Analysis of Load and Time Dependent Software Rejuvenation Policy on Virtualized Environment

Ohnmar Nhway
University of Computer Studies, Mandalay
skynhway@gmail.com

Abstract

In client-server type, due to repeated and potentially faulty usage of long running server is requested by many clients, such software "ages" with time and eventually fails. We show that virtualization promotes a new model that allows tolerating in application domains with little overhead. In this paper, we present load and time dependent software rejuvenation policy that can be applied in virtualized environment. We consider the aging behavior of the system by time, while the actual load of the system as well. The behavior of the system is represented through a Stochastic Petri Net (SPN) model. Numerical analysis of the system availability is carried out the SHARPE tool simulation.

Keywords: Availability, Virtualization, Software Rejuvenation policy, Stochastic Petri Net model

1. Introduction

As software continues to become larger and more complex, it is becoming the dominant source of system failure. System failures due to imperfect software behavior are usually more frequent than failures caused by hardware components faults. These failures are the result of either inherent design defects in the software or from improper usage by clients.

System availability can be further enhanced by taking a proactive approach to detect and predict an impending outage of a specific server in order to initiate planned failover in a more orderly fashion.

In this paper, we describe our proposal to offer high availability mechanism for application servers. Moreover, we consider combination of software rejuvenation policy and virtualization technology to improve the system availability. We evaluate a Stochastic Petri Net model however taking into consideration a different performance measure, the steady state probability of successful service instead of the probability of unavailability.

The rest of the paper is organized as follows. Section 2 address related works. Section 3 briefly introduces the background theory. Section 4 discusses the proposed model and software rejuvenation policy. Details of analysis of the software rejuvenation model can be found in Section 4. Finally conclude the proposed system in Section 5.

2. Related Works

Huang et. al. [6] 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, 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.

The authors [3] presented several individual model types such as availability, confidentiality, integrity, reliability, survivability, safety and maintainability. Moreover, they represented by one of the model representation techniques: combinational (such as reliability block diagrams (RBD), reliability graphs, fault trees, attack trees), state-space (continuous time Markov chains, stochastic Petri nets, fluid stochastic Petri nets, etc) and hierarchical. The show case studies for each individual model types.

A. Rezaei et. al. [1] demonstrated how much the proposed method can improve system availability; the stochastic reward net-based models of a typical virtualized consolidated server in cases of using a prediction-based policy, using a time-based policy, and using their new combinatory rejuvenation technique are presented and compared.

The authors [4] studied and compared two rejuvenation models with performance parameters associated with their service performance using two policies: (1) time and (2) load and time based rejuvenation policies. They considered probability of clock and load in their system.

F. Salfner el. al [2] presented and analyzed a coloured stochastic Petri net model of a redundant fault-tolerant system. They have simulated the Petri net model for different levels of utilization and have computed service unavailability in the different model configurations.

S. Garg et. al [5] proposed time and load based software rejuvenation such as policy, evaluation and optimality using software bugs (faults). Moreover, they applied a rejuvenation policy which takes into account the load (represented as an MMPP/M/1/K) on their system.

The critical role of VM and VMM as the single point of failure of a consolidated system has attracted many researches on how to make it more fault-tolerant. The work most closely related to our work is provided by rejuvenation policy papers as well as load and time based software rejuvenation policy. However, they do not consider the creating and combination of rejuvenation and virtualization technology. They only consider rejuvenation process such as time-based and load and time-based rejuvenation policies.

3. Background

Unplanned computer system outages are more likely to be the result of software failures than of hardware failures. In this section, we introduce the concepts of software aging, software rejuvenation and virtualization.

3.1. Software Aging

Software aging refers to progressive performance degradation in the availability of OS resources, data corruption and numerical error accumulation that may lead to system crashes or undesired hang ups. Aging has not only been observed in software used on a mass scale but also in specialized software used in high-availability and safety critical applications such as web server application and enterprise always-on applications. A proactive fault management method to deal with the software aging phenomenon is software rejuvenation.

3.2. Software Rejuvenation

Software rejuvenation is a proactive fault management technique aimed at cleaning up the internal state of the system to prevent the

occurrence of more severe crash failures in the future. It involves occasionally terminating an application or a system, cleaning its internal state and restarting it. Current Methods of software rejuvenation are system restart, application restart (partial rejuvenation) and node/application failover (in a cluster system). An extreme but well-known example of rejuvenation is a system reboot. There are numerous examples in real-life systems where software rejuvenation is being used.

Two major approaches have been proposed: periodical rejuvenation based on time or work performed, and adaptive or proactive rejuvenation. The proactive approach is more efficient and lower in cost, resulting in higher availability. Any rejuvenation typically involves an overhead, but it prevents more severe failures from occurring.

3.3. Virtualization

Virtualization is a rapidly growing new technology that is transforming the world of Information Technology. Virtualization allows multiple operating system instances to run concurrently on a single computer; it is a mean of separating hardware from a single operating system.

4. Load and time based software rejuvenation policy

The software starts up in a "robust or healthy" state without the probability of failure. Due to repeated and potentially faulty usage of long running client-server type software systems by many clients, such software "ages" with time and eventually fails. According to software degrading, "robust" state changes "failure (crash)" state with a non-zero probability. Once it crashes, it takes a random amount of time to bring the "healthy" again to the clean state and restart it.

4.1. System Architecture

In Figure 1, we describe our proposal to offer high availability mechanism. Our approach makes use of several concepts: a virtualization, a clustering and software rejuvenation. We consider an active/standby virtual machine based on single physical server. If failures are generated in the active virtual machine (VM), standby virtual machine takes over the resources of the active VM and continuously provides a corresponding service based on the resources. Moreover, we consider by adding load and time based rejuvenation policy in our proposed model.

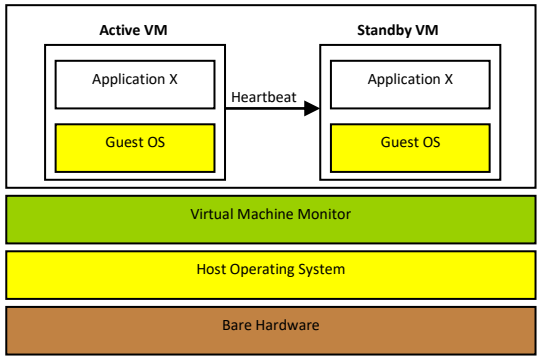

Figure 1. System Architecture of Proposed Model

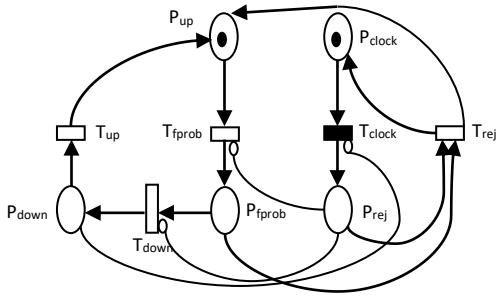

Figure 2. MRSPN Model of the System Behavior without Virtualization

We show the example of Markov Regenerative Stochastic Petri Net (MRSPN) model for the system behavior without virtualization in Figure 2.

If the software was in the robust state when T_{clock} fires, then after rejuvenation is complete, T_{rej} fires to reinitialize the net, with a rate equal to λ_{rej1}. If the software had reached the failure probable state (token in place P_{probf}), then fires to complete the rejuvenation and reinitializes the net; a rate equal to (with $\lambda_{rej1} \geq \lambda_{rej2}$) is assumed in this case.

4.2. Stochastic Petri Net Model

In this paper, we propose a different rejuvenation strategy which is strictly related to the load offered to the system in a given time constant. The strategy we want to analyze determines the Stochastic Petri Net model depicted in Figure 3.

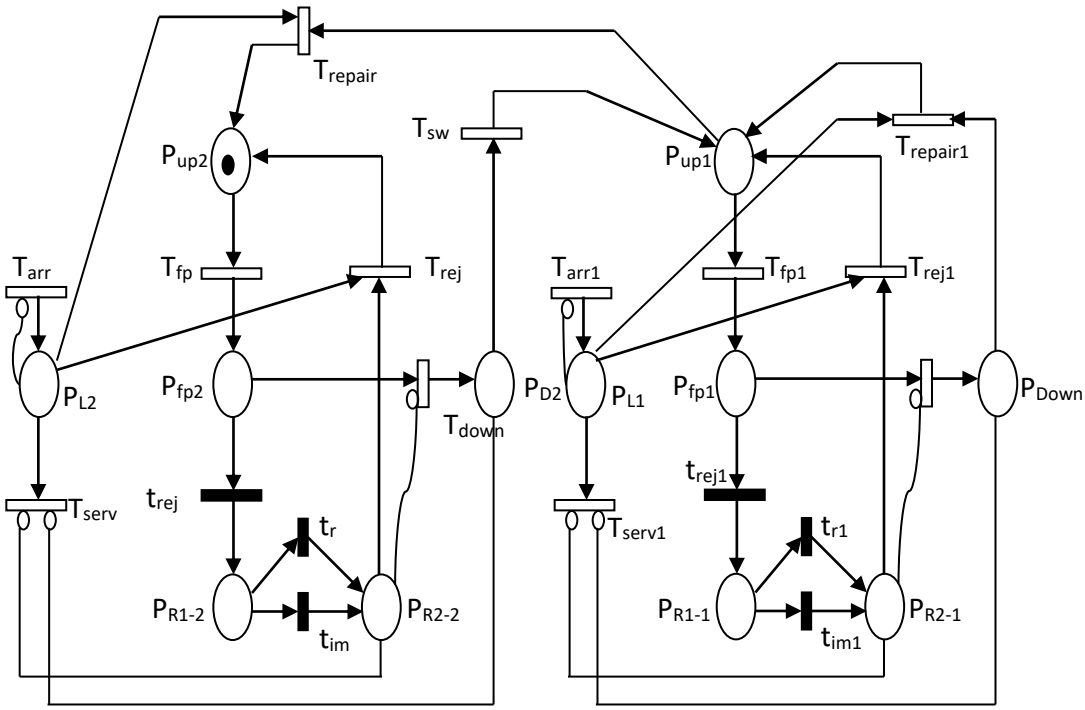

Figure 3. Load and time dependent rejuvenation policy with Virtualized state (2VMs1P)

The SRN model for load and time based rejuvenation policy in single physical server is presented in Figure 3. It consists of active-standby virtual machine servers. The transition T_{arr} and T_{arr1} model the arrival process (exponentially distributed inter-arrival times have been assumed) of requests which are stored in a buffer model through place P_{L2} and P_{L1}. The transition T_{serv} and T_{serv1} model the service time of the process. Both VMs are "healthy" working states, indicated by a token in place P_{up2}. As time progresses, active VM eventually transits to failure probably states in place P_{fp2} and P_{fp1}

through the transition T_{fp} and T_{fp1}. The VM is still operation in this state. At that time, VM can be enter the failure state (Place P_{D2}) through the firing transition $T_{down..}$ But VM can be switch over to another standby VM. When transition T_{sw} is enabled, the operation of active VM is switched to standby VM and a token is moved to a place P_{up1}. After that operation will be restarted on standby VM. On the other hand, the VM can be rejuvenation. In this model, rejuvenation interval is determined by using a clock with guard function ghinterval and there are tokens in the place P_{clock1} and P_{pf} and P_{pf1}. If there is a token in place $P_{trigger}$, and the system enters into a state (modeled by a token in place P_{R1-2} and P_{R1-1}) in which a decision should be taken if rejuvenate or not according to the status of the input buffer, (a token is placed in P_{pf} and P_{pf1}). More precisely, if the load to the system is below a given threshold (inhibitor arcs from P_{L2} and P_{L1}) immediate transitions t_{im} and t_{im1} are enabled 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 by using a clock with guard function ghinterval1 and there are tokens in the place $P_{clock2\ and}$ P_{R1-2} and $P_{R1-1.}$ If there is a token in place $P_{trigger1}$, and the system enters into a state (modeled by a token in place P_{R2-2} and P_{R2-1}) in which a rejuvenation should be taken. After VM has been rejuvenated, it goes back to healthy state with transition T_{rej} and T_{rej1}. Also immediate transition t_{reset} and t_9 are enabled and a token is moved to P_{clock1} and P_{clock2}. Standby VM has the same rejuvenation policy. When there is a physical server is crash (i.e., there is a token is placed in place P_{Down} by using transition T_{down1} with guard function. From a full system outage, the system can be repaired through the transition T_{repair} and $T_{repair1}$, all VMs are in healthy state in place P_{up2}.

Table 1. System Assumptions

Places	Descriptions
Pup	Virtual Machines are 'robust' or healthy state
Pup2	Virtual Machines are 'robust' or healthy state
Pup1	Virtual Machines are 'robust' or healthy state
Prej	Rejuvenation State
PR1-2,PR2-2	Rejuvenation State
PR1-1,PR2-1	Rejuvenation State
Pfprob	Failure Probably State
Pfp2	Failure Probably State
Pfp1	Failure Probably State
PD2	Failure Probably State
Pdown	Failure State
PDOWN	Failure State

4.2.1. Structure of Reachability graph

In this section, we construct the reachability graph for the proposed model. This graph represents the active physical server with two virtual machines (2VMs1P) at both active-standby virtual machines. Let the 12-tuple ($\#P_{up2}$, $\#P_{fp1}$, $\#P_{R1-2}$, $\#P_{R2-2}$, $\#P_{D2}$, $\#P_{up1}$, $\#P_{fp1,}$ $\#P_{R1-1}$, $\#P_{R2-1}$, $\#P_{Down}$, $\#P_{L2,}$ $\#P_{L1}$) denote the markings of the Petri net model. The dimension of the buffer is limited to k elements (the inhibitor arc from P_{L2} and P_{L1} to T_{arr} and T_{arr1} models the finite dimension of the buffer). A marking is reachable from another marking if there exists a sequence of transition firings starting from the original marking that result in the new marking. P_{L2} and P_{L1} represents '*'(buffer size k = 1 in this graph).

Figure 4 illustrates the reachability graph with squares representing the markings and arcs representing possible transition between

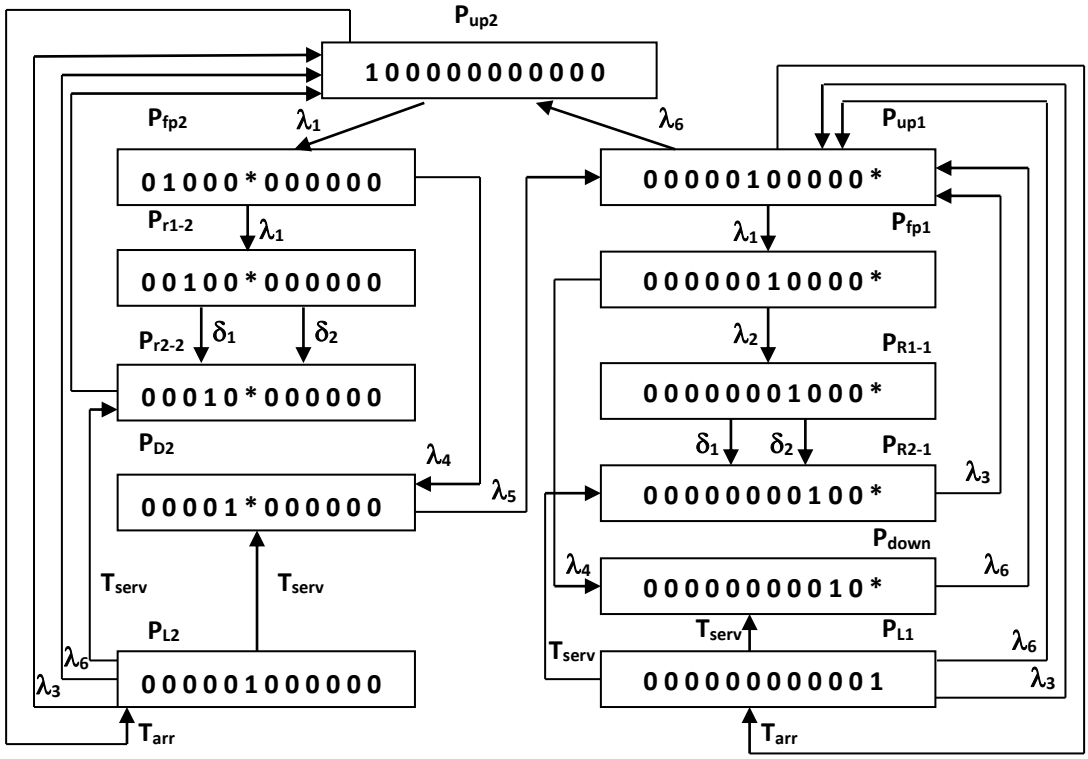

Figure 4. Reachability graph of the load and time based model for 2 VMs on single physical server

the markings. By mapping through actions to this reachability graph with stochastic process, we get mathematical steady-state solution of the chain. We apply λ_1, λ_2, λ_3, λ_4, λ_5, λ_6, δ_1 and δ_2 as well as T_{fp} and T_{fp1}, t_{rej} and t_{rej1}, T_{rej} and Trej1, Tdown and Tdown1, Tsw, Trepair and Trepair1, tr and tr1, and tim and tim1 in reachability graph.

We may compute the steady-state probability by first writing down the steady-state balance equations of figure as follows.

$$\lambda_1 P_{up2} + T_{arr} P_{up2} = \lambda_3 P_{L2} + \lambda_6 P_{L2} + \lambda_3 P_{R2-2} + \lambda_6 P_{up1} \quad (1)$$

$$\lambda_2 P_{pf2} + \lambda_4 P_{pf2} = \lambda_1 P_{up2} \quad (2)$$

$$\delta_1 P_{R1-2} + \delta_2 P_{R1-2} = \lambda_2 P_{pf2} \quad (3)$$

$$\lambda_3 P_{R2-2} = \delta_1 P_{R1-2} + \delta_2 P_{R1-2} + T_{serv} P_{L2} \quad (4)$$

$$\lambda_5 P_{D2} = \lambda_4 P_{pf2} + T_{serv} P_{L2} \quad (5)$$

$$2T_{serv} P_{L2} + \lambda_3 P_{L2} + \lambda_6 P_{L2} = T_{arr} P_{up2} \quad (6)$$

$$(\lambda_6 + T_{arr1}) P_{up1} = \lambda_5 P_{D2} + \lambda_3 P_{R2-1} + \lambda_6 P_{DOWN} + (\lambda_3 + \lambda_6) P_{L1} \quad (7)$$

Solving the steady-state balance equations, we find,

$$P_{fp2} = B * P_{up2} \quad (8)$$

$$P_{R1\text{-}2} = [C * B] * P_{up2} \quad (9)$$

$$P_{R2\text{-}2} = [((\frac{\lambda_2}{\lambda_3}) * B) + ((\frac{T_{serv}}{\lambda_3}) * A)] * P_{up2} \quad (10)$$

$$P_{D2} = [((\frac{\lambda_4}{\lambda_5}) * B) + ((\frac{T_{serv}}{\lambda_5}) * A)] * P_{up2} \quad (11)$$

$$P_{up1} = [\frac{D}{E\text{-}F\text{-}G\text{-}H}] * P_{up2} \quad (12)$$

$$P_{fp1} = [B] * P_{up1} \quad (13)$$

$$P_{R1\text{-}1} = [C * B] * P_{up1} \quad (14)$$

$$P_{R2\text{-}1} = [((\frac{\lambda_2}{\lambda_3}) * B) + ((\frac{T_{serv}}{\lambda_3}) * A)] * P_{up1} \quad (15)$$

$$P_{L2} = A * P_{up2} \quad (16)$$

$$P_{L1} = A * P_{up1} \quad (17)$$

$$P_{DOWN} = [((\frac{\lambda_4}{\lambda_6}) * B) + ((\frac{T_{serv}}{\lambda_6}) * A)] * P_{up1} \quad (18)$$

Where,

$$A = \frac{T_{arr}}{2T_{serv} + \lambda_3 + \lambda_6}$$

$$B = \frac{\lambda_1}{\lambda_2 + \lambda_4}$$

$$C = \frac{\lambda_2}{\delta_1 + \delta_2}$$

$$D = (\lambda_4 * B) + (T_{serv} * A)$$

$$E = (\lambda_6 + T_{arr1})$$

$$F = ((\lambda_1 * B) + (T_{serv} * A))$$

$$G = ((\lambda_4 * B) + (T_{serv} * A))$$

$$H = [(\lambda_3 + \lambda_6) * A]$$

4.2.2. Model Analysis

In the proposed model, services are not available when both active and standby VMs are down.

$$\textbf{\textit{Availability}} = 1 - P_{DOWN} \quad (19)$$

In this section, we focus on the applicability of the proposed model and solution methodology through numerical examples. We apply stochastic Petri Net based approach to build the model. For this purpose, the parameters are chosen as shown in Table 1. The transition firing rates which are required to test the proposed model from literature review in [6].

Table 2. System Operating Parameters

Transition	**Firing Rate (hr^{-1})**
T_{serve}	1 time/1 hour
T_{fail}	3 times/month
T_{probf}	1 time/a day
T_{hw}	$1/10^6$
T_{repair}	2 times/a day
$1/T_{rej1}$, $1/T_{rej2}$	5 mins
$1/T_{sw}$, $1/T_{swbk}$	3 mins
T_{clock1}, T_{clock2}, T_{imm} T_{arr}	Variable

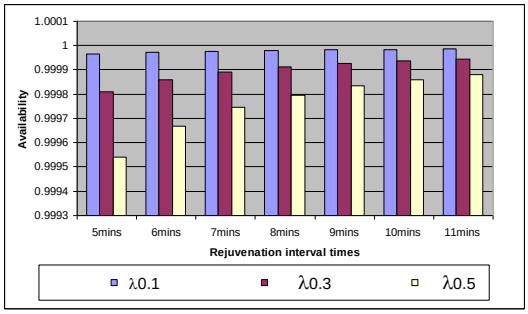

Figure 5. Availability vs different rejuvenation interval and arrival rates

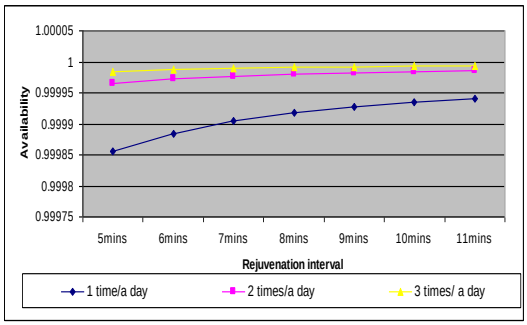

Figure 6. Availability vs different rejuvenation interval and repair rates

To examine the influence of rejuvenation triggering rates and recovery rates (rejuvenation service rates) on system availability, we set the value of arrival rate is range from 0.1 to 0.5. The rejuvenation service time is range from 5 mins to 11mins. The plot of system availability in the case of performing rejuvenation is shown in Figure 5 and 6 using different rejuvenation rates, arrival rates and repair rates. We apply repair rates such as 1 time/a day, 2 times/a day and 3 times/ a day.

From our results, it is apparent that our proposed system is a cost-effective way to build high availability system and virtualized clustering technology can improve the software rejuvenation process.

5. Conclusion

In this paper, we have shown that the time to rejuvenate is equal to the time to recover from a failure, a load and time based policy beneficial whereas simply time based rejuvenation does not. The numerical results are validated with the evaluation through SHARPE tool simulation. It is found that the derivation results and the SHARPE result are the same. Therefore, the results showed that combining virtualization technology and software rejuvenation methodology can improve proposed model's performance.

References

[1] A.Rezaei and M.Sharifi, “Rejuvenating High Available Virtualized Systems”, International Conference on Availability, Reliability and Security, 2010.

[2] F.Salfner and K.Wolter, ”A Petri Net model for Service Availability in Redundant Computing Systems”, Proceedings of the 2009 Winter Simulation Conference, 2009.

[3] K.S.Trivedi, D.S.Kim, A.Roy and D.Medhi, ”Dependability and Security Models”, 7th International Workshop on the Design of Reliable Communication Networks, 2009.

[4] S. Garg, A. Pfening, A. Puliafito, M.Telek and K. S. Trivedi," Modeling and Analysis of Load and Time Dependent Software Rejuvenation Policies", Research Gate, Journal Article, 2001.

[5] S. Garg, Y. Huang, C. Kintala and K. S. Trivedi, "Time and Load Based Software Rejuvenation: Policy, Evaluation and Optimality", Fault-Tolerant Systems, IITMadras, December 20-22, 1995.

[6] Y. Huang, C. Kintala, N. Kolettis and N. D. Fultion, "Software Rejuvenation: Analysis, Module and Applications", June 1995.