A Neural Network Approach to Forecasting Computing-Resource Exhaustion with Workload*

Ke-Xian Xue[1,2], Liang Su[1], Yun-Fei Jia[1] and Kai-Yuan Cai[1]
[1]Department of Automatic Control
Beijing University of Aeronautics and Astronautics
Beijing, China, 100191
[2]Command and Engineering Institute of Chemical Defense, PLA,
Beijing, China, 102205
e-mail: jiayunfei@asee.buaa.edu.cn

Abstract—**Software aging refers to the phenomenon that applications will show growing failure rate or performance degradation after longtime execution. It is reported that this phenomenon usually has close relationship with computing-resource exhaustion. This paper analyzes computing-resource usage data collected on a LAN, and quantitatively investigates the relationship between computing-resource exhaustion trend and workload. First, we discuss the definition of workload, and then a Multi-Layer Back propagation neural network is trained to construct the nonlinear relationship between input (workload) and output (computing-resource usage). Then we use the trained neural network to forecast the computing-resource usage, i.e., free memory and used swap, with workload as its input. Finally, the results were benchmarked against those obtained without regard to influence of workload reported in the literatures, such as non-parametric statistical techniques or parametric time series models.**

Keywords-software aging; neural network; computing-resource exhaustion; workload parameters

I. INTRODUCTION

Availability and reliability of computer systems is an issue of paramount importance that has received special attention from the industry and academia in the last decades. More often than not, system failures are attributed to software than hardware [8].

When an application server executes continuously long period of time, it accumulates many error conditions in its process space or kernel space, such as memory leak, round-off errors, out-of-order processes or threads, file table not released. Eventually, these error conditions will result in critical results, such as exhaustion of computing-resource[1] of computer system, paroxysmal crash, increased response time or degraded performance. This phenomenon is called software aging, which will bring many disastrous results. A real life example is the loss of human life due to aging in safety-critical system--Patriot's software in 1991 [20]. It is considered the root causes of software aging come from residual defects in the software system [9,12], because it is impossible to guarantee the released applications have not contained any defects. Although researchers have proposed many assumptions about its causes and evolvement [4,11,12,13], the influencing factors of software aging are still not well identified. This fundamental question can only be answered by experimental research.

Although several experimental studies on software aging have been reported, they are not the majority of software aging research. So far, less than ten publications [9,10,22] discussing experimental studies on software aging can be found on major software and reliability journals. This is contrast with the growing awareness and widely accepted importance of experiment-based studies. Grottke et al. forecast the computing-resource usage with an Autoregression model [9]. Their experiment employs even workload input for purpose of ease of analysis and/or detection of aging phenomenon. Unfortunately, the workload of application subject to is often uneven. A question naturally arises is that HOW TO FORECAST RESOURCE EXHAUSTION BASED ON WORKLOAD INFORMATION. Kalyanaraman Vaidyanathan and Kishor S. Trivedi has noted this lack, and make effort to combine workload as a factor in their software aging model. They cluster the workload into eight clusters, and then calculate the computing-resource exhaustion rate with respect to each workload cluster. Their finding is that computing-resource exhaustion rate is faster with higher workload [22]. Nevertheless, their finding is only for modeling purpose rather than forecasting resource exhaustion with varying workload. Moreover, their proposed metric "resource exhaustion rate" is still of weakness. More specifically, in many workload states the dynamics of the resources demonstrates very high variance, which results in very broad confidence intervals of their resource exhaustion rate. The highly irregular and oscillatory behavior of the data makes most trend models insufficient.

To sum up, a question not answered in previous work is that, given an arbitrary workload at any time, can we estimate the computing-resource usage and the risk of

* This work was supported by the National Natural Science Foundation of China and Microsoft Research Asia (Grant No. 60633010)

[1] Computing-resource refers to the various available resources of operating system, such as *Real Memory Free*, *CPU usage rate*, *Used Swap Space*, *I/O usage*, etc. In this paper, we focus on the *Real Memory Free* and *Used Swap Space*.

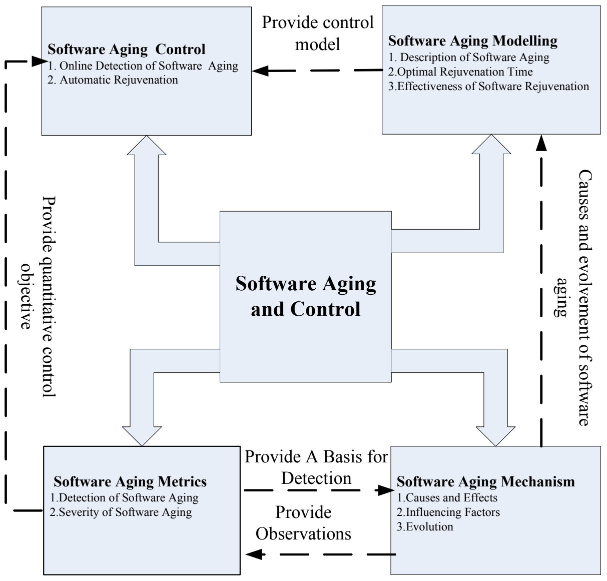
computing-resource exhaustion? This paper is aimed to construct a model between workload (input) and computing-resource usage (output). Our work is meaningful because the workload can be monitored directly, with which the risk of computing-resource exhaustion can be calculated by our model in a straightforward manner. Moreover, our model can provide a basis for how many connections to web sever should be cut off to avoid computing-resource exhaustion based on the admission control reported in [23].

In this work, we investigate the use of neural network based forecasting of computing-resource exhaustion. More specifically, we discuss the definition of workload on Operating System first. Then neural network are used to forecast *Used Swap Space* and *Real Memory Free* of Operating System. Finally, the results are cross benchmarked against those reported in the literature based on parametric and non-parametric statistical techniques and other empirical modeling techniques.

The rest of this paper is organized as follows. Related studies are provided in Section 2. The datasets and observations are illustrated in Section 3. In section 4, neural network approach is introduced. Section 5 applies our approach to *Used Swap Space* forecasting. Section 6 applies our approach to *Real Memory Free* forecasting. In Section 7, we validate the effectiveness of our approach by applying it to dataset 2. Section 8 concludes this paper.

II. Related Studies

Software aging was first reported in Patriot Missile system in [20]. This problem is solved by resetting the Patriot Missile system every 8 hours. Huang et al. construct a model of software aging and propose a counteraction called Software Rejuvenation for the first time. The studies on software aging and control can roughly be classified into four parts: software aging mechanism, software aging metrics, software aging modeling and software aging control, which is illustrated in Figure 1. Software aging mechanism research focuses on causes and effects, influencing factors and evolution of software aging. It provides a basis for extracting a metric of software aging and/or provides observations for modeling research. The objective of software aging metrics research lies in detection and estimation of the severity of software aging. It provides quantitative metrics for mechanism research and control objective for control research. Software aging modeling can formulate the aging process based on observations from experiments and can determine the effectiveness of software control. Further, it can provide a model for aging control research. Software aging control is the ultimate objective of software aging. It ultimate objective is to online detect and estimate the severity of software aging, and select optimal rejuvenation policy to heal the aged software of interest.


Software Aging Control
1. Online Detection of Software Aging
2. Automatic Rejuvenation
Provide control model
Software Aging Modelling
1. Description of Software Aging
2.Optimal Rejuvenation Time
3.Effectiveness of Software Rejuvenation
Provide quantitative control objective
Software Aging and Control
Causes and evolvement of software aging
Software Aging Metrics
1.Detection of Software Aging
2.Severity of Software Aging
Provide A Basis for Detection
Provide Observations
Software Aging Mechanism
1.Causes and Effects
2.Influencing Factors
3.Evolution
Figure 1 Overview of software aging research

Few literatures discussing the nature of software aging, or called mechanism of software aging, are reported. This is due to the failure-prone, intelligent-intensive and/or labor-intensive process of mechanism research. Matias et al. use design of experiment (DOS) technique to characterize the aging phenomenon. They find the “page size” and “page type” factors were responsible for over 99% of memory size variation in httpd processes [19]. Jia et al. analyze the evolution of software aging in Apache httpd and report that the aging process is chaotic and can only be forecasted in limited ahead time [13]. Shereshevsky et al. [21] monitors the Hölder exponent (a measure of the local rate of fractality) of the system parameters and find that system crashes are often preceded by the second abrupt increase in this measure.

To the best of our knowledge, there are only two metrics proposed to measure the degree of software aging. The reason lies in that software aging phenomenon are complicated and related to many factors. It is difficult to propose a metric reflecting all characteristics. For example, software aging phenomenon are characterized by consistent throughout loss, response time increase or computing-resource exhaustion respectively [4,9,14]. A metric “estimated time to exhaustion” is proposed to predict the approximate time of system resource depletion [9]. Nevertheless, their metric only considers two resources of system, i.e., free memory and used swap, which cannot represent the overall aging severity. A comprehensive evaluation function is proposed in [14] to measure the aging speed of the Apache server. The proposed metric abstracts seven important computing-resource usage parameters of system into two primary components via Principal Component Analysis (PCA) method. Both primary components are then combined to a hybrid metrics, namely the Z-metric to represent the average aging speed along the runtime of the system.

Software aging modeling is the majority of software aging research. Software aging modeling begins with making assumptions about the mechanism of software aging (including the causes and the effects of software aging) and constructing mathematical models to describe the process of software aging. Their contributions lie in validating the effectiveness of software rejuvenation and the optimal schedules for software rejuvenation. K. Vaidyanathan, et al. [22] monitors the system activity of Operating System, describe the workload based on four important parameters (*cpuContextSwitch*, *sysCall*, *pageIn*, *pageOut*) through clustering method. Then they describe the aging process with a semi-Markov reward model. In [12], Huang et al. 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 many researchers to answer similar questions [5,6]. Chen et al. introduce a threshold to judge the current pattern of software aging to improve the accuracy of forecasting and describe the nonlinear phenomenon of software aging [3]. Above models are all Markov models, [15] introduces a nonlinear model to describe the evolution of software aging. [7] exploits neural network to mining the patterns of resource usage with parameters of Apace httpd. But their work neglect the influence of workload as well. To sum up, the influence of workload on risk of computing-resource exhaustion is not described quantitatively.

Hong et al. propose an idea of closed-loop design of software rejuvenation to reset the computer system based on the feed back information [11]. Nevertheless, their objective is to determine the optimal rejuvenation time with the feedback information. On the other hand, Jia et al. introduce control theory to software rejuvenation, including how to apply system identification, controller design and evaluation to software rejuvenation [16].

This paper falls into software aging modeling, with the purpose of quantitatively describing the relationship between workload and risk of computing-resource exhaustion.

III. Datasets of Software Aging

A. Data collection

We employ the data reported in [22] to carry on further analysis using a neural network approach. The data collection process can be illustrated as follows. The SNMP (Simple Network Management Protocol)-based distributed resource monitoring tool developed by Garg et al. was used for data collection. The resource monitoring tool was used to collect operating system computing-resource usage data (physical/virtual memory usage, file/process table usage, etc.) and system activity data (paging activity, CPU utilization, etc.) from nine heterogeneous UNIX-like [2] workstations. These workstations were connected by an Ethernet LAN at the Duke Department of Electrical and Computer Engineering. These workstations provide various services, and inputs from clients are unknown. The objects or parameters collected on the workstations 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. More than 100 such parameters were collected at regular intervals (10 min) for more than three months.

In this paper, we focus on the data collected from the workstation named Rossby. Then we apply our approach to the data of Jefferson as cross-validation. It should be noted that the data of Jefferson is not illustrated and studied in [22].

B. Definition of Workload

http connection rate, i.e., the number of http requests coming from clients per unit time, is not a good measure of workload. There are several reasons. First, the http requests may be CPU-intensive or I/O-intensive. For example, static html page request is more often I/O-intensive than CPU-intensive because they rarely involve computation [17]. However, dynamic request, such as querying a database, will involves much CPU usage. The connection rate cannot reveal this fact, because both types of requests will result in different bottleneck of system. Second, even if the type of http requests is identical, say I/O intensive, request of large html files will cost more resources and time than small html files. Finally, the data is collected from real software runtime rather than controlled experiment, i.e., the inputs coming from the clients, the configuration of Operating System are both not controlled. To sum up, it's insufficient to describe workload by http connection rate.

The system workload was characterized by obtaining a number of variables pertaining to CPU activity and file system I/O. In this study, we use the following variables to characterize the workload:

cpuContextSwitch: The number of process context switches performed during the measurement interval (10 min in our case).

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. These variables are thus used to define and characterize the system workload.

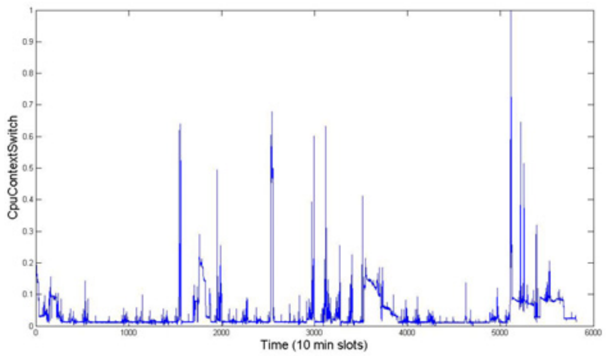
It should be noted that the definition of workload is slightly different from that in [22]. We discard pageOut, because we find this variable is nearly constant in our both datasets. Moreover, fewer variables will result in a simpler model which can be generalized more conveniently.

C. Pretreatment of datasets

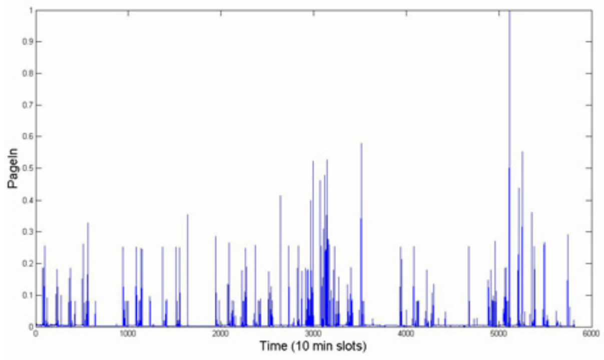
Due to our data are collected at real software runtime, there is not any controlled variable. These workstations may be rebooted for some reasons. System reboot may be

[2] The Unix-like means variants of Unix the workstation are running on, such as SunOS and Linux.

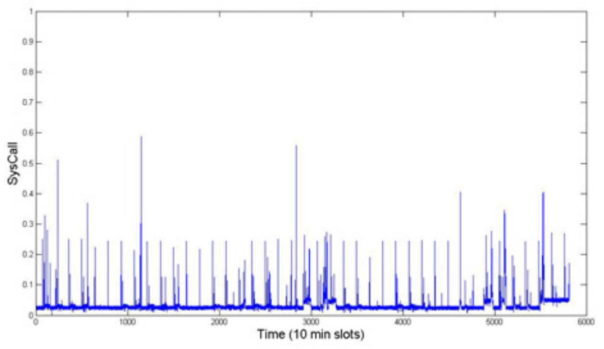
identified by the **accumulated sysCalls.** When this parameters drops to zero, the workstation is rebooted at this time. Note that there are several times of reboots during data collection, only the data between two successive reboot operations are considered in this paper. We select the longest interval during which the workstation operates without reboot. In addition, the unit of elements of our workload vector, say cpuContextSwitch and sysCall, are in different order of magnitude. Thus, for modeling purpose, the first work is to normalize them. The pretreated data are shown in Figure 2 and Figure 3.



(a)

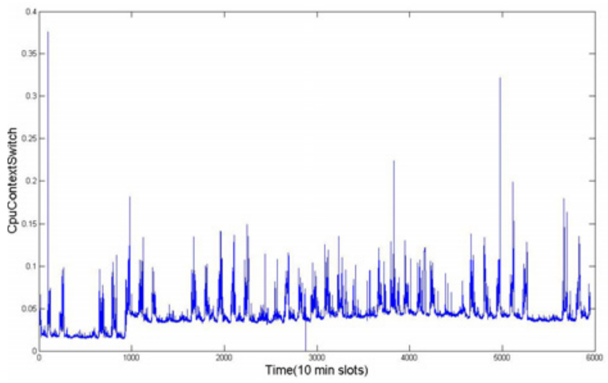


(b)

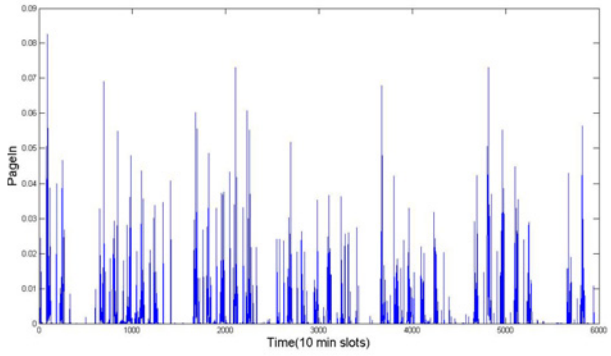


(c)

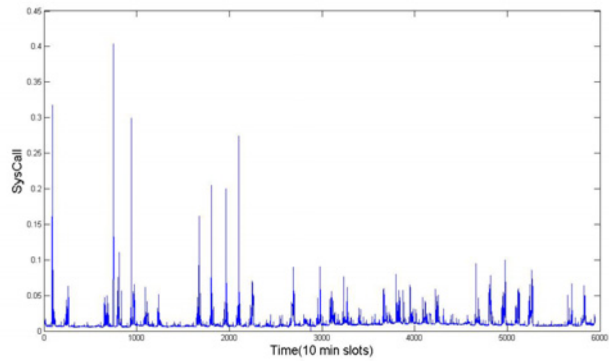
Figure 2. Workload of Rossby



(a)



(b)



(c)

Figure 3. Workload of Jefferson

In Figure 2 we can see there are many spikes reflected by each variable. There are not obvious trend shown in Figure 2 (a), Figure 2(b) and Figure 2(c). However, there is an obvious trend shown in Figure 3(a) and Figure 3(c). Also, Figure 3 (a), Figure (b) and Figure 3(c) show many spikes.

IV. THE NEURAL NETWORK APPROACH

In this section, we first give a brief description of neural network. Then, a more detailed discussion of the back-propagation (BP) neural network is provided.

A. *Introduction of Neural Network*

Neural network is mathematical model that can learn and mimic human movement behavior. A neural network is composed of many simple elements called neurons. Neurons are connected together with weights on the connections so that they can process information collaboratively and store the information on these weights. Although many types of neural network models have been proposed, the most popular one is the Multi-Layer Perceptron (MLP) feed forward model, which is intrinsically composed of non-linear, non-parametric approximators. The advantages of neural network can be summarized as follows:

1) They can capture the nonlinear phenomenon;
2) They have the ability to solve problems that do not have an algorithmic solution or the available solution is too complex to be found;
3) They will show better performance with time because more patterns will be learnt. In fact, neural network has been successfully applied to many areas such as pattern recognition, system identification, and forecasting [1] and intelligent control [2].

Choosing an appropriate network architecture (which means the number of layers and the number of hidden neurons per layer) is an issue of paramount importance. This question can only be answered by experience than mathematics. In our case, we select a three-layer neural network, in which two hidden layers are employed, as is shown in Figure 4. There are several reasons to adopt two hidden layers from perspective of the working principle of neural network. First, using a single hidden layer is that the neurons tend to interact with each other globally. Such interactions can make it difficult to generate approximations of arbitrary accuracy. The advantages of using two hidden layers lie in that the first hidden layer can learn the local features that characterize specific regions of the input space. Global features are extracted in the second hidden layer [18]. It should be noted that certain complex problems require the use of more than two hidden layers, but two hidden layers are sufficient for our case. Another important question should be noted when applying neural network approach is selection of training algorithm.

Back propagation error (BP) is most useful for feed-forward network [24]. Usually, Multi-Layer Perceptron feed forward model, combined with *Backpropagation error* training, are usually called BP neural network. In next subsection, we will illustrate the working principle of BP neural network.

B. *The BP Neural Network*

A BP neural network is a kind of feed forward neural network. Neurons in a BP neural network are connected only with the neurons in adjacent layers. A BP network can learn a complicated nonlinear input-output relationship from a set of sample data (including inputs and the corresponding expected outputs).

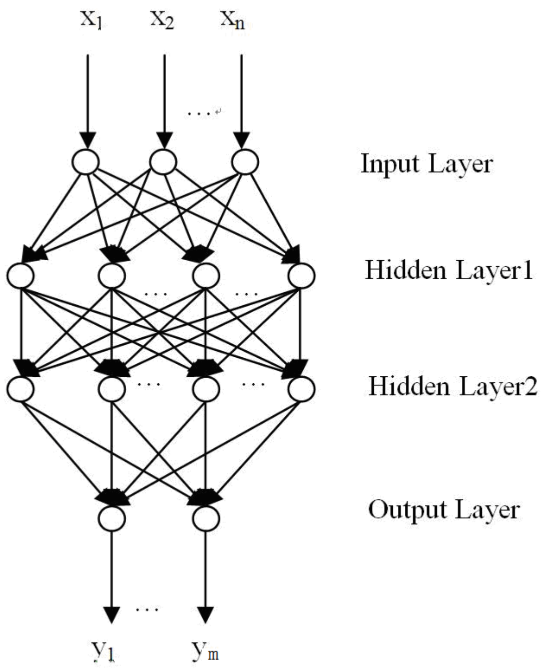


Figure 4. Structure of a BP neural network

Figure 4 shows the structure of a three-layer BP neural network. The output of first network is the input to the second network, and the output of the second network is the input to the third network. Each layer may have a different number of neurons, and even a different transfer function. The input layer has *n* linear neurons that receive real valued external inputs in the form of an n-dimensional vector in R^n . Similarly, the hidden layer has *q* sigmoidal neurons that receive signals from the input layer. The output layer comprises *p* sigmoidal neurons. Neuron layers compute in a strictly feedforward fashion-signals from one layer of neurons act as inputs to the next layer, and so on. Finally, the network signals that emanate from the last layer of neurons comprise a *p*-dimensional vector of real numbers. The neural network thus maps a point in R^n (the input space) to a point in R^m (the output space).To learn such a mapping , the network is provided with a training set of discrete data samples that comprise input-output vector pairs that describe an unknown function.

An error back-propagation algorithm is used in the training of a BP neural network. After the neural network is set up, the BP algorithm uses a set of sample data to train the network by the following steps:

1) Input the sample inputs to the neural network to generate the actual outputs.
2) Calculate the errors between the actual outputs and the expected outputs specified in the sample data. Then, propagate the errors backward to the input layer. In the backward process, the weights

on connections are changed so that the errors are small enough to satisfy the stopping criteria.

In this paper, a BP neural network is employed for computing-resource exhaustion because of the following considerations:

1) BP neural network is capable of approximating complex nonlinear functions. For example, they can be used to simulate the relationship between workload and computing-resource usage;
2) BP neural network is trained by a supervised training algorithm (e.g., an error back-propagation algorithm). Since in computing-resource exhaustion forecasting, we know the expected output of each sample input (i.e., how much the output (computing-resource usage) jumps with respect to input (workload) increase), a supervised neural network is more suitable for the multi-dimensional relationship construction problem.

It is important to correctly choose a set of initial weights for the network. Sometimes it can decide whether or not the network is able to learn the training set function. It is common practice to initialize weights to small random values within some interval [-e e]. Initialization of weights of the entire network to the same value can lead to network paralysis where the network learns nothing-weight changes are uniformly zero. Further, very small ranges of weights randomization should be avoided in general since this may lead to very slow learning in the initial stages.

V. FORECASTING OF USED SWAP SPACE

In this section, we apply our approach to *Used Swap Space* of Rossby. Then we compare the accuracy against other modeling techniques.

A. Results of Used Swap with Workload

The *Used Swap Space* of Rossby is shown in Figure 5, from which we can see that the signal includes many spikes. These spikes correspond to those shown in Figure 2 with slight ahead time or lag. There is a slight growing trend shown in Figure 5, showing sign of software aging. In addition, the signal of *Used Swap Space* shows strongly nonlinear sign.

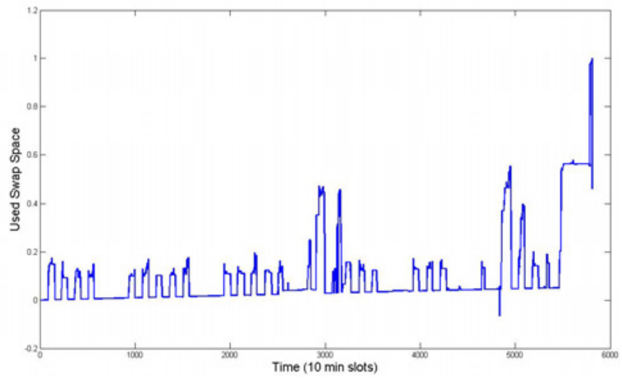


Figure 5 *Used Swap Space* of Rossby

First, we need to train our neural network with some samples from the observations of Rossby. There are 5811 observations included in our modeling. This dataset is partitioned into two subsets, the first subset is used for training our BP neural network, and the latter subset is used for testing our approach. More specifically, the dataset T is partitioned into $T_{training}$ and T_{test}. $T_{training}$ is used for estimation and evaluation of the network. T_{test} is employed to test the performance of the network. To tune the weights of the network, $T_{training}$ is divided into $T_{learning}$ and $T_{validation}$. $T_{learning}$ is the set of patterns that is used to actually train the network using backpropagation. At the end of epoch, the network performance is evaluated using $T_{validation}$. We stop the training process when the error on $T_{validation}$ starts to rise [19].

If learn rates are small enough, then convergence to local minimum in question is guaranteed. However, very small learning is nonuniform, and we stop before the network is trained to an error minimum, some weights will have reached their final "optimal" values; others may not have. In such a situation , the network might perform well on some patterns and very poorly on others. If we assume that the error function can be approximated by a quadratic then we can make the following observations. An optimal learning rate will reach the error minimum in a single learning step.

As a result, the first 4000 observations are used as $T_{training}$ and the remaining 1811 observations are used as T_{test}. As we discussed earlier in Section IV, multilayer network can be used to approximate almost any function, if we have enough neurons in the hidden layers. However, we cannot say, in general, how many neurons are necessary for adequate performance. It's a trial to determine the number of neurons. In our case, we find twenty neurons can output accurate results. The output of our model and the observations are compared in Figure 6.

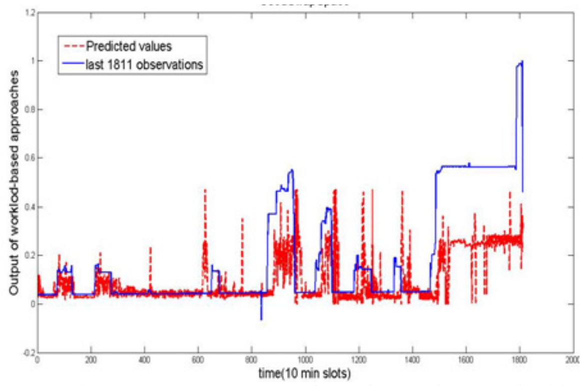


Figure 6. *Used Swap Space* of neural network approach with workload

From Figure 6 we can see that the paroxysmal spikes of *Used Swap Space* can be tracked by our model. However, we can see large difference between the predicted values and the observations shown in the last part, i.e., the right part of Figure 6. This can be attributed to several reasons. First, this is a new pattern which has not learnt by our neural network. Second, the factors influencing computing-resource usage are not completely included in our model, only workload cannot account for all influence factors. However, the neural network will have better performance with time, because it will learn more patterns.

We note that the previous literature on software aging had never explicitly modeled the relationship between workload and computing-resource usage of system. Neglecting such patterns may result in straightforward but inaccurate predictions, and hence in suboptimal decisions regarding when to rejuvenate. Our results are encouraging, as they seem to indicate that a relatively simple and easy-to-use approach can produce adequate forecasts.

B. *Comparison with Other Models*

In this part, we apply AR model which is employed by M. Grottke [9] to our dataset and compare it with neural network approach. Again, to illustrate the importance of using workload, we apply the neural network without workload parameters to our dataset, and then compare it against the neural network models with workload input. The AR model is shown in Figure 7, and the neural network without workload input is shown in Figure 8.

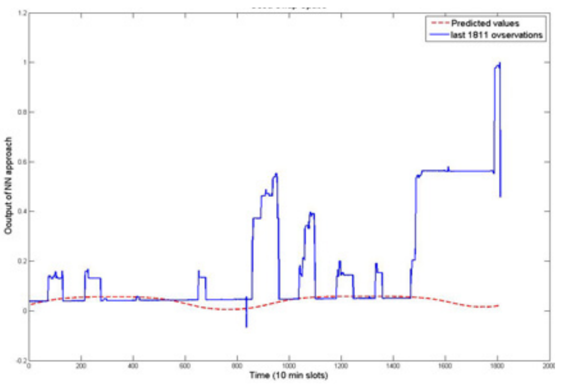


Figure 7 *Used Swap Space* of AR model

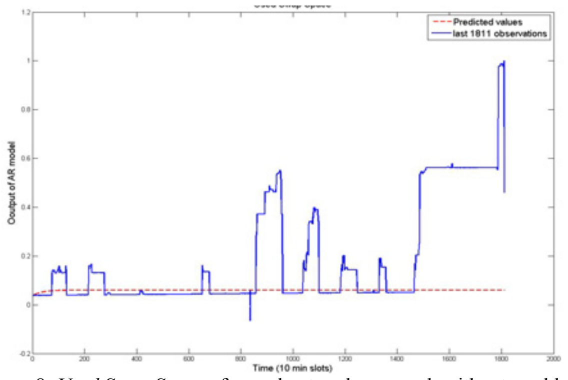


Figure 8. *Used Swap Space* of neural network approach without workload

Figure 7 and Figure 8 show the prediction of *Used Swap Space* by AR model and neural network approach without workload parameters respectively. It is clear that the both methods are not effective.

Neural network consists of layers of interconnected nodes which combine the data in a way to minimize root mean squared error (RMSE), but the researcher may also use some other minimizing criteria such mean absolute percentage error (MAPE) and Symmetric Mean Absolute Percentage Error (SMAPE). One simple example of a network is a pyramid type structure where each brick represents a node. Raw information is fed at the bottom of the pyramid where each node independently processes information and then transmits output, weighted by the importance of the node in question, to all the nodes sitting in the layer above. The nodes in this new layer then process the already processed- data and then pass on their weighted outputs to nodes on the layer above. This process continues until the node at the top of the pyramid finally transmits output of interest to the researcher. The final output is then checked against a RMSE criterion and if the criterion is not met, learning happens by taking into consideration the size of the error and a rule which allows adjusting initial weights assigned to each node and each layer in the pyramid. One key point that deserves mentioning is that each node is equipped with a combination function which combines various data points into a single value using weights. These single values are then transformed into the unit circle normally using a trigonometric function.

Thus, the training and forecasting accuracy is measured by Root Mean Square Error (RMSE) and two other common error measures, MAPE and SMAPE.

Root Mean Square Error (RMSE). RMSE calculates the root mean square error between the actual *y* and the forecast *e* across all observations *t* of the test set of size *n*. It can be calculated as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_t - e_t)^2}{n}} \tag{1}$$

Mean Absolute Percentage Error (MAPE). MAPE is calculated by averaging the percentage difference between the fitted (forecast) line and the original data:

$$MAPE = \frac{1}{n}\sum_t |(e_t - y_t)/y_t| \tag{2}$$

Where *y* represents the original series and e the original series minus the forecast, and *n* the number of observations.

Symmetric Mean Absolute Percentage Error (SMAPE). SMAPE calculates the symmetric absolute error in percent between the actual *y* and the forecast *e* across all observations *t* of the test set of size *n*. It can be formulated by:

$$SMAPE = \frac{1}{n}\sum_t \frac{|y_t - e_t|}{(y_t + e_t)/2} \tag{3}$$

TABLE I. USED SWAP SPACE EVALUATION

Error measures for the predicted data	SMAPE	MAPE	RMSE
AR model	6.2019	4.8545	11.6189
neural network approach without workload	0.8498	0.5302	0.2678
neural network approach with workload	0.0933	0.07245	0.0570

Table 1 shows the RMSE, MAPE and SMAPE for the forecasts of *Used Swap Space* for the testing dataset using the MLP described in Figure6, 7 and Figure 8 with 3 input neurons (time lags), 10 neurons in the hidden layer and a sigmoid nonlinear transfer function. As seen in Table 1, the results obtained by the neural network approach with workload as input are far more accurate than AR model and neural network approach without workload parameters. In fact, the AR model cannot account for the nonlinear phenomena of software aging, and neural network approach without workload input cannot reflect the spikes shown in the software aging phenomenon.

VI. FORECASTING OF REAL MEMORY FREE

In this section, we use the same approach described in Section 5 to model the *Real Memory Free*. In Figure 9, the solid line plots the *Real Memory Free* that was collected in 12-day duration with an interval of 600 seconds. From Figure 9 we can see that *Real Memory Free* first keeps flat, and then increases sharply. This behavior cannot readily be explained by computing-resource exhaustion due to software aging. *Real Memory Free* cannot be lower than a preset threshold value. If physical memory approaches the lower limit, the systems frees up memory by paging[10]. Thus, we get an irregular utilization pattern. *Real Memory Free* shows more spikes than *Used Swap Space*.

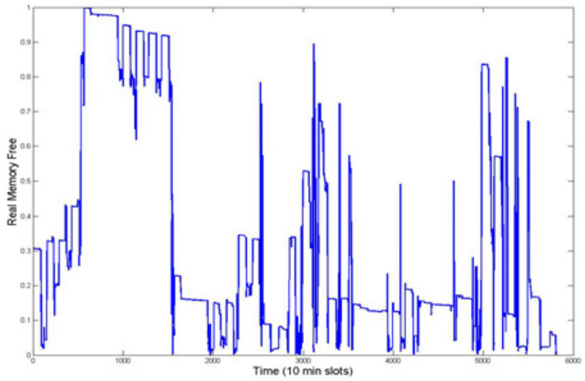


Figure 9. *Real Memory Free* of Rossby

Figure 10 and Figure 11 show the prediction of *Real Memory Free* by AR model and neural network approach without workload input. It is clear that it cannot track the irregular patterns shown in *Real Memory Free*, and both methods are not effective. This can be attributed to the working principle of them.

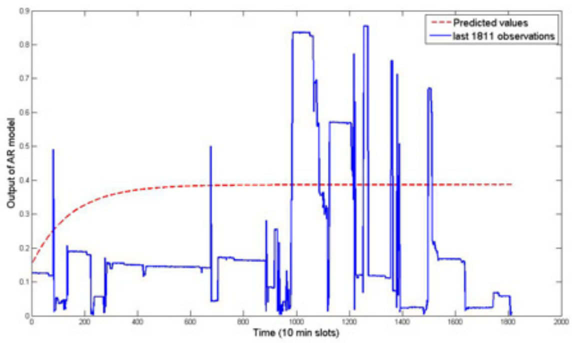


Figure 10. *Real Memory Free* of AR Model

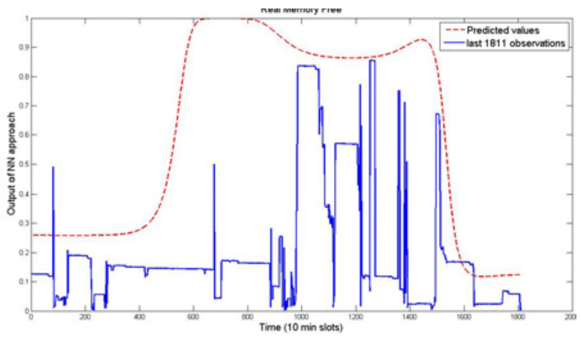


Figure 11. *Real Memory Free* of neural network approach without workload

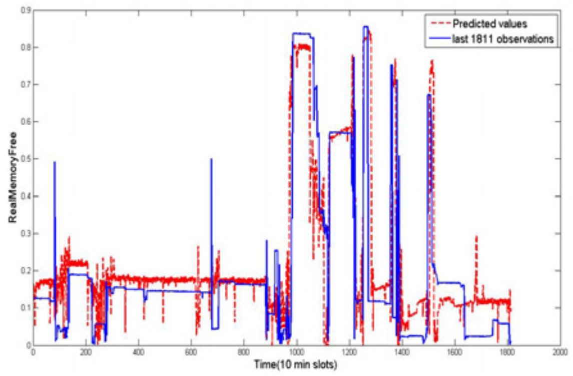


Figure 12 *Real Memory Free* of workload-based approach

In Figure 12, we show a plot of the last 1811 observations of the measured *Real Memory Free* (the testing dataset) and the predicted values obtained by the neural network model. These spikes can be tracked well by the output of our model. This can be explained by most patterns shown in the last 1811 observations have been well learnt by neural network.

Again, we tabulate the results of three goodness evaluation measures mentioned in above section, i.e., SMAPE, MAPE and RMSE to table 2. From that we can see the obtained results by the neural network with workload-based approach are more accurate than the results obtained by AR model approach and neural network approach without workload parameters.

TABLE II. REAL MEMORY FREE EVALUATION

Error measures for the predicted data	SMAPE	MAPE	RMSE
AR model	8.0385	12.301	0.5342
neural network approach without workload	1.0429	7.277	0.5368
neural network approach with workload	0.0673	0.080	0.0394

VII. RESULTS FOR DATASET 2

In this Section, our approach is validated through the dataset of Jefferson. This work is meaningful, because software runtime often shows great variance and random phenomena. Workstation Jefferson is subject to different inputs (which can be seen from Figure 3) and provides different services. Further, the configuration of Jefferson is different from that of Rossby.

We use the same resource variables, i.e. *Used Swap Space* and *Real Memory Free*. Again, we select the longest interval when Jefferson operates without reboot. The workload is shown in Figure 3. There are 6049 observations involved in our validation process. This dataset was split into two segments; the first 4000 observations are used for learning and validation. The last 2049 observations are used for testing. The output of our model and the observations are compared in Figure 13 and Figure 14.

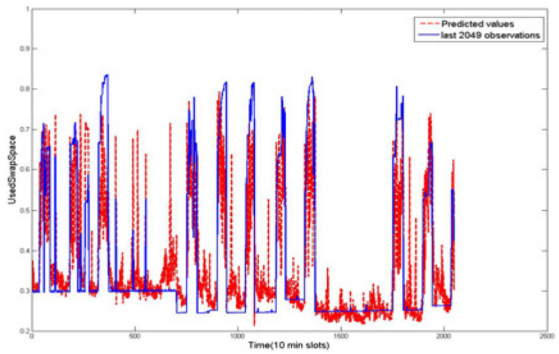


Figure 13. *Used Swap Space* of workload-based approach

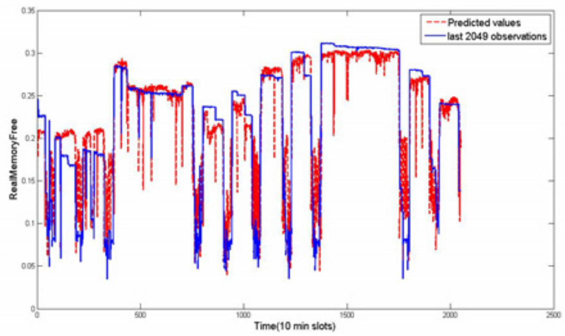


Figure 14. *Real Memory Free* of workload-based approach

From Figure 13 and Figure 14 we can see that this dataset is more strongly nonlinear than that of Rossby. The variance of both resource variables is larger than that of Rossby. However, our approach can still track the spikes of resource usage with higher accuracy than that in Figure 6 and Figure 12. This can be attributed to the working principle of neural network. More specifically, from Figure 13 we can see that most patterns of *Used Swap Space* have been learnt sufficiently even if the variance is larger. Figure 14 confirms this judgement. Recall the large difference between our model and real value shown in the last part of Figure 6, we have reasons to believe that, our approach will have better performance if the training time is long enough and more patterns are learnt.

In addition, the accuracy evaluation is also calculated and compared against AR model and neural network approach without workload. It's obvious the comparison results are positive. Thus, the comparison table is not provided.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we construct the relationship between computing-resource exhaustion and workload. First, we discuss the definition of workload. Then we analyze the activity data of a real workstation. It is observed that the free memory and used swap are affected by the workload greatly. This nonlinear relationship cannot be interpreted by the models previous reported. A BP neural network model is trained by historical data and used to forecast the resource usage with workload as input. Finally, forecasting results compared with other proposed models. The result shows that our approach is superior to others. The main contributions of our work are:

1. It is the first paper to quantitatively investigate the influence of workload on software aging;

2. Our approach can forecast computing-resource exhaustion; particularly can generate approximation of computing-resource usage spikes;

3. Our model can easily be generalized to investigate the influence of other factors on software aging, such as the influence of parameter settings on software aging.

Web traffic forecasting has been studied in depth. For example, the accepted requests stored in the buffer of server can be monitored directly. Thus, we can estimate the risk of computing-resource exhaustion at any time in a straightforward manner. Further, we can employ appropriate admission control policy to balance the cost of rejuvenation with risk of computing-resource exhaustion.

ACKNOWLEDGEMENTS

The authors would like to thank Professor Kishor S. Trivedi, the Hudson Chair in the Department of Electrical and Computer Engineering at Duke University, and Michael Grottke for providing the datasets that are analyzed in this paper.

REFERENCES

[1] A. Andrzejak, “Using Machine Learning for Non-Intrusive Modeling and Prediction of Software Aging,” Coregrid Technical Report, Number TR-0142, 2008

[2] K. J. Åstrom and B. Wittenmark, Computer Controlled System – Theory and Design, Englewood, Cliffs, 1984.

[3] X.E. Chen, Q. Quan, Y. F. Jia and C. Y. Cai, “A Threshold Autoregressive Model for Software Aging”, in Proceedings of the second IEEE International Workshop on Service-Oriented System Engineering, pp. 34-40 Oct. 2006

[4] D. Cotroneo, S. Orlando, and S. Russo, “Characterizing Aging Phenomena of the Java Virtual Machine”, in Proceedings of 26th IEEE International Symposium on Reliable Distributed Systems, pp. 127-136, 2007.

[5] T. Dohi, K. Goseva-Popstojanova, and K. S. Trivedi, “Statistical Non-parametric Algorithms to Estimate the Optimal Software Rejuvenation Schedule,” in Proceedings of International Pacific Rim Symposium on Dependable Computing, pp. 77-84, 2000.

[6] T. Dohi, K. Goseva-Popstojanova, and K. S. Trivedi, “Estimating Software Rejuvenation Schedules in High Assurance Systems,” Computer Journal, 44(6):473-482, 2001.

[7] H. El-Shishiny, S. Deraz, and O. Bahy, “Mining Software Aging Patterns by Artificial Neural Networks,” Lecture Notes in Computer Science, 2008.

[8] J. Gray and D.P. Siewiorek, “High-availability Computer Systems,” IEEE Computer, 24(9): 39–48, Sep. 1991.

[9] M. Grottke, L. Li, Kalyanaraman Vaidyanathan, and K. S. Trivedi, “Analysis of Software Aging in a Web Server,” IEEE Transaction on Reliability, 55(3):411-420, Sep. 2006.

[10] G. A. Hoffmann and K. S. Trivedi, “A Best Practice Guide to Resource Forecasting for Computing Systems,” IEEE Transactions on Reliability, Vol(56): 615-628, Dec. 2007

[11] Y. Hong, D. Chen, L. Li, and K.S. Trivedi, “Closed Loop Design for Software Rejuvenation,” in Proceedings of the Workshop Self-Healing, Adaptive and Self-Managed Systems, June 2002.

[12] Y. Huang, C. Kintala, N. Kolettis, and N. Fulton, “Software Rejuvenation: Analysis, Module and Applications,” in Proceedings of the 25th IEEE International Symposium on Fault-Tolerant Computing, pp. 381-390, Pasadena, USA, June 1995.

[13] Y. F. Jia, X. E Chen and K. Y. Cai “Chaotic Analysis of Software Aging in Web Server” in Proceedings of the second IEEE International Workshop on Service-Oriented System Engineering, pp. 117-120, Oct. 2006

[14] Y. F. Jia, L. Zhao, K. Y. Cai, “On the Relationship between Software Aging and Related Parameters in a Web Server”, in Proceeding of The Eighth International Conference on Quality Software, pp.241 – 246, 2008.

[15] Y. F. Jia, L. Zhao, K. Y. Cai, “A Nonlinear Approach to Modeling of Software Aging in a Web Server”, in Proceeding of the 15th Asia-Pacific Software Engineering Conference, pp.77-84, 2008.

[16] Y. F. Jia and K. Y. Cai “A Feedback Control Approach for Software Rejuvenation in a Web Server”, in Proceeding of the First Workshop on Software Aging and Rejuvenation”, 2008

[17] P. Killelea, Web Performance Tuning (2nd edition), O'Reilly Media, 2003

[18] S. Kumar, Neural Networks, Tsinghua University Press, pp. 193, Beijing, China, 2006

[19] R. Matias Jr. and P. J. Freitas Filho. “An Experimental Study on Software Aging and Rejuvenation in Web Servers”, In Proceedings of the 30th IEEE Annual International Computer Software and Applications Conference, vol (1) :189-196, 2006.

[20] E. Marshall, “Fatal Error: How Patriot Overlooked a Scud,” Science, 255(5050):1347, Mar. 1992.

[21] M. Shereshevsky, J. Crowell, B. Cukic, V. Gandikota, and Y. Liu, “Software Aging and Multifractality of Memory Resources,” in Proceedings of the International Conference on Dependable Systems and Networks, pp. 721-730, 2003.

[22] K. Vaidyanathan, and K.S. Trivedi, “A Comprehensive Model for Software Rejuvenation for Computing Systems.” IEEE Transaction on Dependable and Secure Computing, Vol(2):124-137, 2005

[23] Z. H. Xia, W. Hao, I. L. Yen, “A Distributed Admission Control Model for QoS Assurance in Large-Scale Media Delivery Systems”, IEEE Transaction on Parallel and Distributed Systems, Vol (16): 1143-1153, Dec. 2005.

[24] G. P. Zhang and M. Qi, “Neural Network Forecasting for Seasonal and Trend Time Series”, European Journal of Operational Research, pp. 501-514 2005