Journal of Systems Engineering and Electronics
Vol. 26, No. 2, April 2015, pp.407–414

Neural network based approach for time to crash prediction to cope with software aging

Moona Yakhchi[1], Javier Alonso[2], Mahdi Fazeli[1,3,*], Amir Akhavan Bitaraf[1], Ahmad Patooghy[1,3]

1. School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran 19395-5746, Iran;
2. Institute of Advanced Studies on Cybersecurity, University of Leon, Leon 24005, Spain;
3. Department of Computer Engineering, Iran University of Technology, Tehran 1684613114, Iran

Abstract: Recent studies have shown that software is one of the main reasons for computer systems unavailability. A growing accumulation of software errors with time causes a phenomenon called software aging. This phenomenon can result in system performance degradation and eventually system hang/crash. To cope with software aging, software rejuvenation has been proposed. Software rejuvenation is a proactive technique which leads to removing the accumulated software errors by stopping the system, cleaning up its internal state, and resuming its normal operation. One of the main challenges of software rejuvenation is accurately predicting the time to crash due to aging factors such as memory leaks. In this paper, different machine learning techniques are compared to accurately predict the software time to crash under different aging scenarios. Finally, by comparing the accuracy of different techniques, it can be concluded that the multilayer perceptron neural network has the highest prediction accuracy among all techniques studied.

Keywords: software reliability, software rejuvenation, machine learning.

DOI: 10.1109/JSEE.2015.00047

1. Introduction

Ever-increasing use of computer systems in human life requires high levels of reliability and availability of such systems. Computer systems have a variety of applications ranging from safety critical applications such as medical, industrial, and military systems to non-critical applications such as e-commerce websites or consumer electronic devices.

The complexity of today's computer systems has severely challenged the availability of such systems. Availability has been regarded as an important issue in safety critical applications; however, it has also become a serious concern in today's non-safety critical applications due to the increasing societal dependency on IT systems. Moreover, IT system failures have important profit cost and also erode the company's reputation and negatively impact its future market share [1].

The underlying causes of system failures can be classified into software errors, hardware errors and human errors. Software errors are becoming the main reason of the system failures due to their growing complexity. The software aging phenomenon causes a non-negligible fraction of system failures. Software aging was introduced in [2]. It is defined as an accumulation of software errors with time, which can result in resource exhaustion, performance degradation, and eventually system crash. Software aging is caused by aging-related bugs (ARBs) [3]. Some examples of the effects of ARBs are memory leakage [4], memory fragmentation [4], unterminated threads [5], or data expiration [2,5].

The software aging phenomenon may lead to system crash. Hence it is of decisive importance to cope with software aging effects. As a counter measure of software aging, software rejuvenation was proposed [2]. Software rejuvenation is a proactive approach in contrast with traditional reactive approaches to deal with software failures, which removes the aging effects by stopping the aging system, cleaning its internal state and resuming its operation. This would prevent or mitigate the performance degradation, and even it can avoid or delay software crash.

Software rejuvenation is classified into two main categories based on how the rejuvenation is triggered (See Fig. 1, adapted from [3]): time-based and inspection-based approaches. In the time-based approach, the rejuvenation activation is triggered at regular time intervals [5]. Inspection-based approaches monitor the system state and, when it crosses certain pre-specified limit, the rejuvenation is activated. The inspection-based approach is sub-divided

Manuscript received December 27, 2013.
*Corresponding author.

into three approaches: threshold-based, prediction-based, and mixed approaches. In threshold-based approaches, the activation threshold is prefixed based on the experience or human expertise [6]. In prediction-based approaches, a prediction mechanism (e.g., machine learning and statistical approaches) is used to estimate the time to crash (or time to resource exhaustion) [7,8]. Based on the predicted measure the rejuvenation is triggered when the crash is imminent. Therefore, in this approach a predictive function is required. In mixed approaches, it is a combination of threshold-based and prediction-based approaches together [9].

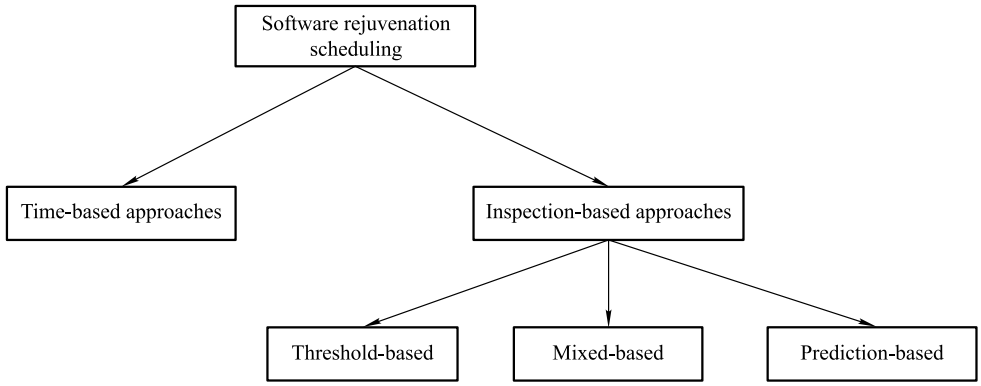

Fig. 1 Software rejuvenation scheduling strategies

According to [10], prediction approaches are gaining attention from industry and academia as a future mechanism to implement software rejuvenation. In order to develop the prediction-based software rejuvenation mechanism, it requires a highly accurate prediction function. Thus, prediction approaches must be able to predict, with high accuracy, the time to crash or time to resource exhaustion due to software aging effects. This will allow triggering the rejuvenation only when it is strictly necessary according to the system state.

Prediction-based approaches which require a prediction function according to the current/past state of the system will predict the future system state. Previous papers have compared the prediction accuracy of different machine learning algorithms to predict the state of the system under the effect of software [7,8,11–13].

However, due to the vast number of machine learning algorithms, there are still different machine learning algorithms not analyzed. In this paper, we present an experimental comparison of multilayer perceptron neural network (MPNN) [14] with the best candidates revealed in previous papers. MPNN is compared with some of the best well-known machine learning algorithms and some of the best candidates revealed in the previous papers [12,13].

We rank the algorithms according to their correlation coefficients, mean absolute error (MAE), root mean squared error (RMSE) and relative absolute error (RAE). We compare the algorithms under four different software aging scenarios. The results reveal that MPNN is the best candidate according to all metrics of interest.

The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 describes the experimental environment. Section 4 briefly introduces the machine learning algorithms compared in our experimental study. Section 5 discusses the obtained results and finally Section 6 concludes the paper.

2. Related work

Huang et al. [2] introduced software rejuvenation as a proactive countermeasure of software aging. As it has been aforementioned, software rejuvenation is classified according to how the rejuvenation is scheduled as time-based or inspection-based strategies. Time-based approaches focus on trying to estimate the most suitable time interval to trigger the rejuvenation [5]. However, time-based rejuvenation strategies do not usually take into account the state of the system under monitoring, and thus, it is possible that they trigger unnecessary rejuvenation actions. In order to fill this gap, inspection-based approaches have been proposed. Inspection-based approaches are characterized by monitoring the state of the system (i.e., monitoring memory usage, CPU usage, or other resources) and determining if the system is aged. If it is the case, the rejuvenation is triggered. The mechanism to determine the aging of the system subdivides inspection-based approaches into threshold-based approaches and prediction-based approaches. The former approaches define a threshold for the aging factor (e.g., memory usage), when the threshold is reached the rejuvenation is triggered [6,15]. These strategies have important limitations since they require knowing the aging factor(s) in advance, which is not always possible. Furthermore, the threshold becomes a key factor for the availability of the

system. If the threshold is not accurately fixed, the system can crash or offer an unacceptable performance before the rejuvenation is triggered.

Several papers have presented different mechanisms to predict/estimate the time to crash or time to resource exhaustion, using statistical or machine learning techniques [7,8,12,13,16,17]. These techniques use historical system state metrics easily collected in any system to estimate the time to crash or to resource exhaustion. Some examples of these system state metrics are throughput, response time, CPU usage, memory consumed by the system or a specific application, number of threads, or number of TCP connections.

In [16] Sens's slope statistical technique is used to estimate the resource exhaustion trend. Sen's slope statistical technique is also able to estimate this trend in presence of seasonal behavioral data. Once they obtain the aging trend, the time series approach is used to estimate the time to resource exhaustion. In [18], the authors compare the Mann-Kendall trend and Sen's slope approaches with the Hodrick-Prescott filter and the multi-scale Mann-Whitney U test. The results show that the Hodrick-Prescott filter and the multi-scale Mann-Whiteney test improve the accuracy of Sen's slope and Mann-Kendall, offering a unified structure without requiring a previous seasonal test. Furthermore, the proposal approach is able to distinguish between abrupt changes and aging degradation.

Time series autoregressive moving average (ARMA) models are used in [17] to estimate the resource exhaustion time. This paper recognizes the relationship between the aging trend and the workload. However, all aforementioned papers assume that the aging factor is known in advance.

There also are a set of papers where different machine learning algorithms have been compared each other under different aging scenarios. In [13], the authors compare Naïve Bayes, decision trees and support vector machine algorithms to predict time to crash due to aging. However, the considered aging is constant along the time. In [7] and [12], the authors compared linear regression, decision trees and a mixed algorithm called M5P under different dynamic and complex scenarios. These papers consider scenarios where the aging trend changes along the time with and without dependency of the workload. The study revealed that M5P was a good candidate to predict the time to crash under different scenarios and different/combined aging factors (i.e., memory and threads).

The main goal of any of the software aging prediction approaches is to decide if the system requires rejuvenation. Thus, it is only required to know the state of the system. Following this idea, different classification algorithms were compared in [12] under the same scenarios presented in [7]. Random forest was determined as the best candidate.

In this paper, we are interested in extending the number of machine learning algorithms analyzed to model the aging phenomena under different aging scenarios. This will provide more information to practitioners and researchers about which the machine learning algorithm will be the better candidate for predicting the time to crash of a system under different circumstances.

3. Experimental system setup

The main goal of this paper is to compare the effectiveness of different machine learning techniques to predict the time to crash of an aging system. To this end, some hypotheses are presented and some experiments have been carried out which will be discussed in the following.

As mentioned earlier, the software aging occurs due to misuses of resources by applications such as memory leakage caused by not properly releasing the allocated memory by running programs. Such misuses of system resources would cause a system crash due to eventual resource exhaustion. To model software aging in the system, we have developed a tool which injects memory leaks creating the conditions under which a system would crash. In this tool, a random number of dummy processes are created. Each created process contains an infinity loop inside which a specific amount of memory is allocated and then released. The main point that should be noted is that the amount of memory allocation is supposed to be more than that of the memory released in order to mimic the behavior of a memory leaking process. In the memory leakage injection tool, the amount of memory allocation in each process is determined by a Poisson process. Briefly, in the tool, we have two random variables including the number of the dummy process that follows a uniform process, and the amount of memory allocation in each process that follows a Poisson process. We analyze the machine learning algorithms in four different scenarios. The scenarios under consideration are as follows.

(i) Scenario 1 (Heavy leakage rate): In this scenario the amount of memory leakage for each dummy process follows a Poisson process with the rate bigger than 0.5 byte per second ($\lambda \geqslant 0.5$). We run several executions of this scenario varying the parameters in order to collect enough data to proper train the models. The number of processes is determined by a uniform distribution. Once the number of processes is chosen, it keeps constant along the run.

(ii) Scenario 2 (Light leakage rate): In this scenario the amount of memory leakage for each dummy process follows a Poisson process with the rate smaller than 0.5 byte

per second ($\lambda \leqslant 0.5$).

(iii) Scenario 3 (Hybrid leakage rate): In this scenario we first divide the number of dummy processes into two same sized groups. In the first group, we use the same approach as Scenario 1 and for the second group we use the same approach as Scenario 2.

(iv) Scenario 4 (Random leakage rate): In this scenario, we do not have any restriction for the rate of memory leakage for each dummy process i.e. λ can be any value between 0 (the minimum value) and 1 (the maximum value). The number of processes is also random between 8 and 15.

Table 1 describes the hardware/software used to build up the experimental environment.

Table 1 Experimental setup

	Environment
Hardware	CPU i7 Intel 1.7–2.9 GHz RAM 6 G
OS	Windows 7 Home premium 64 bit
Software	Stress test implemented by C#

To predict the time to crash by a machine learning technique, we need to profile different system characteristics when our memory leakage injection tool is running. To do so, we employ a performance-monitoring tool of the Windows 7 OS which is called performance monitor [19].

Before each experiment starts, we adjust the performance monitoring tool to collect the required data each second. The collected data are then saved to be analyzed in a log file. The log file contains different system characteristics such as processor utilization, number of processes and memory usage. After the dataset is collected, we add a new column (hold time) into the dataset representing the time interval between running our injection tools and observing a system crash. This process is called completing process.

After collecting the dataset for each scenario, different machine learning algorithms are applied independently on the datasets and their prediction accuracies are compared. We use 80 percent of the collected data to build/train the model and 20 percent to predict the time to crash. The data instances percent of the collected data for the test phase used to train and test the models is chosen randomly, once. Then the same training data and testing data are used for all the models.

Fig. 2 illustrates the overall flow of the work conducted. To train and test the machine learning algorithms under study, we employ the well-known Waikato environment for knowledge analysis (WEKA) [20] machine learning (ML) and data mining package.

4. ML algorithms

In this section, we briefly describe the main characteristics of the compared machine learning algorithms.

Linear regression: Linear regression (LR) is a method to model the linear relationship between a scalar dependent variable y and one or more independent variables x.

Least median square: The least median square (LMS) method is one of the statistical methods for solving the equations. This method is always used in analytical regression. In fact, LMS is a method for fitting the dataset. The LMS must yield the smallest value for the median of squared residuals computed for the entire data set. This residuals refer to the difference between real data and predicted data.

Gaussian process: In probability theory, the Gaussian process (GP) consists of random values associated with every point in a range of times. The random variable has a normal distribution. In [20], it has been mentioned that "a Gaussian process is a stochastic process whose generalization of a Gaussian distribution over a finite vector space to a function space of infinite dimension".

M5P: M5P is a tree learner consisting of a binary decision tree which learns a "model" tree. This is a decision tree with LR functions at the leaves. It is used to predict a numeric target attribute. It may be piecewise fit to the target [14,21].

REPtree: REPtree (RT) builds a decision/regression tree using information gain/variance and prunes it using reduced-error pruning (with back fitting). Only sort values for numeric attributes once. Missing values are dealt with by splitting the corresponding instances into pieces (i.e. as in C4.5) [14].

Sequential minimal optimization: In [4,22], sequential minimal optimization (SMO) was proposed as an iterative algorithm. This algorithm uses support vector machine (SVM) for solving the regression problem and SVM classifier design. The SMO algorithm has two worthy aspects: easy implementation and fast computational speed [23].

Multilayer perceptron: The multilayer perceptron (MLP) is a very simple model of biological neural networks. The network is structured in a hierarchical way. MLP includes some nodes located in different layers where the information flows only from one layer to the next layer. The first and the last layers are the input and the output. Layers between the input and the output layer are called hidden layers. This network is trained based on back propagation error in that the real outputs are compared with network outputs, and the weights are set using supervised back propagation to achieve the suitable model [24].

5. Experiment results

In this section, we comprehensively compare the machine learning algorithms to show which one is more effective

and has a better accuracy in determining the time to crash under the considered scenarios.

In scenarios explained in Section 3, the system is considered that crashes when the memory consumed by the system reaches 100% during two measurements. After running all mentioned scenarios and performing the completing process, each scenario dataset is used to train and compare the algorithms, including LR, SMO, MLP, M5P, LMS, RT and GP.

As mentioned in Section 3, the completing process has been done by adding the real times to crash to our profile. For comparing machine learning algorithms, we use the correlation coefficients, MAE, RMSE and RAE metrics. After computing each of these performance metrics, we rank the algorithms by computing the mean ranked position of each algorithm in each scenario. Thus, we sum the position of a algorithm in each scenario for an specific metric and obtain the average rank. Based on this value, we rank the algorithms according to each metric of interest. At the end, we compute the overall ranked position by the same approach.

Table 2 presents the correlation coefficients of each algorithm per scenario. We can state that predicting values by MLP are directly and closely related to training dataset values, compared with other algorithms. On the other side, the GP experiences worth average performance across the scenarios according to the rank. The rank list of the algorithms is as follows considering the four scenarios together: (i) MLP, (ii) M5P, (iii) SMO, (iv) LR, (v) RT, (vi) LMS and (vii) GP. Note that M5P and SMO obtain the same average ranked value across scenarios. For this reason, both are in the second position.

MAE of each algorithm in each scenario is presented in Table 3. According to the average ranked position across scenarios, MLP offers the best results. It indicates that the MLP algorithm is able to adapt better to different scenarios. The complete average rank list across the four scenarios is: (i) MLP, (ii) SMO, (iii) M5P, (iv) LR, (v) RT, (vi) LMS and (vii) GP.

Table 4 presents the RMSE of each algorithm. The RMSE provides insight about the presence of large errors. Therefore, the presence of a larger RMSE value indicates that it is more likely to observe large errors (i.e. the difference between the real value and the predicted one). Since the goal is to obtain a model to predict the time to crash (or time to resource exhaustion) to trigger the software rejuvenation, we need models with a low probability of large errors. The complete average rank list across the four scenarios is: (i) MLP, (ii) M5P, (iii) LR, (iv) SMO, (v) RT, (vi) LMS and (vii) GP.

Finally, Table 5 presents the RAE of each algorithm in each scenario. The RAE provides the error percentage normalized by the real value. We observe that the RAE is large since none of the algorithms are over 10%. Note that the algorithms are used with their default WEKA parameters. Therefore, it can be expected that adjusting the parameters properly of the algorithms these results can be improved significantly. According to the RAE results, the average rank list is as follows: (i) MLP, (ii) SMO, (iii) M5P, (iv) LR, (v) RT, (vi) LMS and (vii) GP.

Table 2 Correlation coefficient of each algorithm and scenario under study

	Scenario 1		Scenario 2		Scenario 3		Scenario 4	
	Rank	Correlation coefficient	Rank	Correlation coefficient	Rank	Correlation coefficient	Rank	Correlation coefficient
LR	3	0.986	6	0.901 4	3	0.962 6	2	0.985 6
MLP	1	0.993 2	2	0.977 4	1	0.979 5	1	0.990 5
M5P	5	0.944 6	1	0.984 1	2	0.973 3	5	0.980 4
SMO	2	0.989 4	4	0.965 5	4	0.960 4	3	0.984 9
GP	7	0.893 7	5	0.919 4	6	0.928 2	7	0.937 8
LMS	4	0.948 4	7	0.855	7	0.424 1	4	0.984
RT	6	0.933 1	3	0.972	5	0.954 5	6	0.958 1

Table 3 MAE of each algorithm and scenario under study

	Scenario 1		Scenario 2		Scenario 3		Scenario 4	
	Rank	MAE	Rank	MAE	Rank	MAE	Rank	MAE
LR	5	8.501 2	3	4.940 1	3	2.805 5	4	1.855 8
MLR	1	3.128 2	1	3.386	1	2.091 2	1	1.534 9
M5P	2	3.560 6	4	5.945 6	2	2.739 2	5	2.042 6
SMO	4	5.199 2	2	4.250 3	4	2.914 5	2	1.735 1
GP	6	12.198 2	6	13.146 3	6	5.970 5	7	5.044
LMS	7	9.015 6	7	13.831 2	7	15.372 7	3	1.802 8
RT	3	4.508 2	5	6.352 3	5	3.105 1	6	3.078 4

Table 4 RMSE of each algorithm and scenario under study

	Scenario 1		Scenario 2		Scenario 3		Scenario 4	
	Rank	RMSE	Rank	RMSE	Rank	RMSE	Rank	RMSE
LR	3	6.601 6	5	11.636 7	3	3.660 8	2	2.409 1
MLR	1	4.555 7	2	5.710 9	1	2.700 4	1	2.005 9
M5P	4	12.950 9	1	4.716	2	3.610 8	5	2.538 6
SMO	2	5.670 2	4	6.861 4	4	3.764	4	2.494 6
GP	6	19.794 9	7	15.533 3	6	7.393 6	7	6.521 3
LMS	7	23.746 3	6	14.090 9	7	32.230 9	3	2.475 5
RT	5	13.881 4	3	6.220 1	5	3.970 7	6	3.738 9

Table 5 RAE of each algorithm and scenario under study

	Scenario 1		Scenario 2		Scenario 3		Scenario 4	
	Rank	RAE	Rank	RAE	Rank	RAE	Rank	RAE
LR	3	16.434 1%	5	37.702 1%	3	22.562 2%	4	16.166%
MLR	1	11.264 2%	1	13.873 4%	1	16.817 9%	1	13.370 9%
M5P	4	19.779 1%	2	15.791%	2	22.028 7%	5	17.793%
SMO	2	14.139 2%	4	23.057 8%	4	23.438 7%	2	15.114 6%
GP	6	43.733 5%	7	54.097 7%	6	48.015 4%	7	43.938 6%
LMS	7	46.011 9%	6	39.983 1%	7	123.629 1%	3	15.704 3%
RT	5	21.131 9%	3	19.993 3%	5	24.971 8%	6	26.816 3%

After analyzing the four performance metrics for all algorithms in four scenarios under study, we propose to rank the algorithms per scenario, computing the average rank obtained in each performance metric. Then summarize the results per scenario. Finally, we also compute the final rank across scenarios. We can observe that MLP is the best candidate in all scenarios (average rank of the four performance metrics under consideration). After that SMO presents the second average performance, followed by LR and M5P. On the other side, GP obtains the last position in the ranked list, showing its least suitability to predict the time to crash due to aging effects. Finally, Fig. 2 presents the comparison between actual and predicted data in scenario 2. According to Fig. 3, there is high matching between actual and predicted data by MLP in scenario 2.

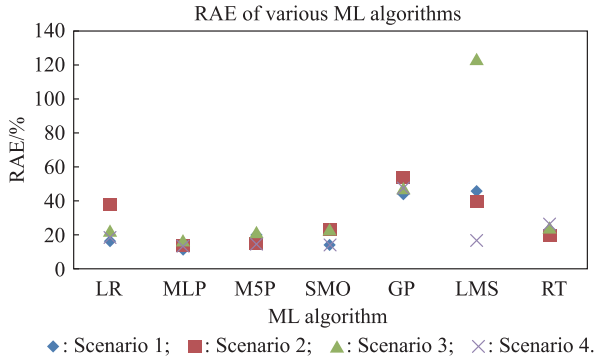

Fig. 2 RAE of each algorithm and scenario under study

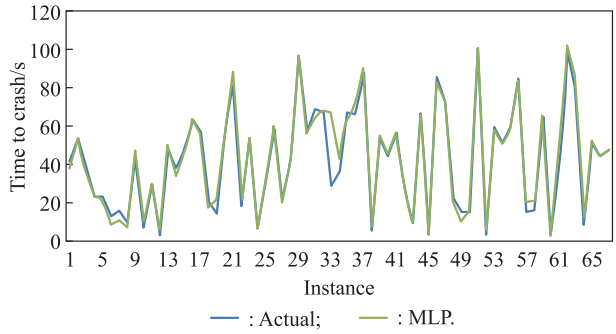

Fig. 3 Comparison between actual and predicted data in scenario 2

6. Conclusions

In this paper, we compare different machine learning algorithms under different aging scenarios. As memory leakage is the most important parameter in software aging, we implement a memory leakage injection which statistically injects memory leakage to understudy system equipped with a software rejuvenation technique to examine the accuracy of the technique. Using the memory leakage injection tool as well as the performance monitoring tool that is available in Windows 7 OS, we comprehensively compare different machine learning algorithms. Our experimental results show that MLP obtains the better overall performance among the studied algorithms under the four scenarios considered. As a future work, it would be necessary to compare the algorithms listed in this paper under more complex scenarios and when more complex aging factors are involved in order to study how generalizable are the results presented in this paper.

References

[1] I. Koren, C. M. Krishna. *Fault tolerant systems*. Amsterdam: Elsevier, 2007.

[2] Y. Huang, C. Kintala, N. D. Funton. Software rejuvenation: analysis, module and applications. *Proc. of the 25th IEEE International Symposium on Fault Tolerant Computing*, 1995: 381 – 390.

[3] J. A. Lopez, R. Matias, E. Vicente, et al. A comparative experimental study of software rejuvenation overhead. *Journal of Performance Evaluation*, 2013, 70(3): 231 – 250.

[4] S. K. Shevade, S. S. Keerthi, C. Bhattacharyya, et al. Improvements to the SMO algorithm for SVM regression. *IEEE Trans. on Neural Networks*, 2000, 11(5): 1188 – 1193.

[5] K. Vaidyanathan, K. S. Trivedi. A comprehensive model for software rejuvenation. *IEEE Trans. on Dependable and Secure Computing*, 2005, 2(2): 124 – 137.

[6] J. A. Lopez, L. Silva, A. Andrzejak, et al. High-available grid services through the use of virtualized clustering. *Proc. of the 8th IEEE/ACM International Conference on Grid Computing*, 2007: 34 – 41.

[7] J. A. Lopez, J. Torres, R. Gavalda. Predicting web server crashes: a case study in comparing prediction algorithms. *Proc. of the 34th International Conference on Distributed Computing Systems*, 2009: 264 – 269.

[8] J. A. Lopez, L. Belanche, D. R. Avresky. Predicting software anomalies using machine learning techniques. *Proc. of the 10th IEEE International Symposium on Network Computing and Applications*, 2011: 163 – 170.

[9] R. Matos, J. Araujo, P. Maciel, et al. Software rejuvenation in eucalyptus cloud computing infrastructure: a hybrid method based on multiple thresholds and time series prediction. *International Transactions on Systems Science and Applications*, 2012(8): 1 – 16.

[10] J. A. Lopez, A. Bovenzi, J. Li, et al. Software rejuvenation – - Do IT & Telco industries use it? *Proc. of the 23rd IEEE International Symposium on Software Reliability Engineering Workshops*, 2012: 229 – 304.

[11] J. A. Lopez. Proactive software rejuvenation solution for web environments on virtualized platforms. Barcelona, Spain: Polytechnic University of Catalonia, 2011.

[12] J. A. Lopez, J. L. Berral, R. Gavalda, et al. Adaptive on-line software aging prediction based on machine learning. *Proc. of the 40th IEEE/IFIP International Conference on Dependable Systems and Networks*, 2010: 497 – 506.

[13] A. Andrzejak, L. Silva. Using machine learning for non-intrusive modeling and prediction of software aging. *Proc. of the IEEE Network Operations and Management Symposium*, 2008: 25 – 32.

[14] I. H. Witten, F. Eibe. *Data mining practical machine learning tools and techniques*. Burlington: Elsevier, 2005.

[15] L. Silva, J. Alonso, J. Torres. Using virtualization to improve software rejuvenation. *IEEE Trans. on Computers*, 2009, 58(11): 1525 – 1538.

[16] M. Grottke, L. Li, K. Vaidyanathan, et al. Analysis of software aging in a Web server. *Proc. of IEEE Transactions on Reliability*, 2006, 55(3): 411 – 420.

[17] L. Li, K. Vaidyanathan, K. S. Trivedi. An approach for estimation of software aging in a web server. *Proc. of the International Symposium on Empirical Software Engineering*, 2002: 91 – 100.

[18] P. F. Zheng, Q. G. Xu, Y. Qi. An advanced methodology for measuring and characterizing software aging. *Proc. of the 23rd IEEE International Symposium on Software Reliability Engineering Workshops*, 2012: 253 – 258.

[19] Windows Performance Monitor. http://technet.microsoft.com/en-us/library/cc749249.aspx.

[20] Y. Wang, I. H. Witten. Intduction model trees for continuous classes. *Proc. of the European Conference on Machine Learning Poster Papers*, 1997.

[21] WEKA 3.6.8. http://www.cs.waikato.ac.nzlmllwekal.

[22] A. J. Smola, B. Schölkopf. A tutorial on support vector regression. London: Royal Holloway College, 1998.

[23] D. J. C. Mackay. Introduction to Gaussian processes. London: Cambridge University, 1998.

[24] A. Macedo, T. B. Ferreira, R. Matias. The mechanics of memory-related software aging. *Proc. of the IEEE International Workshop on Software Aging and Rejuvenation*, 2010: 1 – 5.

Biographies

Moona Yakhchi received her M.S. degree in computer engineering from the Islam Azad University, Boroujerd, Iran, in 2012. Her current research interests are software rejuvenation and cloud computing.
E-mail: m.yakhchi@gmail.com

Javier Alonso received his M.S. degree in computer science in 2004 and Ph.D. degree from the Technical University of Catalonia (UniversitatPolitecnica de Catalunya, UPC) in 2011. From 2006 to 2011, he was an assistant lecturer in the Computer Architecture Department of UPC. From 2011 to 2014, he was a postdoctoral associate in the Electrical and Computer Engineering Department, Duke University. Since 2014, he has been the research manager in the Institute of Applied Sciences on Cybersecurity at University of Leon, Spain. He has published more than 30 papers in premier conferences and journals. He is a member of IEEE. He has also served as a reviewer for IEEE Transactions on Computers, *IEEE Transactions on Dependability and Security Computing, Performance Evaluation, and Cluster Computing*, and several international conferences. His research interests are dependability, reliability, availability, and performance of computer and communication systems. He has special interests in software dependability and software aging and rejuvenation topics.
E-mail: javier.alonso@unileon.es

Mahdi Fazeli received his M.S. and Ph.D. degrees in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2005 and 2011, respectively. He has been with the Department of Computer Engineering, Iran University of Science and Technology (IUST), since 2011, where he is currently an assistant professor. He established the Dependable Systems and Architectures Laboratory (DSA Lab) at IUST in 2012 and he has chaired the Laboratory since then. He has authored or co-authored more than 40 papers in reputable journals and conferences. His current research interests include reliable issues in VLSI circuits and emerging technologies, hardware security, dependable embedded systems, low power circuits and systems, fault-tolerant computer architectures, and reliability modeling and evaluation.
E-mail: m_fazeli@iust.ac.ir

Amir Akhavan Bitaraf received his M.S. degree in software engineering from Islamic Azad University, Broujerd, Iran in 2012. He has been collaborating with Broujerd branch of Sama Technical and Vocational Training College since 2010. He is interested in software engineering, data mining and software fault tolerance, especially software aging and rejuvenation.

E-mail: bitarafaa@gmail.com

Ahmad Patooghy received his M.S. and Ph.D. degrees in computer engineering from Sharif University of Technology, Tehran, Iran, in 2005 and 2011, respectively. He is currently an assistance professor at Department of Computer Engineering, Iran University of Science & Technology, Tehran, Iran. He initiated the Dependable Systems and Architectures Laboratory at Iran University of Science and Technology in 2012 and has chaired the Laboratory since then. His research interests include hardware design and test, architectural design of multi- and many-core chips, dependability and security evaluation of VLSI circuits, fault injection, and analytical modeling.

E-mail: patooghy @iust.ac.ir