Achieving and Assuring High Availability

K. Trivedi *Duke University* *kst@ee.duke.edu,*
G. Ciardo *UC Riverside* *ciardo@cs.ucr.edu,*
B. Dasarathy *Telcordia* *das@research.telcordia.com*
M. Grottke *Univ. Erlangen-Nbg.* *michael.grottke@wiso.uni-erlangen.de,*
A. Rindos *IBM* *rindos@us.ibm.com,*
B. Vashaw *IBM* *vashaw@us.ibm.com*

Abstract

We discuss availability aspects of large software-based systems. We classify faults into Bohrbugs, Mandelbugs and aging-related bugs, then examine mitigation methods for the last two bug types. We also consider quantitative approaches to availability assurance.

1. Overview

High availability is being demanded for military as well as commercial applications such as e-commerce systems, financial systems, stock-trading systems, national and international telecommunication infra-structure (e.g., switches and routers) and several types of life-critical and safety-critical systems. Many techniques to achieve high availability from the hardware perspective are known. However, software remains the main bottleneck in achieving high availability. Despite many advances in formal methods, programming methodology, and testing, the software development process has not reached the stage to allow for the routine production of ultra-low defect software systems [6]. Yet, complex software-based mission-critical systems are expected not to fail.

Bugs invariably remain when an application is deployed. A good, albeit expensive, development process can reduce the number of residual bugs (bugs that remain after the code has been tested and delivered) to the order of 0.1 defects per 1000 lines of code [31]. There are broadly two classes of these residual bugs in an application, known as Bohrbugs and Mandelbugs [25], [26]. Bohrbugs are, in principle, easily isolated and manifest themselves consistently under well-defined sets of conditions; thus, they can be detected and fixed during the software-testing phase, although some of them do remain in production. Preliminary results from our own investigation of a NASA software project suggest that 52% of residual bugs were Bohrbugs [24]. Mandelbugs instead have complex causes, making their behavior appear chaotic or even non-deterministic (e.g., race conditions, complex error propagations), thus are often difficult to catch and correct in the testing phase [57]. Retrying the same operation might not result in a failure manifestation. Sometimes, the literature also calls these software faults Heisenbugs [22], [34]. However, Bruce Lindsay, who invented the term deriving it from Heisenberg's Uncertainty Principle, refers it to faults that change their behavior when probed or isolated [76]. Lindsay's Heisenbugs are actually a subtype of Mandelbugs [25], [26]. Published data suggests that Mandelbugs account for between 15% and 80% of all software faults detected after release [12], [40]. An interesting subtype of Mandelbugs [25], [26] has the characteristic that its failure manifestation rate increases with the time of execution. Such faults have been observed in many software systems and have been called aging-related bugs [3], [19], [23], [44]. Memory leaks and round-off errors are examples of aging-related bugs. There appears to be no concrete data on what percentage of residual bugs are aging-related.

There are effective approaches to dealing with residual Bohrbugs after a software product has been released. If a failure due to a Bohrbug is detected in production, it can be reproduced in the original testing environment, and a patch correcting the bug or a workaround can be issued. Mandelbugs, however, often cannot be easily fixed, thus techniques to recover from Mandelbugs at run-time are needed. Fixing aging-related bugs is possible in some cases. However, broadly applicable cost- and time-effective run-time techniques exist to address aging-related bugs. We focus on Mandelbugs in general and aging-related bugs in particular.

In general, there are two ways to improve availability: increase time-to-failure (TTF) and reduce time-to-recovery (TTR). To increase TTF, proactive failure avoidance techniques known as rejuvenation can be used for aging-related bugs. To reduce TTR, we propose instead escalated levels of recovery, so that most failures are fixed by the quickest recovery method and only few by the slowest ones. We are also interested in quantifiable availability assurance.

In *Proc. 13th IEEE Workshop on Dependable Parallel, Distributed and Network-Centric Systems/22nd IEEE International Parallel & Distributed Processing Symposium*, 2008.

In Section 2, we discuss the use of analytic models for availability assurance. Section 3 describes the approaches to deal with Mandelbugs while Section 4 discusses approaches to deal with aging-related bugs. Section 5 offers some concluding remarks.

2. Quantified Availability Assurance

In practice, availability assurance is provided qualitatively by means of verbal arguments or using checklists. Quantitative assurance of availability by means of stochastic availability models constructed based on the structure of the system hardware and software is very much lacking in today's practice [62], [66], [68], [69]. While such analyses are nowadays supported by software packages [7], [58], they are not routinely carried out on what are touted as high availability products; there are only islands of such competency even in large companies.

Engineers commonly use reliability block diagrams or fault trees to formulate and solve availability models because of their simplicity and efficiency [58], [67]. But such combinatorial models cannot easily incorporate realistic system behavior such as imperfect coverage, multiple failure modes, or hot swap [62], [69]. In contrast, such dependencies and multiple failure modes can be easily captured by state-space models such as Markov chains, semi-Markov processes [67], and Markov regenerative processes [7]. However, the construction, storage, and solution of these state space models can become prohibitive for real systems. The problem of large model construction can be alleviated by using some variation of stochastic Petri nets [7], but a more practical alternative is to use a hierarchical approach using a judicious combination of state space models and combinatorial models [58]. Such hierarchical models have been successfully used on practical problems including hardware availability prediction [39], OS failures [62], [66], [68] and application software failures [18], [69]. Furthermore, user and service-oriented measures can be computed in addition to system availability. Computational methods for such user-perceived measures are just beginning to be explored [59], [69], [74].

As an example, the IBM BladeCenter is a system where the complexity of the system precludes modeling as a single-level state space model. The number of BladeCenter components subject to failure is close to 140. If each component were to be in one of two states only (actually some components have more than two states), the size of the state space of the overall Markov chain would be 2^{140}. However, as dependencies exist in the system, an overall combinatorial model will not suffice. Dependencies within subsystems are modeled in [62] using homogeneous continuous-time Markov chains. Independence across subsystems is assumed, thus a combinatorial model is used to combine the subsystem availabilities into the overall system availability. The top-level model is a fault tree because some of the component failures affect several different portions of the system at the same time. Such effects are captured by fault trees with repeated events but cannot be captured by reliability block diagrams [58], [67]. Other methods to reduce the state space size include state truncation, applicable to high-level model descriptions such as stochastic Petri nets, so that truncation error bounds could be computed, and fixed-point iterations. Besides availability assurance, such models can also be used to find availability bottlenecks [59].

Subsequently, parameter values are needed to solve the models and predict system availability and related measures. Model input parameters can be divided into failure rates of hardware or software components; detection, failover, restart, reboot and repair delays and coverages; and parameters defining the user behavior. Hardware failure rates (actually MTTFs) are generally available from vendors, but software component failure rates are much harder to obtain. Alcatel-Lucent uses residual failure intensity based on a software-reliability growth-model as the failure rate in operation [49]. An alternative is to carry out controlled experiments and estimate software component failure rates. In fact, we are currently performing such experiments for the WebSphere Application Server and the SIP/Proxy at IBM. Fault injection experiments can be used to estimate detection, restart, reboot, and repair delays [33], as in the IBM SIP/SLEE modeling exercise [69]. Statistical inference methods for the estimations are well known [1], [40], [47], [53], [64]. However, passing the confidence intervals of input parameters through an analytical availability model is relatively unexplored [78].

Due to many simplifying assumptions made about the system, its components, and their interactions and due to unavailability of accurate parameter values, the results of the abstract models cannot be taken as a true availability assurance. Monitoring and statistically inferring the observed availability is surely much more satisfactory assurance of availability. Off-line [21] and on-line [28], [50] monitoring of deployed system availability and related metrics can be carried out. The major difficulty is the amount of time needed to get enough data to obtain statistically significant estimates of availability.

3. Recovery from failures caused by Mandelbugs

Reactive recovery from failures caused by Mandelbugs has been used for some time in the context of operating system failures, where reboot is the mitigation method [35], [68]. Restart, failover to a replica, and further escalated levels of recovery such as node reboot and repair are being successfully employed for application failures. Avaya's NT-SwiFT and DOORS systems [20], JPL REE system [13], Alcatel Lucent [48], [49], [72], IBM x-series models [70], CORBA [52], [55], [56], and IBM SIP/SLEE cluster [62], [68], [69] are examples where applications or middleware processes are recovered using one or more of these techniques. To support recovery from Mandelbug-caused failures, multiple run-time failure detectors are employed to ensure that detection takes place within a short duration of the failure occurrence. In all but the rarest cases, manual detection is required. As, by definition, failures caused by non-aging-related Mandelbugs cannot be anticipated and must be reacted to, current research is aimed at providing design guidelines as to how fast recovery can be accomplished and obtaining quantitative assurance on the availability of an application.

Stochastic models discussed in the previous section are beginning to be used to provide quantitative availability assurance [13], [20], [69], [70]. Besides system availability, models to compute user-perceived measures such as dropped calls in a switch due to failures are beginning to be used [36], [69]. Such models can capture the details of user behavior [74] or the details of the call flow [70] and its interactions with failure and recovery behavior of hardware and software resources. Difficulties we encounter in availability modeling are model size and obtaining credible input parameters [7], [54], [62], [69]. To deal with the large size of availability models for real systems, we typically employ a hierarchical approach where the top-level model is combinatorial, such as a fault tree [39], [62], [69] or a reliability block diagram [66], [68]. Lower-level stochastic models for each subsystem in the fault tree model are then built. These submodels are usually continuous-time Markov chains but if necessary non-Markov models [73] can be employed. Weak interactions between submodels can be dealt with using fixed-point iteration [43], [65]. The key advantage of such hierarchical approach is that closed-form solution now appears feasible [59], [62] as the Markov submodels are typically small enough to be solved by Mathematica and the fault tree can be solved in closed-form using tools like our own SHARPE software package [58]. Once the closed-form solution is obtained, we can also carry out sensitivity analysis to determine bottlenecks and provide feedback for improvement to the designers [59]. We are currently working on interfacing SHARPE with Mathematica to facilitate such closed-form solutions. Errors in these approximate hierarchical models can be studied by comparison with discrete-event simulation and exact stochastic Petri net models solved numerically.

A standby copy to failover to can be either an active or a passive replica. For example, the IBM SIP/SLEE system uses active replication, while Avaya's SwiFT system uses passive replication. In active replication, both copies serve different requests at the same time and constant synchronization of data might be required if the data is not partitioned across replica. In passive replication, only one replica, the primary, executes at any one time while one or more backups are waiting to take over when the primary fails [17]. Passive replication can be further divided into two categories: warm and cold [20]. Warm replicas are periodically updated with state information while cold replicas are not. The chosen way to organize the replicas has both performance and availability impact. Performance penalty will be larger as we move from cold to warm to active replication while the availability will likely improve. Detailed availability and performance models can be developed for the three schemes as in [20].

Recovery should be tailored to different kinds of failures and only touch the affected system components. However, a recovery technique may not always successfully recover an application from the current failure, i.e., the conditional probability that a specific recovery technique will bring the system up again, given that it is attempted after a failure. is usually less than one. Since it is not known in advance which recovery technique should be used after a failure occurrence, a sequence of recovery procedures consisting of specific escalated levels or stages of recovery should be employed. Typically, the techniques are ordered according to the expected length of time needed for their execution: the fastest technique is tried first, while the last recovery stage may be slow but guarantees recovery. An example of such a sequence is: micro-rebooting of an individual software component in an application, restart of the application, fail-over, reboot of the entire node, and full system repair. A fundamental question is then whether the sequence is optimal from the perspective of availability. For example, should the fail-over precede application restart, or follow the reboot of the entire node? We can answer such questions using probabilistic availability models. Since the number of possible recovery actions is small, preliminary research suggests that an exhaustive search is adequate to determine the optimal sequence [27].

4. Proactive recovery and aging-related bugs

Aging-related bugs in a system are such that their probability of causing a failure increases with the length of time the system is up and running. For such bugs, besides reactive recovery, proactive recovery to clean the system internal state can effectively reduce the failure rate. This kind of preventive maintenance is known as "software rejuvenation" [5], [26], [34]. Many types of software systems, such as telecommunication software [3], [5], [34], network devices [14], web servers [23], [46], and military systems [44], are known to experience aging. Rejuvenation has been implemented in several kinds of software systems, including telecommunication billing data collection systems [34], transaction processing systems [10], spacecraft flight systems [63], distributed CORBA-based applications [55], and cluster servers [11].

The main advantage of planned preemptive procedures such as rejuvenation is that the consequences of sudden failures (like loss of data and unavailability of the entire system) are postponed or prevented; moreover, administrative measures can be scheduled to take place when the workload is low. However, for each such preemptive action, costs are incurred in the form of scheduled downtime for at least some part of the system. Rejuvenation can be carried out at different granularities: restart a software module, restart an entire application, perform garbage collection in a node, or reboot a hardware node [9], [32], [46], [77]. A key design question is finding the optimal rejuvenation schedule and granularity.

Rejuvenation scheduling can be time-based or condition-based. In the former, rejuvenation is done at fixed time intervals [11], [15], [16], [18], [32], [34], [42], [70], while, in the latter, the condition of system resources is monitored and prediction algorithms are used to determine an adaptive rejuvenation schedule [2], [19], [61], [71], [77].

For time-based rejuvenation, the stochastic models discussed in Section 2 are enhanced to incorporate aging-related bugs and the rejuvenation triggers at various levels of granularity. The resulting models are no longer Markov. We have solved such models using a combination of phase-type expansions and deterministic and stochastic Petri nets (DSPN) type techniques [41], [75]. To solve and optimize these models, besides the input parameters discussed in Section 2, we need parameters for various proactive recovery actions and the time to failure distribution due to aging-related failures. Before the system is deployed, a distribution and its parameters will have to be based on past experience. During the operation of the system, time to failure data can be collected and used to parameterize the optimization model. It is possible to directly use the measured time-to-failure data in the optimization of the rejuvenation schedule via the notion of total time on test (TTT) transform to avoid the error-prone process of fitting of this data to a distribution [4], [15]. The scheme is then a closed-loop feedback control system [29] where the "fixed-time" is adaptive in response to monitored time-to-failure data.

A rejuvenation trigger interval, as computed in time-based rejuvenation, adapts to changing system conditions, but its adaptation rate is slow as it only responds to failure occurrences that are expected to be rare.

Condition-based rejuvenation instead does not need time to failure inputs; it computes rejuvenation trigger interval by monitoring system resources and predicting the time to exhaustion of resources for the adaptive scheduling of software rejuvenation [3], [11], [19], [55], [71]. Garg et al. [19] measured variables such as free main memory, used swap space, and file table size in a network of UNIX workstations. These measured variables showed a statistically significant (decreasing or increasing) trend over time. Using a non-parametric technique, Garg et al. determine the global aging trend and calculate the estimated time until complete exhaustion via linear extrapolation for each resource. In case some form of rejuvenation or periodicity is already implemented by the system, as in the Apache Web server [23], piecewise linear [11], autoregressive time series with deterministic seasonal component [23], nonlinear statistical methods [30], and fractal-based methods [60] have also been used on such data. Regardless of the prediction method used, the resources selected for monitoring must be determined to minimize the monitoring overhead. Design-of-experiment (DOE) and analysis-of-variance (ANOVA) have been used to answer this question [45], [51]. All the published methods predict the times to exhaustion of individual resources, but time to system failure is a complex combination of these times. Predicting time to failure is an open question.

Whatever schedule and granularity of rejuvenation is used, the important question is what improvement this implies on system availability, if any. Published results are based on either analytic models [70] or simulations [15]. Early results of a measurement experiment at Tokyo Institute of Technology are very encouraging, where rejuvenation increased the MTTF by a factor of two [38] on the system reported in [37].

5. Conclusions

We have discussed models for quantifying availability of a software system. We have considered reactive recovery techniques for Mandelbugs and availability models that incorporate these recovery techniques. For aging-related bugs, a powerful proactive recovery technique is rejuvenation. We discussed rejuvenation scheduling and availability models for software systems when rejuvenation is used to deal with aging. We strongly emphasize obtaining model parameters from measurements as a key ingredient in our analytical solution framework.

References

[1] J. Arlat, A. Costes, Y. Crouzet, J.-C. Laprie, and D. Powell. Fault injection and dependability evaluation of fault tolerant systems. *IEEE Trans. Computers*, 42(8):913–923, 1993.

[2] A. Avritzer, A. Bondi, M. Grottke, K. Trivedi and E. J. Weyuker. Performance assurance via software rejuvenation: monitoring, statistics and algorithms. In *Proc. International Conference on Dependable Systems and Networks 2006*, pp. 435-444, 2006.

[3] A. Avritzer and E. J. Weyuker. Monitoring smoothly degrading systems for increased dependability. *Empirical Software Engineering*, vol. 2, no. 1, pp. 59–77, 1997.

[4] R. E. Barlow, and R. Campo. Total time on test processes and applications to failure data analysis. In R. E. Barlow, J. Fussell and N. D. Singpurwalla, editors, *Reliability and Fault Tree Analysis*, pp. 451-481, SIAM, Philadelphia, PA, 1975.

[5] L. Bernstein and C. M. R. Kintala. Software rejuvenation. *CrossTalk*, vol. 17, no. 8, pp. 23–26, 2004.

[6] R. Bharadwaj. Whither verified software? In *Proc. IFIP Working Conference on Verified Software: Theories, Tools, Experiments (VSTTE)*, Bertrand Meyer (ed.), LNCS 4171, Zurich Switzerland, October 2005.

[7] G. Bolch, S. Greiner, H. de Meer, K. S. Trivedi. *Queueing Networks and Markov Chains Modeling and Performance Evaluation with Computer Science Applications, Second Edition*. John Wiley and Sons, New York, NY, 2006.

[8] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Comp*., 35(8):677–691, Aug. 1986.

[9] G. Candea, J. Cutler, and A. Fox. Improving availability with recursive microreboots: a soft-state system case study. *Performance Evaluation*, vol. 56, no. 1–4, pp. 213–248, 2004.

[10] K. J. Cassidy, K. C. Gross, and A. Malekpour. Advanced pattern recognition for detection of complex software aging in online transaction processing servers. In *Proc. International Conference on Dependable Systems and Networks*, 2002, pp. 478–482.

[11] V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter, K. S. Trivedi, K. Vaidyanathan, and W. P. Zeggert. Proactive management of software aging. *IBM Journal of Research and Development*, vol. 45, no. 2, pp. 311–332, 2001.

[12] S. Chandra and P. M. Chen. Whither generic recovery from application faults? A fault study using open-source software. In *Proc. Int'l Conf. Dependable Systems and Networks (DSN)*, , 2000, pp. 97-106.

[13] D. Chen et al. Reliability and availability analysis for the JPL remote exploration and experimentation system. In *Proc. Int'l Conf. Dependable Systems and Networks (DSN)*, Washington, DC, June 2002.

[14] Cisco Systems. Cisco catalyst memory leak vulnerability. ID:13618, *Cisco Security Advisory*, 2001.

[15] T. Dohi, K. Goseva-Popstojanova, and K. S. Trivedi. Statistical Non-Parametric Algorithms to Estimate the Optimal Software Rejuvenation Schedule. In *Proc. of the 2000 Pacific Rim Intl. Symp. on Dependable Computing (PRDC)*, Los Angeles, December 2000.

[16] T. Dohi, K. Goseva-Popstojanova, and K. S. Trivedi. Estimating software rejuvenation schedule in high assurance systems. *The Computer Journal*, vol. 44, no. 6, pp. 473–485, 2001.

[17] T. Dumitras, D. Srivastava, and P. Narasimhan. Architecting and Implementing Versatile Dependability. *Architecting Dependable Systems Vol. III*, C. Gacek, A. Romanovsky and R. de Lemos, editors. Springer-Verlag, 2005.

[18] S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi. Analysis of software rejuvenation using Markov regenerative stochastic Petri net. In *Proc. Sixth International Symposium on Software Reliability Engineering*, 1995, pp. 24–27.

[19] S. Garg, A. van Moorsel, K. Vaidyanathan, and K. S. Trivedi. A methodology for detection and estimation of software aging. In *Proc. Ninth International Symposium on Software Reliability Engineering*, 1998, pp. 283–292.

[20] S. Garg, Y. Huang, C. M. R. Kintala, K. S. Trivedi, and S. Yajnik. Performance and reliability evaluation of passive replication schemes in application level fault

tolerance. In *Proc. 29th Annual International Symposium on Fault Tolerant Computing (FTCS)*, pp. 15–18, Madison, Wisconsin, June 1999.

[21] M. Garzia. Assessing the Reliability of Windows Servers. *Proc. DSN* 2003.

[22] J. Gray. Why do computers stop and what can be done about it? In *Proc. 5th Symp. Reliability in Distributed Systems*, 1986, pp. 3–12.

[23] M. Grottke, L. Li, K. Vaidyanathan, and K. S. Trivedi. Analysis of software aging in a web server. *IEEE Transactions on Reliability*, vol. 55, no. 3, Sept. 2006, pp. 411–420.

[24] M. Grottke, A. Nikora, and K. S. Trivedi. Preliminary results from the NASA/JPL investigation, Classifying Software Faults to Improve Fault Detection Effectiveness. Dec. 2007.

[25] M. Grottke and K. S. Trivedi. Software faults, software aging and software rejuvenation. *Journal of the Reliability Engineering Association of Japan*, vol. 27, no. 7, pp. 425-438, Oct. 2005.

[26] M. Grottke and K. S. Trivedi. Fighting bugs: remove, retry, replicate and rejuvenate. *IEEE Computer*, vol. 40, no. 2, pp. 107–109, Feb. 2007.

[27] M. Grottke and K. S. Trivedi. Analysis of the escalated levels of failure recovery approach. Working paper, University of Erlangen-Nuremberg, 2008.

[28] M. Haberkorn and K. Trivedi. Availability monitor for a software based system. In *Proc. HASE 2007*, Dallas, TX.

[29] J. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computer Systems*, John Wiley & Sons, 2004.

[30] G. Hoffman, M. Malek, and K. S. Trivedi. A best practice guide to resource forecasting for the Apache webserver. In *Proc. Pacific Rim Dependability Conference (PRDC), Riverside*, CA, December 2006.

[31] G. J. Holzmann. Conquering complexity. *IEEE Computer*, Dec 2007.

[32] Y. Hong, D. Chen, L. Li and K. Trivedi. Closed loop design for software rejuvenation. In *Proc. Workshop on Self-Healing, Adaptive and Self-Managed Systems*. New York, NY. 2002.

[33] M-C. Hsueh, T. K. Tsai, and R. K. Iyer. Fault Injection Techniques and Tools. *IEEE Computer*, April 1997.

[34] Y. Huang, C. Kintala, N. Kolettis, and N. Fulton. Software rejuvenation: analysis, module and applications. In *Proc. Twenty-Fifth International Symposium on Fault-Tolerant Computing*, 1995, pp. 381–390.

[35] S. W. Hunter and W. E. Smith. Availability modeling and analysis of a two node cluster. In *Proc. 5th Int. Conf. on Information Systems, Analysis and Synthesis*, Orlando, FL, Oct. 1999.

[36] M. Kaaniche, K. Kanoun, and M. Martinello. A user-perceived availability evaluation of a web based travel agency. In *Proc. Intl. Conf. on Dependable Systems and Networks* (DSN), 2003.

[37] K. Kourai and S. Chiba. A fast rejuvenation technique for server consolidation with virtual machines. In *Proc. International Conference on Dependable Systems and Networks 2007*, pp. 245-255.

[38] K. Kourai. Personal communication. January 9, 2008.

[39] M. Lanus, Liang Yin, and K. Trivedi. Hierarchical composition and aggregation of state-based availability and performability models. *IEEE Transactions on Reliability*, Mar. 2003.

[40] I. Lee and R.K. Iyer. Software Dependability in the Tandem GUARDIAN System. *IEEE Trans. Software Engineering*, May 1995, pp. 455–467.

[41] C. Lindemann. *Performance Modelling with Deterministic and Stochastic Petri Nets*. John Wiley and Sons, 1998.

[42] Y. Liu, Y. Ma, J. Han, H. Levendel, and K. S. Trivedi. A proactive approach towards always-on availability in broadband cable networks. *Computer Communications*, 28(1):51-64, Jan 2005.

[43] V. Mainkar and K. S. Trivedi. Sufficient conditions for existence of a fixed point in stochastic reward net-based iterative methods. *IEEE Transactions on Software Engineering*, vol. 22, no. 9, pp. 640–653, 1996.

[44] E. Marshall. Fatal error: how Patriot overlooked a Scud. *Science*, Vol. 255, p. 1347, 1992.

[45] R. Matias Jr. and P. J. Freitas Filho. Software aging characterization based on a DOE approach. In *Proc 1st Experimental Software Engineering Latin American Workshop*, Brazil, 2004.

[46] R. Matias Jr. and P. J. Freitas Filho. An experimental study on software aging and rejuvenation in web servers. In *Proc. 30th IEEE Annual International Computer Software and Applications Conference*, vol. 1, pp. 189-196, 2006.

[47] W. Q. Meeker and L. A. Escobar. *Statistical Methods for Reliability Data*. John Wiley & Sons, New York, 1998.

[48] V. B. Mendiratta. Reliability analysis of clustered computing systems. In *Proc. Ninth International Symposium on Software Reliability Engineering*, pp. 268–272, 1999.

[49] V. B. Mendiratta, J. M. Souza, and G. Zimmerman. Using software failure data for availability evaluation. *Designer and Developer Forum, GLOBECOM 2007*, November 27, 2007, Washington, D.C.

[50] K. Mishra and K. S. Trivedi. Model based approach for autonomic availability management. In *Proc. Int. Symposium on Service Availability, ISAS*, Helsinki, Finland, May 2006.

[51] D. C. Montgomery. *Design and Analysis of Experiments*. 6th edition, John Wiley & Sons, 2004.

[52] P. Narasimhan, T. Dumitras¸, S. Pertet, C. F. Reverte, J. Slember, and D. Srivastava. MEAD: support for real-time fault tolerant CORBA. *Concurrency and Computation: Practice and Experience*, vol. 17, no. 12, pp. 1527-1545, 2005.

[53] W. Nelson, *Applied Life Data Analysis*. John Wiley and Sons, New York, 1982.

[54] D. Nicol, W. Sanders, and K. S. Trivedi. Model-based evaluation: from dependability to security. *IEEE Transactions on Dependable and Secure Computing*, Vol. 1, 2004.

[55] S. Pertet and P. Narasimhan. Proactive recovery in distributed CORBA applications. In *Proc. DSN*, pp. 357-366, 2004.

[56] S. Pertet and P. Narasimhan. Causes of failure in web applications. *Carnegie Mellon University Parallel Data Lab Technical Report*, CMU-PDL-05-109, December 2005.

[57] E. S. Raymond, *The New Hacker's Dictionary*, MIT Press, 1991.

[58] R. A. Sahner, K. S. Trivedi, and A. Puliafito. *Performance and Reliability Analysis of Computer Systems*. Kluwer Academic Press, 1996.

[59] N Sato, H. Nakamura and K. S. Trivedi. Detecting performance and reliability bottlenecks of composite web services. In *Proc. ICSOC*, 2007.

[60] M. Shereshevsky, J. Crowell, B. Cukic, V. Gandikota, and Y. Liu. Software aging and multifractality of memory resources. In *Proc. Int'l Conf. Dependable Systems and Networks (DSN)*, 2003, pp. 721–730.

[61] L. Silva, H. Madeira, and J. G. Silva. Software aging and rejuvenation in a SOAP-based server. *Proc of Fifth IEEE International Symposium on Network Computing and Applications (NCA'06)*, pp. 56-65, Washington, DC, 2006.

[62] W. Earl Smith, K. S. Trivedi, L. Tomek, and J. Ackeret. Availability analysis of multi-component blade server systems. *IBM Systems Journal*, to appear, 2008.

[63] A. Tai, S. Chau, L. Alkalaj, and H. Hect. On-board preventive maintenance: a design-oriented analytic study for long-life applications. *Performance Evaluation*, vol. 35, nos. 3-4, pp. 215-232, May 1999.

[64] P. Tobias and D. Trindade. *Applied Reliability*, 2nd edition. Kluwer Academic Publishers, Boston, 1995.

[65] L. Tomek and K. S. Trivedi. Fixed-point iteration in availability modeling. *Informatik-Fachberichte, Vol. 283; Fehlertolerierende Rechensysteme*, M.Dal Cin, editor, pp 229-240, Springer-Verlag, Berlin, 1991.

[66] K. S. Trivedi. Availability analysis of Cisco GSR 12000 and Juniper M20/M40. Cisco Technical Report, 2000.

[67] K. S. Trivedi. *Probability & Statistics with Reliability, Queueing and Computer Science Applications*, Second Edition, John Wiley, New York, 2001.

[68] K. S. Trivedi, R. Vasireddy, D. Trindade, S. Nathan, and R. Castro. Modeling high availability systems. In *Proc. Pacific Rim Dependability Conference*, 2006.

[69] K. S. Trivedi, D. Wang, J. Hunt, A. Rindos, M. Peyravian, and B. Pulito. IBM SIP/SLEE cluster reliability model. Internal document, IBM RTP, 2007, also *Globecom 2007*, D&D Forum, Washington DC.

[70] K. Vaidyanathan, R. E. Harper, S.W. Hunter, and K. S. Trivedi. Analysis and implementation of software rejuvenation in cluster systems. In *Proc. ACM SIGMETRICS*, 2001.

[71] K. Vaidyanathan and K. S. Trivedi. A comprehensive model for software rejuvenation. *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, pp. 124–137, 2005.

[72] S. A. Vilkomir, D. L. Parnas, V. B. Mendiratta, and E. Murphy. Availability evaluation of hardware/software systems with several recovery procedures. In *Proc. Twenty-Ninth Annual International Computer Software and Applications Conference*, pp. 473–478, 2005.

[73] D. Wang, R. Fricks, and K. S. Trivedi. Dealing with non-exponential distributions in dependability models. *Performance Evaluation - Stories and Perspectives*, G. Kotsis, editor, Österreichische Computer Gesellschaft, pp. 273-302, 2003.

[74] D. Wang and K. S. Trivedi. Modeling user-perceived service availability. In *Proc. of the 2nd International Service Availability Symposium* (ISAS), Berlin, April 2005.

[75] D. Wang, W. Xie, and K. S. Trivedi. Performability analysis of clustered systems with rejuvenation under varying workload. *Performance Evaluation*, vol. 64, no. 3, pp. 247-265, 2007.

[76] M. Winslett. Bruce Lindsay speaks out. *ACM SIGMOD Record*, June 2005, pp. 71-79.

[77] W. Xie, Y. Hong and K. S. Trivedi. Analysis of a two-level software rejuvenation policy. *Reliability Engineering and System Safety.* vol. 87, no. 1, pp. 13-22, Jan. 2005.

[78] L. Yin, M. Smith, and K. S. Trivedi. Uncertainty analysis in reliability modeling. In *Proc. Annual Reliability, Availability and Maintainability Symposium (RAMS)*, Philadelphia, PA, 2001.