On the Analysis of Software Rejuvenation Policies

Sachin Garg[1], Antonio Puliafito[2], Miklos 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

[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

Software rejuvenation is a technique for software fault tolerance which involves occasionally stopping the executing software, "cleaning" the "internal state" and restarting. This cleanup is done at desirable times during execution on a preventive basis so that unplanned failures, which result in higher costs compared to planned stopping, are avoided. Since during rejuvenation, the software is typically unavailable or in a degraded mode of operation, the operation involves a cost. The necessity to use this technique not only in general purpose computers but also in safety-critical and high availability systems clearly indicates the need of analysis in order to determine the optimal times to rejuvenate.

In this paper, we present an analytical model of a software system which services transactions. Due to "aging", not only the service rate of the software decreases with time but the software itself experiences occasional crash/hang failures. We propose and compare two rejuvenation policies. First policy is purely time dependent while the second also takes into account the number of transactions currently queued for service. The policies are evaluated for the resulting steady state availability as well the probability that a transaction is denied service. We also numerically illustrate the use of our model to compute the optimal rejuvenation interval which manimizes (maximizes) the loss probability (steady state availability).

Keywords: Transaction oriented software systems, Software rejuvenation, Markov Regenerative Stochastic models

1 Introduction

It is now well established that system failures due to imperfect software behavior are usually more frequent than failures caused by hardware components faults [?, ?]. It is also well known that software, regardless of development, testing and debugging time, contains some residual faults [?]. Thus, fault tolerant software has become an effective alternative to virtually impossible fault-free software, at least in safety-critical, high availability applications. Traditional methods of software fault-tolerance, such as N-version programming [?, ?], recovery blocks [?] and N-self checking programming [?] are all based on design diversity. Furthermore, they are all reactive in nature, i.e., the fault tolerance mechanism comes into effect after at least one version has experienced failure.

More recently, from the study of field failure data, it has been observed that a large percentage of software failures are transient in nature [?, ?], caused by phenomena such as overloads or faulty exception handling [?]. A common characteristic of these type of failures is that upon re-execution of the software, the failure is not likely to occur again. 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, it has been observed that the manifestation of the same error condition, when the software is re-executed, is very unlikely thus avoiding the failure. This observation has led to alternative software fault-tolerance techniques such as progressive retry [?]. These techniques based on environment diversity are quite inexpensive compared to design diversity approaches and yet have the potential to provide required reliability/availability.

Further, while monitoring real applications an interesting 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. Memory bloating, unreleased file-locks, data corruption are some typical causes of slow degradation. Huang et. al. report this

phenomenon in general purpose applications [?], where a Unix process results in a crash or a hang failure. Avritzer and Weyuker have witnessed aging in telecommunications' switching software where the effect manifests as degrading performance [1]. The service rate of the software keeps decreasing with time and eventually it starts losing packets. Perhaps the most vivid example of aging can be found in [?], where the failure resulted in loss of human life. Patriot missiles, used during the Gulf was to destroy Iraq's Scud missiles used a computer whose software accumulated errors. 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 death of twenty-eight U.S. soldiers.

It is to counteract this phenomenon of aging that Huang et. al. [4] have proposed the technique of "Software Rejuveantion". It is proactive in nature and 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 migh involve. 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, rejuvenation has been used in high-availability systems such as large telecommunications software [1], where the switching computer is rebooted occasinally upon which the service rate is restored to its peak value. It is also used implicitly in fault tolerant operating system [5], where the process is restarted on a lossely coupled redundant processor. In a safety critical environment also, the necessity of performing rejuvenation is evident from the example in [6]. 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.*

Since during rejuvenation, the software is typically unavailable or in a degraded mode of operation, the operation involves a cost. The use of this technique in safety-critical/high availability systems clearly indicates the need of analysis in order to determine the optimal times to rejuvenate. Prevous work in analysis of software rejuvenation was started

with a continuous time Markov chain model proposed by [4] to determine if rejuvenation is beneficial for systems which experience hang/crash failures. It was improved improved upon by allowing deterministic rejuvenation time and provided a closed form expression for the optimal rejuvenation interval which maximizes availability [2]. Avritzer and Weyuker, collected traffic data on the experimental system and proposed heuristics on good times to rejuvenate [1]. In [**?**], a Markov decision process (MDP) based framework to deal with the problem of determining optimal times to rejuvenate was developed assuming that the software experiences degradation in the service rate.

In this paper, we develop and evaluate a non-Markovian model of a transaction oriented software where the effect of aging manifests as decreasing service rate as well as occasional crash/hang failure. This combination has not been considered by any of the previous model. We propose and compare two rejuvenation policies. First policy is purely time dependent while the second also takes into account the number of transactions currently queued for service. The policies are evaluated for the resulting steady state availability as well the probability that a transaction is denied service. We also numerically illustrate the use of our model to compute the optimal rejuvenation interval which manimizes (maximizes) the loss probability (steady state availability).

The rest of the paper deals with the proposed model, its solution and numerical example and is organized as follows. Section 2 describes the behaviour of the software system being modeled and introduces the rejuvenation policies. The desired measures of steady state probability and loss probability are derived in Section 3. In Section 4, the use of the models in comparing the two policies and in finding optimal rejuvenation interval is illustrated via a numerical example. The paper is concluded in Section 5.

2 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 monotone non-increasing function of time (because of aging) denoted by

$\mu(t)$. Therefore, a transaction which starts service at time t_1, occupies the server for a time whose distribution is given by $1 - e^{-\int_{t_1}^{t} \mu(t)\,dt}$. If the software is busy processing a transaction, arriving customers are queued. Total number of transactions that the software can accomodate 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 at decreasing service rate) is denoted as state "1" (see Figure 1).

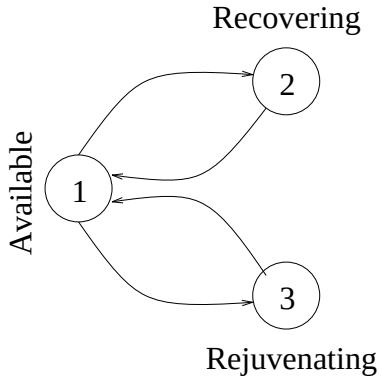

Figure 1: Macro-states representation of the software behavior

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 "2". The rate at which it fails, i.e., at which the software moves from state 1 to state 2 is a monotone non-decreasing funtion of time and is denoted by $\rho(t)$. Let the time to failure be denoted by random variable X. Then, its distribution is given by

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

The effect of aging, therefore, is captured by using decreasing service rate and increasing failure rate, which to the best of our knowledge considered for the first time. This is, however, reasonable as both performance degradation and unavailability are characteristics of a complex transaction oriented software, even though one may be more dominant than the other. The service degradation and crash failures in our model are assumed to be stochastically independent processes. The dependence (in reality) can be approximated by using parameteric dependence in the definitions of $\rho(t)$ and $\mu(t)$. The failure process is

independent of the arrival also and any transactions that are queued at the time of failure are lost. Moreover, any transactions which arrive during recovery are also lost. Time to recover from a failed state is denoted by Y_f with associated distribution F_{Y_f}.

Lastly, the software is occasionally rejuvenated. This state is denoted as state "3" and the software is allowed to rejuvenate only from state 1. We consider two different policies which determine the time to rejuvenate.

1. *Purely time based.* The software is rejuvenated after a constant time δ_I has elapsed since it was started (or restart) referred to as **Policy I**. We shall refer to δ under this policy as the rejuvenation interval.

2. *Load and time based.* A waiting period of δ_{II} must elapse before rejuvenation is attempted. Further, after this time, the software is rejuvenated if and only if there are no transations in the system. This policy shall be referred to as **Policy II** and *delta* under this policy as rejuvenation wait. Note that actual rejuvenation interval under policy **II** is determined by sum of rejuvenation wait and the time it takes for the queue to get empty. The latter quantity can be potentially infinite.

Regardless of the policy used, it takes a random amount of time, denoted by Y_r, to rejuvenate the software. Let F_{Y_r} be its distribution. As will be showed in the following Section, our model does not require any assumption on the nature of F_{Y_f} and F_{Y_r}, and only the expectation $\gamma_f = E[Y_f]$ and $\gamma_r = E[Y_r]$ of the two random variables is required.

Once recovery from the failed state or rejuvenation is complete, the software is reset in state "1" and is as good as new. From this moment on, the whole process stochastically repeats itself. The transition behavior of the software among states 1, 2 and 3 is illustrated in Figure 1

The queueing behaviour of the software, on the other hand, as determined by the two rejuvenation 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 rejuvenation occurs as soon constant

time δ elapses. In accordance with Policy **I**, the software is idle at time instant δ and, even if $N(t) > 0$, the software is rejuvenated and all the transactions already in the queue at time δ are lost.

Figure 2(b) illustrates Policy **II** where at time δ, transactions are queued to receive service (including the transaction being processed), i.e., $N(t) > 0$. In this case, the software waits till the queue is empty upon which it is rejuvenated[1].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 be rejuvenated.

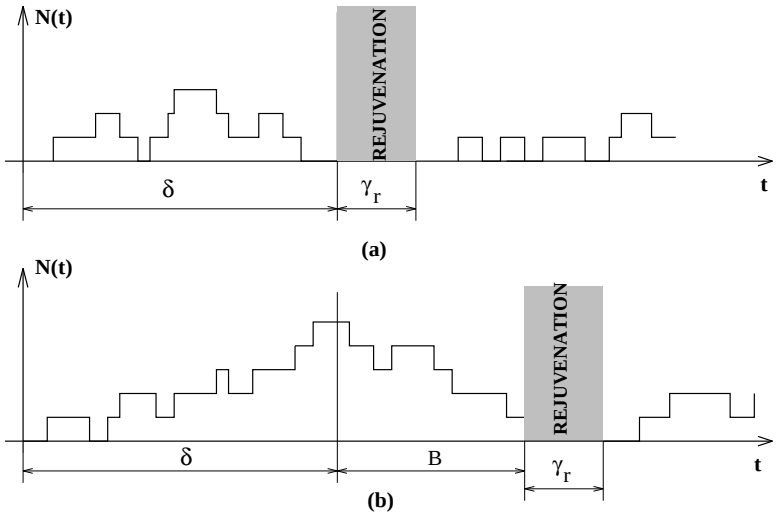

Figure 2: Sample Path of the process

The main assumptions of our model can then be summarized as follows:

1. Aging is captured by decreasing service rate and increasing failure rate;
2. Time to recovery from a failure is generally distributed;
3. Time to complete rejuvenation is generally distributed;
4. δ, (Rejuvenation interval under policy **I** and rejuvenation wait under policy **II** is deterministic;

[1]Eventually this assumption might be relaxed assuming that rejuvenation occurs as soon as $t \geq \delta$ and the number of jobs in the queue is below a given threshold.

5. Transaction arrivals follows the Poisson process;

Assumption 1 is quite realistic as a complex software system experiences unavailability and performance degradations at the same time, even if the effect of one may be more dominant than the other. Each of the previous models considered only one of the two aspects. Specifically in [2] occasional crash/hang failures with increasing failure rates were assumed, while in [7] decreasing service rates were taken into consideration. Furthermore, in [2] the time to failure was assumed to have hypo-exponential distribution. In this work, we relax this assumption by considering a general distribution. Assumptions 2 and 3, similarly are enhancements to previous modeling efforts, where a specific distribution (exponential in [2] and deterministic in [**?**]) for these qauantities was assumed. Assumption 4 is usually accepted for such arrival processes.

Given the above behavioral model, the task at hand involves

1. Evaluate the steady state availability for both policies **I** and **II**.

2. Evaluate the stready state probability that an arriving transaction will not be processed by the software.

3. Determine the optimal value of δ, i.e., rejuvenation interval under policy **I** and rejuvenation wait under policy **II** which would maximize (minimize) the availability (probability of loss).

3 Model Solution

Let the steady state availability be denoted by A_{SS} and the probability that an arriving transaction will be lost be denoted by P_{loss}. The approach we follow in deriving expressions for above qunatities and the intermediate expressions we work with apply 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 simulation approach.

As already mentioned 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 1), rejuvenating (state 3) or recovering, i.e.,in failed (2) state (see Figure 1). If $i\{Z(t), t \geq 0\}$ represents the state of the software at time t, and the sequence of random variables $S_i, i > 0$ represent the transition times among different states, then it is easy to see that $i\{Z(S_i), i > 0\}$ is the embedded discrete time Markov chain (DTMC). The transition probabilities matrix P for this DTMC can also be easily derived and is given by:

$$P = \begin{pmatrix} 0 & P_{12} & P_{13} \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \tag{1}$$

The steady state probability $\pi_i, i = 1, 2, 3$ of the software being in state i can be also be determined in a straightforward manner from the well know relation $\pi = \pi P$. They are given by;

$$\begin{aligned} \pi_1 &= \frac{1}{2} \\ \pi_2 &= \frac{1}{2} P_{12} \\ \pi_3 &= \frac{1}{2} P_{13} \end{aligned} \tag{2}$$

The software behavior as a whole is modeled via the stochastic process $\{(Z(t), N(t)), t \geq 0\}$, where $N(t)$ represents the number of transactions in queue for processing (including one being serviced). If $Z(t) = 1$, then $N(t) \in \{0, 1, \ldots, K\}$, as the software queue can accomodate up to K transactions. If $Z(t) \in \{2, 3\}$, then $N(t) = 0$, since we assumed that all incoming transactions when the software is either rejuvenating or recovering are lost. Further, the transactions already in the queue at the transition instant are also discarded. It can be shown that the process $\{(Z(t), N(t))$ is a Markov regenerative process (MRGP) [**?**, **?**]. The regeneration instants are embedded at times when the process makes transition from state i to state j, $i, j = 1, 2, 3$, i.e., $Z(t)$ changes. 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 1. State 1 is certainly a regeneration state as we explicitly assumed that when the

system enters this state everything is reset to the original initial condition (the system is empty and the software is as good as new). States 2 and 3 are also regeneration states as, once the system enters these states, every other activity but the recovery or rejuvenation one is interrupted. The system forgets the past history and can only come back to the fully operational condition. Note that what makes this 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 1. We defined and solved the embedded DTMC of this MRGP.

To better understand the solution method, the Table 1 lists the adopted notation (R.V. denotes random variable):

P_{12}	Transition probability from state 1 (UP) to state 2 (Recovering)
P_{13}	Transition probability from state 1 (UP) to state 3 (Rejuvenating)
$p_i(t)$	Probability that i transactions are queued at time t
N_l	Number of transactions discarded at the end of the regeneration period started from state 1 (R.V.)
γ_f	Expected time to recover from failure
γ_r	Expected time to rejuvenate
U	Sojourn time spent in state 1 (R.V.)
λ	Transaction arrival rate
$\mu(t)$	Transaction service rate
$\rho(t)$	Failure rate

Table 1: Adopted notation

The steady state availability can then be obtained using standard formulae from MRGP theory and is given as:

$$\begin{aligned} A_{SS} &= Pr\{software\ in\ state\ 1\} \\ &= \frac{\pi_1 E[U]}{\pi_2 \gamma_f + \pi_3 \gamma_r + \pi_1 E[U]} \end{aligned}$$

Substituting values of π_1, π_2 and π_3,

$$A_{SS} = \frac{E[U]}{P_{12}\gamma_f + P_{13}\gamma_r + E[U]} \tag{3}$$

The probability that a transaction is lost is defined as the ratio of expected number of transactions which are lost in an interval and the total number transactions which arrive during that interval. Since the process is stochastically same in succesive visits to state 1, it suffices to consider this interval only. The expected number of transactions lost is given by summation of the expected number lost due to the discarding (when the software failed or was rejuvenate), expected number lost while recovery or rejuvenation is going on and expected number which lost due to the buffer being full. The last quantity is of special significance as due to the degrading service rate, the buffer has a higher probability of being full. The probability of loss is then given by:

$$P_{loss} = \frac{\pi_1 E[N_l] + \lambda\left(\pi_2\gamma_f + \pi_3\gamma_r + \pi_1\int_0^\infty p_K(t)dt\right)}{\lambda\left(\pi_2\gamma_f + \pi_3\gamma_r + \pi_1 E[U]\right)} \tag{4}$$

where:

- $\pi_1 E[N_l]$ is the expected number of jobs already in the buffer when the system is exiting state 1;
- $\lambda\pi_2\gamma_f$ is expected number of jobs arriving while the system is recovering in state 2;
- $\lambda\pi_3\gamma_r$ is the expected number of jobs arriving while the system is rejuvenating;
- $\lambda\pi_1 \int_0^\infty p_K(t)dt$ is the expected number of transactions denied service because of buffer full;
- $(\pi_2\gamma_f + \pi_3\gamma_r + \pi_1 E[U])$ is the average length of time between two consequtive visits to state 1.

Equation 4 is valid for policy **II** only is assumed. 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 1 is limited by δ under policy **I**.

Regardless of the rejuvenation policy, as can be observed from equations 3 and 4, we need to obtain expected soujourn times and the steady state probability of the software in each of the three states $1, 2$ and 3, as well as the transient probability that there 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 2 and 3 is already available as γ_f and γ_r respectively[2]. The quantities still to be derived are related to the behavior of the software in state 1i, viz., P_{12}, P_{13}, $E[U]$ and $p_i(t), i = 0, 1, \ldots, K$ and their evaluation depends on the policy used.

3.1 Behavior of the system in state 1 assuming Policy I

For $Z(t) = 1$, 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 rejuvenation 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 continuous time Markov chain (NH-CTMC) under policy **I**. It is simply 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 whole process terminated.

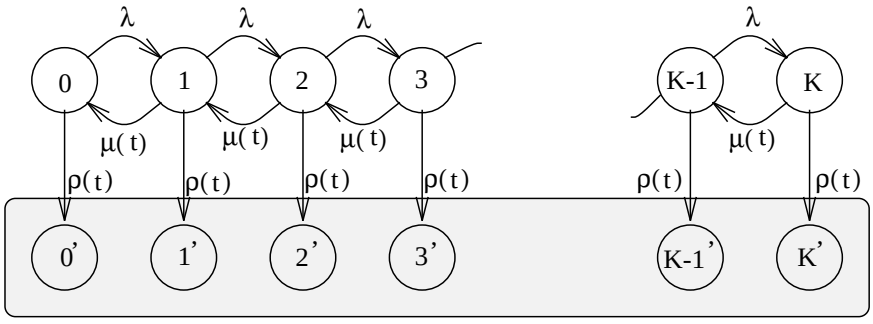

Figure 3: Subordinated Non-homogeneous CTMC for $t \leq \delta$

By our notation, $p_i(t)$ is the probability that there are i transactions queued for service,

[2]The first moment 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.

which is also the probability of being in state i of the NH-CTMC at time t. Note that state $i, i = 0, 1, \ldots, K$ is not 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 NH-CTMC can be lumped into a single absorbing state.

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

$$\begin{aligned}
\frac{dp_0(t)}{dt} &= \mu(t)p_1(t) - (\lambda + \rho(t))\, p_0(t) \\
\frac{dp_i(t)}{dt} &= \mu(t)p_{i+1}(t) + \lambda p_{i-1}(t) - (\mu(t) + \lambda + \rho(t))\, p_i(t), 1 \leq i < K \\
\frac{dp_K(t)}{dt} &= \lambda p_{K-1}(t) - (\mu(t) + \rho(t))\, p_K(t) \\
\frac{dp_{i'}(t)}{dt} &= \rho(t)p_i(t), 0 \leq i \leq K
\end{aligned} \tag{5}$$

The set of simultaneous of differential-difference equations given by 6 do not have a tractable analytical solution and must be evaluated numerically. In our work, this set along with the initial conditions $p_0(0) = 1$, $p_i(0) = 0, 1 \leq i \leq K$ and $p_{i'}(0) = 0, 0 \leq i \leq K$ was coded in *Mathematica* and solved numerically for all $p_i(t)$. Once these probabilities are obtained, the rest of the wuantities can be easily evaluated as follows.

One step transition probability P_{12} is given by:

$$P_{12} = \sum_{i=0'}^{K'} p_i(\delta)$$

and

$$P_{13} = 1 - P_{12}$$

Thereafter, according to 3, the steady state probability that the software is in states 2 and 3 can be obtained.

The expected sojourn time in state 1 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 1 is left, is

evaluated as:

$$E[N_l] = \sum_{i=0}^{K} i\, p_i(\delta)$$

Both A_{SS} and P_{loss} as given in equations 3 and 4 respectively can now be easily calculated.

3.2 Behavior of the system in state 1 assuming Policy II

If Policy II is assumed, the evolution of the system in macro-state 1 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 rejuvenation will occur if and only if the buffer is empty after δ has elapsed. For $t \leq \delta$, exactly the same NH-CTMC of Figure 3 determines the behavior of the software. For $t > \delta$ the NH-CTMC 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 rejuvenation will occur, thus turminating the subordinated NH-CTMC, once the system processes all the transactions in the queue.

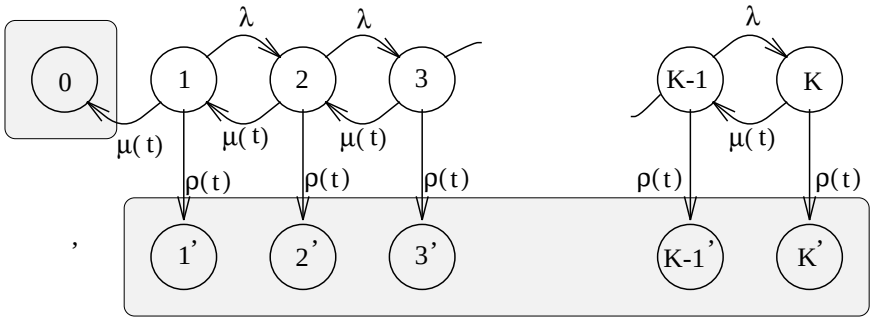

Figure 4: Subordinated Non-homogeneous CTMC fot $t > \delta$

The Chapman-Kolmogorov forward differential equations which are used to determine all transient probabilities are given as follows:

$$\begin{aligned}
\frac{dp_0(t)}{dt} &= \mu(t)p_1(t) - (\lambda' t + \rho(t))\, p_0(t) \\
\frac{dp_1(t)}{dt} &= \mu(t)p_2(t) + \lambda'(t)p_0(t) - (\mu(t) + \lambda + \rho(t))\, p_1(t), \\
\frac{dp_i(t)}{dt} &= \mu(t)p_{i+1}(t) + \lambda p_{i-1}(t) - (\mu(t) + \lambda + \rho(t))\, p_i(t), 2 \leq i < K \\
\frac{dp_K(t)}{dt} &= \lambda p_{K-1}(t) - (\mu(t) + \rho(t))\, p_K(t)
\end{aligned} \tag{6}$$

$$\begin{aligned}\frac{dp_{0'}(t)}{dt} &= \rho'(t)p_0(t) \\ \frac{dp_{i'}(t)}{dt} &= \rho(t)p_i(t), 1 \leq i \leq K\end{aligned}$$

where $\lambda'(t) = \lambda, t \leq \delta$, otherwise it is zero. Similarly, $\rho'(t) = \rho(t), t \leq \delta$, otherwise zero. As before, this set of differential-difference equations along with the initial condition that $p_0(0) = 1$ is solved numerically using *Mathematica* package.

The quantities of interest can then be evaluated using similar expressions as derived for policy **I**. Minor differences arise, which arise are now given. Transient state probabilities P_{12} and P_{13} are evaluated at $t = \infty$ and are given as:

$$P_{12} = \sum_{i=0'}^{K'} p_i(\infty)$$

and

$$P_{13} = 1 - P_{12}.$$

The sojourn time in state 1 is now given by:

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

Computation of $E[N_l]$ is exactly the same as in policy **I**. Using equations 3 and 4, steady state availability and probability of loss of an arriving transaction can be calculated.

4 Experiments and Results

In this Section, we illustrate the usefulness of the models developed to evaluate the steady state availability (A_{SS}) and the probability that a transaction is lost P_{loss}. Further, 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. Table 2 shows the parameter values that were chosen. The values chosen are for illutration purposes only and do not necesasrily represent any physical system.

γ_f	0.15 and 0.85 $(hours)$
γ_r	0.15 $(hours)$
λ	6.0 $(hours^{-1})$
δ	$1 \ldots 200$ $(hours)$
K	20
$\mu(t)$	$\mu_{max}\left[1 - \frac{t}{MTTF}\right]$ $if\ t \leq a$ μ_{min} $if\ t > a$ with $a = \frac{(\mu_{max} - \mu_{min})}{\mu_{max}} MTTF$
$\rho(t)$	$\beta\alpha t^{\lambda-1}$ with $\alpha = 1.5$ and $\beta = \left[\frac{\Gamma(1+\frac{1}{\alpha})}{MTTF}\right]^{\alpha}$

Table 2: Model parameters

Further, the rejuvenation interval (wait) δ is varied between 1 and 200 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 [1]. Note

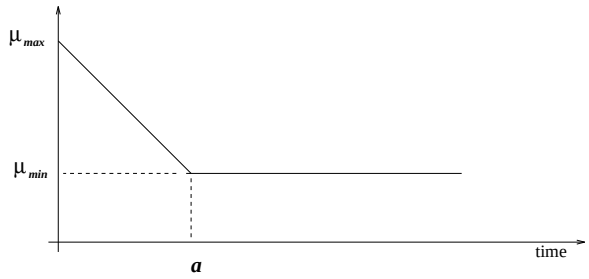

Figure 5: Time variation of the service rate $\mu(t)$

that our model, by itself, is not restrictive and $mu(t)$ can be any general function. As we mentioned in the earlier section, the failure process and the service process are stochastically independent. In reality, they may be dependent. To capture this dependence, we define $\mu(t)$ to have parametric dependence on the Mean Time To Failure (MTTF) of the software. This

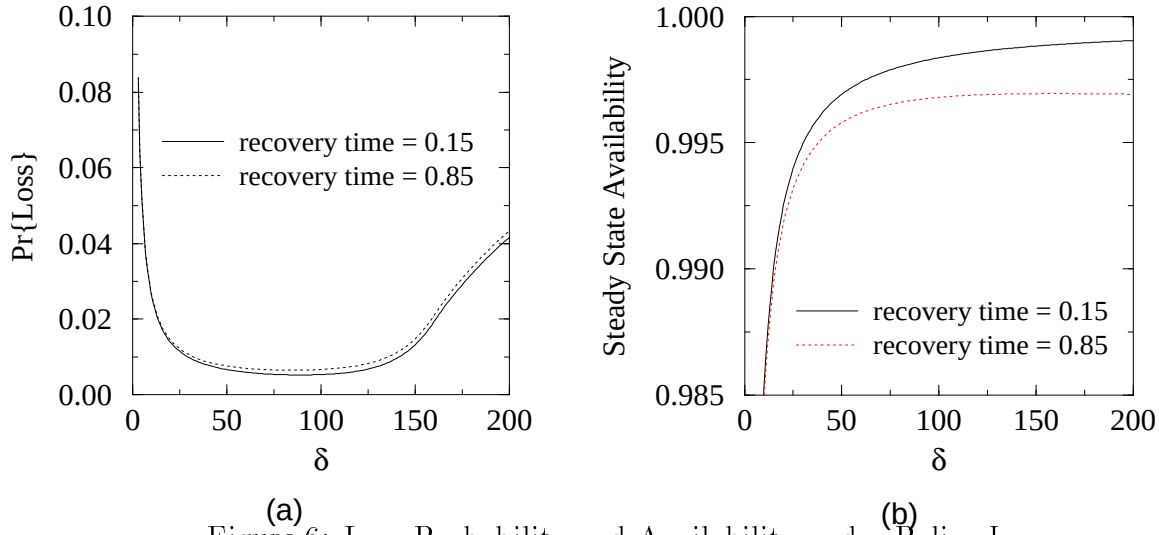

Figure 6: Loss Probability and Availability under Policy I

is shown in table 2 from the way in which the value of a is calculated:

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

We further assumed $\mu_{max} = 15$ and $\mu_{min} = 6$ customers served per hour. Finally, to model the increasing failure rate of the software system, Weibull distribution for time to failure has been assumed with $\alpha = 1.5$ and

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

The results obtained by after running the *Mathematica* program are plotted in Figure 6 for Policy **I** and in Figure 7 for Policy **II**.

Figure 6(a) shows the plot of the probability of loss against the rejuvenation interval, assuming $\gamma_f = 0.15$ and 0.85 hours. As can be observed, if the rejuvenation is done at very small intervals, the probability of loss is quite high. Similarly, with very infrequent rejuvenation, the probability of loss is high. A range of δ, roughly from 50 to 125 hours guarantees a minimum value of loss probability. We can definitely find a given minimum value for the loss probability (that in our plot occurs for $\delta = 87.0$ hours), but it is interesting to note that some flexibility is allowed in choosing the rejuvenation interval. Further, the value of the recovery time γ_f does not affect too much the final results.

Figure 6(b) shows the system availability vs. the rejuvenation interval δ, with varying the recovery time. If $\gamma_f = 0.85$ hours the system availabilty is always less than if $\gamma_f = 0.15$

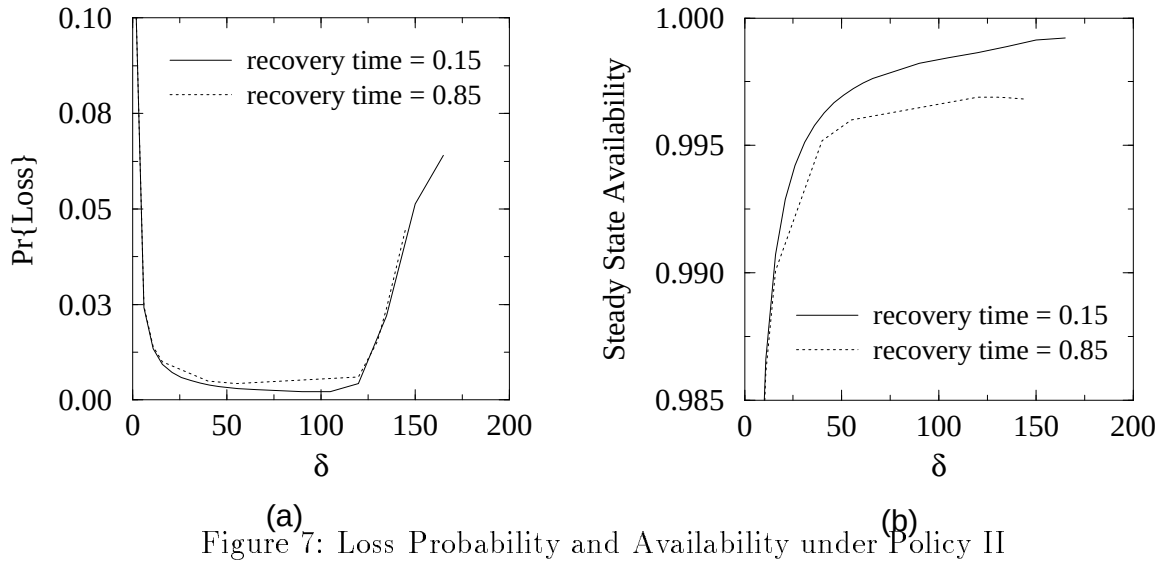

Figure 7: Loss Probability and Availability under Policy II

hours is assumed.

If $\gamma_f = 0.15$ hours, then it can be observed that it is not possible to identify an optimal value for δ, but increasing it as much as possible. It indicates that in this case rejuvenation is not an effective way to increase steady state availability. If $\gamma_f = 0.85$ hours it is possible to idenfy a value of δ such that the availability starts decreasing. It is not easily visible form the Figure, but at $\delta = 170$ hours, the system availability starts descreasing from its maximum value 0.9969. Thus, it is not possible to optimize at the same time loss probability and system availability, but a reasonable comprise can be obtained by choosing δ accordingly to the plots of Figure 6.

Figure 7 shows similar measures, but when Policy II is assumed. As can be observed the behavior of the system is almost the same, but values are quite different. In fact in this case the loss probability values are much smaller, as a consequence of the rejuvenation policy adopted.The optimal value of the rejuvenation interval δ is now equal to 105 hours, and the loss probability is equal to 0.00237 and 0.00424 for $\gamma_f = 0.15$ and $\gamma_f = 0.85$, respectively. For Policy I tyhe corresponding vaules were 0.00526 and 0.0065. Also in this case an optimal value to rejuvenate which maximize system availability can be found only for $\gamma_f = 0.85$ and is equal to 144 hours were $A_{SS} = 0.9968$.

5 Conclusions

Software rejuvenation has been proved to be a pratical way to avoid system crash due to software starvation. Even if it has been adopted till now in a very empirical way by almost all computer users, in this paper we provide an in depth analysis of the phenomen, assuming different rejuvenation policies. An analytical approach is followed which, through numerical solution, allows to compare two rejuvenation policies in order to evaluate the steady state availability (A_{SS}) and the probability that a transaction is lost P_{loss}. Further, 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.

Acknoledgements

This work was done while Antonio Puliafito was visiting Duke University supported by italian CNR under grant n.
Sachin Garg was partially supported by an IBM fellowship.

References

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

[2] S. Garg, A. Puliafito, M. Telek and K. S. Trivedi, "Analysis of software rejuvenation using Markov regenerative stochastic Petri net", *Proc. of the Sixth Intnl. Symposium on Software Reliability Engineering*, Toulouse, France, October 24-27, 1995, pp. 180-187.

[3] 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.

[4] Y. Huang, C. Kintala, N. Koletis, N. D. Fulton, "Software Rejuvenation- design, implementation and analysis", Proc. of *Fault-tolerant Computing Symposium*, Pasadena, CA, June 1995, pp. 381-390.

[5] I. Lee, "Software dependability in the operational phase", *Ph.D. Thesis*, Dept. of Electrical and Computer Engineering, Univ. of Illinois, Urbana-Champaign, 1995.

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

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