Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER Computers and Mathematics with Applications 51 (2006) 335–344
An International Journal
computers & mathematics
with applications
www.elsevier.com/locate/camwa

Discrete-Time Cost Analysis for a Telecommunication Billing Application with Rejuvenation

KAZUKI IWAMOTO, TADASHI DOHI AND HIROYUKI OKAMURA
Department of Information Engineering
Graduate School of Engineering
Hiroshima University
1-4-1 Kagamiyama, Higashi-Hiroshima 739–8527, Japan
<dohi><okamu>@gal.sys.hiroshima-u.ac.jp

NAOTO KAIO
Department of Economic Informatics
Faculty of Economic Sciences
Hiroshima Shudo University
1717 Ohtsuka, Numata-Cho, Asa-Minami-Ku
Hiroshima 731-31, Japan
kaio@shudo-u.ac.jp

Abstract—Software rejuvenation is a proactive fault management technique that has been extensively studied in the recent literature. In this paper, we focus on an example for a telecommunication billing application considered in [1] and develop the discrete-time stochastic models to estimate the optimal software rejuvenation schedules. More precisely, two software cost models with rejuvenation are formulated via the discrete semi-Markov processes, and the optimal software rejuvenation schedules which minimize the expected costs per unit time in the steady state are derived analytically. Further, we develop statistically nonparametric algorithms to estimate the optimal software rejuvenation schedules, provided that the complete sample data of failure times are given. Then, a new statistical device, called discrete total time on test statistics, is introduced. Finally, we examine asymptotic properties for the statistical estimation algorithms proposed in this paper through a simulation experiment.

Keywords—Software rejuvenation, Fault management, Cost, Discrete-time model, Nonparametric estimation, Total time on test.

1. INTRODUCTION

Software aging will affect the performance of the application and eventually cause it to fail [2,3]. Huang *et al.* [1] report this phenomenon in telecommunications billing applications where over time the application experiences a crash or a hang failure. Avritzer and Weyuker [4] discuss the aging in a telecommunication switching software where the effect manifests as gradual performance degradation. Software aging has also been observed in widely-used software like NETSCAPE

This research was partially supported by the Ministry Education, Culture, Sports, Science and Technology, Grant-in-Aid for Exploratory Research; Grant No. 15651076 (2003–2005), Scientific Research (B); Grant No. 16310116 (2004–2006), and Research Program 2005 under the Institute for Advanced Studies of Hiroshima Shudo University.

0898-1221/06/$ - see front matter Typeset by AMS-TEX
doi:10.1016/j.camwa.2005.11.007

and MSN EXPLORER. As the most vivid example of software aging in safety critical systems, we may remember the PATRIOT's software [5], where the accumulated errors led to a failure that resulted in loss of human life.

Resource leaking and other problems causing software to age are due to the software faults whose fixing is not always possible because, for example, application developer cannot access the source code directly. Furthermore, it is almost impossible to fully test and verify whether a piece of software is fault-free. Testing software becomes harder if it is complex, and more testing and debugging cycle times are often reduced due to smaller release time requirements. In fact, common experience suggests that most software failures are transient in nature [6]. Since transient failures will disappear if the operation is retried later in slightly different context, it is difficult to characterize their root origin. Therefore, the residual faults have to be tolerated in the operational phase. Usual strategies to deal with failures in operational phase are reactive in nature; they consist of action taken after failure.

Recently, a complementary approach to handle transient software failures, called *software rejuvenation*, is rather popular [1]. Software rejuvenation is a preventive and proactive solution that is particularly useful for counteracting the phenomenon of software aging. It involves stopping the running software occasionally, cleaning its internal state and restarting it. Cleaning the internal state of a software might involve *garbage collection*, *flushing operating system kernel tables*, *reinitializing internal data structures*, etc. An extreme, but well-known example of rejuvenation is a *hardware reboot*. Apart from being used in an *ad hoc* manner by almost all computer users, software rejuvenation has been used in high availability and mission critical systems [7,8]. Although the fault in the software program still remains, performing software rejuvenation occasionally or periodically can prevent failures due to that fault effectively.

Huang *et al.* [1] use continuous-time Markov chain to model software rejuvenation, and consider the two-step failure model where the application goes from the initial robust (clean) state to a failure probable (degraded) state from which two actions are possible: rejuvenation or transition to failure state. Garg *et al.* [9] introduce a periodic rejuvenation to deal with deterministic interval between successive software rejuvenations. Then, the system behavior is represented by a Markov regenerative stochastic Petri net. Dohi *et al.* [10] extend the original Huang *et al.* model [1] to the semi-Markov models, and further develop statistically nonparametric algorithms to estimate the optimal software rejuvenation schedules.

In this paper, we consider the similar cost problems discussed in [10] under a somewhat different operating condition. That is, we deal with a software system which is operating in discrete time and develop two stochastic models to determine the optimal software rejuvenation schedules minimizing the expected costs per unit time in the steady state. In literature [1,9,10], several continuous-time models are developed for the theoretical determination of the software rejuvenation schedule. In these models, it is assumed that the failure time and the other statistical data have to be measured based on cumulative operation (CPU) time precisely.

Also, to minimize the expected cost per unit time in the steady state with higher accuracy, the optimal software rejuvenation should be performed in the minimum unit of time such as seconds or minutes. This assumption on the continuous setting may be tractable mathematically, but is rather restrictive in practice. In addition, although continuous monitoring of the software system does not need so much cost, it is not always possible in many cases to rejuvenate the system at the timing when the cumulative operation time reaches an optimal level. The above points motivate us to consider the cost models under the discrete-time setting, such as hours, days, number of completed jobs, etc. Further, an effort to collect the statistical data can be reduced drastically if one considers the discrete model, because the data should be observed at the discrete points of time.

As referred in [10], it is noted that the failure time distribution needs to be specified to derive the optimal software rejuvenation schedule and the corresponding expected cost. This seems to be also quite restrictive, since the determination of the theoretical distribution from the real data