SOFTWARE REJUVENATION

A TERM PAPER FOR CSC 532 SOFTWARE ENGINEERING

Submitted By:
Swetha Kendyala
ABSTRACT:
Software rejuvenation is the concept of gracefully terminating an application and immediately restarting it at a clean internal state. In a client-server type of application where the server is intended to run perpetually for providing a service to its clients, rejuvenating the server process periodically during the most idle time of the server increases the availability of that service. In a long-running computation-intensive application, rejuvenating the application periodically and restarting it at a previous checkpoint increases the likelihood of successfully completing the application execution. Software rejuvenation is used in a billing data collection subsystem of a telecommunications operations system and other continuously-running systems and scientific applications in AT&T are described.

INTRODUCTION:
As software continues to become larger and more complex, it is becoming the dominant source of system failures. Even though all software-induced system failures have their cause in flexible design faults, the sheer complexity of modern day software along with inherent limitations in testing make it practically impossible to produce truly fault-free software. Among various kinds of software faults "bugs" of a particularly elusive nature have come to light. These bugs, commonly named "Heisenbugs" are characterized by their non-deterministic activation, i.e. a second execution of the software, even with the same data, may not result in a failure. Transient software failures of this nature are reported in many instances in the field. The reason behind the Heisenbug's elusiveness, during testing as well as in the operational phase, is the dependence
of their activation on the operational environment. Since exactly the same operational environment which led to error and failure is unlikely to be reproduced, the failure upon a second execution is avoided. This is especially true, if the environment is deliberately changed or "cleaned".

Following this reasoning, software rejuvenation has recently been proposed to avoid failures caused by Heisenbugs. Software rejuvenation is the "periodic preemptive rollback of continuously running applications to prevent failures in the future". The implementation of this idea involves "cleaning up the in-memory data structures, respawning the processes at the initial state, logging administrative records, etc." A typical example where rejuvenation can be beneficial is when the software experiences "memory leaks". Causing a continuous reduction in the amount of free memory. Rejuvenation then would consist of garbage collection, or a hardware reboot in the worst case, to reclaim memory. Since rejuvenation typically involves an overhead, an important research issue is to determine when and how often the software should be rejuvenated. Performability modeling of software rejuvenation enables us to answer this question.
Software Rejuvenation

To counteract the phenomenon of software aging, a proactive approach of fault management, called “software rejuvenation” is introduced. 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.

APPOROACHES 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.
**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 or can be based on both time and system workload. Another approach to estimate the optimal time to rejuvenation could be based on system failure data. 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). 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. 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 the adjacent figure.
CONCLUSION:
Software rejuvenation 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. More recently, rejuvenation has been implemented in IBM’s xSeries servers to improve performance and availability.

REFERENCES:
15-17 Nov. 2000 Page(s):25 - 34]
