International Journal of Electrical and Computer Engineering (IJECE)
Vol. 10, No. 6, December 2020, pp. 5985~5991
ISSN: 2088-8708, DOI: 10.11591/ijece.v10i6.pp5985-5991

An analysis of software aging in cloud environment

Shruthi P.[1], Nagaraj G. Cholli[2]
[1]Department of Computer Science and Engineering, Global Academy of Technology, India
[2]Department of Information Science and Engineering, Rashtreeya Vidyalaya College of Engineering, India

Article Info

Article history:

Received Dec 30, 2019
Revised May 1, 2020
Accepted May 16, 2020

Keywords:

Cloud computing
Software aging
Software rejuvenation

ABSTRACT

Cloud computing is the environment in which several virtual machines (VM) run concurrently on physical machines. The cloud computing infrastructure hosts multiple cloud services that communicate with each other using the interfaces. During operation, the software systems accumulate errors or garbage that leads to system failure and other hazardous consequences. This status is called *software aging*. Software aging happens because of memory fragmentation, resource consumption in large scale and accumulation of numerical error. Software aging degrads the performance that may result in system failure. This happens because of premature resource exhaustion. The errors that cause software agings are of special types and target the response time and its environment. This issue is to be resolved only during run time as it occurs because of the dynamic nature of the problem. To alleviate the impact of software aging, software rejuvenation technique is being used. Rejuvenation process reboots the system or reinitiates the softwares. Software rejuvenation removes accumulated error conditions, frees up deadlocks and defragments operating system resources like memory. Software aging and rejuvenation has generated a lot of research interest recently. This work reviews some of the research works related to detection of software aging and identifies research gaps.

Corresponding Author:

Shruthi P.,
Department of Computer Science and Engineering,
Global Academy of Technology,
Ideal Homes Township, Mysore Rd, Aditya Layout, RR Nagar, Bengaluru, Karnataka 560098, India
Email: shrutip@gat.ac.in

1. INTRODUCTION

System is a group of inter operating components. The boundary of the system separates the system from its environment but its services are always towards surrounding environment. Software failure happens if the environment oriented outputs are incorrect. The services in their long-run operation accumulate numerous internal errors and garbage, thus resulting in software aging and probable failure or performance degradation [1]. Software aging phenomenon is defined as the increase of failure rate and/or decrease in performance of a long-running software system. Aging-related-bugs (ARBs) get activated and propagate in the long run causing software aging. These manifest as faults in the software which become visible only after continuous execution of the software for a long period. There will not be any impact because of these bugs immediately. But they make the system slowly shift from a healthy state to a failure prone state. Figure 1 depics the state change of systems affected by software aging.

Aging effects are a result of error accumulation. Error accumulation leads to resource exhaustion, unreleased file locks, un-terminated threads and memory leaks. Software aging effects can be detected by using aging indicators. Examples of aging indicators are free physical memory, swap space used, file and process tables size, application response time, traffic metrics like packet rate and bit rate.

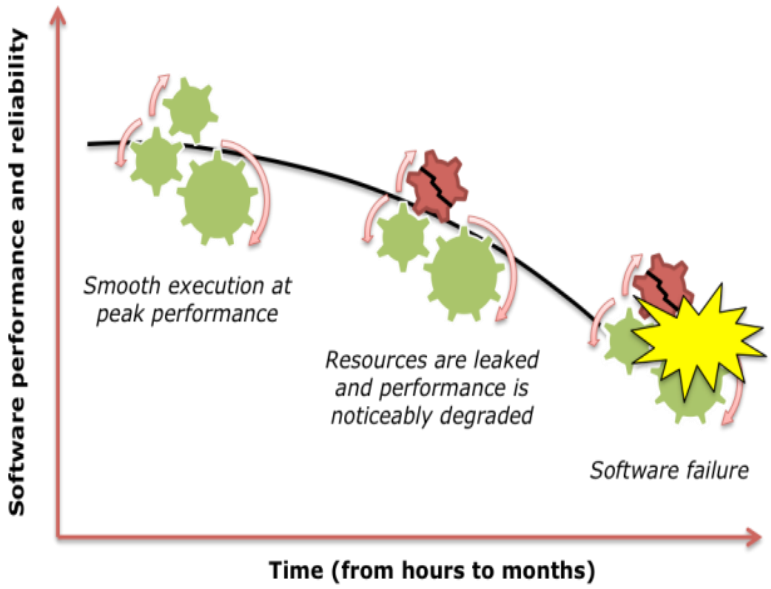

Figure 1. Performance degradation [2]

The propagation of aging-related errors that do not cause a failure hide in the system internal state as a time bomb [3]. This aging-related error accumulation leads to the failure of the system internal environment. Software aging effects are of two types, volatile and non-volatile. Volatile aging effects are created usually in operating system resources. These can be removed by re-initiating the system process that is affected. Examples of volatile aging effects are fragmentation of memoryor operating system resource leakage. But, non-volatile effects are created in storage parts like database and file system. Non-volatile aging effects exist even after re-initialization of system. Examples of non-volatile aging effects are fragmentation in file system or metadata of database. This issue can be detected only when the system is running, by monitoring system parameters which are aging indicators. Just like antigens indicate cancer disease, aging indicators signify software aging.

Major causes for software aging are memory leaks/fragmentation, unreleased file handles, lock contention issues, data corruption, memory/swap space bloat, round off error accumulation and storage space fragmentation. While the system is running, aging effects can be detected by monitoring of aging indicators. Based on their granularity, aging indicators can be classified into two categories. They are system-wide aging indicators and application-specific aging indicators. Metrics related to several subsystems like OS, middleware, virtual machine that are shared by running applications can be obtained by system-wide aging indicators. Using these aging indicators, aging effects of the whole system can be studied. Examples of system-wide aging indicators are used swap space, free physical memory, system load and file table size. Information specific to individual application can be obtained by application-specific aging indicators like response time, Java VM heap size. These aging indicators give more specific information about the application under observation. Aging indicators have been studied in detail by several researchers [4, 5]. Software aging effects happen at various levels including operating system level or application level [6].

Costs of software aging are owner's loss, inability to keep up, reduced performance and decreasing reliability. The owners of the aging software suffer as it is increasingly difficult to keep up with the market. Software grows bigger as it degrades. This happens due to addition of new codes to manage the software. Because of increase in program size, there will be more demands for memory and there will be delays as code is to be swapped in from mass storage. The response of the program becomes slow and hence performance and reliability decreases. Software maintenance activities introduce errors sometimes. Correction of an error may lead to another error. Sometimes it may so happen that each time an attempt is made to decrease the error rate, it may get worse.

Software aging is inevitable even if all the preventive measures are taken. The ability to design for a change depends on an ability to predict the future. If it is done imperfectly and approximately, over a period of time, the software becomes unusable. Preventive measures cannot eliminate aging. Because of aging, lot of money has been lost in various domains across the world. Table 1 shows the cost of software aging on some US based companies of different sectors.

Table 1. Cost of software aging [7]

Domain	Operation Type	Industry cost range per hour	Average cost per hour of downtime
Financial	Brokerage	$ 5.6 M to $ 7.3 M	$ 4.45 M
Financial	Credit card / Sales	$ 2.2 M to $ 3.1 M	$ 2.6 M
Media	View	$ 233 K	$ 150 K
Retail	Home Shopping Network	$ 87 K to $ 140 K	$ 113 K
Transportation	Airlines Reservation	$ 67 K to $ 112 K	$ 28 K

Software rejuvenation is the proactive technique proposed to counter software aging that involves a series of steps such as periodically stopping the application and restarting it after cleaning the internal state. Software rejuvenation performs flushes buffer queues,garbage clearance, reinitializes the internal kernel tables, and cleans up file systems. Intrinsically, it cleans and restores the operating environment. From the literature review, it can be observed that, software rejuvenation approaches are of two types: Time-based and Inspection based [8]. In the first method, rejuvenation is applied at pre-determined intervals of time. This technique is used in environments such as web servers. In the second method, software aging forecasting is done and rejuvenation is triggered pre-emptively.

2. SOFTWARE AGING DETECTION TECHNIQUES

Alonso, Belanche and Avresky [9] used machine learning algorithms for prediction of resource exhaustion that are caused by software anomalies. Evaluation of performance was done through various algorithms like K-nearest neighbors, Naïve Bayes, Random Forest. The researchers created three different scenarios for execution. In each scenario, they used different software anomaly to crash a system using TPC-W benchmark. The numbers of instances used are 2815, 1688 and 3819. There were a total of 29 attributes. The execution results indicate that the Random Forest performed comparativelybetter in all the three scenarios. Naïve Bayes produced the highest number of errors. Lasso regularization was used by the researchers to reduce (up to 60%) the number of parameters that are needed to build the model. This further reduced the errors in several cases. Later, the analysis of accuracyand number of monitored parameters was done.

Toshiaki Hayashi et al. [10] worked to detect performance degradation by measuring traffic between virtual machines. The metric collection method was non-intrusive as the traffic was collected passively from a machine that is not part of virtual environment on which the measuring services are hosted. Using this technique, the metrics can be collected even under extreme performance degradation. The traffic metrics used in the study are bit-rate, packet rate, connection rate and TCP SYN loss rate. For measuring traffic, the researchers used program packets which were built using the C language and “pcap” (TCPDUMP/ LIBPCAP, 2012) library. The metric values were collected and deliveredfor every 60 seconds. Using these metrics, training and test data were constructed. The virtual environment consisted of Apache web server running on virtual machines housed on Xen hypervisor. To measure the request-response rate of files, httperf tool was used on the client machines. Training data which is of 600 instances and test data which is of 1000 instances was constructed using traffic data and request-response rate data. Each of the data set has five attributes. Machine learning classifier C4.5 was used to build decision tree which detected the performance from traffic metrics. The proposed method was found reliable as the error ratio was small (2.2% for 1000 instances of test data).

Jing Liu et al. [11] proposed a comprhesive service rejuvenation based fault tolerant method that guarantees cloud system’sservice availability. The architecture proposedinclude services running on VMs, software rejuvenation manager (SRM) installed on each VM and an Interim Node housed in the separate physical machine. All these entities are in the same local network. When aging is detected on any of the service components, the check point of its running state is created and stored in the interim node. The VM that hosts aging prone service component is migrated to interim node to keep the service requests continuity. During the migration, the trace log file is updated by rejuvenation agent. When migration is done, the original virtual environment is rebooted that brings it to aging free state. Next, the check point image is transferred from interim node to original VM. The trace log file in interim node VMis copied to original VM. Once all the trace log files are copied, there will be a replica of service components in original VM and VM on interim node. The network traffic is redirected to original VM and VM in interim node is shut down.

Yongquan Yan [12] proved that the selection of a proper data set is significant. The data set selection carries more weightage than the approach that is used for software agingprediction. Linear and non-linear methodswere used to detect resource consumption of aged web server. Metrics were collected from the system under true load and that is not given artificial load. In this studyIIS web server is used. A set of health care applications running on the IIS web server are considered for study. To handle the performance

degradation of IIS server, the only option was to restart as there was no hot standby system. Operating system parameters and database parameters in running phase are collected using built-in windows counter without disturbing the running system. Two metrics were collected at every one minute interval. The metrics were collected for available memory that represent operating system level and Java heap memory that represent application level. To forecast the consumption of resource, three models were used: support vector machine (SVM), auto regressive integrated moving average (ARIMA) and Aartificial neural network (ANN). The outcome of heap memory and available memory prediction indicate that non-linear methods do not perform better compared to linear methods in some situations. The researcher conclude that it is more important to choose proper data set than choosing non-linear or linear method.

Ahamad et al., [7] conducted survey regarding software aging issues. Found reasons and effects of software aging. The researchers concluded that concluded that it is impossible to stop software aging but it is possible to reduce its speed and progress.

Fang et al., [13] opined that, in some systems where the memory is shared between operating system and application software, as the memory consumption is closely related to system performance and changes constantly, using empirical thresholds may cause waste of resourcesor system outage. The researchers proposed an adaptive strategy adopted for optimization of the thresholds. Instead of fixed thresholds, the method regularly regulates the thresholds by taking feedback information in the running process into account. Critical equations are constructed to calculate the thresholds by maximizing the system availability and reliability. Simulation results demonstrated that the proposed method achieves higher availability and more stable performance than that based on empirical thresholds. If some parameters of the running environment are uncertain, the necessary parameters are estimated according to the software behavior during the constant running process. This method regulates the threshold by using feedback information in the running process. In the simulation results, it is observed that the proposed method achieves higher availability and stable performance.

Some of the solutions proposed can be seen in [14-19]. Some of the researchers have worked on different platforms. D. Cotroneo et al [20] presented the design of an aging detection and rejuvenation tool for Android ADaRTA which performs selective monitoring of system processes and of trends in system performance indicators. E. Andrade et al [21] proposed a deterministic and stochastic petri net (DSPN) for quantitatively analyzing the impacts of software aging phenomenon on a cyber-physical system using edge computing. Liu et al., [22] proposed a novel hybrid aging prediction model named CSSAP which is an integration of autoregressive integrated moving average (ARIMA) model and long short term memory (LSTM) model for cloud services. Several other works in the area are also surveyed [23-30].

Virtualized containers have been studied in [31-34]. Using containers, developers can design software in the local environment. Containers are smaller than virtual machines and hundreds of them run on a single physical machine. Software aging may affect such systems also. Mobile systems also get affected by software aging as studied in [35-37]. Machine learning has been used for software aging detection in mobile systems [38]. Markov stochastic processes have been used for analyzing software aging behavior [39]. Some researchers have provided asymptotic or steady-state solutions which are insufficient for real-time systems [40]. Machine learning has been used to predict software aging in [41-42]. System softwares may also get affected by software fault and related study can be found in [43]. Reliability of cloud-based systems with multiple software spare components has been studied in [44]. Similar researches can be found in [45-50]. Table 2 (see appendix) presents the summary of previous works in the area of software detection.

3. CONCLUSION

It can be observed from the literature that some of the works are platform specific and can not be applied to different platforms. Some studies have used machine learning for detection of software aging. But the accuracy of the aging detection can be improved by combining more than one method. For aging indicator metric collection, methods used by some of the researchers are intrusive, i.e., the metric collector software also consumes resoures. It can be concluded that it is impossible to stop software aging but it can be addressed only in run time because of the dynamic nature of the problem. The analysis of the previous works indicate that there is a necessity of non-intrusive, platform independent software aging detection techniques that combine the predictive power of machine learning tool and statistical models to strengthen the accuracy. As cloud computing has become the norm, it is worth conducting research in the area of software aging in cloud systems and provide a solution.

APPENDIX

Table 2. Summary of previous works

Researcher	Research work done	Remark
Alonso, et al., [9]	a. Analyzed a set of ML algorithms for predicting system crashes due to the resource exhaustion caused by software anomalies. b. Evaluated classifiers like Decision trees, Support Vector Machines and K- Nearest Neighbour c. Random Forest obtained a much better result (error less than 1%)	Used machine learning algorithms
Toshiaki, et al., [10]	a. Used a non-intrusive method of metrics collection using a machine that is separate from virtual machines and physical hosts. b. Httperf, a web benchmark was run on the client machines that is used to measure the request-response rate of files being transferred between the virtual machines and clients. c. The detection obtained using c4.5 machine learning classifier proved reliable d. A monitoring agent in every VM collects metrics; CPU usage and free memory available. The agent encapsulates this data into a packet and sends it to the aging detector module e. This process achieves two tasks; Expected Arrival Time (EAT) of next packet and the information it brings. If next metrics data packet arrives after EAT or does not arrive, it indicates some failure probability f. The CPU usage and free memory available is used to detect the aging severity.	Used machine learning algorithms
J. Liu, et al., [11]	a. A monitoring agent in every VM collects metrics; CPU usage and free memory available. The agent encapsulates this data into a packet and sends it to the aging detector module b. This process achieves two tasks; Expected Arrival Time (EAT) of next packet and the information it brings. If next metrics data packet arrives after EAT or does not arrive, it indicates some failure probability c. The CPU usage and free memory available is used to detect the aging severity.	Intrusive Method
Y. Yan, et al., [12]	a. Operating system parameters and database parameters in running phase are collected using built-in windows counter without disturbing the running system b. Only two metrics are considered; available memory that represent operating system level and .net common language runtime memory in all heaps (heap memory, for short) as well as Java heap memory in Apache web server that represent application level c. The forecasting of resources is done using three models, ARIMA, ANN, and SVM	Platform Specific Used IIS Webserver Proved choosing proper dataset is more important than choosing linear or non linear methods
S. Ahamad [7]	Conducted survey regarding aging issues. Found reasons of aging, effects of software aging.	Concluded that it is impossible to stop software aging but it is possible to reduce its speed and progress
Y. Fang, et al., [13]	a. Instead of fixed thresholds, the method used in this work regularly regulates the thresholds by taking feedback information in the running process into account. b. Critical equations are constructed to calculate the thresholds by maximizing the system availability.	Recommended Adaptive Threshold for aging detection
J. Liu, et al., [22]	Developed CSSAP which is an integration of Autoregressive Integrated Moving Average (ARIMA) model and Long Short-Term Memory (LSTM)	Hybrid aging prediction model

REFERENCES

[1] D. Cotroneo, et al., "Software Aging and Rejuvenation: Where We Are and Where We Are Going," *Proceedings 3rd International Workshop on Software Aging and Rejuvenation, (WoSAR 2011)*, pp. 1-6, 2011.

[2] Roberto Natella, "Software Aging and Rejuvenation," [Online], Available: http://wpage.unina.it/roberto.natella/.

[3] W. E. Wong and B. Cukic, "Adaptive Control Approach for Software Quality Improvement," *Series on Software Engineering and Knowledge Engineering*, vol. 20, 2011.

[4] R. Matias, et al., "Monitoring Memory-Related Software Aging: An Exploratory Study," *23th International Symposium on Software Reliability Engineering Workshops*, pp. 247-252, 2012.

[5] D. Cotroneo and R. Natella, "Monitoring of Aging Software Systems affected by Integer Overflows," *Proceedings of the IEEE 23rd International Symposium on Software Reliability Engineering*, pp. 265-270, 2012.

[6] A. Gupta, et al., "Prediction of Software Anomalies using Time Series Analysis – A recent study," *International Journal on Advanced Computer Theory and Engineering*, vol. 2, no. 3, pp. 101-108, 2013.

[7] S. Ahamad, "Study of Software Aging Issues and Prevention Solutions," *International Journal of Computer Science and Information Security*, vol. 14, no. 08, pp. 307-31, 2016.

[8] J. Alonso, et al., "A comparative experimental study of software rejuvenation overhead," *Performance Evaluation*, vol. 70, no. 3, pp. 231-250, 2013.

[9] J. Alonso, et al., "Predicting Software Anomalies Using Machine Learning Techniques," *IEEE 10th International Symposium on Network Computing and Applications*, Cambridge, MA, pp. 163-170, 2011.
[10] T. Hayashi, et al., "Performance Degradation Detection of Virtual Machines via Passive Measurement and Machine Learning," *International Journal of Adaptive, Resilient and Autonomic Systems*, vol. 5, no. 2, pp. 40-56, 2014.
[11] J. Liu, et al., "Software Rejuvenation based Fault Tolerance Scheme for Cloud Applications," *IEEE 8th International Conference on Cloud Computing*, pp. 1115-1118, 2015.
[12] Y. Yan, "A Practice Guide of Predicting Resource Consumption in a Web Server," *Review of Computer Engineering Studies*, vol. 2, no. 3, pp. 1-8, 2015.
[13] Y. Fang, et al., "A Rejuvenation Strategy of Two-Granularity Software Based on Adaptive Control," *IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*, Christchurch, pp. 104-109, 2017.
[14] F. Machida, et al., "Aging-Related Bugs in Cloud Computing Software," *IEEE 23rd International Symposium on Software Reliability Engineering Workshops (ISSREW)*, Dallas, TX, pp. 287-292, 2012.
[15] H. Okamura and T. Dohi, "Optimization of Opportunity-Based Software Rejuvenation Policy," *IEEE 23rd International Symposium on Software Reliability Engineering Workshops*, Dallas, TX, pp. 283-286, 2012.
[16] S. Li and Q. Yong, "Software Aging Detection Based on NARX Model," *Web Information Systems and Applications Conference (WISA)*, Haikou, pp. 105-110, 2012.
[17] D. Bruneo and S. Distefano, "Quantitative Assessments of Distributed Systems: Methodologies and Techniques," *Scrivener Publishing LLC*, 2015.
[18] F. Machida, et al., "Job Completion Time on a Virtualized Server Subject to Software Aging and Rejuvenation," *IEEE Third International Workshop on Software Aging and Rejuvenation (WoSAR)*, Hiroshima, pp. 44-49, 2011.
[19] L. Jiang, et al., "Time and Prediction based Software Rejuvenation Policy," *Second International Conference on Information Technology and Computer Science (ITCS)*, Kiev, pp. 114-117, 2010.
[20] D. Cotroneo, et al., "A Configurable Software Aging Detection and Rejuvenation Agent for Android," *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, Berlin, Germany, pp. 239-245, 2019.
[21] E. Andrade and F. Machida, "Analysis of Software Aging Impacts on Plant Anomaly Detection with Edge Computing," *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, Berlin, Germany, pp. 204-210, 2019.
[22] J. Liu, et al., "CSSAP: Software Aging Prediction for Cloud Services Based on ARIMA-LSTM Hybrid Model," *2019 IEEE International Conference on Web Services (ICWS)*, Milan, Italy, pp. 283-290, 2019.
[23] M. Torquato, et al., "An approach to investigate aging symptoms and rejuvenation effectiveness on software systems," *2017 12th Iberian Conference on Information Systems and Technologies (CISTI)*, Lisbon, pp. 1-6, 2017.
[24] I. M. Umesh, et al., "Software Aging Forecasting Using Time Series Model," *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 7, no. 3, pp. 839-845, 2017.
[25] I. M. Umesh, et al., "Software Rejuvenation Model for Cloud Computing Platform," *International Journal of Applied Engineering Research*, vol. 12, no. 19, pp. 8332-8337, 2017.
[26] M. Torquato, et al., "SWARE: A methodology for software aging and rejuvenation experiments," *Journal of Information Systems Engineering & Management*, vol. 3, no. 2, pp. 15-27, 2018.
[27] I. M. Umesh and G. N. Srinivasan, "Dynamic software aging detection-based fault tolerant software rejuvenation model for virtualized environment," in Satapathy S., et al., (eds), *Proceedings of the international conference on data engineering and communication technology. Advances in intelligent systems and computing*, vol. 469, pp. 779-787, 2017.
[28] I. M. Umesh, G. N. Srinivasan, M. Torquato, "Software Aging Forecasting Using Time Series Model," *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 7, no. 3, pp. 839-845, 2017.
[29] M. Torquato, et al., "Models for availability and power consumption evaluation of a private cloud with VMM rejuvenation enabled by VM live migration," *The Journal of Supercomputing*, vol. 74, no. 9, pp. 4817-4841, 2018.
[30] M. Torquato and M. Vieira, "An Experimental Study of Software Aging and Rejuvenation in Dockerd," *15th European Dependable Computing Conference (EDCC)*, Naples, Italy, pp. 1-6, 2019.
[31] A. Mouat, "Using Docker Developing and Deploying Software with Containers," *O'Reilly Media Pub.*, 2015.
[32] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81-84, 2014.
[33] N. Alshuqayran, et al., "A systematic mapping study in microservice architecture," *IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, pp. 44-51, 2016.
[34] D. Messina, "5 years later where are you on your docker journey?" *Docker blog*, 2018. [Online], Available: https://blog.docker.com/2018/03/5-years-later-docker-journey/.
[35] D. Cotroneo, et al., "Software aging analysis of the android mobile OS," *IEEE International Symposium on Software Reliability Engineering (ISSRE)*, pp. 478-489, 2016.
[36] C. Guo, et al., "Use two level rejuvenation to combat software aging and maximize average resource performance," *IEEE 12th International Conference on Embedded Software and Systems*, pp. 1160-1165, 2015.
[37] Y. Qiao, et al., "An empirical study of software aging manifestation in android," *International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 84-90, 2016.
[38] S. Huo, et al., "Using machine learning for software aging detection in Android system," *Tenth International Conference on Advanced Computational Intelligence (ICACI)*, Xiamen, pp. 741-746, 2018.
[39] H. Okamura, et al., "A Statistical Framework on Software Aging Modeling with Continuous-Time Hidden Markov Model," *Proceedings of IEEE 36th Symposium on Reliable Distributed Systems*, Hong Kong, pp. 114-123, 2017.

[40] W. Dang and J. Zeng, "Software System Rejuvenation Modeling Based on Sequential Inspection Periods and State Multi-control Limits," in B. Zou, et al., (eds), "Data Science," *ICPCSEE 2017, Communications in Computer and Information Science*, vol. 728, pp. 350-364, 2017.
[41] S. Jia, et al., "Software Aging Analysis and Prediction in a Web Server Based on Multiple Linear Regression Algorithm," *9th IEEE International Conference on Communication Software and Networks*, pp. 1452-1456, 2017.
[42] Y. Yan and P. Guo, "A Practice Guide of Software Aging Prediction in a Web Server Based on Machine Learning," *China Communications*, vol. 13, no. 6, pp. 225-235, 2016.
[43] M. Grotte, et al., "An empirical investigation of fault types in space mission system software," *Proc. of the International Conference on Dependable Systems & Networks (DSN 2010)*, pp. 447-456, 2010.
[44] J. Rahme and H. Xu, "Dependable and Reliable Cloud-Based Systems Using Multiple Software Spare Components," *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation*, pp. 1-8, 2017.
[45] I M Umesh et al., "Machine Learning Framework for Software Aging Forecasting," *Proc. of the Sixth Intl. Conf. On Advances In Computing, Control And Networking - ACCN 2017*, pp. 52-56, 2017.
[46] J. Zhao, "Modeling of Software Aging Based on Non-stationary Time Series," *International Conference on Information System and Artificial Intelligence (ISAI)*, Hong Kong, pp. 176-180, 2016.
[47] J. Rahme and H. Xu, "Preventive maintenance for cloud-based software systems subject to non-constant failure rates," *IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, San Francisco, CA, pp. 1-6, 2017.
[48] F. Machida, et al., "Lifetime Extension of Software Execution Subject to Aging," in *IEEE Transactions on Reliability*, vol. 66, no. 1, pp. 123-134, 2017.
[49] L. Li, et al., "Aberrant software-aging server detection and analysis using sliding window over LOF," *8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, Beijing, pp. 652-656, 2017.
[50] C. Weng, et al., "Analysis of Software Aging in Android," *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, Ottawa, pp. 78-83, 2016.

BIOGRAPHIES OF AUTHORS

Mrs. Sruthi P., is an Information Science and Engineering graduate. She also holds the M Tech degree in Computer Science and Enginerring from VTU. She is pursuing PhD under VTU in the area of Software aging and rejuvenation. She presently works as Assistant Professor in the Department of Computer Science and Engineering, Global Academy of Technology, Bengaluru, Karnataka. She has a total of 6 years of experience in teaching and industry. She has published several papers in International journals. She is active in research and also a life member CSI society.

Dr. Nagaraj G. Cholli obtained B.E in Computer Science & Engineering from Visvesvaraya Technological University, Belagavi, Karnataka, India and M.Tech in Computer Science and Engineering from IIT-Roorkee India. He completed Ph.D in Visvesvaraya Technological University, Belagavi, India in the year 2016 in the field of "Software Aging and Rejuvenation". He is Presently working as Associate Professor in the Department of Information Science and Engineering, R.V College of Engineering, Bengaluru, India. He has 13+ years of Research, Industry & Teaching experience in India and Abroad.Research Guidance: Total no. of PG guided: 26 and Total no. of Ph.D guiding: 06. He is working on consulting & funded projects approved by government of Karnataka & India. He has Published more than 30 Research papers in various National & International Conferences/Journals and filed 6 patents. He is also Life time member in ISTE, CSI, Sciei, IAENG, IRED professional societies.