# Chapter 8
# Software Aging and Rejuvenation for Increased Resilience: Modeling, Analysis and Applications

**Alberto Avritzer, Ricardo M. Czekster, Salvatore Distefano and Kishor S. Trivedi**

**Abstract** Software aging and rejuvenation research has shown that the application of approaches for software aging modeling, monitoring, and rejuvenation has the potential to significantly increase software resilience. In this chapter, we present an overview of important analytical models and measurement approaches for software aging and rejuvenation. We start by describing the Markov based approaches and renewal process based approaches for software aging and rejuvenation modeling. In addition, we present measurement based approaches using both online and offline methods for software rejuvenation. We conclude by presenting a categorization of the approaches and by presenting a brief overview of applicability of each of the approaches presented in this chapter.

A. Avritzer (✉)
Siemens Corporate Research and Technology, 755 College Road East, Princeton NJ 08540, USA
e-mail: alberto.avritzer@siemens.com

R. M. Czekster
PUCRS/Faculdade de Informatica, Avenida Ipiranga, 6681, Predio 32, Sala 505, CEP 90619-900 Porto Alegre, Brazil
e-mail: ricardo.czekster@pucrs.br

S. Distefano
Dipartimento di Elettronica e Informazione (DEI), Politecnico di Milano, Piazza L. da Vinci, 32, 20133 Milan, Italy
e-mail: distefano@elet.polimi.it

K. S. Trivedi
Department of Electrical and Computer Engineering, Duke University, Durham NC 27708, USA
e-mail: kst@ee.duke.edu

K. Wolter et al. (eds.), *Resilience Assessment and Evaluation of Computing Systems*,
DOI: 10.1007/978-3-642-29032-9_8, 

## 8.1 Introduction

The introduction of software for monitoring and control of mission-critical systems has created a need for the validation of the resilience and safety of these systems. The activities required for the assessment and enforcement of these systems reliability and availability include requirements, architecture, modeling, testing, online monitoring, and software rejuvenation.

In this chapter we present models, algorithms and applications of software rejuvenation to increase software resilience. This chapter is closely related to the chapter on resilience assessment based on performance testing, where performance measurement results of smoothly degrading systems were presented. Smooth performance degradation has been also called software aging and is a consequence of the exhaustion of system resources, such as system memory or kernel structures, invalid pointers, the accumulation of round off errors, database deadlocks, and the contention for a pool of limited software resources. Therefore, transient application faults and operating system faults can be a major source of system performance degradation. Examples of operating system related faults are invalid allocation or deallocation of memory, kernel data corruption, and incorrect or sub-optimal kernel resource management [369, 895, 905].

Software aging research was initially directed towards the implementation of data collection tools for monitoring of application and operating system resources [72, 73, 181, 369, 895]. The development of stochastic models of software aging and the parameterization of these models with the time to failure distribution, the input workload, and its influence on software aging were presented in [303, 585, 905]. The *xSeries Software Rejuvenation Agent* (SRA) [181] is a tool introduced by IBM and Duke University to monitor system resources and to calculate the expected time to resource exhaustion. Approaches to monitor a customer-affecting metric, such as response time, to detect software aging due to resource exhaustion or security intrusions were introduced in [65–68].

The types of software faults that cause software aging have been shown to be very difficult to test, reproduce, and correct [392]. Some examples of major software outages that were attributed to software aging were reported in [111, 400]. The *Patriot* anti-missile software aging event allowed a *Scud* missile to penetrate US defenses, when the *Patriot* software started to miscalculate routes. This software aging event led to the death of U.S. soldiers during the Gulf War [401, 625]. The event investigation concluded that the problem was caused by a numerical accumulation error that was never caught during testing. Therefore, the system was deployed in production with the faulty software. The investigation report recommendation was to periodically restart the guiding system every eight hours of continuous operation to reset the accumulated variables to their initial valid states.

Software Rejuvenation is a mechanism to proactively and efficiently counteract the effects of software aging [72, 74, 449, 799, 895, 906]. Software rejuvenation architecture artifacts are a good match for complex industrial mission-critical applications that are susceptible to software aging. For example, the process of quickly

shutting down and restarting a given process is a successful strategy to clear internal data structures and replenish system resources to their original specification. The main purpose of introducing software rejuvenation into the architecture of mission-critical systems is to proactively restore the critical system resources to full capacity before a customer impacting failure occurs. Software rejuvenation functions as preventive maintenance to ensure high availability. Software rejuvenation cost effectiveness depends on the state of the environment [74], the state of the mission [68], and the extent of system degradation. The addition of monitoring for software aging, and software rejuvenation as architecture artifacts, have been shown to be a cost-effective approach to increase the resilience of large industrial mission-critical systems [41, 86, 304, 305]. These systems are susceptible to software aging because of their complexity and the high cost of finding and correcting transient software faults. Software rejuvenation architecture artifacts have been applied to telecommunication billing and provisioning data [74, 449], transaction processing system [368], operating systems [904, 905], cluster systems [903], cable modem termination systems [599], web Servers [585], worm mitigation in tactical MANETs [67], and virtualization [828]. Dynamic software rejuvenation algorithms that are based on online monitoring of the environment and of the real-time system performance, can outperform static algorithms, for systems where mission success is dependent on real-time performance [66]. In addition, several empirical studies have identified relevant customer affecting metrics and the best software rejuvenation trigger interval for different applications [41, 86, 304, 305].

Examples of software rejuvenation approaches are the rebooting of a process, releasing of memory, clearing of a deadlock, or performing any other fast action that would prevent software aging from manifesting itself as a system wide failure that could lead to a system crash. These system wide crashes can cause significant damage to the mission the software is controlling and to the infrastructure that is being used to support the software system. For example, a database corruption could take significant time to recover from.

In this chapter, we present models of software aging and different algorithms that were developed to counter aging and security intrusions by applying the so called software rejuvenation techniques.

The outline of the chapter is as follows. Section 8.2 presents a review of the analytical models that were developed for capturing the effects of software aging and for providing recommendations for the best times to trigger software rejuvenation. Section 8.3 presents measurement based studies of software aging and rejuvenation. Section 8.4 presents our conclusions.

## 8.2 Analytical Models

One of the aims of developing analytic models of software aging is to determine optimal times to perform software rejuvenation to maximize software availability, to minimize the probability of loss, to minimize the mean response time of a transaction

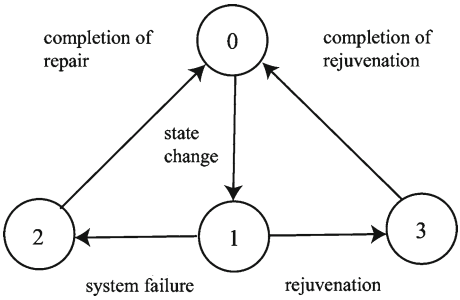


**Fig. 8.1** Basic two-step software rejuvenation model proposed by Huang et al. [449]

(e.g., transaction processing system), or to minimize maintenance costs. Performance optimization is particularly important for business-critical applications for which adequate response times can be as important as system uptime. Modeling and analysis of software aging is done for different kinds of software systems exhibiting varied failure/aging characteristics.

## *8.2.1 Markov Models*

### 8.2.1.1 Continuous-Time Markov Chain

Markov models have been often used in the representation and investigation of software aging and rejuvenation policies [449]. Although the software aging phenomenon is characterized in analytical terms by increasing failure rate (IFR), the first attempts at representing software aging and rejuvenation were based on homogeneous Markov chains [366, 449, 552, 808].

The Markov model used to represent software aging and software rejuvenation is based on a phase-type expansion, where software aging is discretized into a finite number of states of the Markov chain, each characterized by a specific degradation level and a transition rate to the next state. For example, the Markov chain models proposed in [366, 449] restrict the time to failure to be hypo-exponentially distributed. This approach was initially introduced in [449], where a two-step failure model was used with only one degraded state between the initial state (State 0) and the failed state (State 2), as shown in Fig. 8.1. State 1 represents the failure probable state, where a failure would take the system to state 2 and a software rejuvenation trigger would take the system to state 3. The authors solved the model to compute the costs that would be accrued by software rejuvenation and by downtime after a hard failure event. The authors concluded that for the parameters evaluated, software rejuvenation costs would have to be less than 2% of the costs of a hard failure, for software rejuvenation to be cost effective. From this Markov model, the system availability and subsequently the optimal rejuvenation trigger interval was computed.

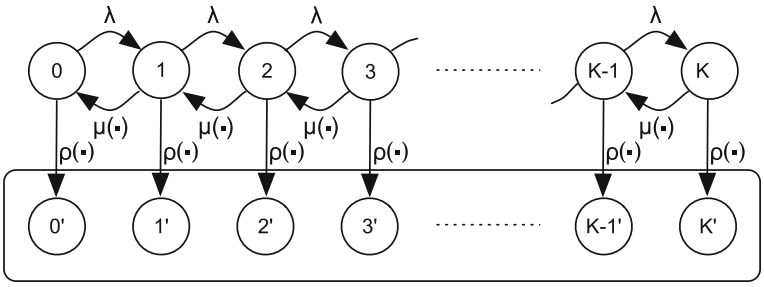


**Fig. 8.2** Subordinated non-homogeneous continuous-time Markov chain of [368]

In [808] a single node system was introduced to investigate the effects of software aging on two different operating systems. The authors compared the effectiveness of several software rejuvenation policies, considering a few alternatives and different system degradation levels (till 10). In this work, system performance degradation was assessed in terms of the system resource utilizations, as for example, processor, memory, and thread utilizations.

#### 8.2.1.2 Non-Homogeneous Continuous-Time Markov Chain

An alternative modeling approach to represent the increased failure rate of the software aging process employs non-homogeneous continuous-time Markov chain model. Non-homogeneous continuous-time Markov chains have been traditionally used for software reliability modeling, and have also been successfully applied to solve software aging and rejuvenation problems [85, 86, 368, 554]. In [368] Garg et al. analyzed a queueing system with preventive maintenance as a mathematical model for a transaction-based software system. The proposed non-homogeneous continuous-time Markov chain model [368] used a time-to-failure function that was generally distributed and a time-varying failure rate to capture the effects of load on software aging. Two software rejuvenation schemes based on the cumulative operation time (before or after the idle time) were investigated. The authors were able to derive the optimal rejuvenation interval $T^*$ under the two policies so as to maximize the steady-state availability, minimize the transactions probability loss, and/or minimize the upper bound on the mean response time. The non-homogeneous continuous-time Markov chain with $K$ states is shown in Fig. 8.2, where the state definition represents the number of transactions queued including the one in service. $K > 1$ is the maximum capacity of the transaction buffer. In Fig. 8.2, $\lambda$ represents the transaction arrival rate, $\mu(\cdot)$ represents the software service rate as an arbitrary function, as it can be constant, or a function of time, load dependent, or a combination of these factors. $\rho(\cdot)$ represents the software failure rate, which is also an arbitrary function. The model is able to capture aging and performance degradation of systems that lose transactions due to software failures.

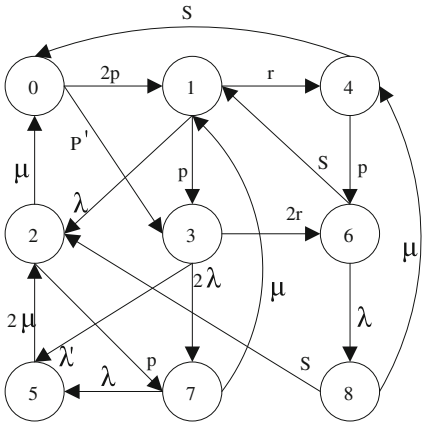


**Fig. 8.3** Markov reward model of [958]

The main goal of [554] was to study the overall behaviour of a software system by modeling the time-dependent rejuvenation rates and deriving an optimal rejuvenation policy using a cyclic non-homogeneous Markov chain. A non-homogeneous continuous-time Markov chain was built to assess the tradeoff between system degradation and software rejuvenation cost.

### 8.2.1.3 Markov Decision Process and Reward Model

Another Markov modelling framework that was applied to assess software aging and rejuvenation is the Markov decision process. In [726] the authors developed a Markov decision process based framework to compute optimal rejuvenation schedules. The optimal rejuvenation schedule solved the optimal stopping problem, as applied to software aging and rejuvenation, by the use of a gradual decrease of the failure rate. The authors have also considered the impact of using realistic cost assumptions and simple rules that could yield an optimal software rejuvenation schedule.

In [958] the authors extended the mathematical characterization of common software-aging-related faults introduced in [449] with a Markov reward model representing a redundant fault-tolerant software system, which is modeled using the software aging and rejuvenation approach introduced in [449]. The proposed model is shown in Fig. 8.3 and represents the states of the joint state of the two parallel software systems, where each individual software system can be in the states defined in [449]. For example, in state 0 both systems are operating correctly. In state 1 one software system is operating correctly and the other one is in the failure probable state. In state 3 both software systems are in the failure probable state, while in state 2 one of the software systems is operating correctly and the other has failed. The other states are derived similarly.

### *8.2.2 Renewal Processes*

Non-Markovian processes shall be employed when the exponential distribution for the time to failure is not sufficient to model the system under study. Renewal theory provides the tools to adequately represent more complex aging processes or rejuvenation policies. Semi-Markov and Markov regenerative processes have been widely used for representing software aging and software rejuvenation.

#### 8.2.2.1 Semi-Markov Process

In [304] the authors developed a semi-Markov model by generalizing the continuous-time Markov chain approach introduced in [449]. The optimal software rejuvenation schedules were analytically derived to optimize the steady-state availability objective and the average cost expended by the software rejuvenation approach. In addition, non-parametric statistical algorithms to estimate the optimal software rejuvenation schedules were also developed.

In [85, 86] the authors used a semi-Markov model to represent a high-level proactive fault management approach. The main contribution of this work was the development of an hierarchical modeling approach composed of a lower level non-homogeneous continuous-time Markov chain and an upper level semi-Markov model. The hierarchical software rejuvenation model triggers different software rejuvenation actions depending on the degradation level that the system has experienced, the time elapsed since the last software rejuvenation event, or any other specific criterion that needs to be modeled [85, 86]. The first-level software rejuvenation, or partial rejuvenation, consists of stopping and rejuvenating certain applications, while the second-level software rejuvenation, or full rejuvenation, consists of stopping all the running applications and restarting the system. Therefore, the first-level software rejuvenation incurs lower cost as measured by system downtime than the second-level software rejuvenation. The hierarchical model proposed in [85, 86] allows for decomposition of the analysis by first evaluating the impact of resource leakage and then assessing the effectiveness of software rejuvenation on the metric of interest.

The tradeoff between using the partial or the full software rejuvenation approach is studied in [553], where a computer system with one standby redundant node was evaluated. The authors considered five different software rejuvenation models for the redundant system, evaluating the steady-state behaviour and the asymptotic availability for each rejuvenation model.

A semi-Markov (SMP) process has been used in [717] to model the availability of personal computer-based active/standby cluster system with software rejuvenation to handle software related system failures. Software rejuvenation and switchover states were mapped into a semi-Markov model whose analysis provided the steady-state availability.

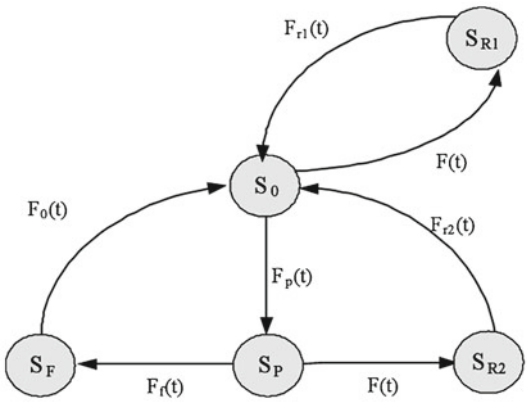


**Fig. 8.4** The MRGP proposed in [423]

### 8.2.2.2 Markov Regenerative Process

Markov regenerative processes have been successfully applied in several software rejuvenation studies. For example, the focus of [423] was to solve the problem of system unavailability caused by the restart operation in the rejuvenation phase. The authors proposed a software rejuvenation model of a hot standby architecture, implementing the software restart by switching between the active copy and the backup copy. The Markov regenerative process availability model was created by changing the Markov chain model of [449] by using two states to represent two different software rejuvenation policies, as shown in Fig. 8.4. The authors have validated their model by comparing the system availability obtained using the model of [449] with the system availability with the proposed hot standby architecture.

Examples of the application of Markov regenerative process software aging and rejuvenation are [309, 368, 926]. As discussed above, in [368] the Markov regenerative process is used on top of a non-homogeneous continuous time Markov chain thus composing a hierarchical model. In [926] three time-based rejuvenation policies used to improve the performability measures of a cluster system under varying workload were evaluated. Similarly, in [309] Markov regenerative process has been used to evaluate the software aging process that was studied by the authors.

In [120] a fine grained software degradation model was proposed, where the current software degradation level could be observed based on the monitoring of a system parameter. In this work, the degradation process consists of a sequence of additive random shocks. The system is considered out of service as soon as the appropriate parameter reaches an assigned threshold level. The system model is a complex reward-renewal processes that is analyzed using the theory of renewal processes with cost/rewards. The approach was used to analyze the impact of system parameters and two alternative rejuvenation policies on a redundant database management system unavailability.

### *8.2.3 Petri Nets*

Petri nets are one of the modeling frameworks used to evaluate software aging and software rejuvenation approaches, because Petri net models can accurately incorporate the most common characteristics of computer systems like concurrency, synchronization, sequencing, and queueing for multiple resources.

Petri nets have been mainly used as a modeling notation. The underlining stochastic processes are derived by using specific techniques. Markov chains, renewal theory, phase type expansions, simulation and similar solution techniques have been used in the evaluation of the Petri nets underlying processes.

Several different Petri net variations have been used to model software aging characteristics and software rejuvenation approaches. One of the specific requirements of software aging modeling is the ability to represent non-Markovian behaviors. Modeling of software aging processes have to take into account the age/history of the software, which can be approximated using Markov models [400].

#### 8.2.3.1 Stochastic Petri Nets

One of the first attempts to apply Petri nets in software rejuvenation is reported in [367]. The authors used the Markov regenerative stochastic Petri net of Fig. 8.5a to deal with a deterministic software rejuvenation trigger interval. The system is fully operational in the place $P_{up}$. When the $T_{fprob}$ transition fires, which represents software aging, a token reaches the place $P_{fprob}$, where the system is the failure probable state. The system is in the crash state after the firing of transition $T_{down}$. While the system is restarting, all transactions are suspended, as shown by the inhibitor arc from $P_{down}$ to the $T_{clock}$ transition, which models the periodic software rejuvenation trigger. $T_{clock}$ fires when the clock expires, if it has not been inhibited. The other transitions are understood similarly.

Another interesting model was described in [117], where the fluid stochastic Petri net shown in Fig. 8.5a was used. The Petri net formalism allows the modeler to represent software aging and software rejuvenation in systems that use specific techniques for software rejuvenation, restoration, and checkpointing. Specifically, a fluid flow approximation approach can be used to model the software aging process also taking into account the workload condition, where the fluid level at a certain time $t$ represents the extent of system degradation that has occurred up to $t$.

Another type of Petri net often used to model software aging and rejuvenation is the deterministic stochastic Petri net, used for representing the cluster system described in [926]. The performability metric was evaluated by the numerical analysis of the underlying subordinated Markov chain. The software rejuvenation policies was there evaluated by considering both the historical data and the current running state of the system.

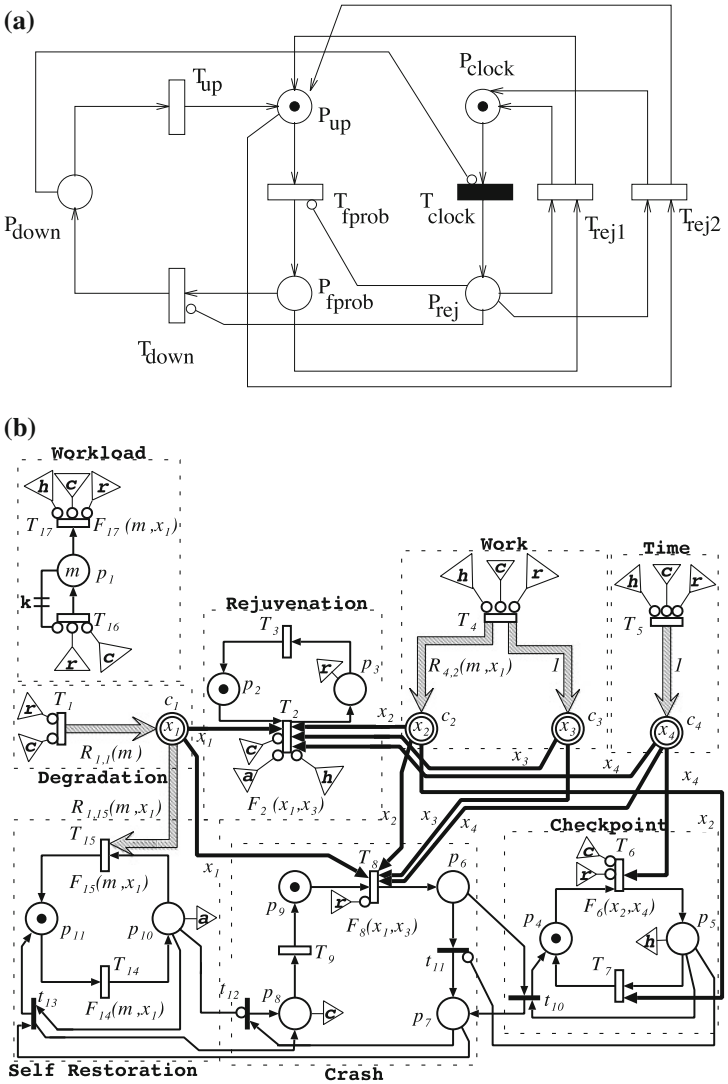


**Fig. 8.5** MRSPN (**a**) and FSPN (**b**) software-aging/rejuvenation models proposed in [117, 367], respectively

### 8.2.3.2 Stochastic Reward Nets

Another Petri net notation used in the software aging and software rejuvenation context is the stochastic reward notation. Stochastic reward nets are particularly suitable to model software aging and rejuvenation approaches, since this formalism allows for modeling the software aging process by using reward rates and guards,

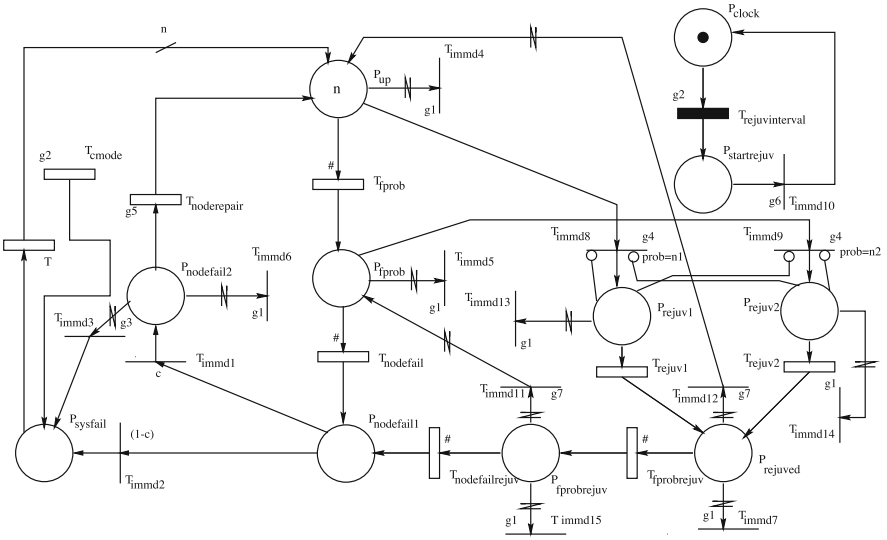


**Fig. 8.6** The SRN proposed in [903] for evaluating cluster systems software rejuvenation policies

among other features. Therefore, stochastic reward nets can be used to represent complex aging processes rejuvenation policies and other quantities such as, for example, the cost [181, 489, 598, 903, 946, 951].

In [903] an evaluation of the application of software rejuvenation to cluster systems was performed. The stochastic reward net introduced in [903] is shown in Fig. 8.6. Software rejuvenation was shown to significantly improve the evaluated system availability and productivity. Both time-based and prediction-based software rejuvenation techniques were there evaluated by modeling, using stochastic reward nets. In [181] the application of software rejuvenation to cluster systems using a real case study related to the xSeries IBM cluster was presented.

Stochastic reward net models were applied to study the cable modem termination system cluster in [598]. Capacity-oriented availability and downtime cost with and without deploying software rejuvenation were evaluated showing significant availability improvement and downtime cost reduction when time-based and condition-based (a prediction-based) rejuvenation strategies were used.

Fan Xin-yuan et al. [946] analyzed the dispatch-worker based cluster system and proposed a Stochastic Reward net model for dispatch-worker based architectures with prediction-based software rejuvenation.

In [951], dependencies among cluster nodes were taken into account in the evaluation of software aging and time/prediction-based software rejuvenation strategies by using a stochastic reward net. The authors evaluated the impact of these rejuvenation strategies on the cluster system reliability by minimizing the software rejuvenation cost, while considering the comprehensive relations between nodes in the entire system.

**Table 8.1** Applicability domain of the software aging and rejuvenation model types

| Model type | Observations |
|---|---|
| Continuous time Markov chains (CTMC) | First attempt to describe software aging and rejuvenation, cost model [449, 552, 808] |
| Non-homogeneous CTMC (NHCTMC) | Derivation of optimal rejuvenation intervals [85, 86, 368, 554] |
| Queueing networks (QN) | Classic approach to performance evaluation |
| Markov decision processes (MDP) | Compute optimal rejuvenation schedules [726, 958] |
| Petri nets (PN) | Higher level representation of models that incorporate synchronization, sequencing, queueing and concurrency [400] |
| Stochastic Petri nets (SPN) | Could be used to compute fluid flow approximations [117, 367, 926] |
| Markov regenerative processes (MRP) | Represent different time-based rejuvenation policies [120, 303–305, 309, 423, 926] |
| Markov regenerative SPN (MRSPN) | Representation of aging for software with two or more components |
| Markov reward models (MRM) | Describe models that calculate costs of different rejuvenation policies [958] |
| Stochastic reward networks (SRN) | [181, 489, 599, 903, 946, 951] |
| Semi-Markov process (SMP) | Hierarchical modeling of rejuvenation and switchover states; allows specification of renewal states in contrast to NHCTMCs; SMPs overcome the NHCTMC models lack in representing rejuvenation, allowing to specify renewal states [717] |
| Semi-Markov reward models (SMRM) | Ability to model aging with rewards [85, 86] |

In [489] a mixed time and prediction based software rejuvenation policy was evaluated using a stochastic reward net. The model was used to evaluate the system availability and downtime cost. The authors were able to show that under the same conditions, a mixed time and prediction based software rejuvenation policy could achieve higher availability and lower downtime cost than either one of the time-based and prediction-based software reliability policies.

Table 8.1 presents a summary of different models presented in this section with a brief description of the domain of applicability of each model.

## 8.3 Measurement Based Approaches

The software aging and rejuvenation analytical models either assume that the time to failure distribution of the software is known (in case of time-based software rejuvenation) or that the degradation level of the software system is known (in case of inspection-based software rejuvenation). To facilitate the latter approach, measure-

ment based approaches monitor and collect data on the attributes responsible for determining the health of the executing software. The data is then analyzed to obtain predictions about possible impending failures due to resource exhaustion. The data analysis can be executed online or offline.

In this section we describe measurement-based approaches for detection and validation of the existence of software aging.

Garg et al. [369] introduced an approach for the detection and the estimation of aging in the UNIX operating system. An SNMP-based distributed resource monitoring tool was used to collect operating system resource usage and system activity data from nine heterogeneous UNIX workstations connected by an Ethernet LAN at Duke University. A central monitoring station was used to run the manager program, which was used to send get requests periodically to each of the agent programs running on the monitored workstations. The agent programs in turn obtained data for the manager from their respective machines by executing various standard UNIX utility programs like pstat, iostat and vmstat. For quantifying the effect of aging in operating system resources the metric *Estimated time to exhaustion* was proposed. The objective of the study was to detect aging or a long term trend (increasing or decreasing) in the measured values. This approach assumed that the accumulated depletion of a resource over a time period depended only on the elapsed time. However, it is intuitive that the rate at which a resource is depleted is also dependent on the current workload. An approach to estimate the rate of exhaustion of operating system resources as a function of both time and the system workload was presented in [905, 906].

A methodology based on time-series analysis was used to detect and estimate resource exhaustion times due to software aging in a web server while subjecting it to an artificial workload [585]. The experiments were conducted on an Apache web server running on the Linux platform. The analysis was done using two different approaches: (1) building a univariate model for each of the outputs or, (2) building only one multivariate model with seven outputs. Seven univariate models were built and then combined into a single multivariate model. First, the parameters were analyzed and incorporated into the model with one output and four inputs for each parameter as follows: connection rate, linear trend, periodic series with a period of one week, and periodic series with a period of one day. The autocorrelation function (ACF) and the partial autocorrelation function (PACF) for the output were computed. The ACF and the PACF were used to select the appropriate model for the data [825]. The autoregressive multiple input single output (MISO) model of order 1 (AR(1)) is considered for the single multivariate model, also taking into account the inputs above identified, an autoregressive model with the exogenous input of order 1 (ARX(1)) is specified for each of them, and obtaining seven ARX(1) models. In summary, the models have been combined into a single multiple input multiple output (MIMO) ARX(1) model. The next step after determination of the model orders is to estimate the coefficients of the model by using the least squares method. The first half of the data is used to estimate the parameters and the rest of the data is then used to verify the model. The obtained results show that the predicted values are very close to the measured values.

In [85] a model was developed to account for the gradual loss of system resources, specifically, the memory resource. The model is able to represent both the correct system operation with no memory leakage and the faulty system operation when a memory leak fault is present. The model relates system degradation to resource request, resource release, or resource holding intervals, and memory leaks. These quantities can be monitored and modeled directly from the system data measurements [585].

Cassidy et al. [180] have developed an approach for software rejuvenation of large online transaction processing servers. The authors monitored various system parameters over a period of time and were able to determine that 13 of these parameters deviate from normal behavior just prior to a crash, thus providing sufficient warning to warrant the initiation of software rejuvenation. A feedback control loop approach for software rejuvenation in a web server was presented in [487].

Machine learning [34], Support Vector Machines (SVM) and similar techniques have also been applied to analyze software aging data [443]. Accelerated life testing and accelerated degradation testing techniques have been applied to reduce the time needed for aging approximations [627].

Algorithms for online monitoring of a defined customer-affecting metric have been applied to the security domain. In [68] the effectiveness of the basic bucket-based online monitoring algorithm introduced in [67] was assessed for mission-critical systems by computing the probability of mission success. The analysis results showed that online monitoring and software rejuvenation are very effective in ensuring a high probability of mission success, when the mission-critical system is under attack by a worm infection.

In [71] an application of the basic bucket-based online monitoring algorithm using known system performance signatures was used to detect security intrusions. The research presented in [71] uncovered a significant difference between the performance signatures associated with failure events and the performance signatures associated with security attacks. The performance signatures obtained from the analysis of system failures showed significant system degradation, i.e., CPU values of up to 100%. In contrast, the performance signatures obtained from the analysis of the execution of security test suites, showed that the observed CPU usage values were constrained to a narrow band. A new version of the bucket-based online monitoring algorithm, which was introduced in [71], was able to successfully distinguish between software aging that results from failure events and software aging that results from security intrusion events.

## 8.4 Conclusions

We have presented in this chapter an overview of several software rejuvenation approaches that can be used to increase software resilience by using analytical modeling, offline and online system measurements.

Different analytical models have been applied to the evaluation of software aging and rejuvenation. We have categorized these models according to the stochastic process and the technique used in the analysis. Software can be modeled as a degrading system that is characterized by the software age. The selection of the stochastic process used to represent software aging and rejuvenation policies is driven by the need to incorporate the software age into the model. Markov models are used in software aging as an approximation where software age is represented by using different degradation states. Approximated models that implement a phase type-like discretization of the software degradation into two or more degradation states have been introduced in [449, 552, 808]. One of the benefits of using Markov models to represent software aging is the reduced model solution cost for simple models. However, a large number of degradation states may be required to model the problem with high accuracy, thus increasing the Markov model solution cost.

Non-homogeneous Markov chains have been used to model software aging [85, 86, 368, 554] but to adequately model software rejuvenation more complex models are required as a single global clock will not provide accurate results. Therefore, semi-Markov and the Markov regenerative processes are required to adequately represent software rejuvenation [85, 86, 120, 303–305, 309, 368, 553, 717, 926]. Specifically, semi-Markov processes can be used to specify renewal states. However, semi-Markov processes cannot adequately represent the aging that occurs between regeneration epochs. As a consequence, Markov regenerative processes are used to model software aging in software hierarchies composed of two or more components, by separately modeling each individual component age.

Markov reward models and variants have been often used when different rejuvenation policies have to be evaluated and compared to establish the optimal software rejuvenation policy [958]. The rewards associated with the Markov model states are used to represent and quantify the cost of the software rejuvenation policies being evaluated.

The Petri net formalism can be considered as a higher level of modeling representation, which can be used to provide clarity to the software modeling process. Petri nets have been used to model software aging and rejuvenation problems, because of their compactness and expressiveness [367, 926]. Specifically, stochastic reward nets have been used when software rejuvenation policies have to be compared in terms of their costs to investigate the optimal software rejuvenation policy [181, 489, 598, 903, 946, 951].

Measurement based approaches rely on data collection and analysis for determining system health and the best time to trigger the software rejuvenation routines. Online monitoring of system resources and/or of a customer affecting metric were shown to be a cost-effective approach to ensure software resilience.

Software aging has been originally observed in the telecommunications domain [70, 72, 111, 449], because of strict engineering efforts that were conducted by telecommunication companies to assess service reliability. The detection of smooth degradation of the available resources or software aging, and the periodic workload characteristics of telecommunication applications, led to the development of techniques to counteract aging that were called software rejuvenation. These software

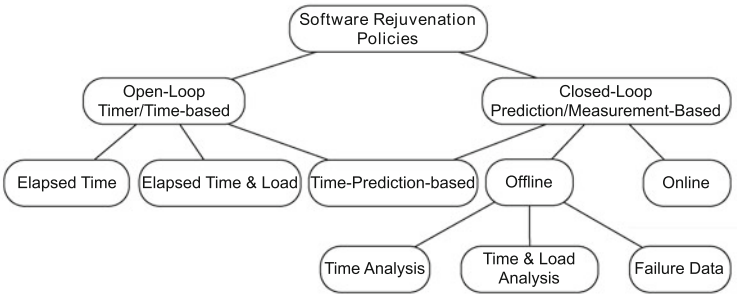


**Fig. 8.7** Software rejuvenation techniques classification

**Table 8.2** Bibliography classification based on Fig. 8.7

| | |
|---|---|
| *Open-loop* | |
| Elapsed time | [55, 85, 117, 120, 304, 367, 449, 553, 554, 717, 726, 808, 958] |
| Elapsed time and load | [86, 309, 366, 368, 423, 926] |
| *Closed-loop* | |
| Offline | |
| Time analysis | [85, 120, 180, 304, 369, 627] |
| Time and load analysis | [585, 905, 906] |
| Failure data | [34, 85, 443] |
| Online | [66–68, 71, 487, 649, 825] |
| *Open/closed-loop* | |
| Time/prediction based | [181, 489, 598, 903, 946, 951] |

rejuvenation techniques were shown to be cost effective to increase system resilience, when the system contains soft faults or is a victim of security attacks. Therefore, we expect that the software aging and rejuvenation approaches described in this chapter will see increased deployment in systems that are designed for resilience.

Figure 8.7 presents a classification of software aging and rejuvenation approaches. The software aging and rejuvenation approaches can be divided broadly into *time-based* (*open loop*) and *prediction-based (closed loop)* approaches. Time-based techniques are usually used in the early stages of software development process and are often implemented by analytical models, as discussed in Sect. 8.2.

In the classification shown in Fig. 8.7, we further characterized the time-based class in terms of the quantities and the parameters taken into account in the model, such as time and load.

On the other hand, in the prediction-based approach, the objective is to monitor and collect data on the attributes responsible for determining the health of the executing software. The data is then analyzed to obtain predictions about possible impending failures due to resource exhaustion. The data analysis can be executed online or offline. Offline techniques have been further characterized according to the type of data available and used in the evaluation:

- *time analysis*—time parameters,
- *time and load analysis*—time and load data,
- *failure data*—for reliability analysis.

Statistical techniques are used to collect and process the data as discussed in Sect. 8.3.

Table 8.2 presents a summary of some of the techniques discussed in the chapter.

**Acknowledgments** We like to thank Dr. Fumio Machida, Ermeson Andrade and Dr. Jing Zhao for their useful comments.