Analysis of Preventive Maintenance in Transactions Based Software Systems

Sachin Garg[1], Antonio Puliafito[2], Miklós Telek[3], Kishor Trivedi[1]

[1] Center for Advanced Comp. & Comm.
Department of Electrical Engineering, Duke University
Durham, NC 27708-0291 - USA
E-mail: {sgarg,kst}@ee.duke.edu
URL: http://www.ee.duke.edu/~{sgarg,kst}

[2] Istituto di Informatica, Università di Catania
Viale A. Doria 6, 95025 Catania - Italy
E-mail: ap@iit.unict.it

[3] Department of Telecommunications
Technical University of Budapest
1521 Budapest - Hungary
E-mail: telek@pyxis.hit.bme.hu

Abstract

Preventice maintenance of operational software systems is a novel technique for software fault tolerance, used specifically to counteract the phenomenon of software "aging". However, it incurs some overhead. The necessity to do preventive maintenance not only in general purpose software systems of mass use but also in safety-critial and highly availabile systems clearly indicate the need to follow an analysis based approach in determining the optimal times to perform preventive maintenance.

In this paper, we present an analytical model of a software system which serves transactions. Due to aging, not only the service rate of the software decreases with time but the software itself experiences crash/hang faliures which result in its unavailability for service. Two policies for preventive maintenance are evaluated for resulting steady state availability, probability that an arriving transaction is lost and an upper bound on the expected response time of a transition. Numerical examples are presented to illustrate the usefulness and applicability of the models.

Keywords: Preventive Maintenance, Transactions based Software Systems, Markov Regenerative Models.

1 Introduction

It is now well established that system outages are caused more due to software faults than due to hardware faults [24, 11]. Given the current growth in software complexity and reuse, the trend is not likely to change. It is also well known that regardless of development, testing and debugging time, software still contains some residual faults. Thus, fault tolerant software has become an effective alternative to virtually impossible fault-free software. The scope of this paper lies in the quantitative evaluation of a novel technique for software fault tolerance, viz., *preventive maintenance of operational software systems*. Although, the use of preventive maintenance is common in physical systems, especially those of mechanical/electro-mechanical nature, its potential effectiveness in enhancing software dependability has only recently been recognized. As we shall exemplify later, in certain situations, it is simply necessary.

Traditional methods of software fault-tolerance, namely N-version programming [2], recovery blocks [23] and N-self checking programming [19] are all based on design diversity, where independently operating teams of developers generate two or more variants of software from the same set of specifications. Another common charateristic of all of the above techniques is their reactive nature, i.e., the fault tolerance mechanism, or rather fault masking, takes effect after at least one version has experienced failure, the idea being that at least a subset of the variants will not fail and therefore be collectively sufficient to provide the correct output. The primary drawback of all the above techniques is the extremely high cost involved in generating multiple versions, which restricts their use to safety-critical software such as for atomic power plants, railroad switching etc. [18], where the cost is justified.

Primarily, the following two factors motivate the need to explore alternate techniques for software fault tolerance.

1. *Reliability/Availability Versus Cost*

   The need for high reliability and availability is not just restricted to safety-critical systems [11]. Telephone switches [7], airline reservation systems, process and production control, stock trading system, computerized banking etc. all demand

very high availability. A survey showed that computer downtime in non-safety critical systems cost over 3.8 Billion in 1991 in U.S. [25]. With current explosive increase in the popularity of network centric computing, web servers too need to be highly available. In most of these systems, the cost providing fault tolerance via use of multiple variants is prohibitive. With commercial considerations driving technology more than ever, release times of software are required to be less and less forcing organizations to reduce testing and debugging cycle times. Further, with software reuse gaining popularity, many times, it is simply not feasible to test the middleware and operating system on which the final software product is based. This leaves no option but to tolerate the residual faults during the operational phase.

2. *Nature of software failures*

   More recently, from the study of field failure data, it has been observed that a large percentage of operational software failures are transient in nature [12, 13, 17], caused by phenomena such as overloads or timing and exception exception errors [24, 5]. A common characteristic of these type of failures is that upon re-execution of the software, the failure does not recur. The error condition, which results in the failure, typically manifests itself in the operating environment of the executing software. Due to the complexity of modern-day operating systems and intermediate layer software, it has been observed that the same error condition, when the software is re-executed, is unlikely to recur, thus avoiding the failure.

A study done by Adams implies that the best approach to masking software faults is to simply restart the system [11]. Based on the above two factors, environment diversity has been proposed as a cheap yet effective technique for software fault-tolerance [15, 18]. Behavior of a process (executing software) is determined by three components; the volatile state, the persistent state and the OS environment. The volatile state consists of the program stack and static and dynamic data segments. Persistent state refers to all the user files related to a program's execution while the OS environment refers to resources that the program must access through the operating system, such as swap space, file systems, communication channels, keyboard, monitors, time etc. [27].

Typical transient failures occur because of design faults in software which result in unacceptable erroneous states in the OS environment of the process. Therefore, the key idea behind environment diversity is to modify the operating environment of the running process. Typically, this has been done on a corrective basis, i.e., upon a failure, the software is restarted after some cleanup, which in most cases results in a different, error free OS environment state thus avoiding further failure.

Recently, the phenomenon of software "aging" [16] has come to light where such error conditions actually accrue with time and/or load. This observation has led to proposals of pro-active approaches to environment diversity in which the operational software is occasionally stopped and "cleaned up" to remove any potential error conditions. Since the preventive actions can be performed at suitable times (such as when there is no load on the system), it typically results in lesser downtime and cost than the corrective approach. Even so, it incurs some overhead and if done more often than necessary will result in higher downtime/cost. *Therefore, an important research issue is to determine the optimal times to perform preventive maintenance of operational software systems.*

In this paper, we present a stochastic model for a transactions based software system which employs preventive maintenance (henceforth referred to as PM). Three measures, availability of the software to provide service, probability of loss of a transaction and response time of a transaction are evaluated. The model is developed under very general conditions and requires numerical solution. We compute each of the three measures under two policies for PM, which were proposed in [9], in the same framework. The rest of the paper is organized as follows. In Section 2, we present real life examples of aging and PM to illustrate the different forms in which they occur and to motivate the need for analysis of such systems. In Section 3, we describe the system model, along with assumptions on modeling aging, failure and PM policies. Section 4 comprises of the analytical solution of the model for availability, loss probability and response time measures. In Section 5, we illustrate the usefulness of the models via numerical examples. The two proposed PM policies are compared along with the effect of model parameters on the derived measures. It is shown that the PM interval which maximizes

availability may be very different from the PM interval which minimizes the probability of loss or the response time. Therefore, selection of the optimum PM interval must be done carefully. Finally, in Section 6, we present the conclusions along with future research directions.

2 Preventive Maintenance of Operational Software

While monitoring real applications, the phenomenon of software "aging" has been reported. It is observed that potential fault conditions slowly accumulate over time since the beginning of the software execution resulting in performance degradation and/or transient failures[1]. Failures of both crash/hang type as well as those resulting in data inconsistency because of aging have been reported. Memory bloating and leaking, unreleased file-locks, data corruption, storage space fragmentation and accumulation of roundoff errors are some typical causes of slow degradation. Examples of aging can be found not only in software used on a mass scale but also in specialized software used in high-availability and safety critical applications.

Widely used web browser "Netscape" is known to suffer from serious memory leaking problems which leads to occasional crash or hang of the application(s), especially in a computer with relatively less swap space. Similar memory leaking problem has been reported in the news-reader program "xrn". All Windows '95 users are familiar with the occasional "switch off and on" of the computer to recover from hangs. Such examples of aging in software of mass-use are probably just an inconvenience, but in systems with high reliability/availability requirements, software aging can result in very cost.Huang et. al. report this phenomenon in telecommunications billing applications where over time the application experiences a crash or a hang failure [16]. Avritzer and Weyuker have witnessed aging in telecommunications' switching software where the effect manifests as gradual performance degradation [3]. The service rate of the

[1]It should be noted that the term "software aging" was used in [1] to mean degradation in the quality of the software by an increase in number or severity of design faults due to repeated bug fixes, which is different from our meaning. Perhaps in our context "process aging" is more appropriate, but we choose to keep the term software aging to be consistent with the usage by Huang et. al.[16].

software decreases with time incresing queue lengths and eventually starts losing packets. Perhaps the most vivid example of aging can be found in [21], where the failure resulted in loss of human life. Patriot missiles, used during the Gulf war to destroy Iraq's Scud missiles used a computer whose software accumulated error. The effect of aging in this case was mis-interpretation of an incoming Scud as not a missile but just a false alarm, which resulted in the death of twenty-eight U.S. soldiers.

One way to counteract aging is to avoid the fault itself. Specifically, for memory leaks, packages like *Purify* and *Insure++* have been made available commercially. These, however, are for use during the development phase with which memory leaks in a software can be detected and corrected. Given that not all faults can be avoided (memory leakage is a very small subset of faults resulting in aging) and that sometimes, it is simply not feasible to fix faults (use of third party software, unavailability of source code etc.), the complimentary approach of tolerating the residual faults must be employed. Moreover, the fault tolerance technique must not only be efficient in terms of providing the necessary reliability or availability but must be cost effective as well. Given the observed nature of failures, PM of operational software is a potential candidate.

Huang et. al.[16] have proposed the technique of "Software Rejuvenation", which simply involves stopping the running software occasionally, removing the accrued error conditions and restarting the software. Garbage collection, flushing operating system kernel tables, reinitializing internal data structures are some examples of what cleaning the internal state of a software might involve. An extreme, but well-known example of rejuvenation is a hardware reboot. It has been implemented in the real-time system collecting billing data for most telephone exchanges in the United States [4]. A very similar technique has been used by Avritzer and Weyuker in a large telecommunications switching software [3], where the switching computer is rebooted occasionally upon which its service rate is restored to the peak value. They call it *software capacity restoration*. Both of the above independently used techniques are specific cases of PM performed on operational software. Grey [14] proposed performing operations solely for fault management in SDI (Strategic Defense Initiative) software which are invoked

whether or not the fault exists and called it "operational redundancy". Tai et. al. [26] have proposed and analyzed the use of on-board PM for maximizing the probability of successful mission completion of spacecrafts with very long mission times. In a safety critical environment also, the *necessity* of performing PM is evident from the example of aging in Patriot's software [21]. In the words of the author, *On 21 February, the office sent out a warning that "very long running time" could affect the targeting accuracy. The troops were not told, however, how many hours "very long" was, or that it would help to switch the computer off and on again after 8 hours.*

The concept of PM as such is not new. All system administrators, whether responsible for small system setup or for large commercial system setups such as banking, reservations systems, inventory control systems etc. routinely perform PM, even if on an ad-hoc basis. In some cases, it is performed on a per process basis, while in others a system wide maintenance is done. In each case, however, the maintenance incurs an overhead which should be balanced against the cost incurred due to unexpected outage caused because of a failure. This in turn demands a quantitative analysis, which in the context of software systems has only recently started getting attention. Mr. Bernstein stresses the need of emergence of a new field "software dynamics" [4] and calls for "developing the design constraints, analytically, to make software behavior periodic and stable in its operational phase".

The contribution of this paper lies in presenting an analytical model under very general assumptions for transaction based software which experiences aging and employs PM to avoid unexpected outages. Further, analyze two policies for PM, namely

1. Purely time based: PM is performed at fixed deterministic interval
2. Time and load based: PM is attempted at fixed intervals and performed only if the software is currently not serving any transactions.

In both cases, equations for expected availability, long run probability of a transaction loss and an upper bound on mean response time are derived.

2.1 Previous Work

The single most important factor (as shall be shown via numerical example) in determining the accuracy of such a model is the assumptions made in capturing aging. Primarily, assumptions regarding the following aspects of aging need to be made.

- *Effect of Aging:* Effect of aging has been witnessed as crash/hang failure, which results in unavailability of the software, and gradual performance degradation[2]. User perceivable impact of one may be dominant than the other, but typically both are present to some degree in a software which experiences aging. In [16, 8, 9, 10, 26] only the failures causing unavailability of the software are considered, while in [22] considers only a gradually decreasing service rate of a software which serves transactions is assumed. In this paper, we consider both the effects together in a single model.
- *Distribution* There is no consensus on the time to failure distribution of an operational software and of the nature of service degradation it experiences. Therefore, for wide applicability, it is essential that a model be able to accomodate general distributions and not be restricted to predetermined ones. This way, with the availability of data, a specific distribution can then be applied on a per system basis. Models proposed in [16, 8, 9] are restricted to hypo-exponentially distributed time to failure. Those proposed in [10, 22, 26] can accomodate general distributions but only the specific aging effect they capture. In our model, we allow for generally distributed time to failure as well as for the service rate to be an arbitrary function of time.
- *Dependence on Load* None of the previous studies capture the effect of load on aging. As it has been noted [24] that transient failures are very much dependent on load, in our model we allow for the time to failure and the degradation in the service rates to be functions of instantaneous as well as mean accumulated load.

[2]Incorrect output because of data inconsistency, from a modeling stadpoint, can be captured in the model for crash/hang failure since it is simply a failure at a specific time point

	Aging captured as:		Accommodates	Model load	Measure Evaluated				
	crash/hang failure	Performance degradation	general distribution?	dependence?	Availability	Loss Probability	Response Time	Completion Time	Prob.
[16]	X				X				
[8]	X				X				
[9]	X				X	X			
[10]	X		X					X	
[22]		X	X			X			
[26]	X		X						X
This	X	X	X	X	X	X	X		

Table 1: Comparison of model assumptions and measures with previous work

Table 1 summarizes the differences in capturing the effect of aging and on the assumptions in the distribution and dependence of these effects. It also shows the differences in the measures evaluated. In [10, 26] software with a finite mission time is considered. Mean completion time in the presence of aging failures is computed in [10] whereas is [26], the probability of successful completion by the mission deadline is computed. In the rest [16, 8, 9] as well as in this paper, measures of interest in a transaction based software intended to run forever are evaluated. Where in [16] and [8] only the steady state availability is computed, both steady state availability as well as the long run probability that a transaction is denied service are computed in [9]. In a transaction based system, the users are sometime more interested in response time. In this paper, we evaluate the steady state availability, the probability of loss of a transaction as well as an upper bound on the response time of a transaction. Optimizing one may result in an unacceptable value for the other. Optimal selection based on constraints on one or more of the measures can then be made via solution of our model. Lastly, all previous models except [10] and [26], because the class of software and therefore the associated measure in these two are not comparable, are just special cases of the model presented in this paper.

The rest of the paper deals with the proposed model, its solution and numerical example.

3 System Model

The system we study consists of a server type software to which transactions arrive at a constant rate λ. Each transaction receives service for a random period. The service rate of the software is an arbitrary function from the last renewal of the software (because of aging) denoted by $\mu(\cdot)$. Therefore, a transaction which starts service at time t_1, occupies the server for a time whose distribution is given by $1 - e^{-\int_{t_1}^{t} \mu(\cdot)\, dt}$. $\mu(\cdot)$ can be a constant, a function of time t, a function of the instantaneous load on the system, a function of total processing done in a given interval or a combination of the above. We shall defer the explicit specification of the parameter in $\mu(\cdot)$ till Section 3.1.

If the software is busy processing a transaction, arriving customers are queued. Total number of transactions that the software can accommodate is K (including the one being processed) and any more arriving when the queue is full are lost. The service discipline is FCFS. This state, in which the software is available for service (albeit with decreasing service rate) is denoted as state "A" (see Figure 1).

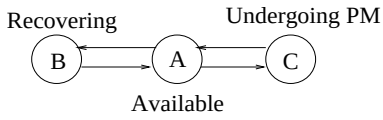

Figure 1: Macro-states representation of the software behavior

Further, the software can fail upon which recovery procedure is started. This state, in which the software is recovering and is unavailable for service is denoted as state "B". The rate at which it fails, i.e., at which the software moves from state A to state B is denoted by $\rho(\cdot)$. Let the time to failure be denoted by random variable X. Then, its distribution is given by

$$F_X(t) = 1 - e^{-\int_0^t \rho(\cdot)\, dt}.$$

The parameters $\rho(\cdot)$ depends on can be time, instantaneous load or mean accumulated load. An explicit representation of $\rho(\cdot)$ shall be deferred till Section 3.1.

The effect of aging, therefore, can be captured by using decreasing service rate and increasing failure rate, where the increase and decrease can be a function of time,

instantaneous load, mean accumulated load or a combination of the above. The service degradation and hang/crash failures in our model are assumed to be stochastically independent processes. Their interdependence, if it exists in the real system, can be approximated by using parametric dependence in the definitions of $\rho(\cdot)$ and $\mu(\cdot)$. The failure process is stochastically independent of the arrival process and any transactions in the queue at the time of failure are assumed to be lost. Moreover, any transactions which arrive while recovery is in progress are also lost. Time to recover from a failed state is denoted by Y_f with associated general distribution F_{Y_f}.

Lastly, the software occasionally undergoes PM. This state is denoted as state "C". PM is allowed only from state "A". We consider two different policies which determine the time to perform PM.

1. *Purely time based.* Under this policy, henceforth referred to as **Policy I**, PM is initiated after a constant time δ has elapsed since it was started (or restarted). We shall refer to δ under this policy as the PM interval.

2. *Instantaneous load and time based.* Under this policy, henceforth referred to as **Policy II**, a constant waiting period δ must elapse before PM is attempted. Further, after this time, PM is initiated if and only if there are no transactions in the system. δ under this policy shall be referred to as PM wait. Note that actual PM interval under Policy **II** is determined by sum of PM wait and the time it takes for the queue to get empty from that point onwards. The latter quantity is dependent on system parameters and can not be controlled. PM wait, therefore has a range $[\delta, \infty)$.

Regardless of the policy used, it takes a random amount of time, denoted by Y_r, to perform PM. Let F_{Y_r} be its distribution. As will be showed in the following section, our model does not require any assumptions on the nature of F_{Y_f} and F_{Y_r}. Only the respective expectations $\gamma_f = E[Y_f]$ and $\gamma_r = E[Y_r]$ are assumed to be finite.

Once recovery from the failed state or PM is complete, the software is reset in state A and is as good as new. From this moment, which constitutes a renewal, the whole process stochastically repeats itself. The transition behavior of the software among states A, B and C is illustrated in Figure 1.

The queuing behavior of the software, on the other hand, as determined by the two PM policies is illustrated in Figure 2. The horizontal axis represents time t and the vertical axis represents the number of transactions queued in the software at time t, denoted by $N(t)$. Figure 2(a) shows a sample path in which PM is initiated as soon constant time δ elapses. In accordance with Policy **I**, the transactions already in the queue at time δ are lost.

Figure 2(b) illustrates Policy **II** where at time δ, some transactions are in the queue , i.e., $N(\delta) > 0$. In this case, the software waits till the queue is empty upon which PM is initiated[3] This wait is a random quantity denoted in the figure by B. Intuitively, if B is very large, it is likely that the software will fail before it has a chance to undergo PM.

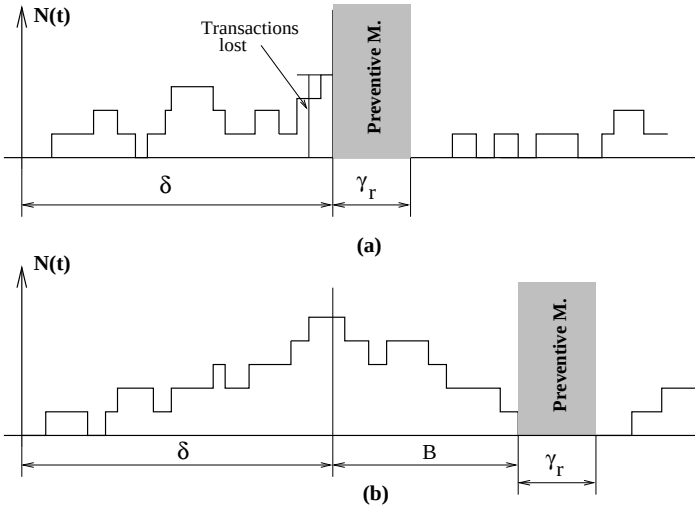

Figure 2: Sample Path of the process

Given the above behavioral model, the task at hand involves

1. Evaluate the steady state availability for policies **I** and **II**.
2. Evaluate the steady state probability that an arriving transaction will be denied service.
3. Evaluate the expected response time of a transaction successfully served.

[3]A generalization of Policy **II** in which after δ PM is initiated if and only if the number of transactions in the queue goes below a certain threshold can easily be accomodated in our mode

4. Determine the optimal value of δ, i.e., rejuvenation interval under policy **I** and rejuvenation wait under policy **II**. which provide an optimal value for availability, probability of loss and response time.

3.1 How to Capture Aging?

The effect of aging, i.e., degradation in performance and failures caousing unavailability are present in varying severity in different systems. Further, each of them may be influenced by different operating parameters in different software systems. We now show how, in our model, the varying severity and dependencies may be captured via proper choice of parameters of $\mu(\cdot)$ and $\rho(\cdot)$. Flexibility in this choice in the same framework greatly enhances the scope of applicability of our model to real software systems.

- $\mu(\cdot) = \mu$ and $\rho(\cdot) = \rho$.

  In the simplest case, the service rate as well as the failure rate are constants. This implies that there is no performance degradation and the time to failure is exponentially distributed, which because of its memoryless property contradicts aging. Therefore, constant μ and ρ do not capture the behaviour of a software system which ages.

- $\mu(\cdot) = \mu(t)$ and $\rho(\cdot) = \rho(t)$

  In this case, the service rate and the failure rates are simply functions of time. Although arbitrary functions are allowed in the model, presumably, service rate will be a monotone non-increasing function and the failure rate will be a monotone non-decreasing function of time in software which ages. If

  $$\rho(t) = \beta\alpha t^{\alpha-1}$$

  where β and α are constants with $\alpha > 1$, the time to failure has Weibull distribution with increasing failure rate, which is commonly used to model aging. To model software systems where only occasional failures are witnessed with no performance degradation, the combination $\mu(\cdot) = \mu$ and $\rho(\cdot) = \rho(t)$ may be used.

Further, to model software systems which undergo performance degradation but are always available, the special case of $\rho(\cdot) = \rho = 0$ and $\mu(\cdot) = \mu(t)$ can be used.

- $\mu(\cdot) = \mu(N(t))$ and $\rho(\cdot) = \rho(N(t))$

  The service rate and the failure rate are functions of instantaneuos load on the system, i.e., their value at time t depends on the number of transactions in the queue at that time. This dependence is useful in capturing overload effects which influence the failure behaviour especially. Of course, more realistic combined dependence on time and instantaneous load, $\mu(\cdot) = \mu(t, N(t))$ and $\rho(\cdot) = \rho(t, N(t))$ is allowed too.

- $\mu(\cdot) = \mu(L(t))$ and $\rho(\cdot) = \rho(L(t))$

  A more complex but also more powerful dependence can be obtained by making $\rho(\cdot)$ and $\mu(\cdot)$ as functions of mean accumulated work done by the software system in a given time interval. Let $p_i(t), 0 \leq i \leq K$ be the probability that there are i transactions in the queue at time t given that the software is in state "A". When the software is not aged, incoming transactions are promptly served and the total amount of time spent in actual processing in an interval $(0, t]$ is usually less than the interval t itself. Since, an idle software is not likely to age, service and failure rates are more realistically a function of the actual processing time rather than the total available time. Let $L(t)$ be defined as:

$$L(t) = \int_{\tau=0}^{t} \sum_{i} c_i p_i(\tau) d\tau$$

  where c_i is a coefficient which expresses how being in state i influences the degradation of the overall system. If $c_0 = 0$ and $c_i = 1$ for $i > 0$ then $L(t)$ represents the average amount of time the software is busy processing transactions in the interval $(0, t]$. if $c_i = 1$ for $i \geq 0$, then $L(t) = t$ given that the software is available. Lastly, our model allows for combination of above dependencies. For example, the failure rate may be a function of not only the mean processeing time in the interval $(0, t]$, but also of the instantaneous laod at time t, In this case, $\rho(\cdot) = \rho(N(t), L(t))$.

In the following section, we derive the three measures for the two PM policies. Table 2 lists the notation used in the rest of the paper. (R.V. denotes random variable):

P_{AB}	Transition probability from state A (Available) to state B (Recovering)
P_{AC}	Transition probability from state A (Available) to state C (Undergoing PM)
$p_i(t)$	Probability that i transactions are queued at time t,
N_l	Number of transactions lost at the end of the available period (R.V.)
γ_f	Expected time to recover from failure
γ_r	Expected time to perform PM
U	Sojourn time in state A (R.V.)
λ	Transaction arrival rate
$\mu(\cdot)$	Transaction service rate
$\rho(\cdot)$	Failure rate
$N(t)$	Number of transaction in the queue at time t
$L(t)$	Mean processing time since the last renewal

Table 2: Adopted notation

4 Model Solution

Let the steady state availability of the software system be denoted by A_{SS}. Let P_{loss} denote the long run probability that an arriving transaction will be lost and let T_res denote the expected response time of a transaction given that it is successfully served. The approach we follow in deriving the expressions for the three measures applies to both policies **I** and **II**. Only when a particular expression is different, will it be noted explicitly. The solution method in general, and the class of stochastic process used to model in particular, provides an elegant, concise and fast alternative to usually expensive discrete-event simulation approach.

As described in the previous section, the software can be in any one of three states at any time t. It can be up and available for service (state A), recovering from a failure (state B) or undergoing PM (state C) (see Figure 1). Let $\{Z(t), t \geq 0\}$ be a stochastic process which represents the state of the software at time t. Further, let a the sequence of random variables $S_i, i > 0$ represent the transition times among different states.

Then, $\{Z(S_i), i > 0\}$ is an embedded discrete time Markov chain (DTMC), since the entrance times S_i constitute renewal points. The transition probability matrix P for this DTMC can be easily derived from the state transition diagram shown in Figure 1 and is given by:

$$P = \begin{pmatrix} 0 & P_{AB} & P_{AC} \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \tag{1}$$

The steady state probability of the software being in state $i, i \in \{A, B, C\}$, denoted by p_i, can be also determined in a straightforward manner from the well know relation $\pi = \pi P$. The the probabilities are given by;

$$\begin{aligned} \pi_A &= \frac{1}{2} \\ \pi_B &= \frac{1}{2} P_{AB} \\ \pi_C &= \frac{1}{2} P_{AC} \end{aligned} \tag{2}$$

The software behavior as a whole is modeled via the stochastic process $\{(Z(t), N(t)), t \geq 0\}$. If $Z(t) = A$, then $N(t) \in \{0, 1, \ldots, K\}$, as the software queue can accommodate up to K transactions. If $Z(t) \in \{B, C\}$, then $N(t) = 0$, since by assumption, all transactions arriving while the software is either recovering or undergoing PM are lost. Further, the transactions already in the queue at the transition instant are also discarded. It can be shown that the process $\{(Z(t), N(t)), t \geq 0\}$ is a Markov regenerative process (MRGP) [6]. The regeneration instants are embedded at times when the process makes transition from state i to state j $(i, j \in \{A, B, C\})$, i.e. $Z(t)$ changes. Trasition to state A from either B or C constitutes a regeneration instant since by assumption, the software is reset to the original initial conditions. At these instants, the system in empty and the software is as good as new. Note that what makes the process an MRGP is the fact that within one regeneration period, the stochastic process changes state. In other words, $N(t)$ assumes different values for some t, during which $Z(t)$ remains in state A. We have already defined and solved the embedded DTMC of this MRGP in Equations (1) and (3) respectively.

The steady state availability can then be obtained using standard formulae from

MRGP theory and is given as:

$$A_{SS} \quad = \quad Pr\{software\ is\ in\ state\ A\} = \frac{\pi_A E[U]}{\pi_B \gamma_f + \pi_C \gamma_r + \pi_A E[U]}$$

where, $E[U]$ is the expected sojourn time in state A. Substituting the values of π_A, π_B and π_C,

$$A_{SS} = \frac{E[U]}{P_{AB}\gamma_f + P_{AC}\gamma_r + E[U]} \tag{3}$$

The probability that a transaction is lost is defined as the ratio of expected number of transactions which are lost in an interval to the expected total number of transactions which arrive during that interval. Since the process is stochastically identical in successive visits to state A, it suffices to consider one interval only. The expected number of transactions lost is given by summation of three quantities; (1) the expected number lost due to discarding because of failure or initiation of PM, (2) the expected number lost while recovery or PM is in progress and (3) the expected number lost due to the buffer being full. The last quantity is of special significance as due to the degrading service rate, the probability of buffer being full if higher.

The probability of loss is then given by:

$$\begin{aligned} P_{loss} \quad &= \quad \frac{\pi_A E[N_l] + \lambda \left(\pi_B \gamma_f + \pi_C \gamma_r + \pi_A \int_0^\infty p_K(t)dt \right)}{\lambda \left(\pi_B \gamma_f + \pi_C \gamma_r + \pi_A E[U] \right)} \\ &= \quad \frac{E[N_l] + \lambda \left(P_{AB}\gamma_f + P_{AC}\gamma_r + \int_0^\infty p_K(t)dt \right)}{\lambda \left(P_{AB}\gamma_f + P_{AC}\gamma_r + E[U] \right)} \end{aligned} \tag{4}$$

where:

- $E[N_l]$ is the expected number of transactions in the buffer when the system is exiting state A;
- $\lambda\gamma_f$ is the expected number of transactions arriving while the system is recovering;
- $\lambda\gamma_r$ is the expected number of transactions arriving while the system is undergoing PM;
- $\lambda \int_0^\infty p_K(t)dt$ is the expected number of transactions denied service because of buffer full while the system is in state A;

- $(P_{AB}\gamma_f + P_{AC}\gamma_r + E[U])$ is the average length of time between two consecutive visits to state A.

Equation 4 is valid only for policy **II**. Under policy **I** the upper limit in the integral $\int_0^\infty p_K(t)dt$ is δ instead of ∞. This is because, the sojourn time in state A is limited by δ under policy **I**.

We now derive an upper bound on the mean response time of a transaction successfully served, denoted by T_res. The mean number of transactions, denoted by E, which are accepted for service while the software is in state A is given by the mean total number of transactions which arrive while the software is in state A minus the mean number of transations which are not accepted due to the buffer being full. That is,

$$E = \lambda(E[U] - \int_{t=0}^{\infty} p_K(t)dt)$$

Out of these transactions, on the average, $E[N_l]$ are discarded later because of failure and initiation of PM. The mean total time the transactions spent in the system while the software is in state A is:

$$W = \int_{t=0}^{\infty} \sum_i i p_i(t)\ dt$$

This time is composed of the mean time spent by the transactions which were served as well as those which were discarded, denoted as W_S and W_D, respectively; Therefore, $W = W_S + W_D$. The response time we are interested in is given by

$$T_{res} = \frac{W_S}{E - E[N_l]}\ .$$

which is upper bounded by[4]

$$T_{res} < \frac{W}{E - E[N_l]}\ . \tag{5}$$

Regardless of the PM policy, as can be observed from equations 3 and 4, we need to obtain expected sojourn times and the steady state probability of the software in each of the three states A, B and C, as well as the transient probability that there

[4] $\frac{W}{E-E[N_l]}$ tends to $\frac{W_S}{E-E[N_l]}$ as the loss ratio of the customers entered into the system (i.e. $\frac{E[N_l]}{E}$) tends to 0.

are $i, i = 0, 1, \ldots, K$ transactions queued up for service. It is the last quantity which forbids a closed form analytical solution and necessitates a numerical approach.

The mean sojourn time in states B and C is already available as γ_f and γ_r respectively[5]. The quantities still to be derived are related to the behavior of the software in state A, viz., P_{AB}, P_{AC}, $E[U]$ and $p_i(t), i = 0, 1, \ldots, K$ and their evaluation depends on the considered system and the policy used.

4.1 Behavior of the system in state A under Policy I

For $Z(t) = A$, the subordinated process, i.e., the process until a regeneration occurs, is determined by the queuing behavior of the software processing transactions. The process is terminated by either a failure (which can happen at any time) or by initiating PM which under policy **I** happens at time δ if the software has not failed by that time. Figure 3 shows the state diagram of the subordinated non-homogeneous process under accumulated load dependent system degradation and policy **I**. It is a birth-death process augmented with one absorbing state associated with each state of the birth-death process. Not included in the figure is the fact that at $t = \delta$, the subordinated process is terminated if it was not terminated before by a transition to an absorbing state $(0', \ldots, K')$. We have chosen $\mu(\cdot) = \mu(L(t))$ and $\rho(\cdot) = \rho(L(t)$ to develop the solution. Other definitions can be directly substituted in the model. In the subsequent analytical treatment we assume that the degradation is proportional to the mean accumulated load, hence

$$L(t) = \int_{\tau=0}^{t} \sum_i c_i p_i(\tau) d\tau \ .$$

By our notation, $p_i(t)$ is the probability that there are i transactions queued for service, which is also the probability of being in state i of the subordinated process at time t. Note that state $i, i = 0, 1, \ldots, K$ is not to be confused with state $i', i = 0, 1, \ldots, K$ which was defined just to be able to evaluate the quantities of interest. As such, all the states under the shaded area of the process can be lumped into a single absorbing state.

[5]The measures evaluated in this paper require only the first moments of Y_f and Y_r and hence no assumptions on the nature of their distribution is made.

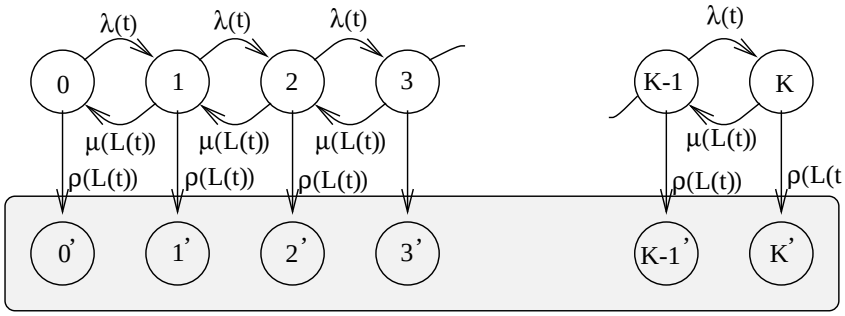

Figure 3: Subordinated Non-homogeneous CTMC for $t \leq \delta$

$p_i(t), i = 0, 1, \ldots, K$ and $p_{i'}(t), i = 0, 1, \ldots, K$ can be obtained by solving the forward differential equations given as:

$$\begin{aligned}
\frac{dL(t)}{dt} &= \sum_i c_i p_i(t) \\
\frac{dp_0(t)}{dt} &= \mu(L(t))p_1(t) - (\lambda + \rho(L(t)))p_0(t) \\
\frac{dp_i(t)}{dt} &= \mu(L(t))p_{i+1}(t) + \lambda p_{i-1}(t) - (\mu(L(t)) + \lambda + \rho(L(t)))p_i(t), 1 \leq i < K \\
\frac{dp_K(t)}{dt} &= \lambda p_{K-1}(t) - (\mu(L(t)) + \rho(L(t)))p_K(t) \qquad (6) \\
\frac{dp_{i'}(t)}{dt} &= \rho(L(t))p_i(t), 0 \leq i \leq K
\end{aligned}$$

The set of simultaneous differential-difference equations given by 6 do not have a tractable analytical solution and must be evaluated numerically along with the initial conditions $p_0(0) = 1, p_i(0) = 0, 1 \leq i \leq K$ and $p_{i'}(0) = 0, 0 \leq i \leq K$. Once these probabilities are obtained, the rest of the quantities can be computed as follows.

One step transition probability P_{AB} is given by:

$$P_{AB} = \sum_{i=0'}^{K'} p_i(\delta)$$

and

$$P_{AC} = 1 - P_{AB}$$

Thereafter, according to equation 3, the steady state probability that the software is in states B and C can be obtained.

The expected sojourn time in state A is given as:

$$E[U] = \int_{t=0}^{\delta} \left(\sum_{i=0}^{K} p_i(t) \right) dt$$

where the upper limit on the integral indicates that the sojourn time is bounded by δ. The average value $E[N_l]$ of customers already in the system at the time when state A is left, is evaluated as:

$$E[N_l] = \sum_{i=0}^{K} i \left(p_i(\delta) + p_{i'}(\delta) \right)$$

A_{SS}, P_{loss} and the upper bound on T_res as given in Equations 3 4 and 5 respectively can now be easily calculated.

4.2 Behavior of the system in state A Policy II

If Policy II is assumed, the evolution of the system in macro-state A is somewhat more complex. In this case, in fact, we need to distinguish between $t \leq \delta$ and $t > \delta$, as Policy II assumes that PM will be initiated if and only if the buffer is empty after δ has elapsed. For $t \leq \delta$, exactly the same process of Figure 3 determines the behavior of the software. For $t > \delta$ the process which models the behavior is shown in Figure 4. As can be observed, the state 0 now belongs to the set of absorbing states because PM will be initiated, thus terminating the subordinated process, once the system processes all the transactions in the queue.

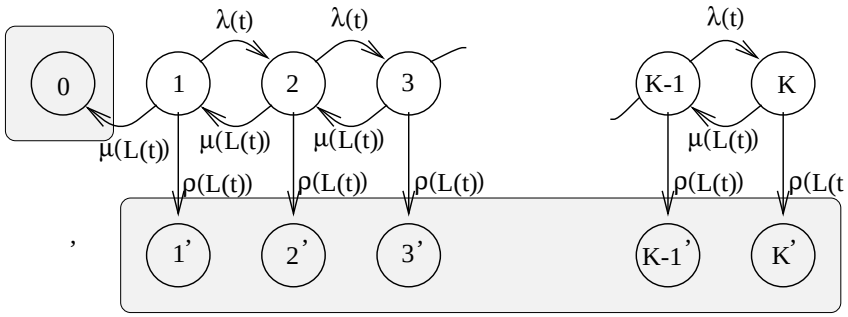

Figure 4: Subordinated Non-homogeneous CTMC if $t > \delta$

The forward differential equations which are used to determine all transient prob-

abilities are given as follows:

$$\begin{aligned}
\frac{dL(t)}{dt} &= \sum_i c_i p_i(t) \\
\frac{dp_0(t)}{dt} &= \mu(L(t))p_1(t) - \left(\lambda'(t) + \rho(L(t))\right) p_0(t) \\
\frac{dp_1(t)}{dt} &= \mu(L(t))p_2(t) + \lambda'(t)p_0(t) - (\mu(L(t)) + \lambda + \rho(L(t)))p_1(t), \\
\frac{dp_i(t)}{dt} &= \mu(L(t))p_{i+1}(t) + \lambda p_{i-1}(t) - (\mu(L(t)) + \lambda + \rho(L(t)))p_i(t), 2 \leq i < K \\
\frac{dp_K(t)}{dt} &= \lambda p_{K-1}(t) - \left(\mu(L(t)) + \rho(L(t))\right) p_K(t) \qquad (7) \\
\frac{dp_{0'}(t)}{dt} &= \rho'(L(t))p_0(t) \\
\frac{dp_{i'}(t)}{dt} &= \rho(L(t))p_i(t), 1 \leq i \leq K
\end{aligned}$$

where $\lambda'(t) = \lambda, t \leq \delta$, otherwise it is zero. Similarly, $\rho'(L(t)) = \rho(L(t)), t \leq \delta$, otherwise zero. As before, this set of differential-difference equations along with the initial condition that $p_0(0) = 1$ requires numerical solution.

The quantities of interest can then be evaluated using similar expressions as derived for policy **I**. Minor differences arise, as described now. Transient state probabilities P_{AB} and P_{AC} are evaluated at $t = \infty$ and are given as:

$$P_{AB} = \sum_{i=0'}^{K'} p_i(\infty)$$

and

$$P_{AC} = 1 - P_{AB} = p_0(\infty).$$

The mean sojourn time in state A is now given by:

$$\begin{aligned}
E[U] &= \int_{t=0}^{\delta} \left(\sum_{i=0}^{K} p_i(t) \right) dt + \int_{t=\delta}^{\infty} \left(\sum_{i=1}^{K} p_i(t) \right) dt \\
&= \int_{t=0}^{\delta} p_0(t) dt + \int_{t=0}^{\infty} \left(\sum_{i=1}^{K} p_i(t) \right) dt
\end{aligned}$$

And the computation of $E[N_l]$ is as follows:

$$E[N_l] = \sum_{i=0}^{K} i\, p_{i'}(\infty).$$

Using equations 3, 4 and 5, the steady state availability, the probability of loss of an arriving transaction and the upper bound on the response time of a transaction can now be calculated.

5 Numerical Illustration

In this Section, we illustrate the usefulness of the models developed to evaluate the steady state availability (A_{SS}), the probability that a transaction is lost (P_{loss}) and the upper bound on the response time T_{res}. The models are solved for multiple values of δ (rejuvenation interval in the case of policy **I** and rejuvenation wait in the case of policy **II**) and optimal values are determined. We also show, how the optimum value of δ may be selected based on combined measures of the above three quantities.

Table 3 shows the parameter values that were kept fixed for all results. The values chosen are for illustration purposes only and do not necessarily represent any physical system.

γ_f	0.85 $(hours)$
λ	6.0 $(hours^{-1})$
K	50
$MTTF$	240 $(hours)$

Table 3: Model parameters

The set of differential-difference equations given in Equations 6 and 8 for policies **I** and **II** respectively were numerically solved using LSODE routine in ANSI FORTRAN. LSODE (Livermore Solver for Differential Equations) is an ODE solver which uses Backward differentiation Formula (BDF) methots for stiff systems of ODEs. LSODE is publicly available as part of ODEPACK from netlib [**?**]. The solution takes less than a minute for $K = 50$. Solution using commercially available packages like Mathematica or MATLAB is also possible, but is likely to be much slower. For large buffer sizes of the order of thousands, sparse matrix methods will need to be used in the ODE solution.

5.1 Experiment 1

. In this experiment, γ_r was varied to ascertain the effect on the measures and on optimal δ. Service rate and failure rate are assumed to be functions of real time, i.e., $\mu(\cdot) = \mu(t)$ and $\rho(\cdot) = \rho(t)$, where $\rho(t)$ is defined to be

$$\rho(t) = \beta\alpha t^{\alpha-1},$$

which is the hazard function of Weibull distribution. α was fixed at 1.5 and β was calculated from the $MTTF$ and α as

$$\beta = \left[\frac{\Gamma(1+\frac{1}{\alpha})}{MTTF}^{\alpha}\right] \tag{8}$$

The model was solved for both policies to show the effect of the cost of PM on the three measures. $\mu(t)$ was defined to be a monotone non-increasing function of t as shown in Figure 5. This behaviour of $\mu(t)$ has been given in [3] as an approximation to service degradation in telecommunications switching software.

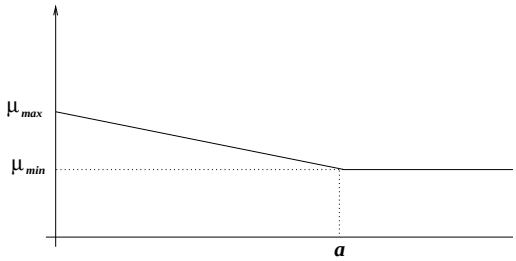

Figure 5: Time variation of the service rate $\mu(t)$

Further, the rejuvenation interval (wait) δ is varied between 1 and 400 hours. The expected time to recover will be assumed to be equal to 0.15 and 0.85, while the expected time to rejuvenate is assumed equal to 0.15 (hours) in all the experiments. The arrival rate is fixed as well ($\lambda = 6\ arrivals\ per\ hours$) and we assume a buffer size of 20. Figure 5 plots the function $\mu(t)$, which is an approximation to what is witnessed in reality [3]. μ_{max} is fixed at 15 $hour^{-1}$ and μ_{min} at 5 $hour^{-1}$ for all the experiments in the paper. For this particular experiment of varying γ_r, $\mu(t)$ (as shown in Figure 5) is defined as:

$$\mu(t) = \begin{cases} \mu_{max}\left[1 - \frac{t}{MTTF}\right], & \text{if } t \leq a \\ \mu_{min}, & \text{if } t > a \end{cases}$$

, where

$$a = \frac{(\mu_{max} - \mu_{min})}{\mu_{max}} MTTF.$$

The use of common parameter $MTTF$ in the definitions of $\mu(t)$ and $\rho(t)$ is simply to illustrate how dependence in the service and failure behaviour can be captured, even though stochastically the two processes are assumed to be independent.

In the numerical evaluation of the measures, evaluation at time ∞ is required which is approximated by respective values at time t_{MAX} where t_{MAX} is associated with the required precision parameter (ϵ) by the following expression:

$$F_X(t_{MAX}) = \int_0^{t_{MAX}} \sum_i p_i(t)\, dt = 1 - \epsilon.$$

Figure 6(a) shows A_{ss} plotted against different values of δ for both PM policies **I** and **II**.

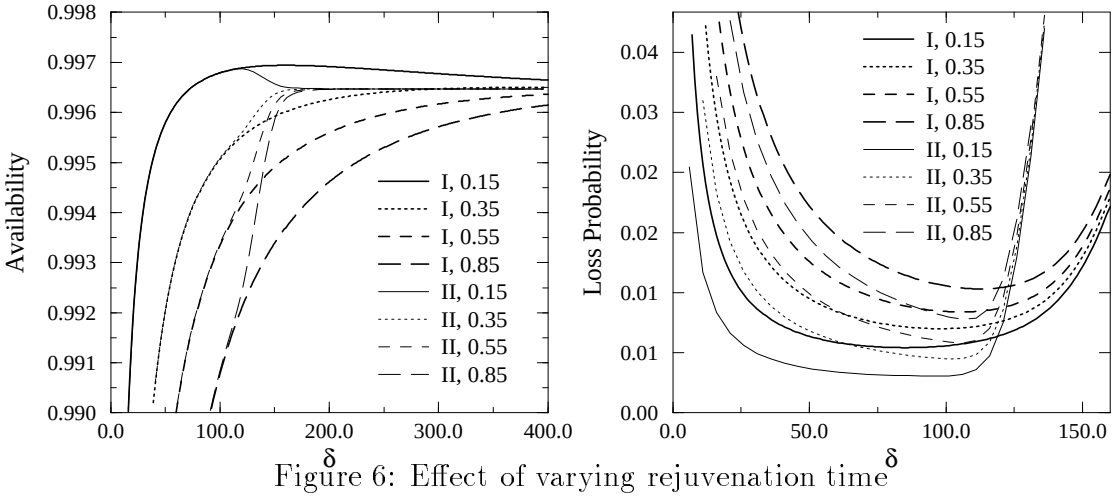

Figure 6: Effect of varying rejuvenation time

Further, for each policy, γ_r was assigned values of 0.15, 0.35, 0.55 and 0.85 hours. As noted already, the expected downtime due to a failure is kept fixed at 0.85 hours. Under both policies, it can be seen that higher the value of γ_r, lower is the availability for any particular value of δ. Under policy **I**, for $\gamma_r = 0.15$ and $\gamma_r = 0.35$, the availability rapidly increases with increase in δ (that is PM is performed less frequently), attains a maximum at $\delta = 160$ and $\delta = 410$ respectively and then gradually decreases. For $\gamma_r = 0.55$ and $\gamma_r = 0.85$, the steady state availabilty turned out to be a monotone function. In all cases, A_{ss} eventually approaches the same value with increase in δ

which corrosponds to the steady state availability if no PM was performed. Therfore, for $\gamma_r = 0.55$ and $\gamma_r = 0.85$, it is better not to perform PM, if the objective is only to maximize availability. Under policy **II**, the steady state availability follows the same behaviour as in policy **I** except that the value of A_{ss} corresponding to the no PM case is reached at much lesser values of δ, which now represents PM wait rather than the PM interval.

Figure 6(b) shows the plots of P_{loss} against δ for both policies with γ_r being assigned values as in Figure 6(a). All the plots attain a minimum. As expected, for any specific value of δ and a specific policy, higher is the value of γ_r, higher is the corrosponding loss probability, because on average, more transactions are denied service while PM is in progress. Since the absolute values of the measures or of optimal δ are not of importance, we shall comment on the relative effects only. It can be seen that for any specific policy, lower the value of γ_r, lower is the value of δ which minimizes the probability of loss for that particular γ_r. For any specific value of γ_r, policy **II** results in a lower minima in loss probability than that achieved under policy **I**. Moreover this minima under policy **II** is achieved at a lower δ as compared to policy **I**. This clearly shows that if the objective is to minimize long run probability of loss, which is the case is telecommunication switching software, policy **II** always fares better than policy **I**. It can also be observed from Figure 6(a) and 6(b), that the δ value which minimizes probability of loss is much lower than the one which maximizes availability. In fact, the probability of loss becomes very high at δ values which maximize availability. Although the behaviour is dependent on system parameter values, caution in proper selection of δ is indicated.

5.2 Experiment 2

In this experiment, γ_r is fixed at 0.15. $\mu(\cdot) = \mu(t)$ and has exactly the same definition as in Experiment 1. $\rho(\cdot) = \rho(t)$, with previously defined Weibull hazard rate, except that the shape parameter α is now varied. Thus for each value of α, corresponding β is calculated using Equation 8 keeping $MTTF$ fixed at 240 hours. In effect, we are interested in studying how the measures and the optimality varies as the failure density

gets peakier with the same mean time to failure. α is assigned values of 1.0, 1.5 and 2.0 respectively.

Figure 7(a) shows the steady state availability under the two policies plotted against δ for different values of α.

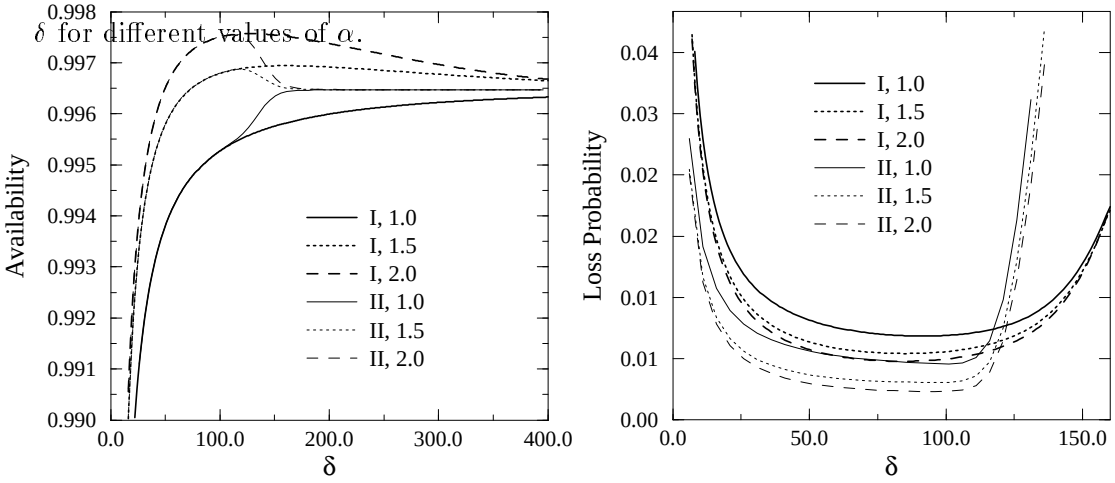

Figure 7: Effect of Varying Weibull Parameter

For $\alpha = 1.0$, the time to failure has an exponential distribution, which, because of its memoryless property, contradicts aging. As seen from the figure, it is better not to perform PM in this case, if the objective is to maximize availability. For the other two values of α, however, PM maximizes availability at certain δ. For a specific policy, peakier the failure density, i.e., higher the value of α, higher is the maximum steady stater availability. Secondly, this maxima occurs at lower δ with higher values of α. Figure 7(b) shows the long run probability of loss of a transaction plotted against δ. In this case, PM proves to be beneficial for all three values of α. Similar observations and arguments as those given in Experiment 1 hold for this case also.

5.3 Experiment 3

The purpose of this experiment is to illustrate the effect of assumptions on $\mu(\cdot)$ and $\rho(\cdot)$ on the three measures. Figure 8(a), (b) and (c) show steady state availability, probability of loss and the upper bound on the mean response time of transactions successfully served plotted against δ under policy **I**.

Each of the figures contains three curves. The solid curve represents a system where $\mu(\cdot) = \mu(t)$ and $\rho(\cdot) = \rho(t)$. The dotted curve represents a second system where

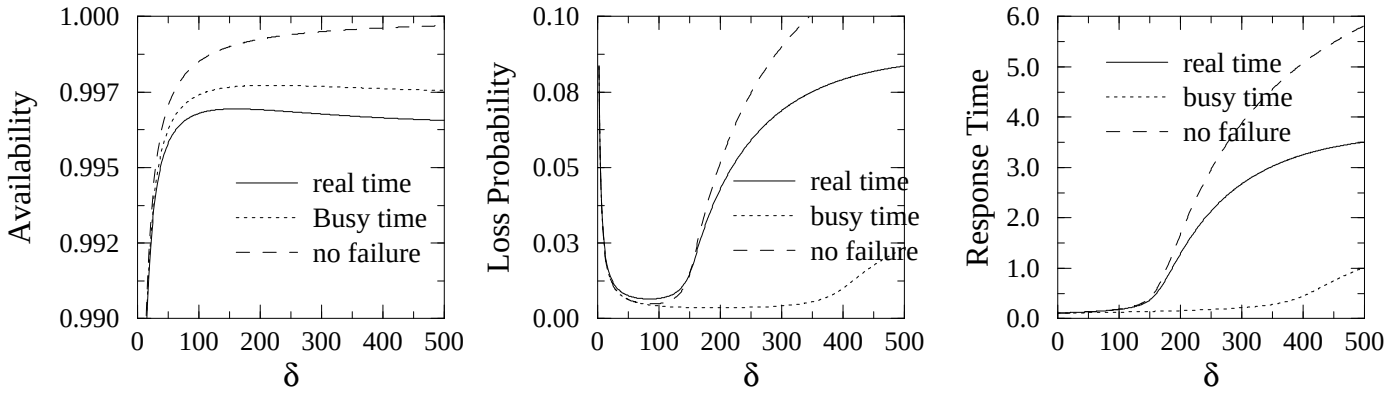

Figure 8: Effect of System assumptions

$\mu(\cdot) = \mu(L(t))$ and $\rho(\cdot) = \rho(L(t))$. The parameters, μ_{max}, μ_{min} and a are kept the same. Similarly, α is kept at 1.5 for both the solid as well as the dotted curve. In other words, $\mu(\cdot)$ and $\rho(\cdot)$ in the solid curve are functions of real time, whereas in the dotted curve, they are functions (with the same parameters) of the mean total processing time. The dashed curve represents a third system in which no crash/hang failures occur, i.e., $\rho(\cdot) = 0$ but service degradation is present with $\mu(\cdot) = \mu(t)$ with same parameters as for $\mu(\cdot)$ of earlier two systems. γ_r for all is kept fixed at 0.15, This experiment only illustrates the importance of making the right assumptions in capturing aging because as seen from the figure, depending on whether in the load dependence via $L(t)$ is captured in the model, the results vary a lot.

Figure 8 plots the upper bound on the mean response time. This was not shown for Experiments 1. and 2. because of its monotone nature. In the type of system under consideration, where queued transactions as well as those arriving during recovery or PM are lost, response time of successful transactions can be trivially minimized by always keeping the software unavailable (with very low δ). This however, will result in unacceptable values of the probability of loss and steady state availability. In many systems, for example ATM switches, QOS requirements are specified using bounds on response time as well as probability of loss. This can be achieved via our model. For example, consider the solid curve in Figure 8(c). If QOS demands that the response time be less than 0.113 hours, δ is restricted approximately in the interval $(0, 70]$. Now, if the QOS further demands that the probability of loss be minimized, the optimal δ corresponds to the minimum P_{loss} within this interval only. The curve identified with

the legend (I,1.5), in Figure 7(b) plots P_{loss} against δ with exactly the same parameter values and it is seen that global minima for P_{loss} occurs at $\delta = 85.$ and the optimal δ for the combined QOS is 70.

6 Conclusion

In this paper, we motivated the need of persuing preventive maintenance in operational software systems on a scientific analytical basis rather than the current ad-hoc practice. We presented a model for a transactions based software system which employs preventive maintenance to either maximize availability, minimize probability of loss, minimize response time or optimize a combined measure. We evaluated the three measures for two different preventive maintenance policies and showed via a numerical example that a policy which takes into account instantaneous load on the system results in lesser optimum probability of loss. The effect of aging is captured as crash/hang failures as well as performance degradation. Systems which experience only one of the two can be modeled as special cases. The main strength of our model, however, is its capability of capturing the dependence of crach/hang failures and performance degradation on time, instantaneous load, mean accumulated load or a combination. This, in our opinion, provides great flexibility in modeling the real situation and enhances the scope of applicability of our model. The main limitation, on the other hand, is that it is applicable to only those software systems in which incoming transactions are lost when either it fails or when PM in initiated. In many database systems which support recovery, transactions are logged when they arrive. Even when failure occurs, the log is not lost, thus violating our assumption.

References

[1] E. Adams, "Optimizing preventive service of the software products", *IBM J. R&D*, 28(1), Jan, 1984, pp. 2-14.

[2] A. Avizienis, "The n-verion approach to fault-tolerant software", *IEEE Trans. on Software Engg.*, Vol. SE-11, No. 12, pp. 1491-1501, December 1985.

[3] A. Avritzer and E. J. Weyuker, "Monitoring smoothly degrading systems for increased dependability", submitted for publication.

[4] L. Bernstein, Text of seminar delivered by Mr. Bernstein, University Learning Center, George Mason University, January 29, 1996.

[5] R. Chillarege, S. Biyani and J. Rosenthal, "Measurements of failure rate in commercial software", in *Proc. of 25th Symposium on Fault Tolerant Computing*, June, 1995.

[6] E. Cinlar, "Introduction to Stochastic Processes", Prentice-Hall, Englewood Cliffs, 1975.

[7] G. F. Clement and P. K. Giloth, "Evolution of fault tolerant switching systems in AT&T", *The Evolution of Fault-Tolerant Computing*, Dependable Computing and Fault-Tolerant Systems Vol. 1, A. Avizienis, H. Kopetz, J. C. Laprie, eds., Springer-Verlag, pp. 37-53, 1987.

[8] S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi, "Analysis of software rejuvenation using Markov regenerative stochastic Petri net", *Proc. 6th Intnl. Symposium on Software Reliability Engineering*, 24-27 October, 1995, Toulouse, France.

[9] S. Garg, Y. Huang, C. Kintala and K.S. Trivedi, "Time and load based software rejuvenation: policy, evaluation and optimality", *Proc. First Fault-tolerant Symposium*, Dec. 22-25, Madras, India, 1995.

[10] S. Garg, Y. Huang and C. Kintala, K.S. Trivedi, "Minimizing Completion Time of a Program by Checkpointing and Rejuvenation", *Proc. 1996 ACM SIGMETRICS Conference,* Philadelphia, PA, pp. 252-261, May 1996.

[11] J. Gray and D. P. Siewiorek, "High-availability computer systems", *IEEE Computer Mag.*, pp. 39-48, Sept. 1991.

[12] J. Gray, "Why do computers stop and what can be done about it?", *Proc. of 5th Symp. on Reliability in Distributed Software and Database Systems*, pp. 3-12, January 1986.

[13] J. Gray, "A census of tandem system availability between 1985 and 1990", *IEEE Trans. on Reliability*, Vol. 39, pp. 409-418, Oct. 1990.

[14] B. O. A. Grey, "Making SDI software reliable through fault-tolerant techniques" *Defense Electronics*, August, 1987, pp. 77-80, 85-86.

[15] Y. Huang, P. Jalote and C. Kintala, "Two techniques for transient software error recovery", *Lecture Notes in Computer Science*, Springer Verlag, pp. 159-170, Vol 774, 1994.

[16] Y. Huang, C. Kintala, N. Kolettis and N. D. Fulton, "Software rejuvenation: Analysis, Module and Applications", *Proc. of 25th Symposium on Fault Tolerant Computing*, Pasadena, California, June, 1995.

[17] R. K. Iyer and I. Lee, "Software fault tolerance in computer operating systems", *Software Fault Tolerance*, M. R. Lyu (Ed.), John, Wiley and Sons Ltd., 1995.

[18] P. Jalote, Y. Huang and C. Kintala, "A framework for understanding and handling transient software failures", *In Proc. of 2nd ISSAT Intnl. Conf. on Reliability and Quality in Design*, Orlando, Florida, 1995

[19] J. C. Laprie, J. Arlat, C. B'eounes, K. Kanoun and C. Hourtolle, "Hardware and software fault tolerance: definition and analysis of architectural solutions", In *Digest of 17th FTCS*, pp. 116-121, Pittsburgh, PA, 1987.

[20] J-C. Laprie, J. Arlat, C. Béounes and K. Kanoun, "Architectural issues in software fault-tolerance", *Software Fault Tolerance*, Ed. M. R. Lyu, John, Wiley & sons. ltd., pp. 47-80, 1995.

[21] E. Marshall, "Fatal error: how Patriot overlooked a Scud", *Science*, March 13, 1992, page 1347.

[22] A. Pfening, S. Garg, M. Telek, A. Puliafito and K. S. Trivedi, "Optimal rejuvenation for tolerating soft failures", *Performance Evaluation*, Vol. 27 & 28, October 1996, North-Holland, pp. 491-506.

[23] B. Randell, "System structure for software fault tolerance", *IEEE Trans. on Software Engg.*, Vol. SE-1, pp. 220-232, June 1975.

[24] M. Sullivan and R. Chillarege, ' 'Software defects and their impact on system availability - A study of field failures in operating systems", in *Proc. IEEE Fault-Tolerant Computing Symposium*, pp. 2-9, 1991.

[25] J. J. Stiffler, "Fault-tolerant architectures – Past, present and future", *Lecture Notes in Computer Science*, Springer Verlag, Berlin, pp. 117-121, Vol 774, 1994.

[26] A. Tai, S. N. Chau, L. Alkalaj and H. Hecht, "On-board preventive maintenance: Analysis of effectiveness and optimal duty period", To appear in *3rd International Workshop on Object-oriented Real-time Dependable Systems*, California, February 97.

[27] Y. M. Wang, Y. Huang and W. K. Fuchs, "Progressive retry for software error recovery in distributed systems", in *Proc. IEEE Fault Tolerant Computing Symposium*, pp. 138-144, June 1993.