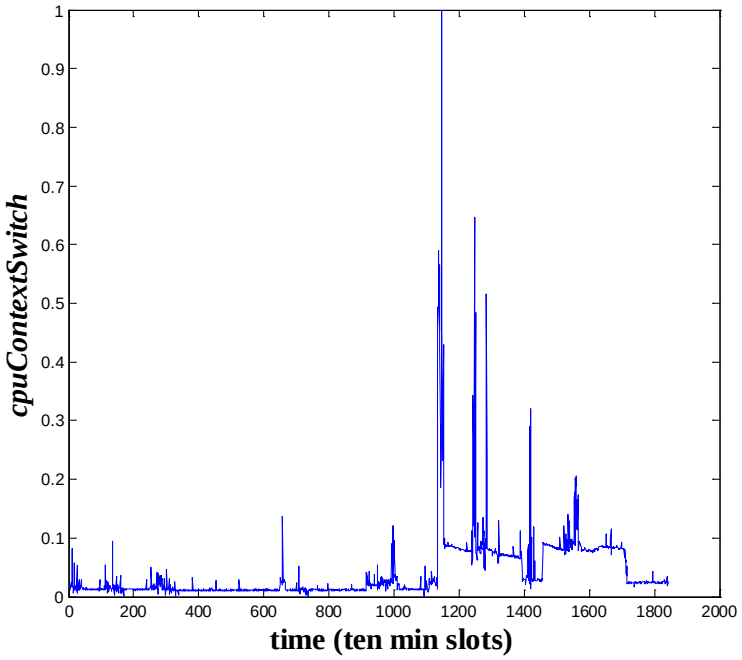
Following [30], we characterize the system workload by variables pertaining to CPU activity and file system I/O. To be more specific, the following variables have been used to characterize the workload in our study:

cpuContextSwitch: The number of process context switches performed during the measurement interval.

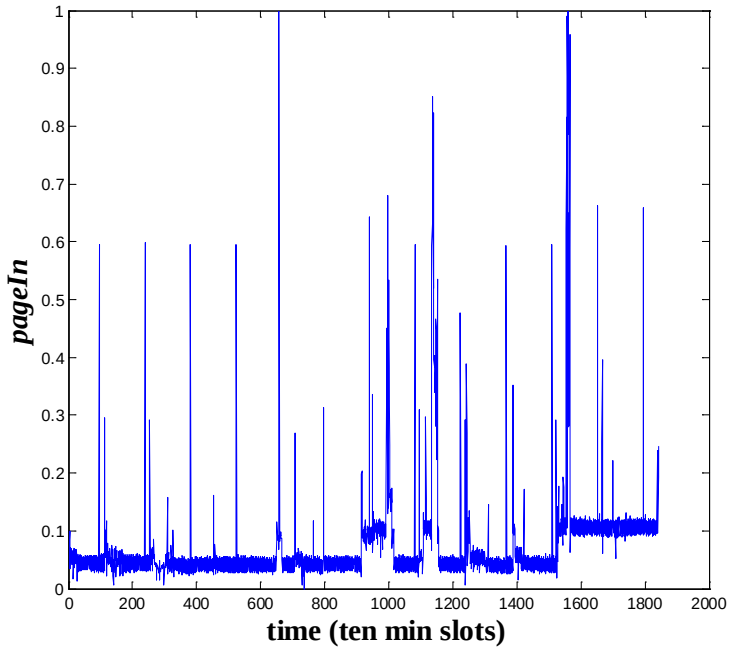
sysCall: The number of system calls made during the interval.

pageIn: The number of page-in operations (pages fetched in from file system or swap device) during the interval.

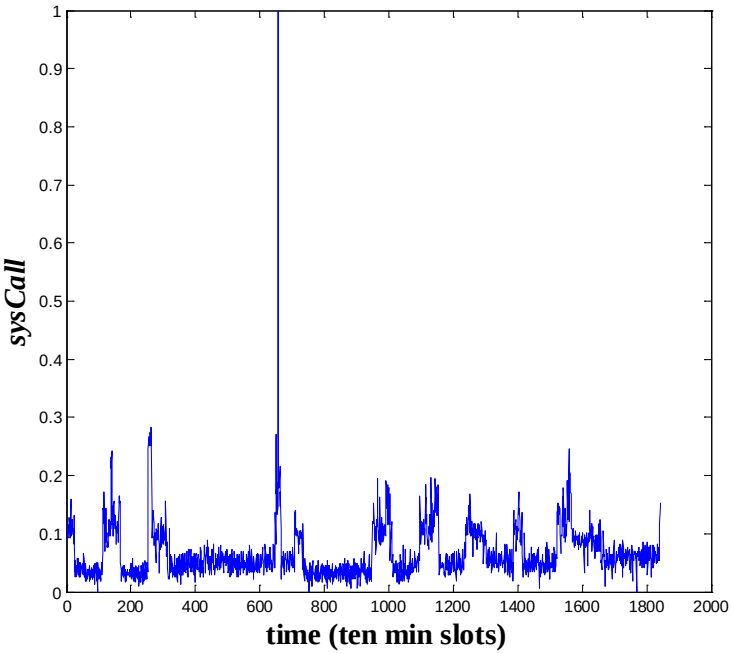
Thus, a point in a three-dimensional space, (*cpuContextSwitch*, *sysCall*, *pageIn*), represents the measured workload for a given interval of time. In our study, the raw data were normalized first. Note that we did not include the factor *pageOut* in the above definition as did in [30]. This is because this variable is almost constant in the data sets. Figure 1 depicts the three dimensions of the workload data collected from the workstation Rossby.



(a)



(b)



(c)

Fig. 1. Workload data observed on Rossby

B. A Neural Network Approach

A neural network approach is adopted in our study. This subsection provides a brief introduction to neural networks.

A neural network is a mathematical model that can learn and mimic human behavior. It is composed of many simple elements called neurons. Neurons are connected (with weights on the connections) so that they can process information collaboratively and can store the information. Although many types of neural network models have been proposed, the most popular one is called *multi-layer perceptron* (MLP) feed forward model, which is composed of non-linear, non-parametric approximators. Neural-network based approaches have many advantages. First, they can capture nonlinear phenomena. Secondly, they can solve problems for which an algorithmic solution does not exist or is too complex to find. Finally, they can show improved performance with time when more and more patterns are learnt. Neural networks have been successfully applied to many areas such as pattern recognition [26], system identification [4,5], forecasting [1] and intelligent control [2].

When applying neural networks, it is important to choose an appropriate network architecture, namely the number of layers and the number of hidden neurons per layer. This question can only be answered by experience. Hornik et al. [16] established that as few as one hidden layer with sufficient neurons can approximate any continuous function with any precision. In our study, we decided to use a three-layer neural network with two hidden layers. There are several reasons for adopting two hidden layers. First, using a single hidden layer could make the neurons tend to interact with each other globally, which is not desirable. Using two hidden layers has an advantage that the first hidden layer can learn the local features that characterize specific regions of the input space, and global features are extracted in the second hidden layer [23].

The selection of a training algorithm for the neural network is another important issue. *Back propagation error* (BP) is most useful for feed forward networks [34]. An MLP feed forward model together with BP training is normally called a *BP neural network*. In a BP neural network, only neurons in adjacent layers are connected. A BP neural network can learn complicated nonlinear input-output relationships from a set of sample data, that is, a set of input-output values. It is also important to correctly choose a set of initial weights for the network. It is a common practice to initialize weights to small random values within a certain interval. The BP neural network is used in our study.

The accuracy of the test results will be measured using all of the following approaches: (1) visual analysis, (2) Root Mean Square Error (RMSE), and (3) Pearson's correlation coefficient (r). The first approach, visual analysis, is straightforward and intuitive, and hence commonly used for researchers to verify the forecasting effect [3, 27]. The second approach, RMSE, is a well-known metric of predictive accuracy. It quantitatively measures the differences between values predicted by the neural network and the values actually observed. The third approach, Pearson's r, is widely used by researchers as a measure of the degree of correlation, or linear dependence, between two variables. When $r \geq 0.5$, the two variables have a strong positive correlation. Furthermore, the statistical significance of the correlation is given by the p-value. A p-value below 0.05 is normally considered to be statistically significant. The Pearson's correlation coefficient is used in the present study to investigate the linear relationship between the observed and predicted values.

IV. ON THE RELATIONSHIP BETWEEN SYSTEM WORKLOAD AND AVAILABLE SYSTEM RESOURCES

Intuitively, there is a relationship between system workload and available system resources. This is because the higher the system activity, the higher is the system workload and, hence, the more will computing resources be consumed, resulting in decreased available system resources. It must

be pointed out, however, that available system resources do not only relate to *normal* usage of resources consumed by running applications, but also relate to software aging caused by accumulated error conditions of the system, such as memory leak. It has been observed that the higher the system activity is, the likelier will the system age [30]. The research question of this section is: how close is the relationship between system workload and available resources, and can the latter be modeled quantitatively by the former? Answers to these questions may also provide hints on whether and when we can control available resources by controlling system workload.

As explained earlier in the paper, in this research we use two indicators, namely *realMemoryFree* and *usedSwapSpace*, to represent system resources as they are leading indicators of aging [30], and three indicators, namely *cpuContextSwitch*, *sysCall*, and *pageIn*, are used to represent system workload.

The workload and resource usage data observed from the real world are strongly nonlinear. For instance, many spikes can be seen in Figure 1. The relationship between workload and resources is also nonlinear and is difficult to formulate. We decided, therefore, to use the BP neural network to study this relationship [1], with the settings given in Section III-B. The input to the network is workload data, namely, three-dimensional vectors (*cpuContextSwitch*, *sysCall*, *pageIn*), and the output is resources data, namely, two-dimensional vectors (*realMemoryFree*, *usedSwapSpace*). As a common practice, the number of neurons was determined through trial and error [23], which will be explained shortly, and we used the first two-thirds or so of the data set to train the neural network and the remaining one-third or so to validate/test the learning effect of the network [23]. We first applied this approach to the Rossby data set. The whole Rossby data set can be divided into a few segments, each of which records observations from system startup to system reset. We used one of these segments in our study, which contains a total of 5,811 observations. Since the

[1] A preliminary version of this approach was proposed in QSIC 2009 [33].

observations were at intervals of ten minutes, the total duration is 10×5,811=58,110 minutes. We cut this segment of 5,811 observations into two parts: the first part contains 4,000 observations and was used to train the neural network, and the second part contains 1,811 observations and was used for testing.

The accuracy is first quantified using the Root Mean Square Error (RMSE), a widely used measure of the difference between values predicted by a model and those actually observed, calculated as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (y_i - e_i)^2}{n}},$$

where n is the total number of observations, and y_i and e_i represent the ith observed and forecasted values (output of the neural network), respectively.

In general, researchers determine the number of neurons in the hidden layers of multi-layer neural networks by trial and error [23]. This approach is adopted in the present study. We increase the number of neurons in the two hidden layers from (5, 5) to (30, 30) and calculate the RMSE values of *realMemoryFree* and *usedSwapSpace*, as shown in Table 1. It can be observed that RMSE decreases when the number of neurons increases, and that the decrease rate becomes small when the number of neurons is sufficiently large. More specifically, when the numbers of neurons in the two hidden layers increase from (10, 10) to (20, 20), the RMSE values of both *realMemoryFree* and *usedSwapSpace* drop noticeably from 0.3194 to 0.3044 (by 4.70%) and from 0.3982 to 0.3877 (by 2.64%), respectively; however, when the numbers of neurons further increase from (20, 20) to (30, 30), the RMSE values decrease only slightly (by 0.33% for *realMemoryFree* and 0.80% for *usedSwapSpace*). We decided, therefore, to set the numbers of neurons to (20, 20) in the experiments because too many neurons in hidden layers can greatly increase the training time.We have also applied the same approach to the Jefferson data set, where

4,000 observations were used to train the neural network, and 2,049 observations were used for testing.

Table 1. RMSE for different numbers of neurons in the two hidden layers

Numbers of neurons in two hidden layers	RMSE of *realMemoryFree*	RMSE of *usedSwapSpace*
(5,5)	0.3216	0.4078
(10,10)	0.3194	0.3982
(20,20)	0.3044	0.3877
(30,30)	0.3034	0.3846

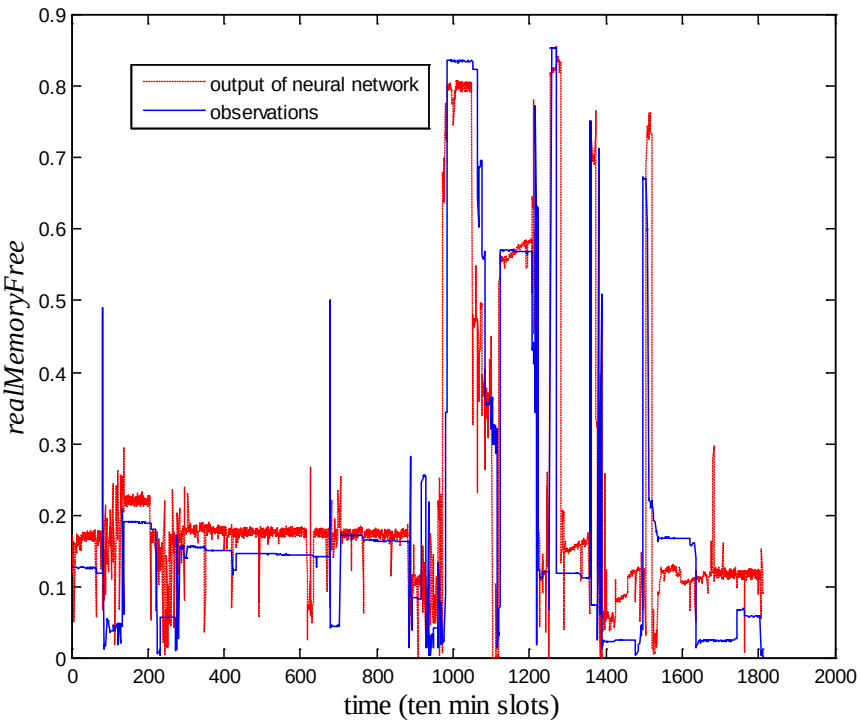


Fig. 2. Test results for *realMemoryFree* (on the Rossby data set)

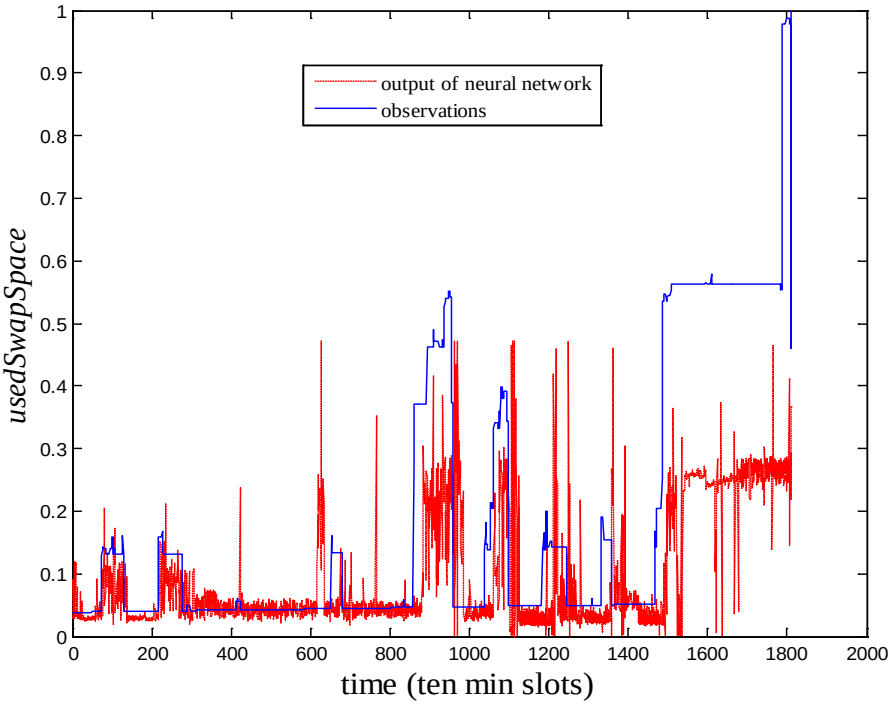


Fig. 3. Test results for *usedSwapSpace* (on the Rossby data set)

The test results on the 1,811 observations based on the Rossby data set for *realMemoryFree* and *usedSwapSpace* are shown in Figure 2 and Figure 3, respectively. The test results based on the Jefferson data set are shown in Figures 4 and 5. A visual analysis shows that, in all four figures, the paroxysmal spikes of the observations can generally be tracked well by the neural network. It is interesting to find that differences between the predicted values and the observations become large in the last parts of Figures 2 and 3. In Figure 2, the last part (starting from around the 1,650th observation point) of observed values of *realMemoryFree* kept very low for a relatively long time – this can imply software aging. The predicted values for this period of time, however, did not go down. This phenomenon is more evident in Figure 3, starting from around the 1,500th observation point, where the predicted values of *usedSwapSpace* did not increase as much as the observed values. This finding suggests that, in normal situations there is a close relationship between system workload and available resources; when the system gets seriously aged, however, the workload

will no longer have a significant impact on the available resources. This is because the system has already accumulated many error conductions, and reducing workload can no longer bring the system back to a normal state. See, for example, Figure 1 (a), which shows that the system workload was not at its peak level around the end of the observations, but Figure 2 shows that *realMemoryFree* reached its lowest level in the end of the observations. This observation shows that our approach can be used to diagnose software aging. As a result, the system was reset after the 1,811th observation point.

In addition to the above visual analysis, the test results are also quantitatively analyzed using RMSE and Pearson's Correlation Coefficient (r), as summarized in Table 2. For Figures 2, 3, 4 and 5, the correlation coefficients are 0.773, 0.716, 0.904, and 0.818, respectively. Because a Pearson's r greater than or equal to 0.5 indicates a strong correlation, and also because all four p-values are smaller than 0.001, we can conclude that there is a strong and statistically significant positive correlation between the observed values and predicted values in all four figures, which means that our approach is effective. This also means that there is a close relationship between workload and available resources. Table 2 indicates that the results on the Jefferson data set are better than those on the Rossby data set. This is because software aging in the Rossby workstation affected the prediction accuracy, as explained in the preceding paragraph. This inaccuracy, however, can be useful: it can help us to diagnose and identify software aging. For instance, Figure 6 (a) shows the trend of the (absolute) prediction errors of Figure 3 (that is, the differences between the observed and predicted values of *usedSwapSpace* on the Rossby data set), and Figure 6 (b) shows the corresponding errors of Figure 5 on the Jefferson data set. It is evident that, in Figure 6 (a), the prediction error increases with time owing to software aging in the Rossby workstation, whereas

Figure 6 (b) does not have this pattern as the Jefferson workstation did not get as aged. Identification of this kind of error pattern may help with the diagnosis of software aging.

Table 2. Quantitative analysis of the test results shown in Figures 2 to 5

Figure	RMSE	Pearson's correlation coefficient (r)	p-value (two-tailed)
Fig. 2	0.1400	0.773	p<0.001
Fig. 3	0.1856	0.716	p<0.001
Fig. 4	0.0338	0.904	p<0.001
Fig. 5	0.1036	0.818	p<0.001

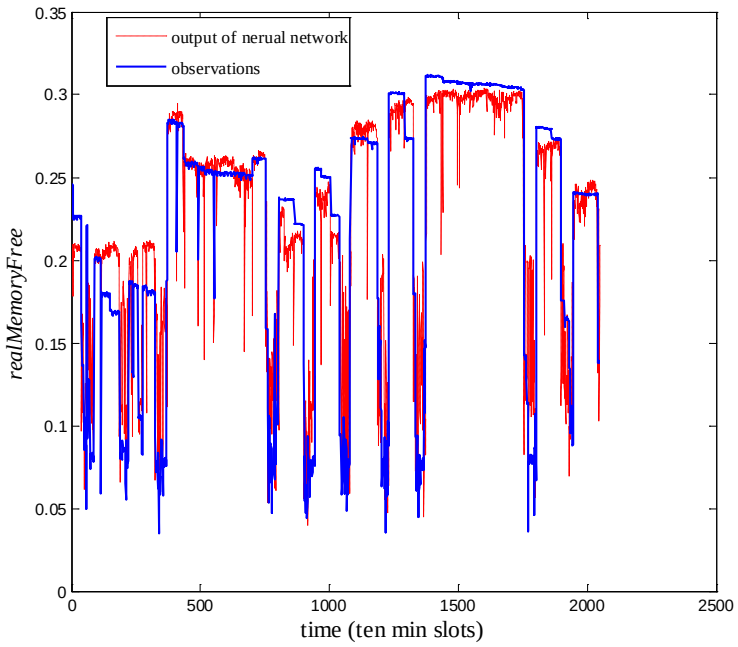


Fig. 4. Test results for *realMemoryFree* (on the Jefferson data set)

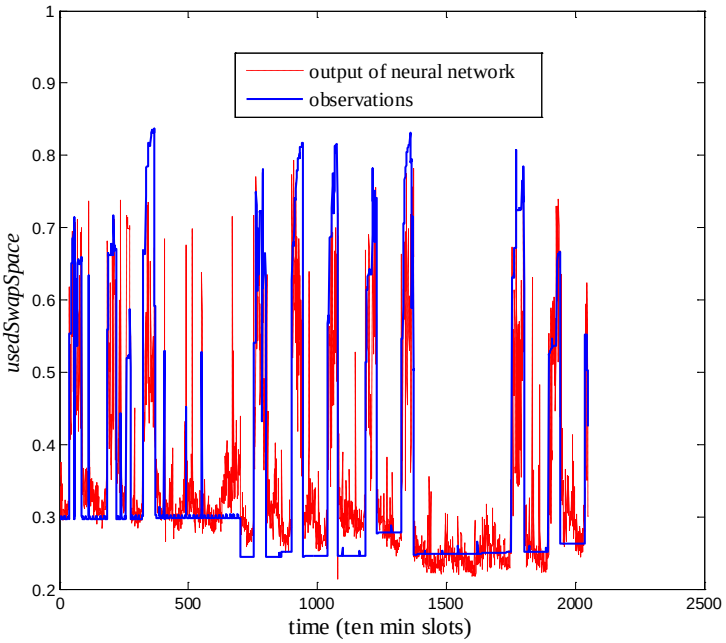
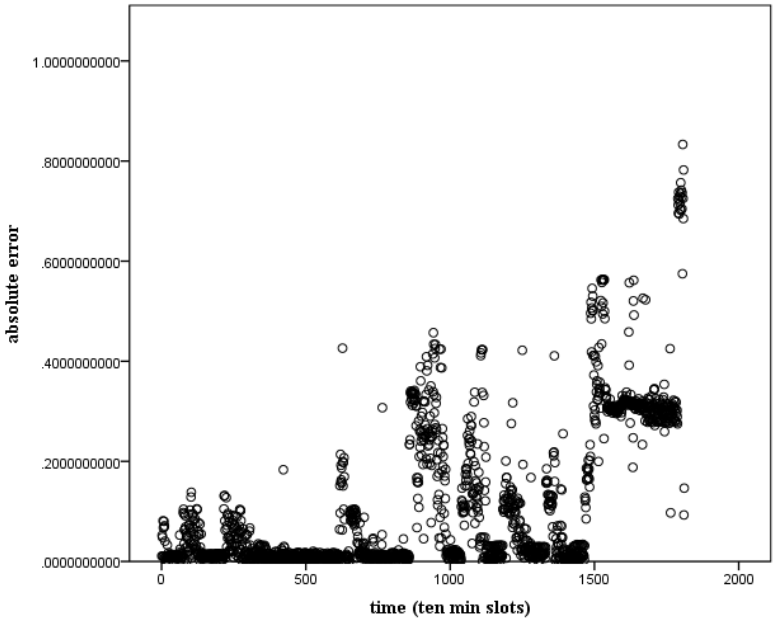
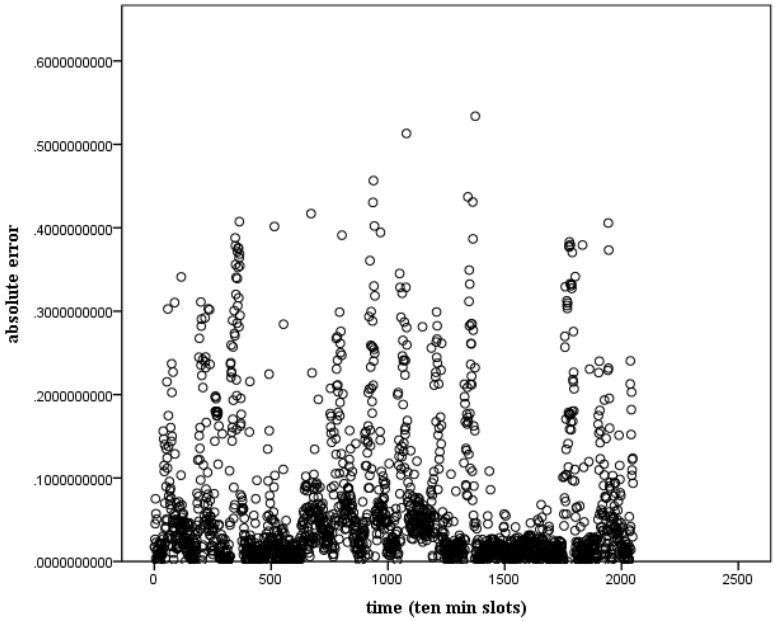


Fig. 5. Test results for *usedSwapSpace* (on the Jefferson data set)



(a) A growing trend of the absolute errors of prediction, based on Fig. 3 results



(b) Absolute errors of prediction, based on Fig. 5 results

Fig. 6. A comparison of the trend of prediction errors: Fig. 3 results vs Fig 5 results

V. FORECASTING FUTURE AVAILABLE SYSTEM RESOURCES

The previous section studied the relationship between system workload and available resources. A further question is: can we forecast future available system resources? The answer to this question will be of practical importance. In this section, we will continue to use the neural network approach to answer this question.

A. One-Step-Ahead Forecasting

To forecast available system resources using neural networks, we need to first decide which input parameters should be provided to the network. Intuitively, these parameters should be relevant to the usage of system resources. Therefore, to forecast the next-step value of *realMemoryFree*, we decided to use the following five parameters: $\Sigma_{cpuContextSwitch}$, $\Sigma_{sysCall}$, Σ_{pageIn}, *systemRunningTime*, and $C_{realMemoryFree}$. The first three parameters are the accumulated values

(accumulated from the last system startup to the present time) of *cpuContextSwitch*, *sysCall*, and *pageIn*, respectively. Note that accumulated rather than present values of the workload variables are used, and this is because of concerns of the impact of software aging: available system resources are not only related to the current system workload, but also related to accumulated error conditions caused by historical activities since the last system startup. For the same reason, a forth parameter *systemRunningTime* is included, which refers to the running time since the system was last started. The fifth parameter $C_{realMemoryFree}$ refers to the current value of *realMemoryFree*. This parameter is included because we believe that the value of *realMemoryFree* at time slot t_{i+1} is related to its previous value at time slot t_i. Note that we did not include the accumulated value of *realMemoryFree* as an input parameter because the current value of *realMemoryFree* has already reflected the effect of historical memory leaks. As an example of illustration, suppose the current time is 10:00 pm and the system was last restarted at 1:00 pm, to forecast the value of *realMemoryFree* at the next observation point (that is, 10:10 pm), our approach will use the following data: accumulated values of the three workload variables (accumulated from 1:00 pm to 10:00 pm), running time since the last startup (that is, nine hours and ten minutes), and the value of *realMemoryFree* at time 10:00 pm. Similarly, to forecast the next-step value of *usedSwapSpace*, we decided to use the following five parameters: $\Sigma_{cpuContextSwitch}$, $\Sigma_{sysCall}$, Σ_{pageIn}, *systemRunningTime*, and $C_{usedSwapSpace}$.

We first applied this approach to the Rossby data set. As explained previously, the whole Rossby data set can be divided into a few segments, each of which records observations from system startup to system reset. We used one of these segments, which contains 5,844 observations, to train the neural network, and another segments, which contains 1,594 observations, for testing. The test results based on the Rossby data set for *realMemoryFree* and *usedSwapSpace* are shown

in Figure 7 and Figure 8, respectively. Using visual analysis, we can find that, in both figures, the 1,594 forecasted values predict the trends of the observed values well. In other words, the neural network approach is effective.

We have also applied the same approach to the Jefferson data set, where 5,811 observations were used to train the neural network, and 3,016 observations were used for testing. The test results are shown in Figures 9 and 10. Using visual analysis, we can find that the observed values can generally be tracked well by the neural network in both figures. These results on Jefferson confirm that our approach is effective for forecasting available system resources.

The results of Figures 7 to 10 are further analysed quantitatively as summarized in Table 3. All four correlation coefficients are well above 0.9 with p-values below 0.001. This means that there is a strong and statistically significant positive correlation between the observed and predicted values in all four figures. In other words, our approach is effective.

Table 3. Quantitative analysis of the test results shown in Figures 7 to 10

Figure	RMSE	Pearson's correlation coefficient (r)	p-value (two-tailed)
Fig. 7	0.1118	0.935	$p<0.001$
Fig. 8	0.0838	0.940	$p<0.001$
Fig. 9	0.0275	0.992	$p<0.001$
Fig. 10	0.0439	0.975	$p<0.001$

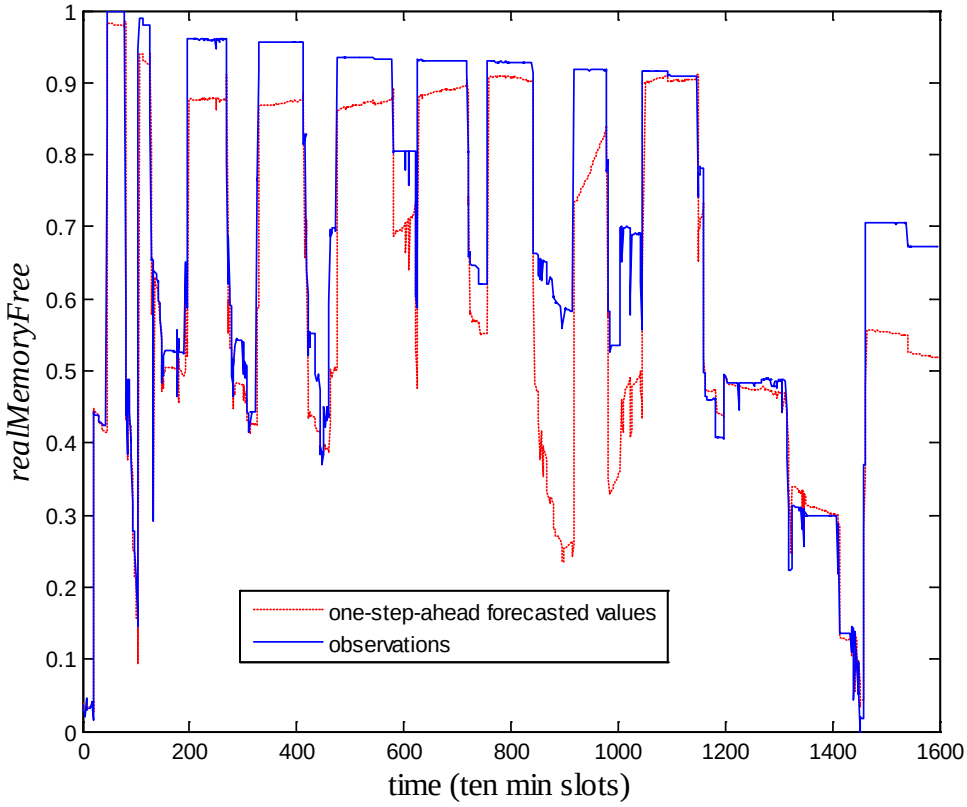


Fig. 7. A comparison of one-step-ahead forecasted and observed results for realMemoryFree (on the Rossby data set)

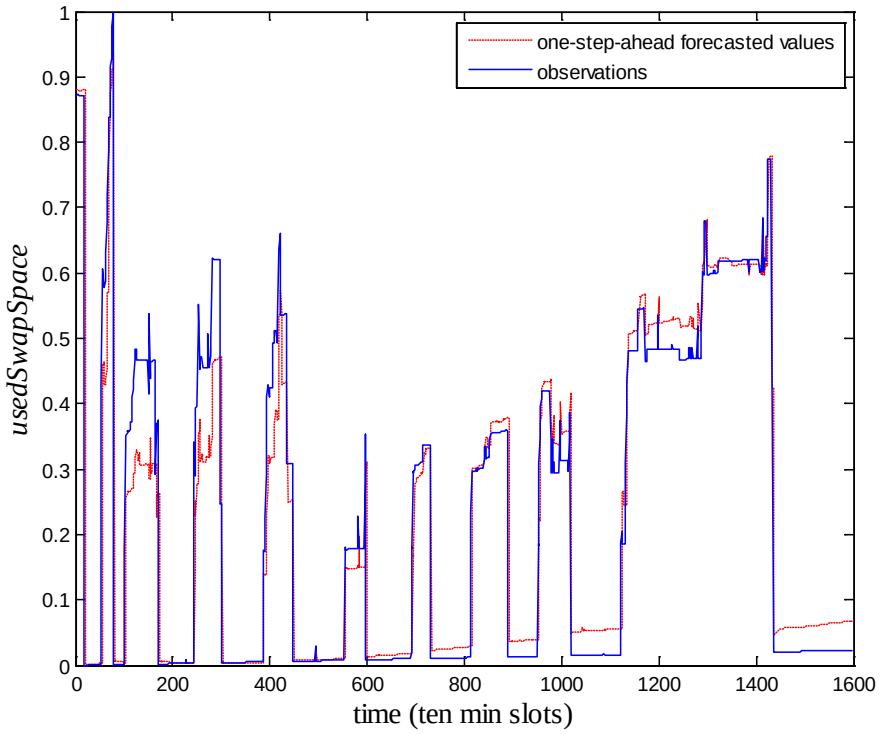


Fig. 8. A comparison of one-step-ahead forecasted and observed results for *usedSwapSpace* (on the Rossby data set)

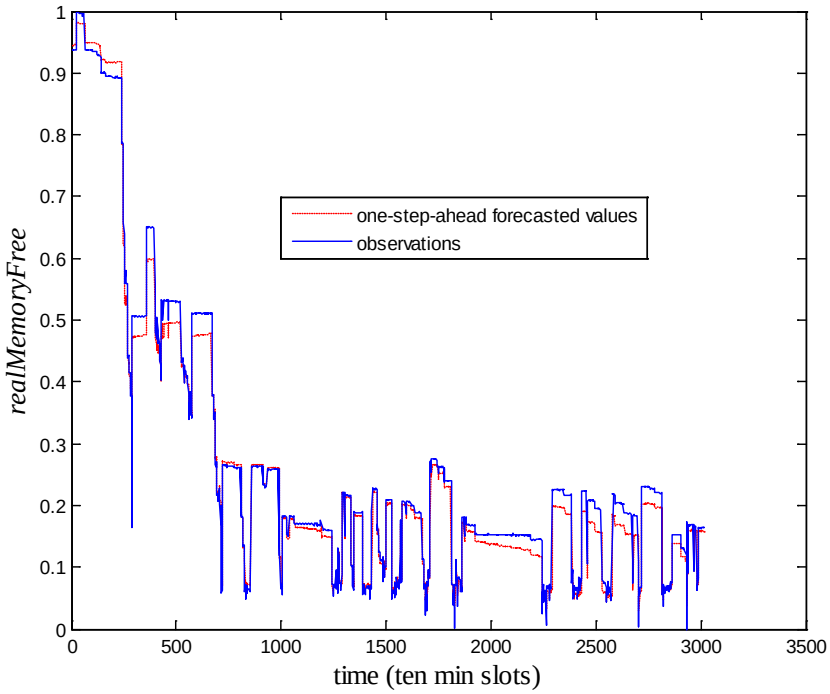


Fig. 9. A comparison of one-step-ahead forecasted and observed results for *realMemoryFree* (on the Jefferson data set)

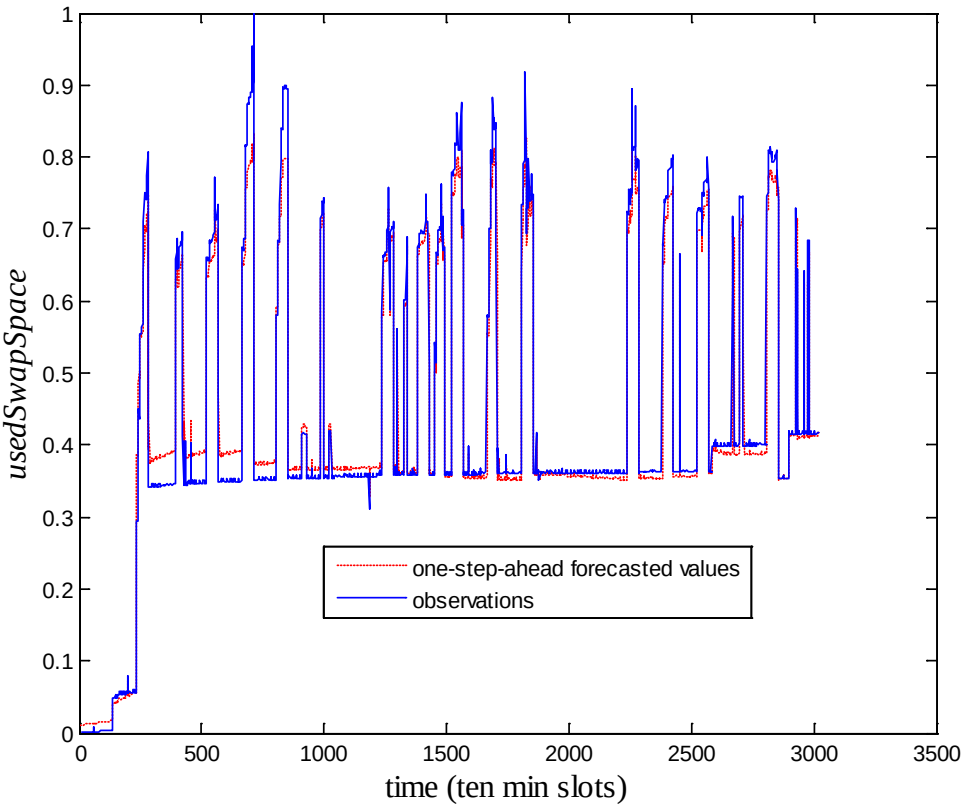


Fig. 10. A comparison of one-step-ahead forecasted and observed results for *usedSwapSpace* (on the Jefferson data set)

B. Two-Step-Ahead Forecasting

The last subsection discussed how to achieve one-step-ahead forecasting. It is natural to ask whether it is possible to go beyond one-step to achieve multi-step-ahead forecasting. In this subsection we will discuss two-step-ahead forecasting. Treatment for n-step-ahead forecasting, where $n>2$, is similar.

Suppose we are now at time slot t_i. To do two-step-ahead forecasting, that is, to forecast the values of *realMemoryFree* and *usedSwapSpace* at time slot t_{i+2}, we should be able to estimate the values of the five input parameters at time t_{i+1}. Without loss of generality, let us consider

realMemoryFree. Among the five parameters, *systemRunningTime* is known, and we can feed back the forecasted value of $C_{realMemoryFree}$ (for time slot t_{i+1}) to the network as its estimated value.

To provide estimated values for the remaining three parameters, namely $\Sigma_{cpuContextSwitch}$, $\Sigma_{sysCall}$, and Σ_{pageIn}, we applied the neural network to forecast each of these values. Without loss of generality, let us consider $\Sigma_{cpuContextSwitch}$. To forecast the value of $\Sigma_{cpuContextSwitch}$ at time slot t_{i+1}, we provided the network with four input parameters, namely the values of $\Sigma_{cpuContextSwitch}$, $\Sigma_{sysCall}$, and Σ_{pageIn} observed at time slot t_i, and *systemRunningTime*. We believe that all these four parameters can have an impact on the value of $\Sigma_{cpuContextSwitch}$ at time slot t_{i+1}. We applied the same data in our experiment, that is, 5,844 observations for training and 1,594 observations for testing using the Rossby data set; and 5,811 observations for training and 3,016 observations for testing using the Jefferson data set. The test results are excellent: the values of $\Sigma_{cpuContextSwitch}$, $\Sigma_{sysCall}$, and Σ_{pageIn} can all be quite accurately forecasted in both the Rossby and the Jefferson servers. An example of the test results is given in Figure 11. We note that it is much easier to forecast the accumulated values of the workload variables than to forecast their individual values at each time slot. This is because the curvature of accumulated values is much smoother than that of the individual values.

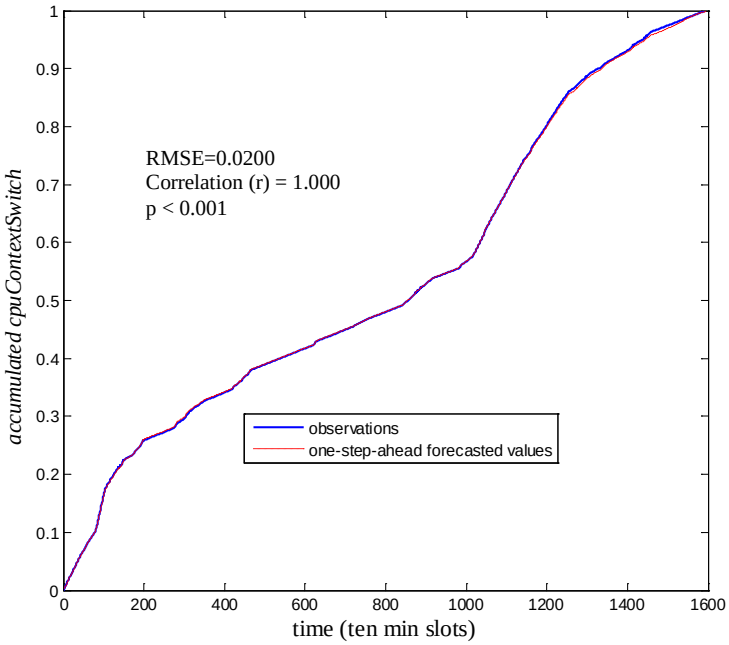


Fig. 11. A comparison of forecasted and observed results for $\Sigma_{cpuContextSwitch}$ (on the Rossby data set)

Having been able to provide the five input parameters (estimated values for time slot t_{i+1}), we are now able to conduct two-step-ahead forecasting. The same data used for one-step-ahead forecasting have also been used in our experiments. The test results based on the Rossby data set for *realMemoryFree* and *usedSwapSpace* are shown in Figure 12 and Figure 13, respectively. The test results based on the Jefferson data set are shown in Figures 14 and 15. Quantitative analysis of the test results for Figures 12 to 15 are summarized in Table 4.

Table 4. Quantitative analysis of the test results shown in Figures 12 to 15

Figure	RMSE	Pearson's correlation coefficient (r)	p-value (two-tailed)
Fig. 12	0.2755	0.894	p<0.001
Fig. 13	0.0918	0.933	p<0.001
Fig. 14	0.0384	0.988	p<0.001
Fig. 15	0.0549	0.962	p<0.001

We can compare the accuracy of one-step-ahead and two-step-ahead forecasting by comparing Figures 7, 8, 9, and 10 against Figures 12, 13, 14, and 15, respectively. A visual analysis can reveal that the accuracy of two-step-ahead forecasting is not as good as that of one-step-ahead forecasting. We can further compare the respective rows of Table 3 and Table 4: the RMSE values one-step-ahead forecasting increased from Table 3's 0.1118, 0.0838, 0.0275, and 0.0439 to Table 4's 0.2755, 0.0918, 0.0384, and 0.0549, respectively. Furthermore, all correlation coefficients decreased. In short, both visual and quantitative analyses show that the accuracy of two-step-ahead forecasting has dropped. Nevertheless, in all of the two-step-ahead forecasting results, the trends of the observed values can still be reasonably tracked, and there is still a strong positive correlation between the forecasted and observed values with a strong statistical significance. It can be expected that, for *n*-step-ahead forecasting, the accuracy will decrease when *n* increases.

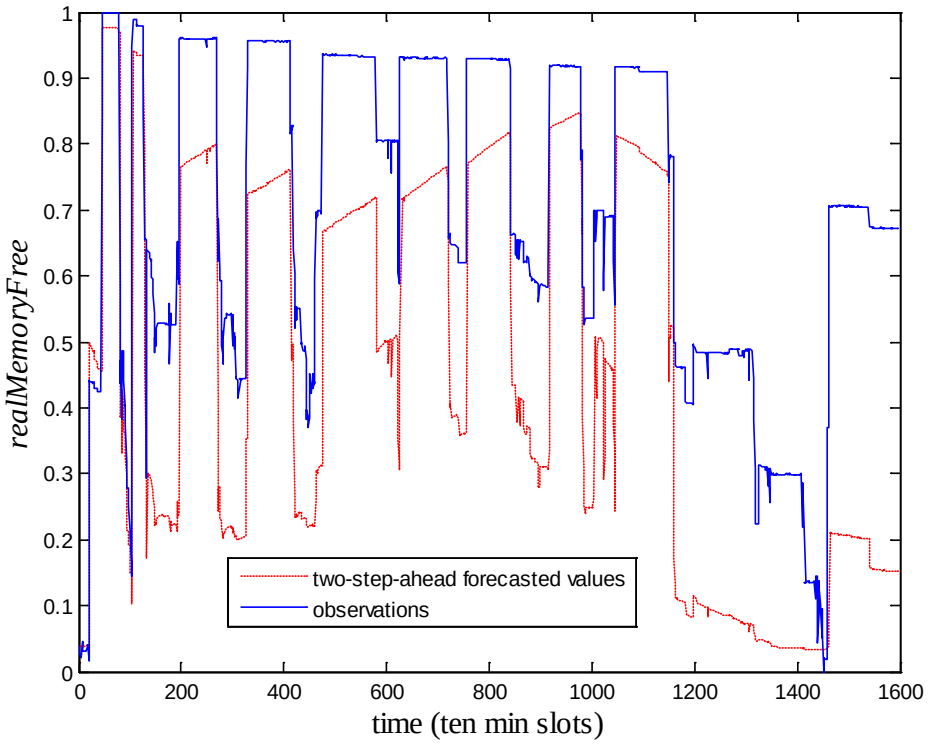


Fig. 12. A comparison of two-step-ahead forecasted and observed results for *realMemoryFree* (on the Rossby data set)

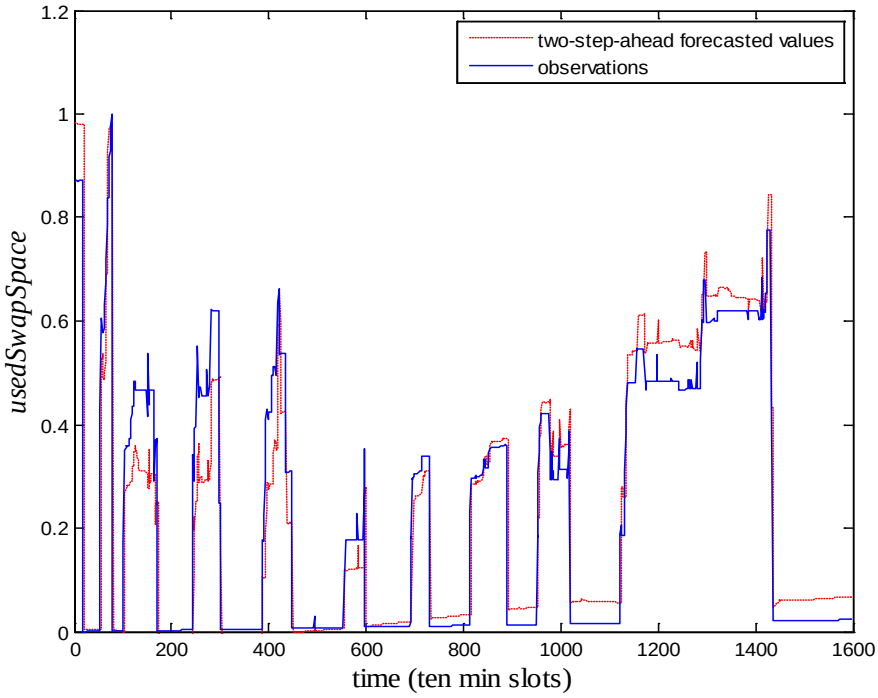


Fig. 13. A comparison of two-step-ahead forecasted and observed results for *usedSwapSpace* (on the Rossby data set)

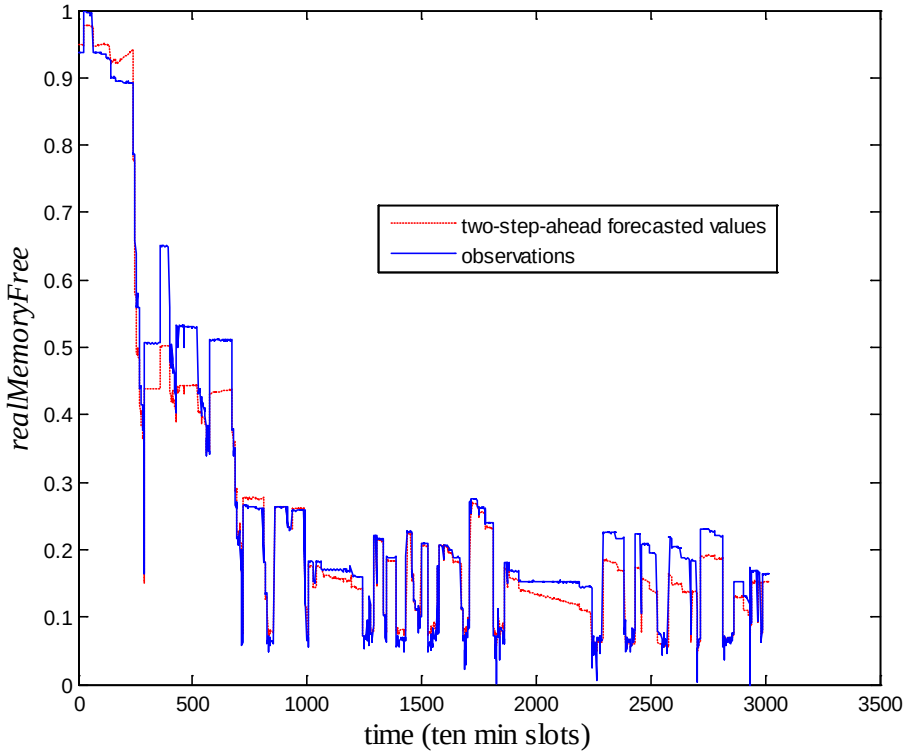


Fig. 14. A comparison of two-step-ahead forecasted and observed results for *realMemoryFree* (on the Jefferson data set)

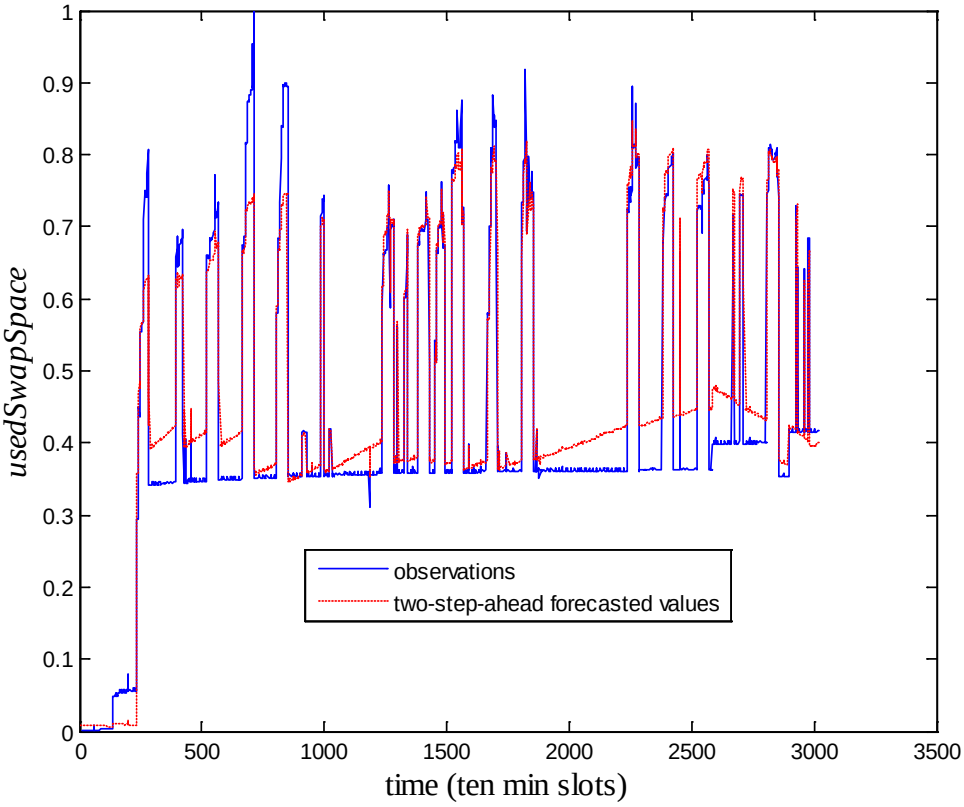


Fig. 15. A comparison of two-step-ahead forecasted and observed results for *usedSwapSpace* (on the Jefferson data set)

VI. Conclusion and Future Work

This is the first paper to present a method for quantitatively forecasting future available system resources under the real-world situation where system workload changes dynamically. We have also quantitatively investigated, for the first time, the relationship between changing system workload and available resources. The experimental results demonstrate that the proposed neural network approach is effective and, hence, contribute to software aging research.

Regarding the internal validity of this research, we have carefully verified all the programs, data, and the experiment procedures. Regarding the external validity, the proposed neural-network-based method has been applied to data sets from two servers, namely Rossby and

Jefferson, which provided different services with different configurations. The experimental results on the data sets from both servers show that the proposed method is effective. Nevertheless, the generalization of our approach is still threatened by the use of data sets from only two servers. Control for this threat can be achieved only through additional studies using different servers in different environments. Another future research topic is to apply the neural network approach to investigate the influence of other factors on software aging.

ACKNOWLEDGEMENTS

We would like to thank Kishor S. Trivedi and Michael Grottke for providing the data sets for this research.

REFERENCES

[1] A. Andrzejak, L. Silva, "Using machine learning for non-Intrusive modeling and prediction of software aging," in *Proc. 11th IEEE Network Operations and Management Symposium*, 2008, pp. 25-32.

[2] K. J. Astrom and B. Wittenmark, *Computer Controlled System – Theory and Design*, 3rd ed. Upper Saddle River: Prentice Hall, 1996.

[3] T.G. Barbounis, J.B. Theocharis, M.C. Alexiadis, and P.S. Dokopoulos, "Long-term wind speed and power forecasting using local recurrent neural network models," *IEEE Transactions on Energy Conversion*, vol. 21, no. 1, pp. 273-284, 2006.

[4] V.M. Becerra, F.R. Garces, S.J. Nasuto, and W. Holderbaum, "An efficient parameterization of dynamic neural networks for nonlinear system identification," *IEEE Transactions on Neural Networks*, vol. 16, no. 4, pp. 983-988, 2005.

[5] S. Bhama and H. Singh, "Single layer neural networks for linear system identification using gradient descent technique," *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 884-888, 1993.

[6] D. Bruneo, S. Distefano, F. Longo, A. Puliafito, and M. Scarpa, "Workload-based software rejuvenation in Cloud systems," *IEEE Transactions on Computers*, vol.62, no.6, pp.1072-1085, 2013.

[7] K.-Y. Cai, "Software reliability experimentation and control," *Journal of Computer Science and Technology*, vol. 21, no. 5, pp. 697-707, 2006.

[8] X.-E. Chen, Q. Quan, Y.-F. Jia, and K.-Y. Cai, "A threshold autoregressive model for software aging," in *Proc. 2nd IEEE International Workshop on Service-Oriented System Engineering*, 2006, pp. 34-40.

[9] T. Dohi, K. Goseva-Popstojanova, and K. S. Trivedi, "Statistical non-parametric algorithms to estimate the optimal software rejuvenation schedule," in *Proc. International Pacific Rim Symposium on Dependable Computing*, 2000, pp. 77-84.

[10] T. Dohi, K. Goseva-Popstojanova, and K. S. Trivedi, "Estimating software rejuvenation schedules in high assurance systems," *Computer Journal*, vol. 44, no. 6, pp. 473-482, 2001.

[11] H. El-Shishiny, S. Deraz, and O. Bahy, "Mining software aging patterns by artificial neural networks," *Lecture Notes in Computer Science* vol 5064, 2008.

[12] J. Gray and D.P. Siewiorek, "High-availability computer systems," *IEEE Computer*, vol. 24, no. 9, pp. 39–48, 1991.

[13] M. Grottke, L. Li, K. Vaidyanathan, and K. S. Trivedi, "Analysis of software aging in a Web server," *IEEE Transactions on Reliability*, vol. 55, no. 3, pp. 411-420, 2006.

[14] G. A. Hoffmann and K. S. Trivedi, "A best practice guide to resource forecasting for computing systems," *IEEE Transactions on Reliability*, vol. 56, no. 4, pp. 615-628, 2007.

[15] Y. Hong, D. Chen, L. Li, and K.S. Trivedi, "Closed loop design for software rejuvenation," in *Proc. Workshop on Self-Healing, Adaptive and Self-Managed Systems*, 2002.

[16] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359-366, 1989.

[17] Y. Huang, C. Kintala, N. Kolettis, and N. Fulton, "Software rejuvenation: analysis, module and applications," in *Proc. 25th IEEE International Symposium on Fault-Tolerant Computing*, 1995, pp. 381-390.

[18] Y.-F. Jia, X.-E Chen and K.-Y. Cai, "Chaotic analysis of software aging in Web server," in *Proc. 2nd IEEE International Workshop on Service-Oriented System Engineering*, 2006, pp. 117-120.

[19] Y.-F. Jia, L. Zhao, and K.-Y. Cai, "On the relationship between software aging and related parameters in a Web server," in *Proc. 8th International Conference on Quality Software*, 2008, pp. 241 – 246.

[20] Y.-F. Jia, L. Zhao, and K.-Y. Cai, "A nonlinear approach to modeling of software aging in a Web server," in *Proc. 15th Asia-Pacific Software Engineering Conference*, 2008, pp.77-84.

[21] Y.-F. Jia and K.-Y. Cai, "A feedback control approach for software rejuvenation in a Web server," in *Proc. 1st Workshop on Software Aging and Rejuvenation*, 2008.

[22] P. Killelea, *Web Performance Tuning*, 2nd ed. Sebastopol: O'Reilly Media, 2002.

[23] S. Kumar, *Neural Networks*, Beijing: Tsinghua University Press, 2006.

[24] R. Matias, P.A. Barbetta, K.S. Trivedi, P.J.F. Filho, "Accelerated Degradation Tests Applied to Software Aging Experiments," *IEEE Transactions on Reliability*, 2010, vol. 59, pp. 102-114.

[25] E. Marshall, "Fatal error: how patriot overlooked a scud," *Science*, vol. 255, no. 5050, pp. 1347, 1992.

[26] S.L. Phung and A. Bouzerdoum, "A pyramidal neural network for visual pattern recognition," *IEEE Transactions on Neural Networks*, vol. 18, no. 2, pp. 329-343, 2007.

[27] L.M. Saini and M.K. Soni, "Artificial neural network-based peak load forecasting using conjugate gradient methods, " *IEEE Transactions on Power Systems*, vol. 17, no. 3, pp. 907-912, 2002.

[28] M. Shereshevsky, J. Crowell, B. Cukic, V. Gandikota, and Y. Liu, "Software aging and multifractality of memory resources," in *Proc. International Conference on Dependable Systems and Networks*, 2003, pp. 721-730.

[29] D.I.K. Sjøberg, J.E. Hannay, O. Hansen, V.B. Kampenes, A. Karahasanovic, N.-K. Liborg, and A.C. Rekdal, "A survey of controlled experiments in software engineering," *IEEE Transaction on Software Engineering*, vol. 31, no. 9, pp.733-753, 2005.

[30] K. Vaidyanathan and K. S. Trivedi, "A comprehensive model for software rejuvenation," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, pp. 124-137, 2005.

[31] Z. H. Xia, W. Hao, and I. L. Yen, "A distributed admission control model for QoS assurance in large-scale media delivery systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 12, pp. 1143-1153, 2005.

[32] W. Xie, Y. Hong, and K.S. Trivedi, "Software rejuvenation policies for cluster systems under varying workload," in *Proc. 10th IEEE Pacific Rim International Symposium on Dependable Computing*, 2004, IEEE Computer Society Press.

[33] K.-X. Xue, L. Su, Y.-F. Jia, and K.-Y. Cai, "A neural network approach to forecasting computing-resource exhaustion with workload," in *Proc. 9th International Conference on Quality Software*, 2009, IEEE Computer Society Press.

[34] G. P. Zhang and M. Qi, "Neural network forecasting for seasonal and trend time series," *European Journal of Operational Research*, vol. 160, no. 2, pp. 501-514, 2005.

[35] J. Zhao, K. S. Trivedi, M. Grottke, J. Alonso, and Y. Wang, "Ensuring the performance of Apache HTTP server affected by aging," *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no.2, pp.130-141, 2013.

[36] J. Zhao, Y. Wang, G. Ning, K. S. Trivedi, R. Matias, and K.-Y. Cai, "A comprehensive approach to optimal software rejuvenation," *Performance Evaluation*, vol. 70, no.11, pp. 917-933, 2013.