Modeling and Analysis of Load and Time Dependent Software Rejuvenation Policies

S. Garg[1], A. Pfening[3], A. Puliafito[2], M. Telek[3], K. S. Trivedi[1]

[1] Center for Advanced Comp. & Comm.
Dept. of Electrical Engineering
Duke University
Durham, NC 27708

[2] Ist. di Informatica e Telecom.
Università di Catania, Viale A. Doria, 6
95125 Catania, Italy

[3] Dept. of Telecommunications
Technical University of Budapest
1521 Budapest, Hungary

Abstract

Due to repeated and potentially faulty usage of continuously running client-server type software systems by many clients, such software "ages" with time and eventually fails. Huang et. al. proposed a technique called "software rejuvenation" [3] in which the software is periodically stopped and then restarted in a "robust" state after proper maintenance. This "renewal" of software prevents, or at least postpones, the crash failure.

In this paper, we present a quantitative analysis of two software rejuvenation policies. The first one considers only the ageing behaviour of the system by time, while the second one considers the actual load of the system as well. The behaviour of the system is represented through a Markov Regenerative Stochastic Petri Net (MRSPN) model. Numerical analysis of the system performance regarding the probability of successful service of clients is provided.

Keywords:

Continuously running client-server software systems, Software rejuvenation, Markov Regenerative Stochastic Petri Net model, Performance analysis, Rejuvenation policies.

1 Introduction

Software life cycle can broadly be classified into development and operational phase. The development phase is roughly divided into design, coding and testing phase. Traditionally, software quality improvement with respect to factors like performance and non-faultyness has been concentrated in the design phase. However, coding and testing are not perfect and extensive enough to guarantee a fault free operational software.

System failures due to imperfect software behaviour are usually more frequent than failures caused by hardware components faults [2]. These failures are the result of either inherent design defects in the software or from improper usage by clients [6]. Thus fault tolerant software has become an effective alternative to virtually impossible fault-free software. A wide literature exists in this field where the software has the ability to recover from a *transient* fault [1, 4, 5, 7]. Most of the approaches for example N-version programming [1] and recovery block [7] are corrective in nature, i.e. only after a failure has occured, recovery is started. The overhead incurred by such recovery strategies remains high and much research was done to reduce it.

Huang et. al. have suggested a complimentary technique which is preventive in nature. It involves periodic maintenance of the software so as to prevent crash failures. They call it *Software Rejuvenation* [3], and define it as the *periodic preemptive rollback of continuously running applications to prevent failures.*

While monitoring real applications, it was observed that software typically "ages" as it is run. Potential fault conditions are thus slowly accumulated since the beginning of the software activity. Consider, for example, a server module interacting with many client modules. Memory bloating, unreleased file-locks, data corruption are the typical causes of slow degradation which, if not taken care of, leads to crash failure. Software rejuvenation involves periodically stopping the system, cleaning up, and restarting it from a clean internal state. This "renewal" of software prevents (or in the least postpones) a crash failure.

In [8] a quantitative analysis of a software rejuvenation model by the mean of the probability of system unavailability is presented. In this paper we evaluate a similar non-Markovian system model however taking into consideration a different performance measure, the steady state probability of successful service instead of the probability of system unavailability.

The rest of the paper is organized as follows. Section 2 briefly introduces the applied model description tool and the associated analysis method. Section 3 discusses the system model and the considered rejuvenation policies. Details of analysis of the considered software rejuvenation models can be found in Section

4. Results of numerical experiments are presented in Section 5, and the paper is concluded in Section 6.

2 Introduction to DSPNs

One difficulty in modeling a stochastic system such as software with rejuvenation arises because the rejuvenation interval is deterministic, which renders the system *"non-Markovian"* and standard evaluation method using the theory of continuous time Markov chains can not be applied. In this case, the approach is to study the underlying stochastic process of such non-Markovian systems. In our cases the underlying process can be shown to be a Markov regenerative one (MRGP) and therefore Markov renewal theory can be applied for its long-run behaviour [9, 11].

A complementary issue is to specify the system behaviour in a concise way from which the underlying stochastic process can be extracted and analyzed. Non-Markovian Stochastic Petri nets on the one hand with their remarkable flexibility and potential for capturing concurrency, contention and synchronization in a system, and on the other hand with their ability to quantitatively evaluate stochastic models can be used as the high-level specification tool.

For analyzing a non-Markovian SPN, with underlying MRGP [10], we need to identify certain time points in the underlying stochastic process at which it is possible to forget the past history. These points, indicated as *regeneration points*, are such that the future evolution of the stochastic process only depends on the present state entered when a regeneration time point occurs. The analysis of a non-Markovian SPN with regeneration time points is composed by the following steps:

i) evaluation of the processes (called subordinated processes) to the consecutive regeneration time point starting from all possible states in which the memoryless property can occur.

ii) evaluation of the steady state (or the transient) behaviour of the whole process based on the analysis of the subordinated processes.

In this paper the considered performance models are associated with the steady state behaviour of the DSPN underlying the MRGP model of the system. According to the above steps the steady state analysis is as follows:

i) analysis of the subordinated processes

- state transition probabilities of the embedded DTMC:

$$\pi_{ij} = Pr\{\mathcal{M}(\tau_1^*) = j \mid \mathcal{M}(\tau_0^*) = i\} \quad (1)$$

where τ_n^* is the nth *regeneration point* and $\mathcal{M}(t)$ is the state of the process at time t.

- Mean time a subordinated process spends in a state is:

$$\alpha_{ij} = \int_0^\infty Pr\{\mathcal{M}(t) = j\ ,\ \tau_1^* > t \mid \mathcal{M}(\tau_0^*) = i\}\ dt \quad (2)$$

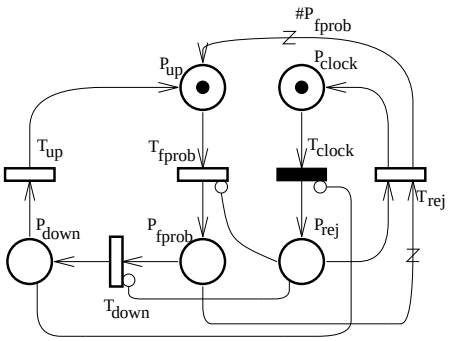

Figure 1: MRSPN Model of the System Behaviour

ii) analysis of the whole process

- steady state probabilities of the embedded DTMC (ν_i):

$$\nu_i = \sum_j \nu_j \pi_{ji}\ ; \quad \sum_i \nu_i = 1 \quad (3)$$

- and final steady state probabilities of the MRGP (γ_j):

$$\gamma_j = \frac{\sum_k \nu_k \alpha_{kj}}{\sum_k \nu_k \sum_l \alpha_{kl}} \quad (4)$$

3 The system model and the considered rejuvenation policies

The software starts up in a "robust" state in which the probability of failure is zero. As it is used, it ages with time and if no rejuvenation is done eventually transits to another state in which it provides normal service but can fail (crash) with a non-zero probability. Once it crashes, it takes a random amount of time to bring the system up again to the clean state and restart it. Rejuvenation is performed at a fixed interval from the start (or restart) of the software in the robust state. At the time of rejuvenation, if the software has not already crashed, it is either in the clean or the failure probable state. It is then stopped, cleaned and restarted all of which takes a random amount of time. We assume that the time for which software remains clean, the time to fail from the failure probable state and the time to restart both from rejuvenation and crash failures are all exponentially distributed. The rejuvenation interval, however, is deterministic.

Figure 1 shows the Petri net model of the above described software system itself. The robust state is modeled by place P_{up}. The exponentially distributed transition T_{fprob} models the aging of the software. When this transition fires, (a token reaches place P_{fprob}), i.e., the software enters the failure probable state. The exponentially distributed transition T_{down} models crash failure of the software. During

the software maintenance (exponentially distributed transition T_{up}), every other activity is suspended: the inhibitor arc from place P_{down} to transition T_{clock} is used to model this fact.

Deterministic transition T_{clock} models the rejuvenation period. It is competitively enabled with T_{fprob} and fires when the clock expires if T_{down} has not fired by that time. Once it fires, a token moves to place P_{rej} and the activity related with software rejuvenation (transition T_{rej}) starts. The marking dependent arcs from P_{fprob} to T_{rej} and from T_{rej} to P_{up} stand for removing the token from P_{fprob} if any at rejuvenation.

During the rejuvenation phase, every other activity in the system is suspended. This is modeled by inhibitor arcs from place P_{rej} to transitions T_{fprob} and T_{down}. Upon rejuvenation, the net has to be re-initialized into a condition with one token in place P_{up} and one in place P_{clock}, and all the other places empty. If the software was in the robust state when T_{clock} fired, then after rejuvenation is complete, T_{rej} fires to re-initialize the net, with a rate equal to λ_{rej1}. If the software had reached the failure probable state (token in place P_{fprob}), then T_{rej} fires to complete the rejuvenation and re-initializes the net: a rate equal to λ_{rej2} (with $\lambda_{rej1} \geq \lambda_{rej2}$ is assumed in this case.

3.1 Time Based Policy

In this paper we extend the previous model, originally provided in [8], by including the arrival/departure processes of the requests addressed to the server. Figure 2 is derived from Figure 1 where transitions T_{arr} and T_{serve} and place P_{load} have been added. Transition T_{arr} models the arrival process (exponentially distributed inter-arrival times have been assumed) of requests which are stored in a buffer modeled through place P_{load}. The dimension of the buffer is limited to k elements (the inhibitor arc from P_{load} to T_{arr} models the finite dimension of the buffer). Transition T_{serve} models the exponentially distributed service time. To model the server unavailability, either because under software rejuvenation or under recovery from a crash condition, two inhibitor arcs are provided from places P_{rej} and P_{down} to transition T_{serve}. Finally, a variable cardinality arc from place P_{load} to transition T_{up} and one to transition T_{rej} have been added which flush the buffer from all the pending requests before the system is restored to the full functioning condition. Having related the processing power of the system to its functioning state allows us to evaluate the loss probability due to the system unavailability as well as its effective productivity.

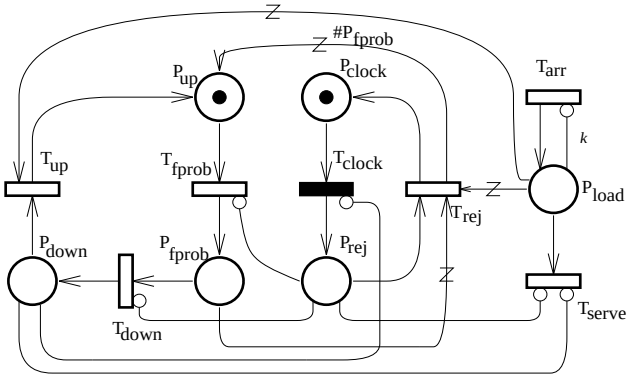

Figure 2: Time dependent rejuvenation policy

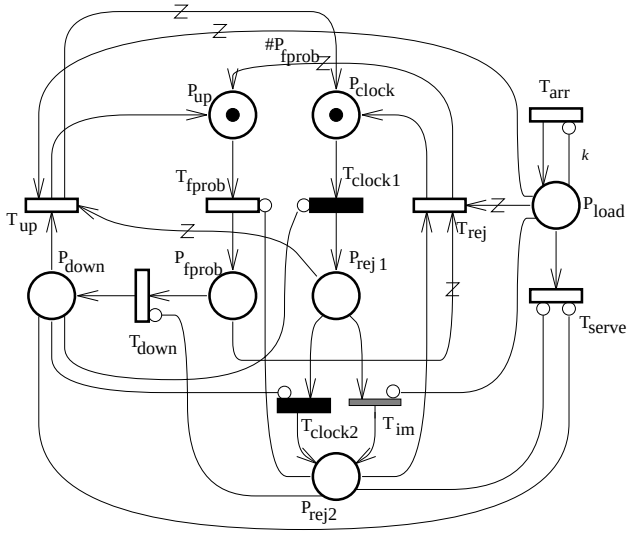

Figure 3: Load and time dependent rejuvenation policy

3.2 Load and Time Based Policy

We propose a different rejuvenation strategy which is strictly related to the load offered to the system in a given time instant. The strategy we want to analyze determines the Petri net model depicted in Figure 3. It is derived from Figure 2, with some changes to the part modeling the rejuvenation activity. The rejuvenation time is now divided into two parts. After the first time interval, modeled by transition T_{clock1}, the system enters into a state (modeled by a token in place P_{rej1}) in which a decision should be taken if rejuvenate or not according to the status of the input buffer. More precisely, if the load to the system is below a given threshold (inhibitor arc from P_{load} to the immediate transition T_{im}) then transition T_{im} will fire immediately and the rejuvenation will start. If the number of requests in the buffer is over a given threshold, it might be more convenient to delay the rejuvenation for a while, in order to process some more requests, thus reducing the number of lost requests. However, after a maximum given amount of time (deterministic transition T_{clock2}) the rejuvenation phase will start independently on the status of the requests queue.

4 Analysis of rejuvenation policies

The performance measure considered in this paper is the steady state probability of successful service of

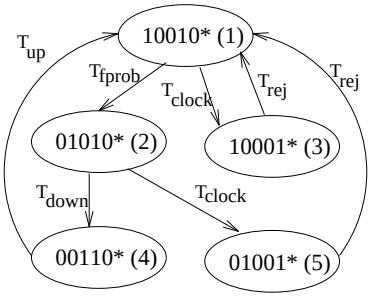

Figure 4: Structure of the reachability graph of the time based model

requests, i.e.

$$\chi = \lim_{t\to\infty} \frac{\#\text{ requests succesfully served in } (0,t)}{\#\text{ requests arrived to the system in } (0,t)}$$

$$= \frac{Pr\{T_{serve} \text{ is enabled}\}\ \mu}{Pr\{T_{arr} \text{ is enabled}\}\ \lambda}$$

$$= \frac{\mu \sum_{\{i:T_{serve} \text{ is enabled}\}} \gamma_i}{\lambda \sum_{\{j:T_{arr} \text{ is enabled}\}} \gamma_j} \tag{5}$$

Which means that the performance measure of the introduced models can be evaluated by summing up the steady state probabilities of the states in which T_{arr} (T_{serve}) is enabled.

4.1 Time Based Policy

Let the 6-tuple ($\#P_{up}$, $\#P_{fprob}$, $\#P_{down}$, $\#P_{clock}$, $\#P_{rej}$, $\#P_{load}$) denote the markings of the Petri net model, where $\#P_x$ is the number of tokens in place P_x. Since there is only a single deterministic transition (T_{clock}) in the net whose behaviour is independent of the service process the structure of the reachability graph can be studied disregarding the service process (i.e. $\#P_{load}$)

The evolution of the stochastic process can be represented over the sets of states (10010$*$), (01010$*$), (10001$*$), (00110$*$) and (01001$*$). '$*$' means that the number of tokens in $\#P_{load}$ can take any value from $0, 1, \ldots, k$. Each sets are composed by $k+1$ states. Hence the number of reachable markings are $5k+5$.

Figure 4 shows the structure of the reachability graph with the ovals representing the sets of markings and the arcs representing the possible transitions between the sets. The five sets of markings are labeled one through five respectively. The arc from a marking set i to j is labeled by the name of the transition whose firing brought about the change.

There is only a single subordinated process with internal state transition in this model. The subordinated process starting from marking (100100) is a CTMC, with internal state transitions due to the firing of T_{fprob}, T_{arr} and T_{serve}. This subordinated process can be concluded by the firing of T_{clock} (at its deterministic firing time) or by a preceding firing of T_{down}. All the other subordinated processes are concluded by the firing of an exponential transition (T_{rej}, T_{up}, T_{arr} or T_{serve}). From sets (10010$*$) and (01010$*$) only marking (100100) can be a regeneration state with the given initial marking, while only exponential transitions are enabled in the other 3 sets, hence there are $3k+4$ regeneration markings out of the $5k+5$ markings.

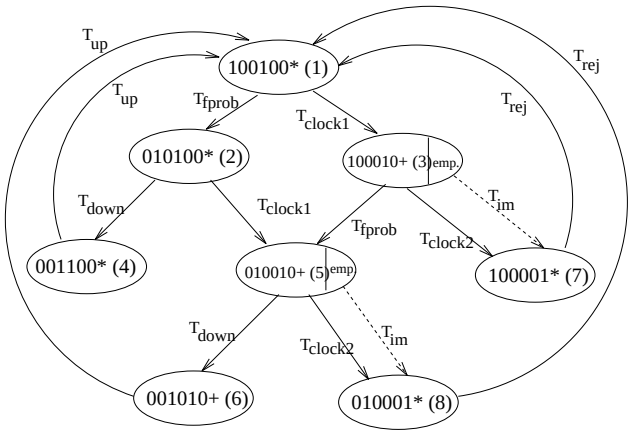

Figure 5: Structure of the reachability graph of the load and time based model

The considered performance parameter is associated with the particular Petri net model in the following way:

$$\chi = \frac{Pr\{T_{serve} \text{ is enabled}\}\ \mu}{Pr\{T_{arr} \text{ is enabled}\}\ \lambda} = \frac{Pr\{(\#P_{down}=0)\&(\#P_{rej}=0)\&(\#P_{load}>0)\}\ \mu}{Pr\{\#P_{load}<k\}\ \lambda} \tag{6}$$

4.2 Load and Time Based Policy

Let the 7-tuple ($\#P_{up}$, $\#P_{fprob}$, $\#P_{down}$, $\#P_{clock}$, $\#P_{rej1}$, $\#P_{rej2}$, $\#P_{load}$) denote the markings of the Petri net model.

The evolution of the stochastic process can be represented over the sets of states depicted in Figure 5. '$+$' means that the number of tokens in $\#P_{load}$ can take any value from $1, 2, \ldots, k$. Sets (1), (2), (4), (7), (8) are composed by $k+1$ tangible states. Sets (3), (5) are composed by k tangible and one vanishing states. Set (6) contains k states. Hence the number of reachable markings are $8k+5$.

There are two exclusively enabled transitions with deterministic firing time (T_{clock1}, T_{clock2}) in the model. The subordinated processes associated with the firing of these transitions are CTMCs. The states of sets (1) and (2) can not be regeneration states apart of (1001000). This way the number of regeneration states is $6k+4$.

The successful service probability is associated with the Petri net model as in Eq. (6) but $\#P_{rej}$ has to be replaced by $\#P_{rej2}$ in the denominator.

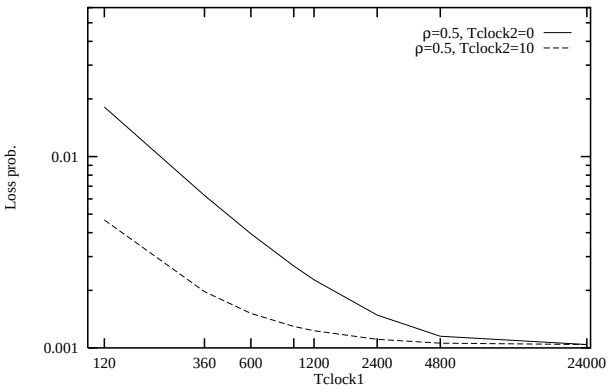

Figure 6: Performance of rejuvenation policies, $\rho = 0.5$

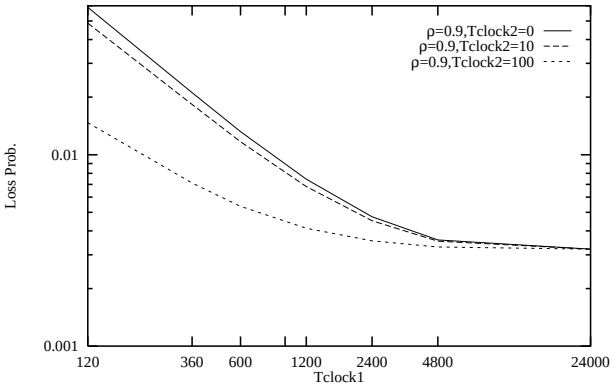

Figure 7: Performance of rejuvenation policies, $\rho = 0.9$

5 Numerical results on the performance of rejuvenation policies

Let λ_1, λ_2, λ_3 and λ_4 be the transition rates associated with T_{fprob}, T_{down}, T_{rej} and T_{up} respectively. Values of λ_1 through λ_4 are fixed for all the results and are taken from [3]. $\lambda_1^{-1} = 240h.$, $\lambda_2^{-1} = 2160h.$, $\lambda_3 = 6/h.$ and $\lambda_4 = 2/h.$. Two values of the utilization (ρ) were considered in the numerical experiments: $\rho = 0.5$ ($\lambda = 0.5$, $\mu = 1$) and $\rho = 0.9$ ($\lambda = 0.9$, $\mu = 1$). The "buffer size" (k) was 20 in all cases.

In Figure 6 and Figure 7 the dependence of the loss probability of customers $(1 - \chi)$ on the rejuvenation time(s) is depicted. The solid lines titled "Tclock2=0" refers to the time dependent model, while the lines where Tclock2>0 refer to the load and time dependent model.

6 Conclusion

We studied and compared two software rejuvenation models with performance parameters associated with their service performance. We found that significant performance gain can be obtained when the number of customers in the system is considered at rejuvenation.

Acknowledgement

A. Pfening and M. Telek would like to thank Hungarian Scientific Research Found (OTKA) for grant no. T-16637.

References

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

[2] 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.

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

[4] Y. Huang and C.M.R. Kintala, "Software implemented fault tolerance: Technologies and experience", In *Proc. of 23rd Int. Symposium on Fault-Tolerant Computing*, Toulouse, France, pp. 2-9, June, 1993.

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

[6] P.A. Lee, "Software-faults: The remaining problem in fault tolerant systems?", In *Eds. M. Banatre and P.A. Lee, Hardware and Software Architectures for Fault Tolerance: Experiences and Perspectives*, LNCS, Springer Verlag, Vol. 774, pp. 171-181, 1994.

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

[8] S. Garg, A. Puliafito, M. Telek, K.S. Trivedi, "Analysis of software rejuvenation using Markov Regenerative stochastic Pertri net", In *Int. Symposium on Software Reliability Engineering (ISSEE'95*, Toulose (France), Oct. 24-27 1995.

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

[10] H. Choi, V.G. Kulkarni, K.S. Trivedi, "Markov Regenerative Stochastic Petri Nets", *Performance Evaluation*, 20:337-357, 1994.

[11] R. Fricks, M. Telek, A. Puliafito, K. Trivedi, "Markov regenerative theory applied to performability evaluation.", In K.K. Bagchi and G. Zobrist, eds., *Modeling and Simulation of Advanced Computer Systems: Applications and Systems*, pp. 193–236. Gordon and Breach Publishers, 1996.