Software Aging and Rejuvenation: Where we are and where we are going

Domenico Cotroneo, Roberto Natella, Roberto Pietrantuono, Stefano Russo
Dipartimento di Informatica e Sistemistica
Università degli Studi di Napoli Federico II
Via Claudio 21, 80125, Naples, Italy
{*cotroneo, roberto.natella, roberto.pietrantuono, sterusso*}*@unina.it*

***Abstract*—After 16 years, a significant body of knowledge has been established in the area of Software Aging and Rejuvenation (SAR). In this paper, we survey papers about SAR that appeared in IEEE conferences and journals, identify where SAR research has been mostly focused, and highlight some aspects deserving more attention, with the aim to provoke a constructive discussion among SAR researches about where SAR has arrived and where it should be headed in the next future.**

***Keywords*-Software Aging; Software Rejuvenation; Survey**

I. Introduction

Since 1995, year of publication of the seminal work of Huang et al. [1], much efforts have been devoted to characterize and mitigate the *Software Aging* phenomenon, that is, the accumulation of errors occurring in long-running operational software systems that leads to progressive resource depletion, performance degradation, and eventually to the hang or crash of the software system. As a result, a significant body of knowledge has been established and an international community of researchers in the area of Software Aging and Rejuvenation (SAR) has grown. Therefore, after 16 years, it is reasonable to look at what has been made, what has still to be accomplished to transfer the results to the industry world, and which are the future challenges for the SAR community. Starting from the analysis of 71 papers, which appeared in IEEE conferences and journals, this paper tries to identify where SAR research has been mostly focused, and to highlight some aspects which still deserve more attention by the SAR community. The aim is to stimulate a constructive discussion among SAR researches about where SAR has arrived and where it should be headed in the next future.

In Section II, we describe how SAR research papers have been collected. Section III reviews SAR literature with respect to four dimensions. Section IV concludes the paper with open issues and research opportunities.

II. Analysis of Literature

To have a picture of the current status of SAR literature, we analyzed some of the most important conference proceedings and journals. The steps followed for the analysis are:

Search engine. We relied on IEEE Xplore (http://ieeexplore.ieee.org/) to conduct the analysis.

Selection of conferences and journals. This preliminary analysis focused on the most relevant conferences/journals in the field of dependability, also considering the workshops held jointly with them: International Conference on Dependable Systems and Networks (DSN), International Symposium on Fault Tolerant Computing (FTCS), Pacific Rim International Symposium on Dependable Computing (PRDC), International Symposium on Software Reliability Engineering (ISSRE), International Symposium on Reliable Distributed Systems (SRDS), Transactions on Software Engineering (TSE), Transactions on Reliability (TR), Transactions on Computers (TC), Transactions on Dependable and Secure Computing (TDSC). In addition, we also considered part of the software engineering community (by querying for "software engineering" in the proceedings or journal title), addressing conferences/journals like International Symposium on Empirical Software Engineering (ISESE), International Conference on Software Engineering (ICSE), World Congress on software Engineering (WCSE), and other minor conferences.

Search criteria. The research has been carried out by querying for the words "aging", or "rejuvenation", or "leak" in the metadata of the IEEEXplore research engine (e.g., title, abstract, keywords, etc.). Note that in the case of software engineering, the word "aging", as well as "rejuvenation" has a different meaning: it indicates the software becoming obsolete, e.g., because of changed requirements, maintenance actions, and so on. As a result, most of the papers found in that case is not related to software aging as intended by the dependability community. Actually, only 8 papers of software engineering community are related to SAR. It should be noted that some papers on SAR appeared in venues not published by the IEEE or related to other computer science fields, and that we focus our analysis on the ones previously mentioned.

In Figure 1, the number of papers per year is reported. As may be noted, since 1995, the trend is increasing. A sharp increase of scientific productivity of the SAR community was fostered by the 2008 WoSAR workshop, which took place for the first time in that year. The venues preferred for SAR studies were ISSRE (30), DSN (12), and PRDC (8).

III. Where we are

Past work on SAR can be analyzed with respect to several dimensions. We here consider four dimensions, namely:

- *the type of analysis that has been conducted*,
- *the type of system to which the work is related*,

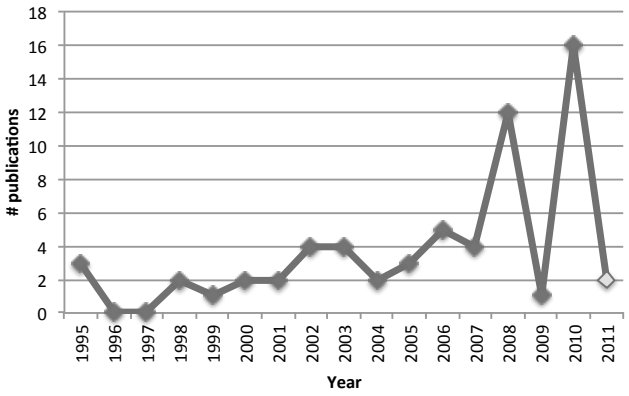

Figure 1. SAR publications per year in IEEE journals and conference proceedings.

- *the aging indicators analyzed*, and
- *the rejuvenation approach adopted or proposed*.

We believe that these dimensions provide insights on interesting research directions that could be pursued by the Software Aging and Rejuvenation community.

A. Type of analysis

From the point of view of the type of analysis, most SAR work focuses on predicting the *time-to-aging-failure* and on optimal scheduling of software rejuvenation strategies. Two main categories can be identified among these works, that is, *model-based analyses* and *measurement-based analyses*. In model-based analyses, a mathematical model of the system is considered, that include states in which the system is correctly working, states in which the system is failure-prone, and states in which software rejuvenation is taking place. Several kinds of model have been considered for this purpose, such as Markov Decision Processes and Stochastic Petri Nets [2], [3]. These studies typically aim at identifying the optimal time for applying software rejuvenation strategies in order to maximize availability or performability in the long term. Measurement-based analyses are instead based on data collected from a system about resource usage (e.g., free physical memory and used swap space) and performance (e.g., response throughput and latency) [4], [5]. These data are processed using algorithms for time series analysis and machine learning, in order to identify resource exhaustion / performance degradation trends. One of the aims of these studies is to provide a support to on-line planning of software rejuvenation, in order to predict aging failures in the short term and to adapt rejuvenation to the actual workload and resource consumption of the system. Another aim of measurement-based analysis is to provide empirical data about the software aging phenomenon; for instance, this data could be used to populate mathematical models. We also consider *hybrid analyses*, in which the approaches of measurement- and model-based approaches are combined, e.g., by populating models with actual measurements and to validate the effectiveness of models with respect to real aging failures. Finally, we consider another class of studies (referred to as "*other*") that do not belong to the classes previously mentioned. These studies are focused on various topics such as field-data analysis of aging-related bugs [6] and debugging techniques tailored to these bugs [7], and software rejuvenation techniques (these are discussed in section III-D) [8].

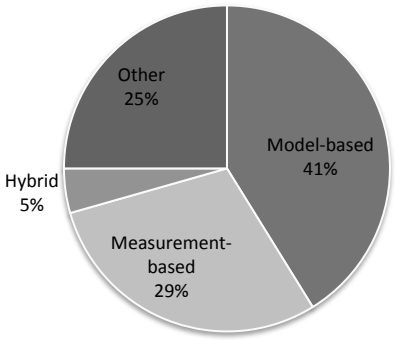

Figure 2. Type of analysis.

The distribution of the types of analysis is shown in Figure 2. Model-based analyses represent the largest share. Many studies have been devoted to analyze alternative models of systems affected by software aging: indeed, these models are needed in order to take into account different kinds of system (e.g., single-server and cluster systems) and rejuvenation (e.g., perfect and imperfect rejuvenation) [9], [10]. Instead, **hybrid analyses represent a very small part of SAR studies**. We believe that this kind of analysis deserves more attention, since it is fundamental to the adoption of model-based approaches in real-worlds systems. Indeed, following the hybrid approaches it is possible to provide examples on how measurements, which are already collected by modern systems for monitoring and debugging purposes, can be exploited to mitigate software aging and improve availability, and performability. Furthermore, the application of software rejuvenation schedules to real systems could serve to provide evidence that model-based approaches are effective at improving availability and performability. A notable example of cross-check between models and real failures can be found in [11].

B. Type of system

An important issue regards the type of system on which aging is analyzed, and/or rejuvenation actions are implemented. Software aging has been shown to affect many kinds of long-running software systems. We distinguish the analyzed papers in three classes: *safety critical systems*, *non-safety-critical systems*, *unspecified*, according to the employment scenarios. The first class indicates systems employed in scenarios that are critical from the safety point of view, i.e., systems whose malfunctioning may cause serious damages or loss of human lives; the non-safety-critical systems class includes business and mission-critical scenarios,

but not safety-critical ones. For instance, a Web Server or a DBMS are typically employed in non-safety-critical scenarios. The class *unspecified* refers to papers that do not present an experimentation on real systems, but that use simulation or numerical examples to demonstrate the validity of their results.

Figure 3 shows that, although safety-critical systems are designed to respect stringent dependability requirements and are tested extensively, a non-negligible percentage of papers have reported aging phenomena in this class of systems. This confirms that aging problems are very difficult to detect in the development phase. Moreover, it should be noted **that many papers (49%) do not perform experiments on real software systems**. The greatest part of these papers present model-based approaches for time-based rejuvenation, and validate their approach by numerical examples.

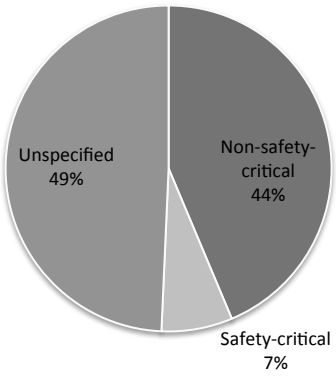

Figure 3. Type of system.

C. Aging indicators

Past studies have also been categorized with respect to the kind of software aging issue that has been investigated. Many studies propose models and approaches to deal with *resource depletion trends*, regardless of which specific resource is affected by depletion (i.e., it seems it has been assumed that the approach could be applied to rejuvenate any kind of resource). These papers are marked as "*unspecified*" in Figure 4. Most of remaining work focuses on software aging trends representing *memory consumption*, *performance degradation* or both (see for instance [4], [5]). These two aspects are the most frequent issues occurring in non-safety-critical systems (see section III-B). These issues have less relevance for safety-critical systems. For instance, in the case of software that undergoes a safety certification process, dynamic memory management is typically avoided in order to accomplish the most stringent safety levels. By contrast, **none of analyzed papers tackled arithmetic issues, such as the accumulation of round-off errors**. These errors are much more relevant in safety-critical contexts, since software is adopted in control systems. Suffice to think that many of the analyzed works, including ours, cited in their introduction the problem of the Patriot missile system but unfortunately only one work discussed issues related to it [12].

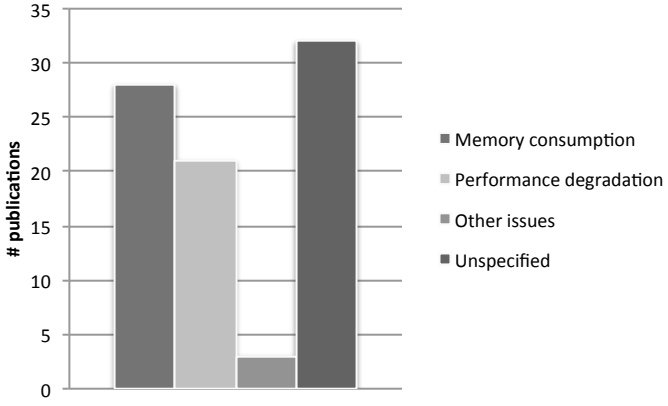

Figure 4. Aging indicators considered in past work.

D. Software rejuvenation approaches

The fourth dimension against which we evaluated SAR studies is with respect to the software rejuvenation actions that were proposed or adopted to counteract software aging. The most of SAR papers are focused on determining the optimal schedule to perform rejuvenation, by either analytical models (i.e., time-based rejuvenation), or by measurements (i.e., prediction-based rejuvenation). In this section, the attention is focused on techniques adopted to rejuvenate the system. Rejuvenation aims to bring the software from a *failure-prone* state (which is the result of errors accumulated due to software aging) to an *aging-free* state. Therefore, rejuvenation techniques can be compared with respect to how the state is processed and the resulting *aging-free* state that is achieved after rejuvenation.

We distinguish between *Application-generic* actions, *Application-specific* actions, and *Unspecified*. Application-generic actions are further classified in: *Component Restart*, *Application Restart*, *VM/VMM (Virtual Machine, or Virtual Machine Monitor) Restart*, *Node Reboot*, *Cluster Failover*. These actions are generic since they do not make use of application-specific features, but rely on *restarting* the system or its components to perform software rejuvenation, or they activate another copy of the system. By following this approach, the system or the component being rejuvenated is brought to its *initial state*, which is assured to be aging-free. This kind of rejuvenation is simple to implement since it makes use of initialization mechanisms of the system, and for this reason it is the most widespread.

By contrast, application-specific rejuvenation is tailored for a specific system: it aims at reducing the cost required to perform rejuvenation (i.e., the downtime and performance loss due to rejuvenation), by cleaning a specific aging-affected resource. This kind of rejuvenation introduces additional mechanisms in the system, and developers exploit peculiar features of the system or the resource. Some examples of application-specific rejuvenation are represented by garbage collection, kernel table flushing, defragmentation. State checkpointing mechanisms could also be adopted, although they need to be tailored to

the specific application in order to save only the relevant part of the system state and avoid to include aging-related errors in the checkpoint [13]. These techniques are effective at reducing the cost of rejuvenation since they do not bring the system to its *initial state*, and avoid to redo work for reconstructing the relevant system state (e.g., to restart transactions that were taking place at the time of rejuvenaton). This issue is negligible in the case of *stateless* applications (e.g., a web server), although it has great importance in the case of *stateful* applications that are adopted in critical systems.

The third class is "unspecified": this class includes those papers where the focus is on determining the optimal rejuvenation time, no matter what specific policy is adopted (e.g., many of these are model-based papers, where there is no application on which experiments are performed). In these studies, application-generic rejuvenation is often implicitly assumed.

Figure 5 shows the distribution among the three mentioned classes, as well as the distribution for the application-generic subcategories. **The "unspecified" class accounts for 59% of the total, which denotes that the focus is often on optimal time scheduling rather than on the design of more effective rejuvenation actions**. Among application-generic actions, the most common one is the *Application Restart*, whereas selective rejuvenation actions (e.g., VMM or component restarts) represent a small share.

IV. Where we are going

The preliminary survey presented in this paper attempts to reveal the trends followed by the software aging researchers in these years. Dimensions adopted for classifying the papers help us getting useful hints.

The first dimension highlighted that a lot of effort has been spent on designing analytical models for rejuvenation time scheduling. These models are becoming more and more refined and comprehensive; however, the works addressing aging by models often lack experimentation on real systems. Most often, models are validated by numerical examples, or by simulation, but to make them actually applicable in operational systems assumptions need to be verified against real system behavior. One relevant research opportunity we identify is about the further development of hybrid (i.e., model and measurement based) approaches, along with an increasing application in real-world systems. This would also provide examples of how software rejuvenation strategies can be applied, and encourage their adoption by practitioners. Toward this direction, the implementation of online monitoring and aging estimation frameworks may represent a valuable support in order to increase in the industrial world the perception and the awareness of the software aging problem.

The need for additional experimentation on real systems is also highlighted from the second dimension that we adopted. The non-negligible percentage of aging symptoms in safety-critical systems suggests us that: *i)* it is certainly worth to further investigate aging phenomena in safety-critical systems, since although well-tested they show to suffer from aging; *ii)* more attention to aging-related bugs is needed in the design and validation of safety-critical systems, since aging-related bugs can affect reliability requirements that are imposed on long-running systems by safety certification standards.

The analysis of aging indicators in Section III-C reports that memory-related and performance indicators received the same attention; however, further research is needed for the investigation of more complex software aging manifestations. There is evidence of several other types of aging bugs, such as numerical errors, storage-related bugs (e.g., when a bug consumes disk space), bugs related to the management of system-dependent data structures (e.g., shared memory pools in DBMSs). In particular, we did not find any suitable approach to cope with the accumulation of numerical errors, for which there is not an aging indicator that could be used by traditional model- or measurement-based analyses. Another missing piece of the puzzle is represented by a comprehensive analysis of real aging bugs.

Finally, in Section III-D we analyzed software rejuvenation approaches. From results, it is clear that much literature is devoted to the optimal schedule determination. A potential direction to further reduce the cost of software rejuvenation regards the development of more efficient techniques for rejuvenating a system. The problem of accounting for the state of the application may be addressed in the future, as well as the implementation of additional mechanisms that try to minimize the downtime by selectively rejuvenating part of the system. The adoption of these rejuvenation strategies among developers could be encouraged by the integration of rejuvenation mechanisms at the OS or middleware level and in programming languages.

References

[1] Y. Huang, C. Kintala, N. Kolettis, and N. Fulton, "Software rejuvenation: analysis, module and applications," in *Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers, Twenty-Fifth Int'l. Symp.*

[2] S. Garg, A. Puliafito, M. Telek, and K. Trivedi, "Analysis of software rejuvenation using markov regenerative stochastic petri net," in *Software Reliability Engineering, 1995. Proc., Sixth Int'l. Symp.*

[3] Y. Bao, X. Sun, and K. Trivedi, "A workload-based analysis of software aging, and rejuvenation," *Reliability, IEEE Transactions on*, vol. 54, no. 3, sept. 2005.

[4] S. Garg, A. van Moorsel, K. Vaidyanathan, and K. Trivedi, "A methodology for detection and estimation of software aging," in *Software Reliability Engineering, 1998. Proc. Ninth Int'l. Symp.*

[5] M. Grottke, L. Li, K. Vaidyanathan, and K. Trivedi, "Analysis of software aging in a web server," *Reliability, IEEE Transactions on*, vol. 55, no. 3, sept. 2006.

[6] M. Grottke, A. Nikora, and K. Trivedi, "An empirical investigation of fault types in space mission system software," in *Dependable Systems and Networks (DSN), 2010 Int'l. Conf.*

[7] H. Zhang, G. Wu, K. Chow, Z. Yu, and X. Xing, "Detecting resource leaks through dynamical mining of resource usage patterns," in *Dependable Systems and Networks Workshops (DSN-W), 2011 41st Int'l. Conf.*

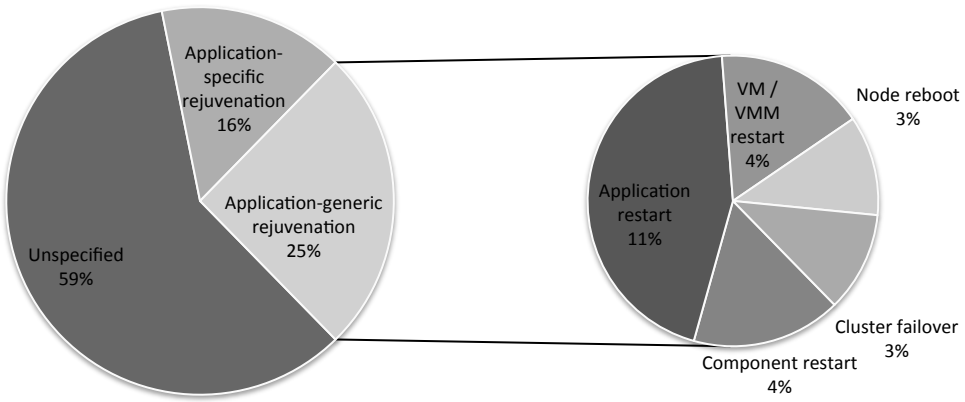

Figure 5. Software rejuvanation approaches.

[8] K. Kourai and S. Chiba, "Fast software rejuvenation of virtual machine monitors," *Dependable and Secure Computing, IEEE Transactions on*, vol. 8, no. 6, nov.-dec. 2011.

[9] W. Xie, Y. Hong, and K. Trivedi, "Software rejuvenation policies for cluster systems under varying workload," in *Dependable Computing, 2004. Proc. 10th IEEE Pacific Rim Int'l. Symp.*

[10] V. Koutras and A. Platis, "Modeling perfect and minimal rejuvenation for client server systems with heterogeneous load," in *Dependable Computing, 2008. 14th IEEE Pacific Rim Int'l. Symp.*

[11] R. Matias, K. Trivedi, and P. Maciel, "Using accelerated life tests to estimate time to software aging failure," in *Software Reliability Engineering (ISSRE), 2010 IEEE 21st Int'l. Symp.*

[12] M. Grottke, R. Matias, and K. Trivedi, "The fundamentals of software aging," in *Software Reliability Engineering Workshops, 2008. IEEE Int'l. Conf.*

[13] Y.-M. Wang, Y. Huang, K.-P. Vo, P.-Y. Chung, and C. Kintala, "Checkpointing and its applications," in *Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth Int'l. Symp.*

[14] K. Cassidy, K. Gross, and A. Malekpour, "Advanced pattern recognition for detection of complex software aging phenomena in online transaction processing servers," in *Dependable Systems and Networks, 2002. Proc. Int'l. Conf.*

[15] K. Kourai and S. Chiba, "A fast rejuvenation technique for server consolidation with virtual machines," in *Dependable Systems and Networks, 2007. 37th Int'l. Conf.*

[16] J. Alonso, J. Torres, J. Berral, and R. Gavalda, "Adaptive on-line software aging prediction based on machine learning," in *Dependable Systems and Networks (DSN), 2010 Int'l. Conf.*

[17] A. Avritzer, A. Bondi, M. Grottket, K. Trivedi, and E. Weyuker, "Performance assurance via software rejuvenation: Monitoring, statistics and algorithms," in *Dependable Systems and Networks, 2006. Int'l. Conf.*

[18] A. Tai, K. Tso, W. Sanders, and S. Chau, "A performability-oriented software rejuvenation framework for distributed applications," in *Dependable Systems and Networks, 2005. Proc. Int'l. Conf.*

[19] R. Hanmer and V. Mendiratta, "Rejuvenation with workload migration," in *Dependable Systems and Networks Workshops (DSN-W), 2010 Int'l. Conf.*

[20] Y. Bao, X. Sun, and K. Trivedi, "Adaptive software rejuvenation: degradation model and rejuvenation scheme," in *Dependable Systems and Networks, 2003. Proc. 2003 Int'l. Conf.*

[21] M. Shereshevsky, J. Crowell, B. Cukic, V. Gandikota, and Y. Liu, "Software aging and multifractality of memory resources," in *Dependable Systems and Networks, 2003. Proc. 2003 Int'l. Conf.*

[22] G. Candea, J. Cutler, A. Fox, R. Doshi, P. Garg, and R. Gowda, "Reducing recovery time in a small recursively restartable system," in *Dependable Systems and Networks, 2002. Proc. Int'l. Conf.*

[23] L. Silva, V. Batista, and J. Silva, "Fault-tolerant execution of mobile agents," in *Dependable Systems and Networks, 2000. Proc. Int'l. Conf.*

[24] Y.-F. Jia, J.-Y. Su, and K.-Y. Cai, "A feedback control approach for software rejuvenation in a web server," in *Software Reliability Engineering Workshops, 2008. IEEE Int'l. Conf.*

[25] H. Suzuki, T. Dohi, N. Kaio, and K. Trivedi, "Maximizing interval reliability in operational software system with rejuvenation," in *Software Reliability Engineering, 2003. 14th Int'l. Symp.*

[26] H. Okamura, S. Miyahara, and T. Dohi, "Dependability analysis of a client/server software system with rejuvenation," in *Software Reliability Engineering, 2002. Proc. 13th Int'l. Symp.*

[27] H. Shetty, M. Nambiar, and H. Kalita, "Analysis and application of conditional software rejuvenation—a new approach," in *Software Reliability Engineering Workshops, 2008. IEEE Int'l. Conf.*

[28] F. Machida, D. S. Kim, J. S. Park, and K. Trivedi, "Toward optimal virtual machine placement and rejuvenation scheduling in a virtualized data center," in *Software Reliability Engineering Workshops, 2008. IEEE Int'l. Conf.*

[29] A. Avritzer, R. Cole, and E. Weyuker, "Methods and opportunities for rejuvenation in aging distributed software systems," in *Software Reliability Engineering Workshops, 2008. IEEE Int'l. Conf.*

[30] A. Bobbio, S. Garg, M. Gribaudo, A. Horvath, M. Sereno, and M. Telek, "Compositional fluid stochastic petri net model for operational software system performance," in *Software Reliability Engineering Workshops, 2008. IEEE Int'l. Conf.*

[31] Y. Liu, K. Trivedi, Y. Ma, J. Han, and H. Levendel, "Modeling and analysis of software rejuvenation in cable modem termination systems," in *Software Reliability Engineering, 2002. Proc. 13th Int'l. Symp.*

[32] D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "Software aging analysis of the linux operating system," in *Software Reliability Engineering (ISSRE), 2010 IEEE 21st Int'l. Symp.*

[33] K. Vaidyanathan and K. Trivedi, "A measurement-based model for estimation of resource exhaustion in operational software systems," in *Software Reliability Engineering, 1999. Proc. 10th Int'l. Symp.*

[34] V. Koutras, A. Platis, and N. Limnios, "Availability and reliability estimation for a system undergoing minimal, perfect and failed rejuvenation," in *Software Reliability Engineering*

Workshops, 2008. IEEE Int'l. Conf.

[35] H. Okamura and T. Dohi, "Availability optimization in operational software system with aperiodic time-based software rejuvenation scheme," in *Software Reliability Engineering Workshops, 2008. IEEE Int'l. Conf.*

[36] K. Rinsaka and T. Dohi, "Non-parametric predictive inference of adaptive software rejuvenation schedule," in *Software Reliability Engineering Workshops, 2008. IEEE Int'l. Conf.*

[37] X. Wang, J. Xu, and C. Pham, "An effective method to detect software memory leakage leveraged from neuroscience principles governing human memory behavior," in *Software Reliability Engineering, 2004. 15th Int'l. Symp.*

[38] G. Carrozza, D. Cotroneo, R. Natella, A. Pecchia, and S. Russo, "An experiment in memory leak analysis with a mission-critical middleware for air traffic control," in *Software Reliability Engineering Workshops, 2008. IEEE Int'l. Conf.*

[39] F. Salfner and K. Wolter, "A queuing model for service availability of systems with rejuvenation," in *Software Reliability Engineering Workshops, 2008. IEEE Int'l. Conf.*

[40] J. Zhao, K. Trivedi, Y. Wang, and X. Chen, "Evaluation of software performance affected by aging," in *Software Aging and Rejuvenation (WoSAR), 2010 IEEE Second Int'l. Workshop on*.

[41] R. Matias, I. Beicker, B. Leitao, and P. Maciel, "Measuring software aging effects through os kernel instrumentation," in *Software Aging and Rejuvenation (WoSAR), 2010 IEEE Second Int'l. Workshop on*.

[42] D. Cotroneo, R. Natella, and R. Pietrantuono, "Is software aging related to software metrics?" in *Software Aging and Rejuvenation (WoSAR), 2010 IEEE Second Int'l. Workshop on*.

[43] H. Okamura and T. Dohi, "Performance-aware software rejuvenation strategies in a queueing system," in *Software Aging and Rejuvenation (WoSAR), 2010 IEEE Second Int'l. Workshop on*.

[44] A. Macedo, T. Ferreira, and R. Matias, "The mechanics of memory-related software aging," in *Software Aging and Rejuvenation (WoSAR), 2010 IEEE Second Int'l. Workshop on*.

[45] P. Reinecke and K. Wolter, "A simulation study on the effectiveness of restart and rejuvenation to mitigate the effects of software ageing," in *Software Aging and Rejuvenation (WoSAR), 2010 IEEE Second Int'l. Workshop on*.

[46] A. Platis and V. Koutras, "Software rejuvenation on a pki," in *Software Aging and Rejuvenation (WoSAR), 2010 IEEE Second Int'l. Workshop on*.

[47] J. Magalhaes and L. Silva, "Prediction of performance anomalies in web-applications based-on software aging scenarios," in *Software Aging and Rejuvenation (WoSAR), 2010 IEEE Second Int'l. Workshop on*.

[48] F. Machida, D. S. Kim, and K. Trivedi, "Modeling and analysis of software rejuvenation in a server virtualized system," in *Software Aging and Rejuvenation (WoSAR), 2010 IEEE Second Int'l. Workshop on*.

[49] M. Moreno and L. Soares, "Resilient hypermedia presentations," in *Software Aging and Rejuvenation (WoSAR), 2010 IEEE Second Int'l. Workshop on*.

[50] K. Vaidyanathan, D. Selvamuthu, and K. Trivedi, "Analysis of inspection-based preventive maintenance in operational software systems," in *Reliable Distributed Systems, 2002. Proc. 21st IEEE Symp.*

[51] D. Cotroneo, S. Orlando, and S. Russo, "Characterizing aging phenomena of the java virtual machine," in *Reliable Distributed Systems, 2007. 26th IEEE Int'l. Symp.*

[52] T. Dohi, T. Danjou, and H. Okamura, "Optimal software rejuvenation policy with discounting," in *Dependable Computing, 2001. Proc. 2001 Pacific Rim Int'l. Symp.*

[53] T. Dohi, K. Goseva-Popstojanova, and K. Trivedi, "Statistical non-parametric algorithms to estimate the optimal software rejuvenation schedule," in *Dependable Computing, 2000. Proc. 2000 Pacific Rim Int'l. Symp.*

[54] G. A. Hoffmann, K. S. Trivedi, and M. Malek, "A best practice guide to resources forecasting for the apache webserver," in *Dependable Computing, 2006. 12th Pacific Rim Int'l. Symp.*

[55] T. Tsai, K. Vaidyanathan, and K. Gross, "Low-overhead run-time memory leak detection and recovery," in *Dependable Computing, 2006. 12th Pacific Rim Int'l. Symp.*

[56] C. Fetzer and K. Hogstedt, "Rejuvenation and failure detection in partitionable systems," in *Dependable Computing, 2001. Proc. 2001 Pacific Rim Int'l. Symp.*

[57] V. Sundaram, S. HomChaudhuri, S. Garg, C. Kintala, and S. Bagchi, "Improving dependability using shared supplementary memory and opportunistic micro rejuvenation in multi-tasking embedded systems," in *Dependable Computing, 2007. 13th Pacific Rim Int'l. Symp.*

[58] A. van Moorsel and K. Wolter, "Analysis of restart mechanisms in software systems," *Software Engineering, IEEE Transactions on*, vol. 32, no. 8, aug. 2006.

[59] L. Silva, J. Alonso, and J. Torres, "Using virtualization to improve software rejuvenation," *Computers, IEEE Transactions on*, vol. 58, no. 11, nov. 2009.

[60] S. Garg, A. Puliafito, M. Telek, and K. Trivedi, "Analysis of preventive maintenance in transactions based software systems," *Computers, IEEE Transactions on*, vol. 47, no. 1, jan 1998.

[61] K. Vaidyanathan and K. Trivedi, "A comprehensive model for software rejuvenation," *Dependable and Secure Computing, IEEE Transactions on*, vol. 2, no. 2, april-june 2005.

[62] R. Matias, P. Barbetta, K. Trivedi, and P. Filho, "Accelerated degradation tests applied to software aging experiments," *Reliability, IEEE Transactions on*, vol. 59, no. 1, march 2010.

[63] G. Hoffmann, K. Trivedi, and M. Malek, "A best practice guide to resource forecasting for computing systems," *Reliability, IEEE Transactions on*, vol. 56, no. 4, dec. 2007.

[64] L. Li, K. Vaidyanathan, and K. Trivedi, "An approach for estimation of software aging in a web server," in *Empirical Software Engineering, 2002. Proc. 2002 Int'l. Symp.*

[65] J. Guo, W. Li, X. Song, B. Zhang, and Y. Wang, "Software rejuvenation strategy based on components," in *Software Engineering, 2010 Second World Congress on*.

[66] L. Jiang, X. Peng, and G. Xu, "Software rejuvenation practice," in *Software Engineering, 2009. WRI World Congress on*.

[67] Y.-F. Jia, L. Zhao, and K.-Y. Cai, "A nonlinear approach to modeling of software aging in a web server," in *Software Engineering Conf., 2008.15th Asia-Pacific*.

[68] K. Gross, V. Bhardwaj, and R. Bickford, "Proactive detection of software aging mechanisms in performance critical computers," in *Software Eng. Workshop, 2002. Proc. 27th Annual NASA Goddard/IEEE*.

[69] G. Xu and A. Rountev, "Precise memory leak detection for java software using container profiling," in *Software Engineering, 2008. ACM/IEEE 30th Int'l. Conf.*

[70] B. Qu, J. Shu, Y. Huang, and Y. Lu, "Memory leak dynamic monitor based on hook technique," in *Computational Intelligence and Software Engineering, 2009. Int'l. Conf.*

[71] Z. Alzamil, "Application of computational redundancy in dangling pointers detection," in *Software Eng. Advances, Int'l. Conf.*, 2006.