

# Extended Classification of Software Faults Based on Aging

Kalyanaraman Vaidyanathan and Kishor S. Trivedi  
Dept. of ECE, Duke University  
Durham, NC 27708, USA  
{kv,kst}@ee.duke.edu

## 1. Introduction

Jim Gray classifies software faults into *Bohrbugs* and *Heisenbugs*. Bohrbugs are essentially permanent design faults and hence almost deterministic in nature. They can be identified easily and weeded out during the testing and debugging phase (or early deployment phase) of the software life cycle. Heisenbugs, on the other hand, belong to the class of temporary internal faults and are intermittent. They are essentially permanent faults whose conditions of activation occur rarely or are not easily reproducible. Hence these faults result in transient failures, i.e., failures which may not recur if the software is restarted. Some typical situations in which Heisenbugs might surface are boundaries between various software components, improper or insufficient exception handling and interdependent timing of various events. It is for this reason that Heisenbugs are extremely difficult to identify through testing. Most recent studies on failure data have reported that a large proportion of software failures are transient in nature caused by phenomena such as overloads or timing and exception errors. In this paper, we extend Gray's classification of software faults based on the phenomenon of software aging which has recently gained recognition and importance.

## 2. Software Aging and Related Faults

The phenomenon of *software aging* has been reported by several recent studies [3, 4]. It was observed that once the software was started, potential fault conditions gradually accumulated with time leading to either performance degradation or transient failures or both. Failures may be of crash/hang type or those resulting from data inconsistency because of aging. Typical causes of aging, i.e., slow degradation, are memory bloating or leaking, unreleased file-locks, data corruption, storage space fragmentation and accumulation of round off errors. Software aging has not only been observed in software used on a mass scale but also in specialized software used in high availability and safety-critical applications [3].

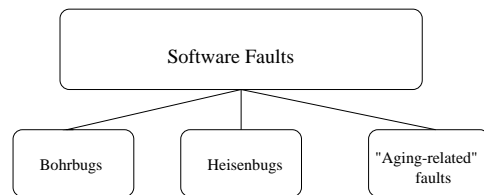


Figure 1. Extended classification

We designate faults attributed to software aging, which are quite different from Bohrbugs and Heisenbugs, as *aging-related* faults. These faults are similar to Heisenbugs in that they are activated under certain conditions (for example, lack of OS resources) which may not be easily reproducible. However, as discussed in Section 3, their modes and methods of recovery differ significantly. Figure 1 shows our extended classification.

## 3. Software Rejuvenation

To counteract the phenomenon of software aging, Huang et al. [4] proposed a proactive approach of fault management, called *software rejuvenation*. It involves occasionally terminating an application or a system, cleaning its internal state and restarting it. This process removes the accumulated errors and frees up operating system resources, thus preventing in a proactive manner, the unplanned and potentially expensive system outages due to the software aging. Since the preventive action can be done at optimal times, for example when the load on the system is low, it reduces the cost of system downtime compared to reactive recovery from failure. Thus, software rejuvenation is a cost-effective technique for dealing with software faults that include protection not only against hard failures, but against performance degradation as well. Numerous examples of software rejuvenation exists in real-life applications [3]. More recently, rejuvenation has been implemented in IBM's xSeries servers to improve performance and availability [1]. The important difference in the treatment of

Heisenbugs and aging-related faults is that in the former, the treatment is reactive while in the latter, it can be proactive as well.

### 3.1 Approaches to Software Rejuvenation

Software rejuvenation can be divided broadly into two approaches as follows.

- **Open-loop approach:** In this approach, rejuvenation is performed without any feedback from the system. Rejuvenation in this case, can be based just on elapsed time (periodic rejuvenation) and/or instantaneous/cumulative number of jobs on the system [5].
- **Closed-loop approach:** In the closed-loop approach, rejuvenation is performed based on information on the system “health”. The system is monitored continuously (in practice, at small deterministic intervals) and data is collected on the operating system resource usage and system activity. This data is then analyzed to estimate time to exhaustion of a resource which may lead to a component or an entire system degradation/crash. This estimation can be based purely on time [1, 3] or can be based on both time and system workload [5]. Another approach to estimate the optimal time to rejuvenation could be based on system failure data [2].

The closed-loop approach can also be classified based on whether the data analysis is done off-line or on-line. Off-line data analysis is done based on system data collected over a period of time (usually weeks or months) [3, 5]. The analysis is done to estimate time to rejuvenation. This off-line analysis approach is best suited for systems whose behavior is fairly deterministic. The on-line closed-loop approach, on the other hand, performs on-line analysis of system data collected at deterministic intervals [1]. The analysis is done after every new set of data is collected to estimate time to rejuvenate. This approach is very general and can work with systems with unpredictable behavior or whose behavior cannot be easily determined. In this case, future system behavior is computed based on the current system parameter values and weighted historical values.

This classification of approaches to rejuvenation is shown in Figure 2.

### 3.2 Rejuvenation Granularity

Rejuvenation is a very general proactive fault management approach and can be performed at different levels - the system level or the application level. An example of a

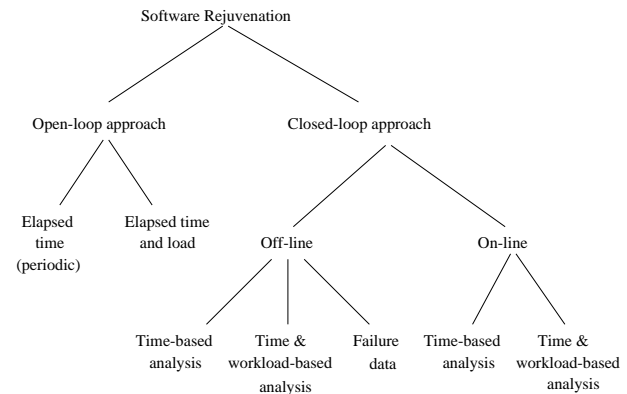


Figure 2. Rejuvenation approaches

system level rejuvenation is a hardware reboot. At the application level, rejuvenation is performed by stopping and restarting a particular offending application, process or a group of processes. This is also known as a *partial rejuvenation*. The above rejuvenation approaches when performed on a single node can lead to undesired and often costly downtime. Rejuvenation has been recently extended for cluster system, in which two or more nodes work together as a single system [1]. In this case, rejuvenation can be performed by causing no or minimal downtime by failing over applications to another spare node.

## 4 Conclusion

In this paper, we extended Gray’s classification of software faults to include aging-related faults. We also dealt with the treatment of these faults by software rejuvenation and discussed the various approaches and current methods of rejuvenation in practice.

## References

- [1] V. Castelli et al. Proactive Management of Software Aging. *IBM JRD*, Vol. 45, No. 2, March 2001.
- [2] T. Dohi, K. Goševa-Popstojanova and K. Trivedi. Statistical Non-parametric Algorithms to Estimate the Optimal Rejuvenation Schedule In *Proc. PRDC 2000*, Los Angeles, CA.
- [3] S. Garg, A. van Moorsel, K. Vaidyanathan and K. Trivedi. A Methodology for Detection and Estimation of Software Aging. In *Proc. ISSRE-98*, Paderborn, Germany.
- [4] Y. Huang et al. Software Rejuvenation: Analysis, Module and Applications. In *Proc. FTCS-25*, Pasadena, CA.
- [5] K. Trivedi, K. Vaidyanathan and K. Goševa-Popstojanova. Modeling and Analysis of Software Aging and Rejuvenation, In *Proc. of the 33rd Ann. Simul. Symp.*, Washington D.C., April 2000.