Chapter 1

Future Directions for Software Aging and Rejuvenation Research

Alberto Avritzer, Roberto Pietrantuono, Kishor Trivedi
eSulabSolutions, Inc., Princeton, USA
Universit degli Studi di Napoli Federico II, Napoli, Italy
Duke University, USA

1. Introduction

In this chapter we present a summary of some future directions for software aging and rejuvenation research. This chapter follows the book organization and is divided into four sections: 1) fundamentals, 2) data-driven approaches, 3) model-driven approaches, and, 4) applications.

2. Fundamentals

2.1. *Control Theory*

Measurements of software aging, followed by the analysis of measured data and prediction of the time to exhaustion of OS resources can allow the execution of a possible rejuvenation action. This MAPE (Measure, Analyze, Predict and Execute) loop is well known in feedback control systems. We expect that researchers in future will take this on-line feedback control view of software aging and rejuvenation.[1] These ideas are also being proposed for cyber security.[2,3] Essential components of such a control system are the controlled entity, the controller and the feedback loop. Furthermore there is also a goal to achieve.

In our case, the controlled entity would be the software system – e.g., the OS, VMM, VM, an application or an application component. The goal could be to maximize the software availability, minimize response time, minimize the total cost of system operation, etc. Alternatively, the goal could be to prevent or postpone software failures caused by aging-related bugs.

If rejuvenation scheduling is time-based then we need the distribution of time to failure as one of the inputs. So measured sequence of inter-failure times is needed. Once this sequence is available, we can either fit a distribution to this data (such as the Weibull or the hypo-exponential) and then use the 3-state model SMP (as

discussed in Chapter 5 Markov chains and Petri nets by Fumio Machida and Paulo R. M. Maciel,of this book) to determine the optimal rejuvenation schedule.

Alternatively we can use the non-parametric method discussed in the work by Dohi et *al.*[4] In the original model, inter-failure time data is assumed to be measured for a system without rejuvenation. But to use this method in an on-line feedback control setting we will need to account for the fact that data collected is modified, as it is inter-failure time data with rejuvenation included. This requires considerable amount of future research.

In the case that inspection-based approach is used for rejuvenation scheduling, the data to be measured/collected is for health indicator performance variables that show the effects of aging. Once the specific performance variables are chosen, we may determine threshold values for these variables to trigger rejuvenation[5] or use prediction methods to determine the time to resource exhaustion and hence trigger rejuvenation. Determining what performance variables to monitor and how to determine threshold values are topics for future research. Though methods of prediction of individual resource exhaustion are known, prediction of the overall time to system failure needs further research. On-line measurement of aging will be increasingly important as a closed-loop feedback control becomes more prevalent. Finally, software rejuvenation scheduling in the context of multi-granularity software rejuvenation needs a lot of future research.[5,6]

3. Data-Driven Approaches

3.1. *Data Analysis and Machine Learning*

We expect to see an increased use of machine learning techniques because they represent a hybrid between model based and measurement based approaches. Indeed, as systems become more and more complex, several indicators will need to be analyzed to detect software aging.

Moreover, it will become increasingly difficult to distinguish between software aging and the expected fluctuations of the system performance, because of inter-relations among multiple parameters.

All these call for more complex statistical techniques that are able of analyzing the cause/effect relations (besides correlations) among multiple indicators and being able to spot those critical patterns relevant for aging identification in complex systems.

3.2. *Experimental approaches to detect aging root causes*

Experimental approaches to detect software aging are involved in the detection of aging root causes and approaches to eliminate aging causes.

There is a need to research experimental approaches to deal with aging bugs pre-release detection, removal or prevention. While many studies assume that aging is there, and thus deal with how to "tolerate" aging effects and when/how to

rejuvenate the system affected by aging, little effort has been spent to *i)* **prevent** the introduction of aging bugs (e.g., design methodologies), *ii)* **detect** and **remove** the root causes before software delivery (i.e., at development time) – such as proper testing and debugging techniques besides dynamic analysis tools used currently, *iii)* **predict** aging bugs before they are triggered at runtime and cause aging. These are all open topics which future works can cope with.

A few attempts have been made about aging bugs classification and prediction, which can be a starting point for the aging prediction task. The software aging and rejuvenation repository[7] describes a software aging repository that can be used to support experimental approaches for software aging detection. Data are organized into two separate repositories. The former contains information on aging-related bugs found in two open-source projects (the Linux kernel and the MySQL DBMS). This dataset has been used to investigate defect prediction approaches for aging-related bugs, by using software complexity metrics and machine learning techniques. The associated repository can be accessed on: `https://zenodo.org/record/581659#.XFqeR_x7nVo`.

The latter contains a list bugs from four open-source projects (the Linux kernel, the MySQL DBMS, the Apache HTTPD web server, and the Apache AXIS WS framework), classified into Mandelbugs, Bohrbugs, or Aging-Related Bugs, by analyzing the conditions that exercise the bug (i.e., the "fault trigger"). The associated repository can be found at: `https://zenodo.org/record/581660#.XFqePfx7nVo`

4. Model-Driven Approaches

4.1. *Cloud Computing and Virtualized Infrastructures*

There is a need for comprehensive models that are able to account for the cost/benefits of rejuvenation actions at several layers of the cloud stack, and to be able to account for the increasing complexity of virtualization technologies. In addition, research into new aging indicators for the different layers and components of a cloud-based system, and on measuring cause-effect relations across virtualization layers are also needed.

Software rejuvenation research in cloud computing infrastructures may include the following topics:

- Investigating the cost/benefit of (passive/active) replication-based strategies (for rejuvenation of VMM, VM, container, application) not only in relation to the downtime reduction, but also considering the cost of setting up and managing replicas, their energy consumption, the side-effects on security as well as on scalability,
- Analyzing the overhead of rejuvenation in virtualized systems, e.g., the effect on memory or storage fragmentation,
- Micro-reboot for parts of the virtualization infrastructure,

- Security and aging in cloud-based systems: aging-related vulnerability/attacks in the cloud (e.g., induced VM leakage),
- Energy consumption as key factor in cloud-based systems, multi-objecive aging detection and multi-objective rejuvenation (considering availability, resource consumption, energy, security,),
- SAR in emerging contexts related to the cloud/virtualization, such as: edge and fog computing, network function virtualization/software-defined networks,
- Impact of deployment on several layers and interactions between them. How to define a holistic approach for availability improvements when aging can occur at several layers of the infrastructure, cloud and virtualization?

4.2. *Restart approaches*

Chapter 7 on Models for Restart, Reboot and Rejuvenation contains a detailed section (1.7) describing proposals for future work related to black-box models of restart.

5. Applications

5.1. *Operating Systems*

Chapter 9 on applications to Operating Systems contains suggestions for future research related to Operating Systems.

5.2. *Software Architecture*

Monitoring is a key facility in microservice architectures. Therefore monitoring could be exploited for software aging detection.

Identifying architecture patterns that are likely to cause software aging is a topic that is related to research on software performance anti-patterns.[8–11] Related research to software performance anti-patterns is a characterization of the sub-set of performance anti-patterns that could be defined as software aging anti-patterns. Examples of some software performance anti-patterns that could be related to software aging anti-patterns are:

- Circuitous Treasure Hunt - when database statements are poorly organized, data might be retrieved from several tables in sequence, as if the programmer is engaged in a circuitous treasure hunt. As a consequence, response times of programs that implement this performance anti-pattern are usually long. As time passes, it is possible that database tables will grow and the software will appear to age.
- Extensive Processing - if a long running process is in control of a resource and a thread is blocked waiting for that resource, the extensive processing

anti-pattern will occur. In addition, if the execution time of the long running process is a function of the process restart time, the software might show aging.
- Wrong Cache Strategy - when a cache strategy is wrongly implemented, it might be related to memory leaks and excessive number of garbage collection events. The accumulation of such events over time can be related to software aging.

5.3. *AI Systems*

The resilience requirements of mission-critical systems and the expected ubiquity of AI systems and Robotics, will require that such system to be tolerant to software aging. In these systems, a further difficulty is caused by the complex and dynamic interaction with surrounding environment, which can cause the software aging dynamics to be much harder to, model, to detect and to tolerate.

5.4. *Mobile Devices and Embedded Systems*

Software aging and rejuvenation of embedded and mobile systems have been addressed.[12–14]

In the work by Kintala[12] memory and stack overflow problems in mobile devices were attributed to the lack of tools to properly configure memory in mobile devices. A technique called micro-rejuvenation was introduced to detect and correct memory and stack overflow problems. This book Chapter 12 on implementation of micro-rejuvenation with cooperative stack leasing extends on the research presented by Kintala[12] to propose a collaborative approach among system stacks.

This is a very promising future research direction, as software aging and rejuvenation to address mobile devices is a very effective approach to address mobile systems unreliability issues, such as, stack and heap overflow problems.

5.5. *Anomaly Detection Systems*

Anomaly detection systems based on system performance metrics have been proposed for the detection of security intrusions.[15–18] In addition, software rejuvenation was introduced to counteract worm epidemics in large distributed systems.[19]

Related future research on software aging and rejuvenation could address some of the following topics:

- How to define metrics to detect software aging related to security intrusions[16]?
- How to distinguish between software aging related to security intrusions and software aging related to performance and reliability issues[20]?
- How to design and implement an automated approach for software aging and rejuvenation to counteract security intrusions[21]?

6. Other Topics for future research

Prognostics and Health Management-PHM could be combined with software aging detection and rejuvenation techniques in the future.

Rejuvenation can be also combined with the Remaining Useful Life-RUL prediction, in the sense that rejuvenation schedule could be planned according to RUL predicted values.

When rejuvenation is implemented, the software system is not shut down. So, under rejuvenation the system can still operate although in a reduced capability mode. Thus, this concept cannot be considered when availability is examined, since in this case the system is considered as down when being rejuvenated. Alternative performance measures concerning the capability of the system to provide service (even under rejuvenation) could be defined and examined.

References

1. Y. Hong, D. Chen, L. Li, and K. Trivedi. Closed loop design for software rejuvenation. In *Workshop on Self-Healing, Adaptive and self-MANaged Systems (SHAMAN)*, (2002).
2. M. Platania, D. Obenshain, T. Tantillo, R. Sharma, and Y. Amir. Towards a practical survivable intrusion tolerant replication system. In *33rd IEEE International Symposium on Reliable Distributed Systems, SRDS 2014, Nara, Japan, October 6-9, 2014*, pp. 242–252, (2014).
3. R. Romagnoli, B. H. Krogh, and B. Sinopoli, Design of software rejuvenation for CPS security using invariant sets, *CoRR*. **abs/1810.10484**, (2018).
4. T. Dohi, K. Goševa-Popstojanova, and K. S. Trivedi, Estimating software rejuvenation schedule in high assurance systems, *Comput. J.* **44**(6), 473–485, (2001).
5. G. Ning, J. Zhao, Y. Lou, J. Alonso, R. Matias, K. S. Trivedi, B. B. Yin, and K. Y. Cai, Optimization of two-granularity software rejuvenation policy based on the Markov regenerative process, *IEEE Trans. Rel.* **65**(4), 1630–1646, (2016).
6. W. Xie, Y. Hong, and K. Trivedi, Analysis of a two-level software rejuvenation policy, *Reliability Engineering and System Safety*. **87**(1), 13–22, (2005).
7. D. Cotroneo, A. K. Iannillo, R. Natella, R. Pietrantuono, and S. Russo. The software aging and rejuvenation repository: Http://openscience.us/repo/software-aging/. In *2015 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 108–113 (Nov, 2015). doi: 10.1109/ISSREW.2015.7392054.
8. C. U. Smith and L. G. Williams. Software performance antipatterns. In *Proceedings of the 2Nd International Workshop on Software and Performance*, WOSP '00, pp. 127–136, New York, NY, USA, (2000). ACM. ISBN 1-58113-195-X. doi: 10.1145/350391.350420. URL `http://doi.acm.org/10.1145/350391.350420`.
9. C. Trubiani, A. Bran, A. van Hoorn, A. Avritzer, and H. Knoche, Exploiting load testing and profiling for performance antipattern detection, *Information & Software Technology*. **95**, 329–345, (2018). doi: 10.1016/j.infsof.2017.11.016. URL `https://doi.org/10.1016/j.infsof.2017.11.016`.
10. T.-H. Chen, W. Shang, Z. M. Jiang, A. E. Hassan, M. Nasser, and P. Flora. Detecting performance anti-patterns for applications developed using object-relational mapping. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE

2014, pp. 1001–1012, New York, NY, USA, (2014). ACM. ISBN 978-1-4503-2756-5. doi: 10.1145/2568225.2568259. URL http://doi.acm.org/10.1145/2568225.2568259.

11. A. Wert, M. Oehler, C. Heger, and R. Farahbod. Automatic detection of performance anti-patterns in inter-component communications. In *Proceedings of the 10th International ACM Sigsoft Conference on Quality of Software Architectures*, QoSA '14, pp. 3–12, New York, NY, USA, (2014). ACM. ISBN 978-1-4503-2576-9. doi: 10.1145/2602576.2602579. URL http://doi.acm.org/10.1145/2602576.2602579.
12. C. M. Kintala, Software rejuvenation in embedded systems, *J. Autom. Lang. Comb.* **14**(1), 63–73 (Jan., 2009). ISSN 1430-189X. URL http://dl.acm.org/citation.cfm?id=1643359.1643364.
13. J. Xiang, C. Weng, A. Andrzejak, Z. Dongdong, T. Jing, X. Shengwu, and L. Lin. A new software rejuvenation model for android. In *10th International Workshop on Software Aging and Rejuvenation, WoSAR 2018* (10, 2018). doi: 10.1109/ISSREW.2018.00021.
14. D. Cotroneo, F. Fucci, A. K. Iannillo, R. Natella, and R. Pietrantuono. Software aging analysis of the android mobile OS. In *27th IEEE International Symposium on Software Reliability Engineering, ISSRE 2016, Ottawa, ON, Canada, October 23-27, 2016*, pp. 478–489, (2016). doi: 10.1109/ISSRE.2016.25. URL https://doi.org/10.1109/ISSRE.2016.25.
15. A. Avritzer, R. G. Cole, and E. J. Weyuker. Using performance signatures and software rejuvenation for worm mitigation in tactical manets. In *Proceedings of the 6th International Workshop on Software and Performance, WOSP 2007, Buenes Aires, Argentina, February 5-8, 2007*, pp. 172–180, (2007).
16. A. Milenkoski, M. Vieira, S. Kounev, A. Avritzer, and B. D. Payne, Evaluating computer intrusion detection systems: A survey of common practices, *ACM Computing Surveys (CSUR)*. **48**(1), 12, (2015).
17. C. Wang, Z. Zhao, L. Gong, L. Zhu, Z. Liu, and X. Cheng, A distributed anomaly detection system for in-vehicle network using htm, *IEEE Access*. **6**, 9091–9098, (2018).
18. C. Kruegel and G. Vigna. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, CCS '03, pp. 251–261, (2003). ISBN 1-58113-738-9.
19. A. Avritzer, R. G. Cole, and E. J. Weyuker, Methods and opportunities for rejuvenation in aging distributed software systems, *Journal of Systems and Software*. **83**(9), 1568–1578, (2010). doi: 10.1016/j.jss.2010.05.026. URL https://doi.org/10.1016/j.jss.2010.05.026.
20. A. Avritzer, A. Bondi, and E. J. Weyuker. Ensuring stable performance for systems that degrade. In *Proceedings of the 5th International Workshop on Software and Performance*, WOSP '05, pp. 43–51, New York, NY, USA, (2005). ACM. ISBN 1-59593-087-6. doi: 10.1145/1071021.1071026. URL http://doi.acm.org/10.1145/1071021.1071026.
21. A. Avritzer, V. Ferme, A. Janes, B. Russo, H. Schulz, and A. van Hoorn. A quantitative approach for the assessment of microservice architecture deployment alternatives by automated performance testing. In *12 European Conference on Software Architecture*, (2018).