SAR Handbook Chapter: Measurements-based aging analysis

Javier Alonso[†,*], Kalyan Vaidyanathan[‡], Roberto Pietrantuono[§]
[†] Amazon, Seattle, USA, jjava@amazon.com
[‡] BAE Systems, San Diego, USA, kalyan.vaidyanathan@baesystems.com
[§] Università degli Studi di Napoli Federico II, roberto.pietrantuono@unina.it

***Abstract*—This paper summarizes the main methods adopted for the analysis and detection of software aging phenomena based on measurements (measurements-based aging analysis) as well as the metrics more commonly used as aging indicators.**

I. Measurements for software aging analysis

One broad class of techniques to cope with software agin is the measurement-based strategy [1]. In measurement-based analysis, the focus is on defining a proper set of aging indicators, on observing them to determine the existence of the phenomenon and, once detected, estimating the remaining time to live of the system [2]. This in turn allows determining the *optimal* moment to trigger the software rejuvenation process. The way in which aging indicators are analyzed range from considering simpler but less accurate techniques, such as observing the value of some key metric, to more accurate but complex strategies relying on hypothesis tests for trend detection, time series analysis, up to machine learning algorithms to support the diagnosis of the system's aging state. We focus on two dimensions to classify measurement-based techniques: *i)* the indicators to analyze the aging phenomenon, *ii)* the analysis technique. These are briefly discussed in the following. A detailed explanation is in the *Handbook of Software Aging and Rejuvenation*, Chapter 4 [3].

II. Aging indicators

The correct identification of metrics capturing the aging of the system is of paramount importance to have a clear view on the system health's state. Aging indicators can refer to *resource usage* and to *performance*. Table I synthesizes commonly used metrics, which are about **Performance degradation** – the main user-perceived effect caused by aging - **Memory consumption**, such as *free physical memory* and *swap space*, which are by far the most commonly adopted ones, and other **resource consumption** besides memory, which monitor resources such as the filesystem, CPU, storage, network, or application-specific metrics (e.g., DBMS shared pool latches). In cloud systems, indicators can be measured at any layer of the virtualization technology stack, e.g., at guest, host or VMM layer. Other indicators of interest are related to VMs, e.g., the number of VM allocations/releases or migrations. Further indicators aim at capturing specific effects like *accumulation of numerical errors* and *memory fragmentation* [4].

[*]Javier Alonso's contributions to this paper are based on his research expertise before joining Amazon.

III. Aging analysis technique

A. Threshold-based

The simplest way to warn about aging in the system is to define thresholds on some key indicators, so as rejuvenation is triggered when these are exceeded. Indicators may refer to response time or memory consumption. Difficulties arise in identifying the best indicators and the right thresholds for them (especially in presence of strong correlations among multiple indicators): the choice of the indicators should be able to prevent actual failures and useless rejuvenation actions at the same time. An example is in the work by Silva *et al.* [5], which adopts thresholds on mean response time and on quality of service indicators. Authors propose a rejuvenation based on self-healing techniques that exploits virtualization to optimize recovery. They implemented a framework, called *VM-Rejuv*, in which an *Aging Detector* module revealing aging based on the mentioned thresholds.

B. Statistical time series analysis

With time series analysis, variables potentially related to software aging, such as system's resources consumption or performance indicators, are monitored periodically over time, and the resulting time series are analyzed to assess the presence of aging. This approach copes with two main problems: 1) detecting the presence of a degradation trend (*trend detection*) and 2) quantitatively estimating the extent of the degradation (*trend estimation*) and its characteristics

TABLE I
Example of aging indicators (# denotes *number of*)

Performance degradation	**Memory consumption**	**Resources consumption** *(beside memory)*
Response time	Free RAM memory	File handles, file table size
Throughput	Cache/Buffer size	Temporary files size
Latency	Virtual memory	CPU utilization
Transaction rate	Resident Set Size (RSS)	#threads/processes
#SLA violations	Swap space	#locks
	Active/Inactive	Socket descriptors
	(more/less recently used mem.)	DBMS res. *(e.g., shared pool)*
	Slab (in-kernel cache)	Power consumption
	Shared memory	Disk space
	At VM-level	VM creation/term./migration
	(e.g., Free RAM in the VM	At VM-level
	Heap/GC stats in the JVM)	*(e.g., #threads in the VM)*

(e.g., seasonality in data). Trend detection applies statistical hypothesis test to determine whether the values within a time series generally increase (or decrease) over a long period of time. Trend estimation allows quantitatively characterizing the long-term component of a time series.

The most used *trend detection* techniques in software aging research are, by far, the Mann-Kendall and Seasonal Kendall test. They are hypothesis tests (for non-periodic and periodic time series, respectively) to accept/reject the hypothesis of no trend in data. Being non-parametric tests, they do not make assumptions on the distribution of data and are more robust to outliers than parametric tests; on the other hand, they are generally conservative, namely some actual trend could be missed. Mann-Kendall (and Seasonal Kendall) test is usually applied in conjunction with the Sen's (and Seasonal Sen's) procedure for *trend estimation*. The Sen's procedure is also a non-parametric linear regression technique (also insensitive to outliers) that fits a linear model and computes the slope as the median of all slopes between paired values. More recently, the Mann-Kendall test is being questioned because of the high rate of false positives, suitability for linear trends only, sensitivity to noise. Zheng *et al.* proposed a modified version of the Cox-Stuart test for trend detection and the iterative Hodrick-Prescott Filter for (linear and non-linear) trend estimation as alternative to Mann-Kendall/Sen's procedure pair [6].

Beside conventional regression, time-series ARMA/ARX models have been extensively used for trend estimation, e.g., [7], as well as ARIMA and Holt-Winters (Triple Exponential Smoothing) model [8]. Non-linear time-series analyses are used by Araujo *et al.* [9], where four time-series models allows to schedule rejuvenation properly: the linear, quadratic, exponential growth and Pearl-Reed logistic model.

Time-series analysis has been also adopted to study the relationship of the software aging and workload in complex systems, including the Linux Kernel code [10], and the Java Virtual Machine [11]. The impact of the workload is also investigated by a Design of Experiment (DoE) method [12].

C. Machine-learning-based

If we consider software aging as a result of two factors (the system resource presenting the aging bug and time), time series approaches can be considered ideal. However, these approaches are limited by the forehand knowledge about the resource/metric that is affected by the aging phenomena, especially in complex systems with many interdependency between resources at different system layers. Machine Learning is a more sophisticated form of data analysis, which adopts regression and/or classification to identify trends (and predict the time to exhaustion (TTE)) and classify a system state as robust or failure-prone. Often time series analysis and classification are used together to identify trends and then classify the system state. Alonso *et al.* compare different regression algorithm families (i.e., regression trees, linear regression and hybrids) in different scenarios and multiple aging phenomena involved [13]. Many classifiers have been explored for aging detection, including naive Bayes, decision trees, neural network, random forest, LDA/QDA, SVM, K-NN, and ZeroR [8], [14].

Eto, Dohi and Ma [15] use reinforcement learning to estimate the optimal rejuvenation schedule, hence not requiring the complete knowledge on system failure time distribution.

IV. Conclusion

The advantage of measurements-based solutions lies in the possibility to gather accurate and detailed information about the aging state, and there is no need to make assumptions on model parameters because real data are available. However, the gathered data are hardly generalizable to other systems. This requires a tuning of the technique used based on the system to be analyzed. Combination of measurements-based approaches with model-based ones make sense in order to be able of parametrizing models with real observations and exploiting the advantage of both approaches. This is a strategy that we will likely witness more and more in the future.

References

[1] 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.*, 1998.

[2] 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.*, 1999.

[3] R. Pietrantuono, J. Alonso, and K. Vaidyanathan, *Measurements for Software Aging*. World Scientific, 2020, ch. Chapter 4, pp. 73–90.

[4] M. Grottke, R. Matias, and K. Trivedi, "The Fundamentals of Software Aging," in *IEEE International Conference on Software Reliability Engineering Workshops*, 2008.

[5] L. Silva, J. Alonso, and J. Torres, "Using Virtualization to Improve Software Rejuvenation," *IEEE Transactions on Computers*, vol. 58, no. 11, pp. 1525–1538, 2009.

[6] P. Zheng, Y. Qi, Y. Zhou, P. Chen, J. Zhan, and M. R. T. Lyu, "An automatic framework for detecting and characterizing performance degradation of software systems," *IEEE Transactions on Reliability*, vol. 63, no. 4, pp. 927–943, Dec 2014.

[7] 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.*, 2002.

[8] 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*, 2010.

[9] J. Araujo, R. Matos, P. Maciel, F. Vieira, R. Matias, and K. Trivedi, "Software Rejuvenation in Eucalyptus Cloud Computing Infrastructure: A Method Based on Time Series Forecasting and Multiple Thresholds," in *3rd International Workshop on Software Aging and Rejuvenation (WoSAR)*, 2011, pp. 38–43.

[10] 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.*, 2010.

[11] D. Cotroneo, S. Orlando, R. Pietrantuono, and S. Russo, "A measurement-based ageing analysis of the JVM," *Software Testing, Verification and Reliability*, vol. 23, no. 3, pp. 199–239, 2013.

[12] A. Bovenzi, D. Cotroneo, R. Pietrantuono, and S. Russo, "On the aging effects due to concurrency bugs: A case study on mysql," in *IEEE 23rd International Symposium on Software Reliability Engineering*, 2012, pp. 211–220.

[13] 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.*, 2010.

[14] J. Alonso, L. Belanche, and D. Avresky, "Predicting software anomalies using machine learning techniques," in *10th IEEE Intl. Symposium on Network Computing and Applications*. IEEE, 2011, pp. 163–170.

[15] H. Eto, T. Dohi, and J. Ma, "Simulation-based optimization approach for software cost model with rejuvenation," *Lecture Notes in Computer Science*, vol. 5060 LNCS, pp. 206–218, 2008.