Maximize System Reliability for Long Lasting and Continuous Applications

Chunhui Guo[1], Hao Wu[1], Xiayu Hua[1], Shangping Ren[1]*, and Jerzy M. Nogiec[2]

[1] Illinois Institute of Technology, Chicago, IL 60616, USA
[2] Fermi National Accelerator Laboratory, Batavia, IL 60510, USA
{cguo13, hwu28, xhua}@hawk.iit.edu, ren@iit.edu, nogiec@fnal.gov

Abstract. In this paper, we use software rejuvenation as a preventive and proactive fault-tolerance technique to maximize the level of reliability for continuous and safety critical systems. We take both transient faults caused by software aging effects and network transmission faults into consideration and mathematically analyze the optimal software rejuvenation period that maximizes system's reliability. The theoretical result is verified through empirical studies.

Keywords: Software Rejuvenation, Reliability Maximization, Long Lasting System

1 Introduction

Reliability is a critical criteria for many computer applications, specially for systems that directly interact with physical environment. For long lasting and continuous applications, such as factory control systems and deep space exploration vehicles, software aging caused system performance degradations, such as increased resource usage or prolonged execution time, can result in catastrophe consequences [1, 2]. Maintaining long lasting and continuous system's reliability has been both a research and an engineering challenge for many years [3–5].

Aging happens both at hardware and software levels, and both hardware and software aging can affect system reliability. Hardware aging not only increases the system's transient failure rate but also slows down the system performance. However, hardware aging often takes much longer time to show effects on a system [6]; while on the other hand, software aging happens more frequently as compared to hardware aging. As pointed in [7], nowadays, computer system outages are caused more by software failures than by hardware failures.

It is an easily observable trend that software systems become larger and more complex over time. Applications are built on top of operating systems and frameworks; they run in virtual environments and use third party software components and services. The situation makes it more difficult or virtually impossible to develop a non-trivial software to be completely error-free. A class of residual

* The research is supported in part by NSF under grant number CAREER 0746643, CNS 1018731, and CNS 1035894.

software errors produces non-catastrophic results where applications continue to provide their functionality but with a degraded performance or increased use of resources. This process is typically referred to as software aging [8]. Those software errors that cause software aging are often difficult and costly to find and verify. Even well established softwares may have aging effects caused by such errors.

To provide evidences for such slowdown phenomena, we have conducted a simple experiment which opens and closes the Matlab R2012b [9] and records the Matlab startup time. The experiment runs on a virtual machine which is pinned to an Xeon E5-2400 core with 1.9Ghz frequency, and has 2048M RAM, and 40G HDD. The operating system of the virtual machine is Window 7 and the test program is the only application running on the virtual machine.

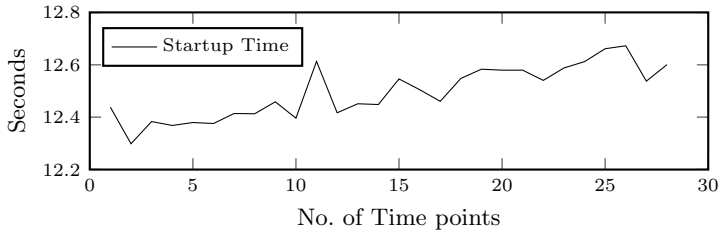


Fig. 1. Aging Effect on Matlab Startup Time

Fig. 1 shows the measurements of the Mathlab startup times over a week. Each time points represents the average Matlab startup time over 6-hour time interval. As shown in the figure, opening the Matlab takes 10% more time than when the system starts a week ago.

For a short time period after system starts running, some software aging effects only degrade the system response time rather than lead to failure. However, for long lasting and continuous control systems, if software aging issue is not dealt with, performance degradation, resource utilization, and error accumulation can lead to catastrophe consequences. For instance, the Mars Surveyor '98 Orbiter that launched in 1998 is designed for long term mission on studying the climate on Mars. However, due to a small software error in unit conversion component, the accumulated conversion error caused the Orbiter to lose connections nine months after it was launched [2]. Another tragic instance is the Ariane 5 rocket explosion occurred in 1996. The rocket exploded 37 seconds after it was launched due to digit conversion error [1]. Hence, software aging has to be probably handled for systems that have a stringent reliability requirement.

Fault-tolerance is a widely studied topic for ensuring system reliability. Commonly used fault-tolerance mechanisms include time redundancy, such as checkpointing and re-execution [10, 11], and space redundancy, such as replication and voting [12–14]. However, all these mechanisms tend to improve the system relia-

bility in a passive way, i.e., they handle faults after their occurrences. Other than the passive fault-tolerance methods, another group of mechanisms are proposed to proactively improve the system reliability, such as fault prediction [15] and software rejuvenation [8, 16, 13].

In this paper, we present an approach that uses software rejuvenation to maximize long lasting and continuous (24×7) system's reliability. Different from existing work in the literature, we take both transient faults caused by software aging and network transmission faults when migrating tasks between main and backup processors into consideration, and decide an optimal software rejuvenation period that maximizes system reliability.

The rest of the paper is organized as follows: we first discuss related work in Section 2. System models and assumptions the paper is based upon, and the formal definition of the problem the paper is to address are presented in Section 3. Section 4 mathematically analyzes system reliability under the models and assumptions defined in Section 3. We experimentally verify the theoretical analysis and discuss the empirical results in Section 5. Conclusions and future work are pointed out in Section 6.

2 Related Work

System reliability issues have been studied since pretty much the time when computers are used in safety critical systems. Many fault-tolerance mechanisms have been developed to improve system's reliability. Most commonly used fault-tolerance mechanism is redundancy [12, 13]. Typically, redundancy refers to systems that use backup components with the same functionality as the running components. When failures occur, systems switch the functionality to their backup components to maintain operation continuity. Replication is also a widely used fault-tolerance mechanism [14]. Replication ensures computation and data are duplicated on the replicas and a voting scheme is used to decide the correct answers of the system. Another widely adapted fault-tolerance technique to deal with system failure is time redundancy, ie., checkpointing and re-execution [10, 11]. With checkpointing, the failed system is recovered from previously stored correct state and re-executed only from the checkpointed state. All the fault-tolerance techniques mentioned above are passive mechanisms in the sense that they deal with failures when the failures occur.

Software rejuvenation is a preventive and proactive maintenance solution for handling system aging effects. It can be utilized in many applications, such as telecommunication systems [8, 13] and long-life deep-space missions [17, 3, 4]. Software rejuvenation is first proposed by Huang et al. [8]. In [8], Huang et. al. developed a four-state model in which a computer system operates, i.e., the *Robust* State, *Failure Probable* State, *Failure* State, and *Rejuvenation* State.

Since then, many rejuvenation models have been developed by the research committee [8, 13]. For instance, the five-state model [13] added the *Preparing* State to represent when systems finish executing tasks or migrate the tasks to another processor when the system has at least one redundancy. Koutras et al.

extended the initial rejuvenation model [8] by considering two levels of rejuvenation actions [12, 18], i.e., perfect rejuvenation action and minimal rejuvenation action. The perfect rejuvenation (cold rejuvenation) results in system returning to the Robust State (initial state), while the minimal rejuvenation (warm rejuvenation) results in system returning to the Failure Probable State (the state before rejuvenation). The cost of minimal rejuvenation is much less than the perfect rejuvenation.

To analyze software aging and study aging related failures, Trivedi et al. [19] presented two approaches: analytical modeling approach for determining optimal times to rejuvenate and measurement based approach for detection and validation. Tai et al. [3] identified key factors that may impact system reliability and developed an approach to maximizing system reliability by analyzing the optimal interval between maintenances. Okamura et al. [20] discussed an maintenance policy that combines aperiodic rejuvenations and periodic checkpoints to maximize the system availability. The estimators of reliability and availability were analyzed in [18, 21].

In this paper, we study how we can use both software rejuvenation and backup mechanism to improve system's reliability. In the study, both transient failures caused by aging effects and network transmission failures caused by migrating applications between main and back processors are taken into consideration in determining an optimal rejuvenation period that maximizes system reliability for long lasting and continuous applications.

3 System Models and Problem Formulation

In this section, we first introduce the models and assumptions our work is based upon and then formulate the problem we are to address in the paper.

3.1 Models and Assumptions

System State Transition Model

We adopt the same model and assumptions used in [8], i.e., we assume the system has four states, and the state transition model is shown in Fig. 2.

- Robust State S_0: the system starts in this state.
- Failure Probable State S_P: the system goes into this state after running for some time.
- Failure State S_F: the system may go into the failure state from the *failure probable state* S_P. Once the system is in *failure state* it has to be reboot in order to go back into the *robust state* S_0.
- Rejuvenation State S_R: from the *failure probable state* S_P the system may also go into *rejuvenation state* S_R, the system performs software rejuvenation once it enters into the *rejuvenation state* and goes back to the *robust state* S_0.

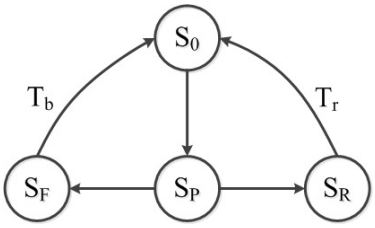


Fig. 2. System State Transition Model with Rejuvenation

The system is unavailable when it goes through either reboot or rejuvenation process. The system downtime caused by each reboot or rejuvenation is assumed to be a constant T_b and T_r, respectively. We assume T_r is much smaller than T_b.

System Model

We adopt the similar system model as in [3]. The system contains two processors, a main processor $\mathcal{P}_1$ and a backup processor $\mathcal{P}_2$. Both processors can execute application tasks, but we assume that only one processor works on the application tasks at any given time, while the other processor either being idle or performing system maintenance. The system model is shown in Fig. 3.

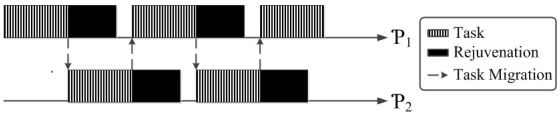


Fig. 3. System Model

To avoid failure caused by software aging effects, we assume that both processors perform rejuvenation periodically with a period t_s. Rejuvenation has overhead T_r, however, we assume $T_r < t_s$. Before one processor starts a rejuvenation process, it must first migrate all its tasks to the other processor. For planned rejuvenations, the start time of a rejuvenation process is known at a priori, hence we can reasonably assume that the overhead of task migration between $\mathcal{P}_1$ and $\mathcal{P}_2$ is being integrated into tasks' execution time.

Network Failure Model

We assume the network transmission failure model follows Poison distribution, i.e., it has a constant failure rate λ_0. The task migration between $\mathcal{P}_1$ and $\mathcal{P}_2$ may fail because of network transmission failures. With constant network transmission failure rate, the probability of a successful task migration is hence a constant p and does not change over time.

Aging Model

Since transient faults are more frequent than permanent faults [22], we only consider the transient faults for both processors. As the system deteriorates with aging, we assume that the transient failure rate $\lambda(t)$ increases with time t. The CDF (Cumulative Distribution Function) of transient fault is modeled as $F(t) = 1 - e^{-\int_0^t \lambda(x)dx}$ [23].

After each rejuvenation, the system is as good as new, i.e., the failure rate and the cumulative distribution function after rejuvenation are reset to $\lambda(t_f) = \lambda(0) = 0$ and $F(t_f) = F(0) = 0$, where t_f is the time when a rejuvenation process is completed.

Fig. 4 illustrates the behaviors of system rejuvenation and failure rate.

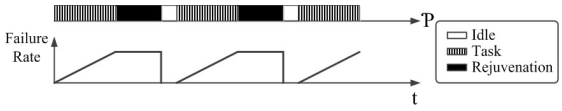


Fig. 4. System Rejuvenation and Failure Rate

3.2 Problem Formulation

Based on the models and assumptions defined in Section 3.1, the system reliability decreases over time because of the increased failure rate caused by software aging. To maintain system reliability level, on one hand, the system should perform rejuvenation as frequently as possible, but on the other hand, every rejuvenation requires tasks being migrated to and back from the other processor. Due to unreliable network, frequent migration between processors can negatively affect the system reliability. Hence, there is a balanced point as to how frequently the system shall perform rejuvenation so that the system reliability can be maximized.

Problem 1. Given two processors $\mathcal{P}_1$ and $\mathcal{P}_2$ which are connected through a network. Assume the transient failure rate of both processors is $\lambda(t)$, the network transmission failure rate is λ_0, and the system is to operate for L time, determine an optimal rejuvenation period t_s that maximizes the system reliability $R(L, t_s)$ within its operation interval $[0, L]$.

4 System Reliability Maximization

4.1 Reliability

System reliability is defined as the probability that the system operates without failure within a given time interval [23]. Assume the time interval the system operates is $[0, L]$, and the system performs $(\lceil L/t_s \rceil - 1)$ times rejuvenation, then

tasks migrate $2(\lceil L/t_s \rceil - 1)$ times. The system reliability within its longevity interval $[0, L]$ is

$$R(L, t_s) = p^{2(\lceil \frac{L}{t_s} \rceil - 1)} \cdot \overline{F}(t_s)^{\lceil \frac{L}{t_s} \rceil - 1} \cdot \overline{F}(t') \tag{1}$$

where $t' = L - t_s \cdot (\lceil L/t_s \rceil - 1)$ and $\overline{F}(t_s) = 1 - F(t_s) = e^{-\int_0^{t_s} \lambda(t)dt}$.

The following lemma gives the worst case system reliability.

Lemma 1. *Let system longevity be L and rejuvenation period be t_s, if $L \bmod t_s = 0$, then the system has the lowest reliability given by Eq.* (2)

$$R(L, t_s) = p^{2(\frac{L}{t_s} - 1)} \cdot \overline{F}(t_s)^{\frac{L}{t_s}} \tag{2}$$

□

Proof. In Eq. (1), the reliability has three factors that are all positive. As L and t_s are given, the first two factors in Eq. (1) are fixed. Hencc, the reliability is minimal when $\overline{F}(t')$ is minimal.

As $\overline{F}(t)$ decreases with t, $\overline{F}(t')$ is minimal when $t' = t_s$, i.e., $L \bmod t_s = 0$. The reliability is minimal when $L \bmod t_s = 0$, and the minimal reliability is

$$R(L, t_s) = p^{2(\lceil \frac{L}{t_s} \rceil - 1)} \cdot \overline{F}(t_s)^{\lceil \frac{L}{t_s} \rceil - 1} \cdot \overline{F}(t_s) = p^{2(\frac{L}{t_s} - 1)} \cdot \overline{F}(t_s)^{\frac{L}{t_s}} \tag{3}$$

■

For the following analysis, we focus on the worst case reliability, i.e., Eq. (2).

4.2 Reliability Maximization

Based on Eq. (2), system reliability is a function of two variables, i.e., L and t_s. To identify the relationship between reliability and rejuvenation period, we derive the partial derivative of $R(L, t_s)$ with respect to the variable t_s as follows.

$$\begin{aligned} \frac{\partial R(L,t_s)}{\partial t_s} &= -\frac{2L}{t_s^2} \cdot p^{2(\frac{L}{t_s} - 1)} \cdot \overline{F}(t_s)^{\frac{L}{t_s}} \cdot \ln p \\ &\quad + p^{2(\frac{L}{t_s} - 1)} \cdot \overline{F}(t_s)^{\frac{L}{t_s}} \cdot (-\frac{L}{t_s^2} \cdot \ln \overline{F}(t_s) + \frac{L}{t_s \overline{F}(t_s)} \cdot \frac{d\overline{F}(t_s)}{dt_s}) \end{aligned}$$

Let $\frac{\partial R}{\partial t_s}(L, t_s) = 0$, we have

$$\frac{t_s}{\overline{F}(t_s)} \cdot \frac{d\overline{F}(t_s)}{dt_s} - \ln \overline{F}(t_s) - 2 \ln p = 0. \tag{4}$$

As Eq. (2) is a concave function, the optimal rejuvenation period that maximizes the system reliability can be calculated by solving Eq. (4) with given $\lambda(t)$ and p.

Lemma 2. *The optimal rejuvenation period is only influenced by network transmission failure rate λ_0 and transient fault occurrence rate $\lambda(t)$, but not by system longevity L.* □

Proof. The lemma can be directly proven by Eq. (4), where $\overline{F}(t) = e^{-\int_0^t \lambda(x)dx}$, and p is a constant with fixed λ_0. ■

The Weibull distribution is commonly used to model the distribution of transient faults [23], with failure rate $\lambda(t) = kt^{k-1}/r^k$ and cumulative distribution function $F(t) = 1 - e^{-(t/r)^k}$, where $r > 0$ and $k > 0$ are scale and shape parameters. The failure rate increases with time t if $k > 1$.

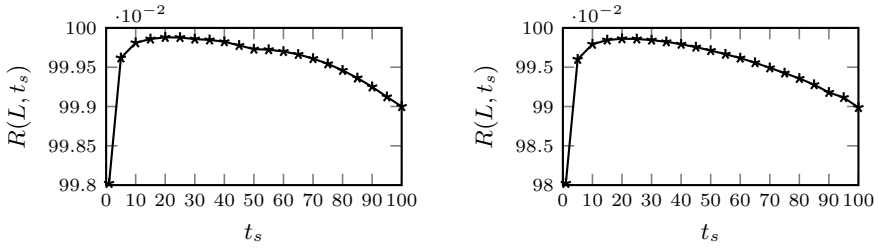
In Section 3, we have made the assumption that due to aging effects, the system transient failure rate increases with time. Hence, we can use Weibull distribution with $k > 1$ to model aging effects. Substitute $\overline{F}(t) = e^{-(t/r)^k}$ into Eq. (4) and solve the equation, we obtain the optimal rejuvenation period that maximizes the system reliability as follows

$$t_s^* = \sqrt[k]{\frac{2r^k \ln p}{1-k}}. \tag{5}$$

5 Experimental Results

In this section, we empirically evaluate the relationship between rejuvenation period and system reliability. In the experiments, we assume the probability of a successful task migration between $\mathcal{P}_1$ and $\mathcal{P}_2$ is $p = 0.99999$ and the system transient fault distribution follows Weibull distribution with $r = 1000$ and $k = 3$, i.e., $\lambda(t) = 3t^2/10^9$ and $F(t) = 1 - e^{-(t/1000)^3}$. We set rejuvenation period ranging from 1 to 100 as $\{1, 5, 10, \cdots, 95, 100\}$, and system operation time $L = 100$, and $1,000$ respectively.

For each rejuvenation period, we use Eq. (1))to calculate the system reliability $R(L, t_s)$. Fig. 5 shows the system reliability under different rejuvenation periods for both longevity settings.



(a) $L = 100$ (b) $L = 1000$

Fig. 5. Reliability vs Rejuvenation Period

From Fig. 5, we have the following observations:

1. When the rejuvenation period increases, the system reliability first increases and then decreases.

2. Neither too small rejuvenation period nor too large rejuvenation period has positive impact on the system reliability. Too frequent rejuvenation in fact lowers system reliability.
3. The experimental optimal rejuvenation period that maximizes the system reliability is consistent with the mathematical analysis (Eq (4)). In particular, for both experiment settings the optimal rejuvenation period $t_s = 20$ is the nearest value with the mathematical analysis result $t_s^* = 21.54$ (Eq (5)) among all provided rejuvenation periods.
4. The system longevity, i.e., its operation time, does not impact the optimal rejuvenation period for maximizing system reliability which is consistent with Lemma 2. In particular, the optimal rejuvenation is $t_s = 20$ for both $L = 100$ and $L = 1000$ experiment settings.

6 Conclusion

Passive fault-tolerance techniques are not sufficient to maintain system reliability for long lasting and continuous applications. Preventive and proactive fault-tolerance techniques are needed to guarantee system's reliability. In this paper, we use both backup and software rejuvenation mechanisms to maximize system's reliability. In our study, we take both transient faults caused by software aging and network transmission faults into consideration and have mathematically analyzed the optimal rejuvenation period that maximizes system reliability. The empirical study confirms with the theoretic analysis.

References

1. Jacques-Louis Lions et al. Ariane 5 flight 501 failure, 1996.
2. Arthur G Stephenson, Daniel R Mulville, Frank H Bauer, Greg A Dukeman, Peter Norvig, LS LaPiana, PJ Rutledge, D Folta, and R Sackheim. Mars climate orbiter mishap investigation board phase i report, 44 pp. *NASA, Washington, DC*, 1999.
3. AT. Tai, S.N. Chau, L. Alkalaj, and H. Hecht. On-board preventive maintenance: analysis of effectiveness and optimal duty period. In *Object-Oriented Real-Time Dependable Systems, 1997. Proceedings., Third International Workshop on*, pages 40–47, Feb 1997.
4. AT. Tai, L. Alkalai, and S.N. Chau. On-board preventive maintenance for long-life deep-space missions: a model-based analysis. In *Computer Performance and Dependability Symposium, 1998. IPDS '98. Proceedings. IEEE International*, pages 196–205, Sep 1998.
5. Sandeep Chatterjee, Mohammad Fawaz, and Farid N Najm. Redundancy-aware electromigration checking for mesh power grids. In *Computer-Aided Design (ICCAD), 2013 IEEE/ACM International Conference on*, pages 540–547. IEEE, 2013.
6. James R Black. Electromigrationa brief survey and some recent results. *Electron Devices, IEEE Transactions on*, 16(4):338–347, 1969.
7. Sachin Garg, Aad van Moorsel, Kalyanaraman Vaidyanathan, and Kishor S Trivedi. A methodology for detection and estimation of software aging. In *Software Reliability Engineering, 1998. Proceedings. The Ninth International Symposium on*, pages 283–292. IEEE, 1998.

8. Y. Huang, C. Kintala, N. Kolettis, and N.D. Fulton. Software rejuvenation: analysis, module and applications. In *Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth International Symposium on*, pages 381–390, June 1995.
9. Matlab R2012b. `http://www.mathworks.com/products/new_products/release2012b.html`.
10. A Bobbio, S. Garg, M. Gribaudo, A Horvath, M. Sereno, and M. Telek. Modeling software systems with rejuvenation, restoration and checkpointing through fluid stochastic petri nets. In *Petri Nets and Performance Models, 1999. Proceedings. The 8th International Workshop on*, pages 82–91, 1999.
11. Zheng Li, Li Wang, Shangping Ren, and Gang Quan. Energy minimization for checkpointing-based approach to guaranteeing real-time systems reliability. In *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2013 IEEE 16th International Symposium on*, pages 1–8, June 2013.
12. V.P. Koutras and A.N. Platis. Semi-markov availability modeling of a redundant system with partial and full rejuvenation actions. In *Dependability of Computer Systems, 2008. DepCos-RELCOMEX '08. Third International Conference on*, pages 127–134, June 2008.
13. R.S. Hanmer and V.B. Mendiratta. Rejuvenation with workload migration. In *Dependable Systems and Networks Workshops (DSN-W), 2010 International Conference on*, pages 80–85, June 2010.
14. C Singh. Reliability modeling of tmr computer systems with repair and common mode failures. *Microelectronics Reliability*, 21(2):259 – 262, 1981.
15. T.M. Khoshgoftaar and N. Seliya. Tree-based software quality estimation models for fault prediction. In *Software Metrics, 2002. Proceedings. Eighth IEEE Symposium on*, pages 203–214, 2002.
16. András Pfening, Sachin Garg, Antonio Puliafito, Miklós Telek, and Kishor S. Trivedi. Optimal software rejuvenation for tolerating soft failures. *Perform. Eval.*, 27-28:491–506, October 1996.
17. AT. Tai and L. Alkalai. On-board maintenance for long-life systems. In *Application-Specific Software Engineering Technology, 1998. ASSET-98. Proceedings. 1998 IEEE Workshop on*, pages 69–74, Mar 1998.
18. Amr Sadek and Nikolaos Limnios. Nonparametric estimation of reliability and survival function for continuous-time finite markov processes. *Journal of Statistical Planning and Inference*, 133(1):1 – 21, 2005.
19. K.S. Trivedi, K. Vaidyanathan, and K. Goseva-Popstojanova. Modeling and analysis of software aging and rejuvenation. In *Simulation Symposium, 2000. (SS 2000) Proceedings. 33rd Annual*, pages 270–279, 2000.
20. H. Okamura and T. Dohi. Availability optimization in operational software system with aperiodic time-based software rejuvenation scheme. In *Software Reliability Engineering Workshops, 2008. ISSRE Wksp 2008. IEEE International Conference on*, pages 1–6, Nov 2008.
21. V.P. Koutras, A.N. Platis, and N. Limnios. Availability and reliability estimation for a system undergoing minimal, perfect and failed rejuvenation. In *Software Reliability Engineering Workshops, 2008. ISSRE Wksp 2008. IEEE International Conference on*, pages 1–6, Nov 2008.
22. Nagarajan Kandasamy, J.P. Hayes, and B.T. Murray. Transparent recovery from intermittent faults in time-triggered distributed systems. *Computers, IEEE Transactions on*, 52(2):113–125, Feb 2003.
23. R. Barlow and F. Proschan. *Mathematical Theory of Reliability*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 1996.