Comprehensive Evaluation of Aperiodic Checkpointing and Rejuvenation Schemes in Operational Software System ⋆

Hiroyuki Okamura and Tadashi Dohi

Department of Information Engineering, Graduate School of Engineering, Hiroshima University, 1–4–1 Kagamiyama, Higashi-Hiroshima 739-8527 Japan

Abstract

This paper examines comprehensive evaluation of aperiodic time-based checkpointing and rejuvenation schemes maximizing the steady-state system availability in an operational software system. We consider two kinds of maintenance policies: checkpointing prior to rejuvenating (CPTR) and rejuvenating prior to checkpointing (RPTC). These schemes are complementary from each other to schedule checkpoints and rejuvenation points. In addition, under a periodic full maintenance operation, we show that aperiodic checkpointing or rejuvenation scheme is optimal to maximize the steady-state system availability by applying the dynamic programming. In numerical examples, CPTR and RPTC are comparatively examined with same overhead parameters, and the effects of CPTR and RPTC on maximizing the steady-state system availability are investigated.

Key words: checkpointing, software rejuvenation, software aging, steady-state system availability, dynamic programming

⋆ This research was supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Young Scientists (B), Grant No. 21700060 (2009-2011) and Scientific Research (C), Grant No. 21510167 (2009-2012).

Email address: {okamu,dohi}@rel.hiroshima-u.ac.jp (Hiroyuki Okamura and Tadashi Dohi).

URL: http://www.rel.hiroshima-u.ac.jp/ (Hiroyuki Okamura and Tadashi Dohi).

1 Introduction

Software fault tolerance is widely accepted as an effective approach to enhance the dependability of software system. The fundamental concept of software fault tolerance is diversity and redundancy. For example, N-version programming, which is a well-known software fault tolerance technique, implements N software components. Each of components has the same function but their designs are different from each other. Once a software component has failed, the system switches over to another component so that it could keep processing without any suspension process. Such software fault tolerance is categorized to the design-diversity technique, namely, it provides an architectural redundancy.

On the other hand, the environment diversity receives much attention as another technique for the software fault tolerance. In general, the design-diversity technique requires a large development cost due to its architectural redundancy. The environment diversity focuses on the computation environment of operational software system. Typical examples of the environment-diversity techniques are reboot, retry and restart of the system. Once a failure occurs on an operational software, the system retries the failed process after refreshing the operation environment with system reboot or restarting. Empirically we know that such a reboot is quite effective to recover transient errors. The environment diversity is based on such empirical experiments, and requires less cost than the design-diversity technique. The environment-diversity technique generally counteracts transient errors that are caused by the faults related to resource exhaustion like memory leaking. Such phenomenon is recognized as *software aging*. The detailed taxonomy of software faults and software aging phenomena are presented in [1].

This paper focuses on two environment-diversity techniques; checkpointing and software rejuvenation. Checkpointing is known as one of the most important techniques in the software fault tolerance. This is a quite simple technique to reduce downtime caused by a system failure in operational phase. Each checkpoint preserves status of a process running on memory at a secondary storage devices such as a hard disk. Even if a system failure occurs, the process can be restarted from the latest checkpoint by referring to the status in the secondary storage device. Then the downtime caused by the system failure may become shorter by controlling placement of checkpoints appropriately. Therefore, an appropriate checkpoint placement maximizes the system availability of operational software systems. On the other hand, placing a checkpoint wasts a time overhead, called a checkpoint overhead, so that the system availability may not be improved if checkpoints are unnecessarily and excessively placed. Of course, since the lack of checkpoints may increase the recovery overhead that is a time overhead to refer to the preserved status,

On contrary, there is a trade-off relationship in the problem of placement of checkpoints. In fact, a huge number of checkpoint creation problems have been considered during the last three decades.

Young [2] obtained the optimal checkpoint interval approximately for restarting a computation process after a system failure. Chandy et al. [3,4], Vaidya [5], Ziv and Bruck [6] proposed some performance models for database recovery and calculated the optimal checkpoint intervals which maximize system availability or minimize an average overhead during normal operation. Since these early contributions, many authors developed checkpoint models to determine the optimal checkpointing schedules with respect to various dependability measures [7–19]. For a good survey of this topic, see Nicola [20]. Most of the above works focused on periodic checkpoint policies, i.e., the case where checkpoint intervals are constant over time. Theoretically this type of policies may be applicable only when system failure time is described as an exponential distributed random variate. However, the periodic policy has been applied in the case where a system failure occurs according to a non-exponential distribution.

Toueg and Babaõglu [21] considered a non-exponential failure case and developed a dynamic programming (DP) algorithm to determine aperiodic checkpointing. Variational calculus approaches in [22–24] are regarded as efficient approximation methods to treat aperiodic checkpoint placement problems. Recently, Dohi et al. [25] and Ozaki et al. [26] reconsidered a sequential checkpoint placement algorithm and dealt with non-constant checkpointing schedules. Okamura et al. [27] also developed a DP-based optimal checkpointing algorithm for real-time applications and refined Toueg and Babaõglu's [21] discrete DP algorithm by extending to continuous time domain. In this way, aperiodic checkpoint schemes have been extensively studied under general operational circumstances described by non-exponential failure times.

Apart from the checkpointing, the environment-diversity technique has been discussed as another fault tolerant technique. In general, we often encounter a system failure caused by the software aging, while the software continuously run for long time. As mentioned before, such aging-related bugs, which lead to resource exhaustion, may exist in operating systems, middleware and application software. The software aging will affect the performance of applications and eventually causes failures [28–30], and has been observed in widely-used communication software like Internet Explorer, commercial operating systems and middleware. In such a situation, the aging phenomenon should be represented by a non-exponential failure time.

A complementary approach to handle software aging and its related transient failures, called *software rejuvenation*, has already become popular as a typical and low cost environment diversity technique of operational software. Software

rejuvenation is a preventive and proactive solution that is particularly useful for counteracting the phenomenon of software aging. It involves stopping the running software occasionally, cleaning its internal state and restarting it. Cleaning the internal state of a software might involve garbage collection, flushing operating system kernel tables, reinitializing internal data structures and hardware reboot. In general, there is also a trade-off relationship between a rejuvenation overhead and downtime due to a system failure.

Huang et al. [31] proposed a continuous-time Markov chain model with random software rejuvenation. Dohi et al. [32, 33] generalized the same model to semi-Markov models with different dependability measures and developed the statistical estimation methods of the optimal software rejuvenation schedule. As an alternative modeling approach, the work in Garg et al. [34] involved arrival and service of transactions in the system, and computed the load and time-based rejuvenation schedule by taking account of the effect of aging as crash/hang failure, referred to as hard failures, and performance degradation, referred to as soft failures. Okamura et al. [35] considered the situation where one running process can be rejuvenated at several time points, and derived an aperiodic rejuvenation sequence to minimize total computation time based on dynamic programming. Subsequently, many authors considered the similar problems as [31], i.e., how to determine the optimal software rejuvenation schedules [32–34, 36–53]. This motivates us to handle the optimal rejuvenation schedule as well as checkpointing schedule.

Although these two software fault tolerant techniques are used for different purposes, it should be noted that they are implemented in a real software operational phase to complement each other. In other words, a unified model to incorporate both checkpointing and rejuvenating is useful to quantify both effects on system availability improvement. In the past literature, very a few papers challenged to this interesting modeling. Gare et al. [54] took account of both checkpointing and rejuvenation for a software system and evaluated its expected completion time. Since their model assumes that a software rejuvenation is triggered at a given checkpoint unless the system fails, the minimization of the expected completion time was solved under a specific maintenance schedule which consists of periodic checkpoint interval and the number of rejuvenation points. Bobbio et al. [55, 56] focused on a modeling technique for software system with rejuvenation, restoration and checkpointing. As an extension of usual stochastic Petri nets [39], they applied so-called fluid stochastic Petri nets to model behavior of the software system and assessed quantitative system dependability. On the other hand, the stochastic models to handle aperiodic checkpointing and rejuvenating are separately discussed in [57] and [58]. However there is no comprehensive evaluation of checkpoint and rejuvenation schemes with respect to maximizing the steady-state system availability.

This paper discusses a general modeling framework to determine the optimal

checkpointing and rejuvenation schedule maximizing the steady-state system availability. In particular, we compare two different checkpoint/rejuvenation placement policies: checkpointing prior to rejuvenating (CPTR) and rejuvenating prior to checkpointing (RPTC). These are simplest policies among the classes belonging to mixed policies with checkpointing and rejuvenating. Since it is well known that global optimal controls tend to be simple under general control arguments, this paper deals with these two policies. Under these policies, the optimal checkpoint or rejuvenation time sequence is aperiodic when one of each is given, although most of existing rejuvenation schemes are periodic. For such aperiodic checkpointing and rejuvenation schemes, we provide DP algorithms to derive the optimal time sequence of checkpoints or rejuvenation points.

This paper is organized as follows. In Section 2, we describe a stochastic model and formulate the steady-state system availability with an aperiodic checkpointing or rejuvenation policy. Section 3 describes the DP approach to obtain the optimal checkpoint or rejuvenation time sequence maximizing the system availability. Numerical examples are presented in Section 4. In these examples, we calculate the aperiodic optimal checkpoint and rejuvenation times in the case where the system failure follows the Weibull distribution. In addition, we fairly compare the maximum steady-state availabilities for CPTR and RPTC under several parametric conditions. Finally the paper is concluded with some remarks in Section 5.

2 Availability Modeling of Operational Software System

2.1 Model Description

Consider an operational software system with software aging phenomenon. Suppose that the system is allowed to generate checkpoints and rejuvenation points. A full maintenance which executes both checkpointing and rejuvenation is performed at every time interval. Without loss of generality, we assume that the last maintenance action is taken at $t = 0$.

Let $F(t)$ be a cumulative distribution function (c.d.f.) of the system failure time, which has the corresponding probability density function (p.d.f.) $f(t)$ with a finite mean time to failure (MTTF). Since the resources of the software system deteriorate with aging, $F(t)$ is supposed to have an IFR (Increasing Failure Rate) property, where the failure rate $r(t) = f(t)/\overline{F}(t)$ is increasing in t, and in general, $\overline{\psi}(\cdot) = 1 - \psi(\cdot)$. The system is supposed to be as good as new at $t = 0$. Let T (> 0) be a time interval of full maintenances. If the software system does not fail until the time T, a full maintenance is applied

to rejuvenate the system and to make a checkpoint.

When a system failure occurs, the system immediately undergoes a recovery process. During the recovery process, the system restores the lost computation with the data stored at the last checkpoint. The recovery process causes a time overhead. More specifically, if a failure occurs at the time when the elapsed time from the last checkpoint is $x \in (0, T)$, the restoration (recovery) overhead is given by $\rho(x)$; for example, if $\rho(x) = \alpha x + \beta$, then αx denotes the time needed to re-execute for the lost computation by the failure, and the second term is a fixed overhead. After the completion of restoration, the system also executes a full maintenance to reduce the overheads caused by additional system failures.

For the software system, we deal with two maintenance policies with respect to the placement of checkpoints and rejuvenation points. One policy generates checkpoints over the time interval of full maintenances (CPTR: checkpointing prior to rejuvenating). In contrast, another policy places rejuvenation points over the time interval of full maintenances (RPTC: rejuvenating prior to checkpointing). Either CPTR or RPTC is applied to maximize the system availability depending on checkpoint and rejuvenation time overheads. Here $\boldsymbol{\pi}_C$ and $\boldsymbol{\pi}_R$ are schedule vectors for CPTR and RPTC, respectively. More concretely, $\boldsymbol{\pi}_C = \{c_1, c_2, \ldots, c_N\}$ and $\boldsymbol{\pi}_R = \{r_1, r_2, \ldots, r_N\}$ are sequences of checkpoints and rejuvenation points over time interval of full maintenances, where N (> 0) is the number of checkpoints or rejuvenation points allowable to be placed during the time interval. Furthermore, μ_c (> 0) and μ_r (> 0) denote time overheads for checkpointing and rejuvenating, respectively. At the completion of checkpointing, the status of software system is recorded for the recovery. On the other hand, if rejuvenation operation is completed, the system becomes as good as new.

Figures 1 and 2 illustrate possible behaviors of software system under CPTR and RPTC, respectively. The main differences of these policies are the length of recovery overhead and the age of failure time distribution. CPTR essentially reduces the length of recovery overhead after the system failure occurs, and RPTC prevents the system failure by executing software rejuvenation.

2.2 Formulation of System Availability

Of our concern here is the formulation of steady-state system availability which is defined as the probability that the software system is operative in the steady state. For this purpose, we define the renewal points at which the full maintenance is executed, and focus on the probabilistic behavior between two successive renewal points, i.e., during one cycle. From the familiar renewal reward argument [59], the steady-state system availabilities with CPTR or RPTC and

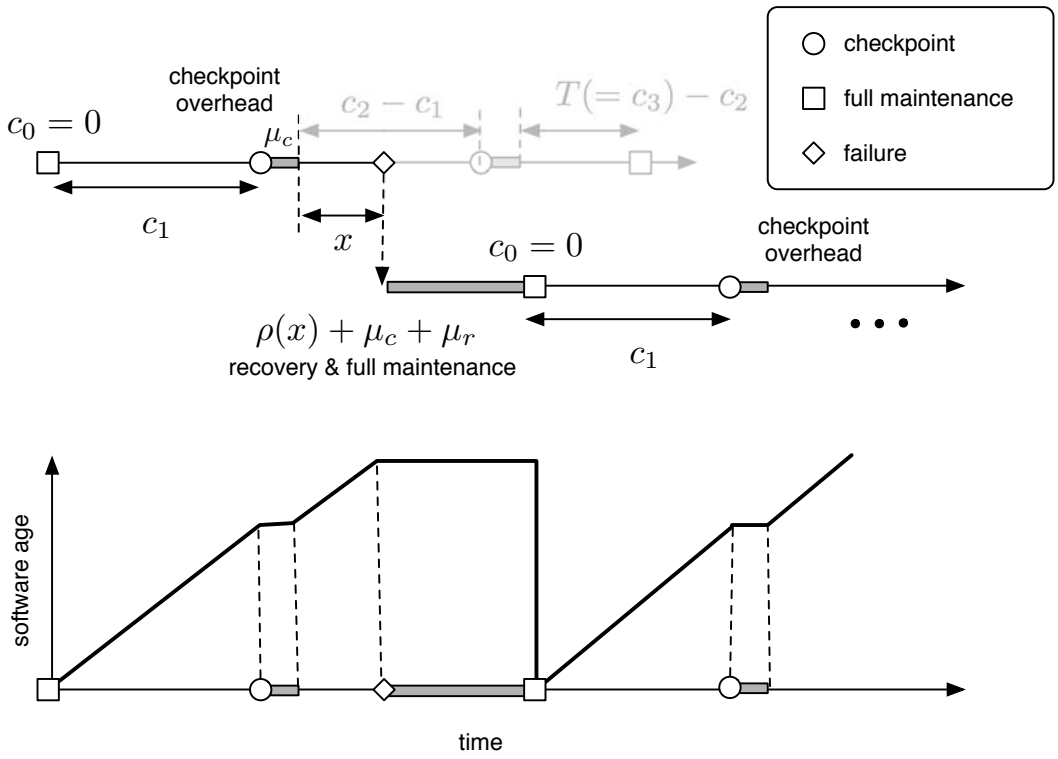


Fig. 1. Possible realization of software system under CPTR.

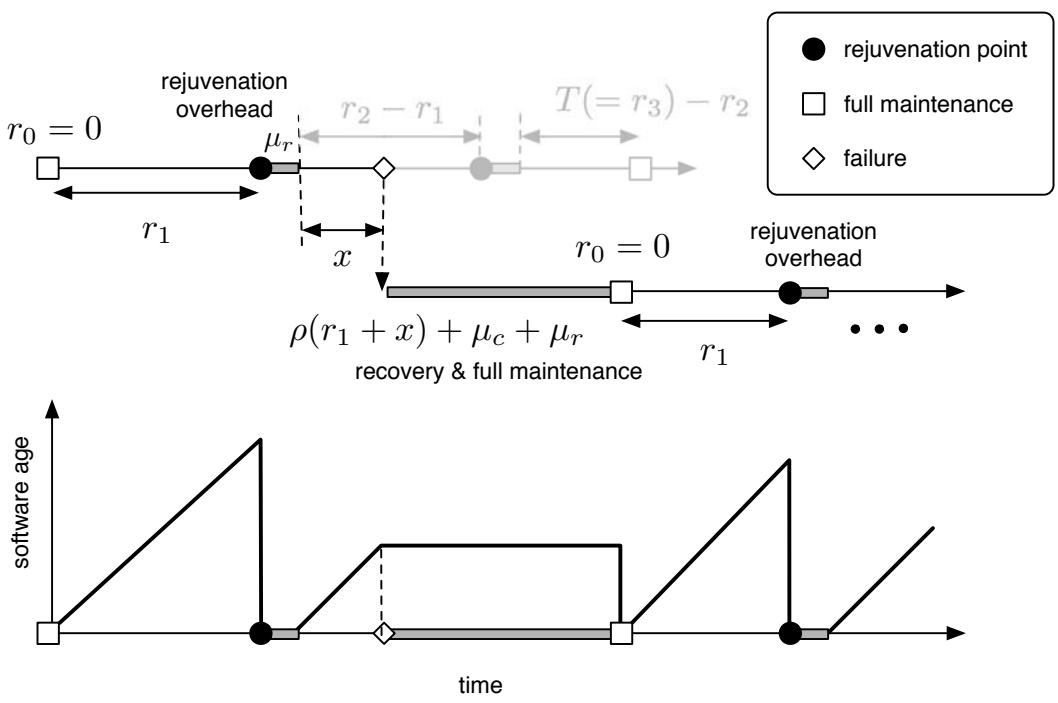


Fig. 2. Possible realization of software system under RPTC.

the full maintenance time interval T are represented by

$$A_C(\boldsymbol{\pi}_C, T) = \lim_{t\to\infty} \frac{\text{E[system operation time during } [0,t)]}{t} = \frac{\text{E[system operation time during one cycle]}}{\text{E[time length of one cycle]}}, \tag{1}$$

and

$$A_R(\boldsymbol{\pi}_R, T) = \frac{\text{E[system operation time during one cycle]}}{\text{E[time length of one cycle]}}, \tag{2}$$

where E denotes the mathematical expectation operator. Then, the problem is to find the optimal schedules $(\boldsymbol{\pi}_C^*, T^*)$ and $(\boldsymbol{\pi}_R^*, T^*)$ under both CPTR and RPTC policies maximizing the steady-state system availabilities, and determine the better policy between CPTR and RPTC by comparing the maximum steady-state system availabilities $A_C(\boldsymbol{\pi}_C^*, T^*)$ and $A_R(\boldsymbol{\pi}_R^*, T^*)$.

For the sake of simplicity, we rewrite the schedules as $\tilde{\boldsymbol{\pi}}_C = \{c_0, c_1, \ldots, c_N, c_{N+1}\}$ and $\tilde{\boldsymbol{\pi}}_R = \{r_0, r_1, \ldots, r_N, r_{N+1}\}$, where $c_0 = r_0 = 0$ and $c_{N+1} = r_{N+1} = T$. During the interval between two successive checkpoints or rejuvenation points, we derive the expected up times (expected operative times) $U_C(c_i|c_{i-1})$ and $U_R(r_i|r_{i-1})$, and the expected total times $S_C(c_i|c_{i-1})$ and $S_R(r_i|r_{i-1})$ which include checkpoint, rejuvenation and restoration overheads:

$$U_C(c_i|c_{i-1}) = \int_0^{c_i-c_{i-1}} x dF(x|c_{i-1}) + (c_i - c_{i-1})\overline{F}(c_i - c_{i-1}|c_{i-1}), \tag{3}$$

$$S_C(c_i|c_{i-1}) = \int_0^{c_i-c_{i-1}} \{x + \rho(x) + \mu_c + \mu_r\} dF(x|c_{i-1}) + (c_i - c_{i-1} + \mu_c)\overline{F}(c_i - c_{i-1}|c_{i-1}), \tag{4}$$

$$U_R(r_i|r_{i-1}) = \int_0^{r_i-r_{i-1}} x dF(x) + (r_i - r_{i-1})\overline{F}(r_i - r_{i-1}), \tag{5}$$

$$S_R(r_i|r_{i-1}) = \int_0^{r_i-r_{i-1}} \{x + \rho(r_{i-1} + x) + \mu_c + \mu_r\} dF(x) + (r_i - r_{i-1} + \mu_r)\overline{F}(r_i - r_{i-1}), \tag{6}$$

where $F(c_i|c_{i-1})$ is a conditional cumulative distribution provided that a failure does not occur until time c_{i-1}, i.e.,

$$F(c_i|c_{i-1}) = 1 - \overline{F}(c_i)/\overline{F}(c_{i-1}). \tag{7}$$

At the last time periods $[c_N, c_{N+1}]$ and $[r_N, r_{N+1}]$, it is noted that the full maintenance is carried out unless the system failure occurs. Then, the expected

up times and the expected total times are given by

$$U_C(c_{N+1}|c_N) = \int_0^{c_{N+1}-c_N} x dF(x|c_N) + (c_{N+1} - c_N)\overline{F}(c_{N+1} - c_N|c_N), \quad (8)$$

$$S_C(c_{N+1}|c_N) = \int_0^{c_{N+1}-c_N} \{x + \rho(x) + \mu_c + \mu_r\} dF(x|c_N) + (c_{N+1} - c_N + \mu_c + \mu_r)\overline{F}(c_{N+1} - c_N|c_N), \quad (9)$$

$$U_R(r_{N+1}|r_N) = \int_0^{r_{N+1}-r_N} x dF(x) + (r_{N+1} - r_N)\overline{F}(r_{N+1} - r_N), \quad (10)$$

$$S_R(r_{N+1}|r_N) = \int_0^{r_{N+1}-r_N} \{x + \rho(r_N + x) + \mu_c + \mu_r\} dF(x) + (r_{N+1} - r_N + \mu_c + \mu_r)\overline{F}(r_{N+1} - r_N), \quad (11)$$

respectively. Based on the above results, the steady-state system availabilities with CPTR and RPTC are formulated as

$$A_C(\tilde{\boldsymbol{\pi}}_C) = \frac{\sum_{i=1}^{N+1} \overline{F}(c_{i-1}) U_C(c_i|c_{i-1})}{\sum_{i=1}^{N+1} \overline{F}(c_{i-1}) S_C(c_i|c_{i-1})}, \quad (12)$$

$$A_R(\tilde{\boldsymbol{\pi}}_R) = \frac{\sum_{i=1}^{N+1} \prod_{j=1}^{i-1} \overline{F}(r_j - r_{j-1}) U_R(r_i|r_{i-1})}{\sum_{i=1}^{N+1} \prod_{j=1}^{i-1} \overline{F}(r_j - r_{j-1}) S_R(r_i|r_{i-1})}. \quad (13)$$

3 DP Algorithms

Since the steady-state system availability for each policy is given as a function of $\tilde{\boldsymbol{\pi}}_C$ or $\tilde{\boldsymbol{\pi}}_R$, the problem is reduced to a non-linear maximization problem $\max_{\tilde{\boldsymbol{\pi}}_C} A_C(\tilde{\boldsymbol{\pi}}_C)$ or $\max_{\tilde{\boldsymbol{\pi}}_R} A_R(\tilde{\boldsymbol{\pi}}_R)$, provided that the number of checkpoints or rejuvenation points N is given. It is worth noting that there is no effective algorithm to find the optimal pair $(\tilde{\boldsymbol{\pi}}_C^*, N^*)$ or $(\tilde{\boldsymbol{\pi}}_R^*, N^*)$ simultaneously, and thus the number of checkpoints and rejuvenation points must be carefully adjusted according to any heuristic manner. On the other hand, in the case of a fixed N, the most popular method to find the optimal schedule might be the Newton's method or its iterative variant. However, since the Newton's method is a general-purpose non-linear optimization algorithm, it may not often function better to solve the maximization problem with many parameter constraints. In our maximization problems, the decision variables $\tilde{\boldsymbol{\pi}}_C$ and $\tilde{\boldsymbol{\pi}}_R$ are restricted. For such sequential optimization problems, it is well known that the dynamic programming (DP) can be used effectively.

In this section, we develop DP algorithms for finding the optimal time sequences $\tilde{\boldsymbol{\pi}}_C^*$ and $\tilde{\boldsymbol{\pi}}_R^*$. The proposed DP algorithms for both CPTR and RPTC are quite similar. Thus we first discuss the DP algorithm for CPTR, and mention the difference between two algorithms for CPTR and RPTC.

The idea behind our DP algorithms is to solve recursively the optimality equations which are typical functional equations. Hence, it seems to be straightforward to give the optimality equations which the optimal maintenance schedule $\tilde{\boldsymbol{\pi}}_C^*$ must satisfy. Suppose that there exists the unique maximum steady-state system availability ξ. From the principle of optimality [60], we obtain the following optimality equations for the maximization problem of the steady-state system availability under CPTR:

$$h_i = \max_{c_i} W_C(c_i|c_{i-1}^*, h_1, h_{i+1}), \qquad i = 1, \ldots, N, \tag{14}$$

$$h_{N+1} = \max_{c_{n+1}} W_C(c_{N+1}|c_N^*, h_1, h_1), \tag{15}$$

where the function $W_C(c_i|t_{i-1}, h_1, h_{i+1})$ is given by

$$\begin{aligned} W_C(c_i|c_{i-1}, h_1, h_{i+1}) &= U_C(c_i|c_{i-1}) - \xi S_C(c_i|c_{i-1}) \\ &\quad + h_1 F(t_i - t_{i-1}|t_{i-1}) + h_{i+1}\overline{F}(t_i - t_{i-1}|t_{i-1}) \end{aligned} \tag{16}$$

and the functions h_i, $i = 1, \ldots, N+1$, are called the *relative value functions*.

Since Eqs. (14) and (15) are necessary and sufficient conditions of the optimal maintenance schedule, the next step is to solve the above optimality equations. In the long history of the DP research, there are a couple of algorithms to solve the optimality equations. In this paper, we apply the *policy iteration* scheme [61] to derive the optimal maintenance scheduling. Our policy iteration consists of two steps; the optimization of the maintenance schedule under a given relative value function and the computation of the relative value function based on the updated maintenance schedule. These steps are repeatedly executed until the resulting maintenance schedule converges to the optimal value.

In the optimization phase, it should be noted that the functions

$$W_C(c_i|c_{i-1}, h_1, h_{i+1}), \quad i = 1, \ldots, N, \tag{17}$$

are not always concave with respect to the decision variables c_i. Our problem is the case. This fact leads to the difficulty for maximizing the steady-state system availability. In order to overcome this problem, we define the following composite function:

$$W_C(c_i|c_{i-1}, h_1, W_C(c_{i+1}|c_i, h_1, h_{i+2})), \quad i = 1, \ldots, N \tag{18}$$

and propose to use it instead of $W_C(c_i|c_{i-1}, h_1, h_{i+1})$. Because the above composite function is a concave function, it is possible to find the optimal maintenance schedule in each iteration phase by maximizing the composite function for $i = 1, \ldots, N$.

In the computation phase, we solve the following linear system to obtain the relative value functions h_i and the maximum steady-state system availability

ξ for a given maintenance schedule:

$$\boldsymbol{M}_C \boldsymbol{x} = \boldsymbol{b}_C, \tag{19}$$

where

$$[\boldsymbol{M}_C]_{i,j} = \begin{cases} -\overline{F}(c_i - c_{i-1}|c_{i-1}) & \text{if } i = j \text{ and } j \neq N+1, \\ 1 & \text{if } i = j+1, \\ S_C(c_i|c_{i-1}) & \text{if } j = N+1, \\ 0 & \text{otherwise}, \end{cases} \tag{20}$$

$$\boldsymbol{x} = (h_2, \ldots, h_N, h_{N+1}, \xi)', \tag{21}$$

$$\boldsymbol{b}_C = (U_C(c_1|c_0), \ldots, U_C(c_N|c_{N-1}), U_C(c_{N+1}|c_N))'. \tag{22}$$

In Eq. (20), $[\cdot]_{i,j}$ denotes the (i,j)-element of matrix, and the prime ($\prime$) represents transpose of vector. The above results come from the direct application of the optimality equations (14) and (15). Note that $h_1 = 0$, since we are here interested in the relative value function h_i and ξ

Finally, we derive the DP algorithm to derive optimal checkpoint sequence under CPTR as follows.

DP Algorithm under CPTR

- **Step 1:** Give initial values

$$\begin{aligned} k &:= 0, \\ c_0 &:= 0, \\ \tilde{\boldsymbol{\pi}}_C^{(0)} &:= \{c_1^{(0)}, \ldots, c_N^{(0)}, c_{N+1}^{(0)}\}. \end{aligned}$$

- **Step 2:** Compute $h_1^{(k)}, \ldots, h_{N+1}^{(k)}, \xi^{(k)}$ for the linear system (19) with the maintenance schedule $\tilde{\boldsymbol{\pi}}_C^{(k)}$.
- **Step 3:** Solve the following optimization problems:

$$\begin{aligned} c_i^{(k+1)} &:= \underset{c_{i-1}^{(k)} \leq c \leq c_{i+1}^{(k)}}{\operatorname{argmax}} W_C(c|c_{i-1}^{(k)}, 0, W_C(c_{i+1}^{(k)}|c_i, 0, h_{i+2}^{(k)})), \\ &\quad \text{for } i = 0, 1, \ldots, N-1, \\ c_N^{(k+1)} &:= \underset{c_{N-1}^{(k)} \leq c \leq c_{N+1}^{(k)}}{\operatorname{argmax}} W_C(c|c_{N-1}^{(k)}, 0, W_C(c_{N+1}^{(k)}|c, 0, 0)), \\ c_{N+1}^{(k+1)} &:= \underset{c_N^{(k)} \leq c < \infty}{\operatorname{argmax}} W_C(c|c_N^{(k)}, 0, 0). \end{aligned}$$

- **Step 4:** For all $i = 1, \ldots, N+1$, if $|c_i^{(k+1)} - c_i^{(k)}| < \delta$, stop the algorithm, where δ is an error tolerance level. Otherwise, let $k := k+1$ and go to Step

2.

In Step 3, an arbitrary optimization technique can be applied. Since the composite function is concave in the range $[c_{i-1}, c_{i+1})$, it is not so difficult to calculate the optimal checkpoint sequence and the optimal full maintenance time. In fact, the golden section method [62] would be effective to find the solution.

The DP algorithm for RPTC is given by replacing $U_C(\cdot|\cdot)$, $S_C(\cdot|\cdot)$ and $F(\cdot|\cdot)$ with $U_R(\cdot|\cdot)$, $S_R(\cdot|\cdot)$ and $F(\cdot)$, respectively. That is, Eqs. (16) and (19) are rewritten as

$$\begin{aligned} W_R(r_i|r_{i-1}, h_1, h_{i+1}) = U_R(r_i|r_{i-1}) - \xi S_R(r_i|r_{i-1}) \\ + h_1 F(t_i - t_{i-1}) + h_{i+1}\overline{F}(t_i - t_{i-1}) \end{aligned} \tag{23}$$

and

$$\boldsymbol{M}_R \boldsymbol{x} = \boldsymbol{b}_R, \tag{24}$$

$$[\boldsymbol{M}_R]_{i,j} = \begin{cases} -\overline{F}(r_i - r_{i-1}) & \text{if } i = j \text{ and } j \neq N+1, \\ 1 & \text{if } i = j+1, \\ S_R(r_i|r_{i-1}) & \text{if } j = N+1, \\ 0 & \text{otherwise,} \end{cases} \tag{25}$$

$$\boldsymbol{b}_R = (U_R(r_1|r_0), \ldots, U_R(r_N|r_{N-1}), U_R(r_{N+1}|r_N))'. \tag{26}$$

Then the DP algorithm for RPTC is described as follows.

DP Algorithm under RPTC

- **Step 1:** Give initial values

$$\begin{aligned} k &:= 0, \\ r_0 &:= 0, \\ \tilde{\boldsymbol{\pi}}_R^{(0)} &:= \{r_1^{(0)}, \ldots, r_N^{(0)}, r_{N+1}^{(0)}\}. \end{aligned}$$

- **Step 2:** Compute $h_1^{(k)}, \ldots, h_{N+1}^{(k)}, \xi^{(k)}$ for the linear system (24) with the maintenance schedule $\tilde{\boldsymbol{\pi}}_R^{(k)}$.

- **Step 3:** Solve the following optimization problems:

$$
\begin{aligned}
r_i^{(k+1)} &:= \underset{r_{i-1}^{(k)} \le r \le r_{i+1}^{(k)}}{\operatorname{argmax}} W_R(r | r_{i-1}^{(k)}, 0, W_R(r_{i+1}^{(k)} | r_i, 0, h_{i+2}^{(k)})), \\
&\qquad \text{for } i = 0, 1, \ldots, N-1, \\
r_N^{(k+1)} &:= \underset{r_{N-1}^{(k)} \le r \le r_{N+1}^{(k)}}{\operatorname{argmax}} W_R(r | r_{N-1}^{(k)}, 0, W_R(r_{N+1}^{(k)} | r, 0, 0)), \\
r_{N+1}^{(k+1)} &:= \underset{r_N^{(k)} \le r < \infty}{\operatorname{argmax}} W_R(r | r_N^{(k)}, 0, 0).
\end{aligned}
$$

- **Step 4:** For all $i = 1, \ldots, N+1$, if $|r_i^{(k+1)} - r_i^{(k)}| < \delta$, stop the algorithm, where δ is an error tolerance level. Otherwise, let $k := k+1$ and go to Step 2.

4 Numerical Comparison of CPTR and RPTC

This section presents numerical comparison of two maintenance policies and examines the effects of CPTR and RPTC on the optimal scheduling and the maximum steady-state system availability.

Suppose that the system failure time obeys the Weibull distribution;

$$
F(t) = 1 - \exp\left\{ -\left(\frac{t}{\eta}\right)^{\phi} \right\}, \tag{27}
$$

where η (> 0) and ϕ (> 0) are scale and shape parameters. If $\phi > 1$, then the system failure time distribution is IFR. The MTTF and the failure rate for the Weibull distribution are given by $\eta\Gamma(1 + 1/\psi)$ and $\phi t^{\phi-1}/\eta^{\phi}$, respectively, where $\Gamma(\cdot)$ is the standard gamma function. In our experiments, we set $\phi = 2.0$ or $\phi = 5.0$ and adjust the scale parameter such that MTTF equals 1.0. Furthermore, we assume that the recovery (restoration) overhead is given by $\rho(x) = \alpha x + \beta$ with $\alpha = 1.0$ and $\beta = 0.0, 0.5$. This means that the restoration operation requires exactly the same time amount as the lost processing time by a failure, and the parameter β corresponds to a fixed time overhead for the restoration. On the overhead parameters of checkpointing and rejuvenation, we set the following four cases:

Case 1: $\mu_c = 0.001$ and $\mu_r = 0.001$,
Case 2: $\mu_c = 0.001$ and $\mu_r = 0.002$,
Case 3: $\mu_c = 0.002$ and $\mu_r = 0.001$,
Case 4: $\mu_c = 0.002$ and $\mu_r = 0.002$.

Here the overhead time $\mu_c = 0.001$ corresponds to 0.1% of MTTF. For the purpose to calculate the optimal maintenance schedule, we make a computation program written by C language with GSL (GNU Scientific Library).

Figures 3 through 6 illustrate the optimal maintenance schedule (checkpoint, rejuvenation and full maintenance times) under CPTR and RPTC in Cases 1 and 4, provided that the shape parameter of failure time distribution is $\phi = 2.0$ and the fixed time overhead of restoration is $\beta = 0.0$. On the other hand, Figs 7 and 10 show the optimal maintenance schedule under CPTR and RPTC, provided that the shape parameter is 5.0 and $\beta = 0.0$. Figures 11 and 18 are the similar results as Figs. 11 and 18 where the fixed time overhead of restoration is given by $\beta = 0.5$. In the figures, the horizontal lines correspond to respective cases where the numbers of checkpoints or rejuvenation points are $N = 1, \ldots, 10$. Since the horizontal line indicates the elapsed time, each point (dot) represents the time at which a checkpoint or a rejuvenation point is placed, and the last one indicates the full maintenance timing.

In these figures, it can be seen that the optimal checkpoint intervals under CPTR are characterized as a decreasing sequence. The optimal rejuvenation intervals also indicate a decreasing sequence, but they are close to constant time intervals in all the cases. Comparing the results in the case of $\phi = 2.0$ and $\beta = 0.0$, the optimal checkpoint and rejuvenation times are quite different in the case of $\phi = 5.0$ and $\beta = 0.0$ This is because the Weibull distribution with $\phi = 5.0$ is closer to a constant distribution than $\phi = 2.0$. In fact, the coefficient of variation (CV) of the Weibull distribution with $\phi = 5.0$ is given by CV ≈ 0.23. In addition, the checkpoint and rejuvenation overheads in the case of $\beta = 0.0$ are relatively small. As a result, the effects of checkpointing and rejuvenating are almost same in the case of $\phi = 2.0$ and $\beta = 0.0$. On the other hand, in the case of $\beta = 0.5$, the optimal checkpoint and rejuvenation times are far from each other. This implies that two typical redundant techniques; checkpointing and rejuvenating are similar but they provide completely different effects on the system availability. In this case, the restoration operation is very expensive because the fixed time overhead reaches to 50% of MTTF. Thus the system failure should be avoided to improve the system availability. At the same time, both time intervals of checkpoint and rejuvenation times should be shorter than the cases of $\beta = 0.0$. However since the checkpointing essentially cannot avoid the system failure, the full maintenance should be frequently executed under CPTR. Hence the optimal time sequences of checkpointing and rejuvenating are quite different.

Next we examine the maximum steady-state system availability under CPTR and RPTC policies. Figures 19 and 20 depict the maximum steady-state system availability for each $N = 1, \ldots, 10$ in the case of the shape parameter of failure time distribution $\phi = 2.0$ and $\phi = 5.0$, respectively, provided that the fixed time overhead is 0.0. The four lines in these figures indicate the steady-

state system availabilities for Cases 1–4. Similarly, Figs. 21 and 22 present the maximum steady-state system availabilities provided that $\beta = 0.5$. By comparing the results of CPTR with those of RPTC, we find that the maximum availabilities for CPTR and RPTC are almost same in the case of $\phi = 2.0$ and $\beta = 0.0$. On the other hand, in the other cases, RPTC is much more effective to maximize the steady-state system availability than CPTR. Although the rejuvenation overheads might be higher than checkpointing overheads in practical cases, these numerical experiments conclude that the rejuvenation operation is superior to the checkpointing in terms of maximization of the availability if the overheads of rejuvenation are not so expensive.

5 Concluding Remarks

In this paper, we have introduced a general stochastic model for aperiodic checkpointing and software rejuvenation to evaluate the effect of them on the steady-state system availability. Based on the model, we have developed DP algorithms for finding the optimal checkpointing and rejuvenation time sequences. In numerical experiments, we have examined comprehensive evaluation of CPTR and RPTC with identical overhead parameters. Lessons learned from the numerical results are that (i) both the optimal checkpoint and rejuvenation times are given by decreasing sequences, (ii) the optimal checkpoint time intervals are sensitive for the aging property of failure distribution, but the optimal rejuvenation time intervals tend to be constant, regardless of the aging property, (iii) rejuvenation is superior to checkpointing in terms of maximizing the steady-state system availability if checkpoint and rejuvenation overheads are almost same.

In future, we will present a mixed policy of CPTR and RPTC, i.e., the policy where checkpoint and rejuvenation are generated at arbitrary time points. Although such a generalized policy has not been treated in this paper, either CPTR or RPTC is expected to be optimal in the optimization based on steady-state analysis. There remains a mathematical proof for the problem. Furthermore we will explore the possibility of an on-line control scheme based on Bayesian learning.

References

[1] M. Grottke, K. S. Trivedi, Fighting bugs: Remove, retry, replicate, and rejuvenate, IEEE Computer 40 (2) (2007) 107–109.

[2] J. W. Young, A first order approximation to the optimum checkpoint interval, Communication of the ACM 17 (9) (1974) 530–531.

[3] K. M. Chandy, J. C. Browne, C. W. Dissly, W. R. Uhrig, Analytic models for rollback and recovery strategies in database systems, IEEE Transactions on Software Engineering SE-1 (1) (1975) 100–110.

[4] K. M. Chandy, A survey of analytic models of roll-back and recovery strategies, Computer 8 (5) (1975) 40–47.

[5] N. H. Vaidya, Impact of checkpoint latency on overhead ratio of a checkpointing scheme, IEEE Transactions on Computers C-46 (8) (1997) 942–947.

[6] A. Ziv, J. Bruck, An on-line algorithm for checkpoint placement, IEEE Transactions on Computers C-46 (9) (1997) 976–985.

[7] F. Baccelli, Analysis of s service facility with periodic checkpointing, Acta Informatica 15 (1981) 67–81.

[8] T. Dohi, N. Kaio, S. Osaki, The optimal age-dependent checkpoint strategy for a stochastic system subject to general failure mode, Journal of Mathematical Analysis and Applications 249 (2000) 80–94.

[9] T. Dohi, N. Kaio, K. S. Trivedi, Availability models with age dependent-checkpointing, in: Proceedings of 21st Symposium on Reliable Distributed Systems (SRDS 2002), IEEE CS Press, 2002, pp. 130–139.

[10] E. Gelenbe, M. Hernandez, Optimum checkpoints with age dependent failures, Acta Informatica 27 (1990) 519–531.

[11] E. Gelenbe, On the optimum checkpoint interval, Journal of the ACM 26 (2) (1979) 259–270.

[12] E. Gelenbe, D. Derochette, Performance of rollback recovery systems under intermittent failures, Communication of the ACM 21 (6) (1978) 493–499.

[13] P. B. Goes, U. Sumita, Stochastic models for performance analysis of database recovery control, IEEE Transactions on Computers C-44 (4) (1995) 561–576.

[14] P. B. Goes, A stochastic model for performance evaluation of main memory resident database systems, ORSA Journal on Computing 7 (3) (1997) 269–282.

[15] V. Grassi, L. Donatiello, S. Tucci, On the optimal checkpointing of critical tasks and transaction-oriented systems, IEEE Transactions on Software Engineering SE-18 (1) (1992) 72–77.

[16] V. G. Kulkarni, V. F. Nicola, K. S. Trivedi, Effects of checkpointing and queueing on program performance, Stochastic Models 6 (4) (1990) 615–648.

[17] P. L'Ecuyer, J. Malenfant, Computing optimal checkpointing strategies for rollback and recovery systems, IEEE Transactions on Computers C-37 (4) (1988) 491–496.

[18] V. F. Nicola, J. M. van Spanje, Comparative analysis of different models of checkpointing and recovery, IEEE Transactions on Software Engineering SE-16 (8) (1990) 807–821.

[19] H. Okamura, Y. Nishimura, T. Dohi, A dynamic checkpointing scheme based on reinforcement learning, in: Proceedings of The 10th International Symposium on Pacific Rim Dependable Computing (PRDC 2004), IEEE CS Press, 2004, pp. 151–158.

[20] V. F. Nicola, Checkpointing and the modeling of program execution time, in: M. R. Lyu (Ed.), Software Fault Tolerance, John Wiley & Sons, 1995, pp. 167–188.

[21] S. Toueg, Ö. Babaoğlu, On the optimum checkpoint selection problem, SIAM Journal of Computing 13 (3) (1984) 630–649.

[22] S. Fukumoto, N. Kaio, S. Osaki, Optimal checkpointing strategies using the checkpointing density, Journal of Information Processing 15 (1992) 87–92.

[23] S. Fukumoto, N. Kaio, S. Osaki, A study of checkpoint generations for a database recovery mechanism, Computers Math. Applic. 24 (1992) 63–70.

[24] Y. Ling, J. Mi, X. Lin, A variational calculus approach to optimal checkpoint placement, IEEE Transactions on Computers 50 (7) (2001) 699–707.

[25] T. Dohi, T. Ozaki, N. Kaio, Optimal sequential checkpoint placement with equality constraints, in: Proceedings of The 2nd IEEE International Symposium on Dependable Autonomic and Secure Computing (DASC 2006), IEEE CS Press, 2006, pp. 77–84.

[26] T. Ozaki, T. Dohi, H. Okamura, N. Kaio, Distribution-free checkpoint placement algorithms based on min-max principle, IEEE Transactions on Dependable and Secure Computing 3 (2) (2006) 130–140.

[27] H. Okamura, K. Iwamoto, T. Dohi, A DP-based checkpointing scheme in real-time applications, International Journal of Reliability, Quality and Safety Engineering 13 (4) (2006) 323–340.

[28] E. Adams, Optimizing preventive service of the software products, IBM J. Research & Development 28 (1984) 2–14.

[29] A. Avritzer, E. J. Weyuker, Monitoring smoothly degrading systems for increased dependability, Empirical Software Engineering 2 (1997) 59–77.

[30] V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter, K. S. Trivedi, K. Vaidyanathan, W. P. Zeggert, Proactive management of software aging, IBM J. Research & Development 45 (2001) 311–332.

[31] Y. Huang, C. Kintala, N. Kolettis, N. D. Fulton, Software rejuvenation: analysis, module and applications, in: Proc. 25th Int'l Sympo. Fault Tolerant Computing, 1995, pp. 381–390.

[32] T. Dohi, K. Goševa-Popstojanova, K. S. Trivedi, Estimating software rejuvenation schedule in high assurance systems, The Computer Journal 47 (2001) 473–485.

[33] T. Dohi, H. Suzuki, K. S. Trivedi, Comparing software rejuvenation policies under different dependability measures, IEICE Transactions on Information and Systems (D) E87-D (8) (2004) 2078–2085.

[34] S. Garg, S. Pfening, A. Puliafito, M. Telek, K. S. Trivedi, Analysis of preventive maintenance in transactions based software systems, IEEE Transactions on Computers 47 (1998) 96–107.

[35] H. Okamura, K. Iwamoto, T. Dohi, Optimal rejuvenation scheduling of distributed computation based on dynamic programming, Journal of Computer Science 2 (6) (2006) 505–512.

[36] A. Bobbio, M. Sereno, C. Anglano, Fine grained software degradation models for optimal rejuvenation policies, Performance Evaluation 46 (45–62).

[37] Y. Bao, X. Sun, K. S. Trivedi, A workload-based analysis of software aging, and rejuvenation, IEEE Transactions on Reliability 54 (3) (2005) 541–548.

[38] H. Eto, T. Dohi, Analysis of a service degradation model with preventive rejuvenation, in: D. Penkler, M. Reitenspiess, F. Tam (Eds.), Service Availability: 3rd International Service Availability Symposium (ISAS 2006), Vol. LNCS 4328, Springer-Verlag, 2006, pp. 17–29.

[39] S. Garg, M. Telek, A. Puliafito, K. S. Trivedi, Analysis of software rejuvenation using Markov regenerative stochastic petri net, in: Proceedings of 6th International Symposium on Software Reliability Engineering (ISSRE 1995), IEEE CS Press, 1995, pp. 24–27.

[40] H. Okamura, S. Miyahara, T. Dohi, Dependability analysis of a client/server software systems with rejuvenation, in: Proceedings of 13th International Symposium on Software Reliability Engineering (ISSRE 2002), IEEE CS Press, 2002, pp. 171–180.

[41] H. Okamura, S. Miyahara, T. Dohi, Dependability analysis of a transaction-based multi server system with rejuvenation, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences (A) E86-A (8) (2003) 2081–2090.

[42] H. Okamura, H. Fujio, T. Dohi, Fine-grained shock models to rejuvenate software systems, IEICE Transactions on Information and Systems (D) E86-D (10) (2003) 2165–2171.

[43] H. Okamura, S. Miyahara, T. Dohi, Rejuvenating communication network system with burst arrival, IEICE Transactions on Communications (B) E88-B (12) (2005) 4498–4506.

[44] S. Pfening, S. Garg, A. Puliafito, M. Telek, K. S. Trivedi, Optimal rejuvenation for tolerating soft failure, Performance Evaluation 27/28 (4) (1996) 491–506.

[45] P. Reinecke, A. van Moorsel, K. Wolter, A measurement study of the interplay between application level restart and transport protocol, in: M. Malek, M. Manfred, J. Kaiser (Eds.), Service Availability: First International Service

Availability Symposium (ISAS 2004), Lecture Notes in Computer Science, Vol. LNCS 3335, Springer-Verlag, 2004, pp. 86–100.

[46] K. Rinsaka, T. Dohi, Behavioral analysis of fault-tolerant software systems with rejuvenation, IEICE Transactions on Information and Systems (D) E88-D (12) (2005) 2681–2690.

[47] K. Rinsaka, T. Dohi, A faster estimation algorithm for periodic preventive rejuvenation schedule maximizing system availability, in: M. Malek, M. Reitenspiess, A. van Moorsel (Eds.), Service Availability: 4th International Service Availability Symposium (ISAS 2007), Vol. LNCS 4526, Springer-Verlag, 2007, pp. 94–104.

[48] A. T. Tai, L. Alkalai, S. N. Chau, On-board preventive maintenance: a design-oriented analytic study for long-life applications, Performance Evaluation 35 (1999) 215–232.

[49] K. Vaidyanathan, R. E. Harper, S. W. Hunter, K. S. Trivedi, Analysis of software rejuvenation in cluster systems, in: Proceedings of ACM SIGMETRICS 2001/Performance 2001, ACM Press, 2001, pp. 62–71.

[50] K. Vaidyanathan, K. S. Trivedi, A comprehensive model for software rejuvenation, IEEE Transactions on Dependable and Secure Computing 2 (2) (2005) 124–137.

[51] A. van Moorsel, K. Wolter, Optimal restart times for moments of completion time, IEE Proceedings of Software 151 (5) (2004) 219–223.

[52] A. van Moorsel, K. Wolter, Analysis of restart mechanisms in software systems, IEEE Transactions on Software Engineering 32 (8) (2006) 547–558.

[53] D. Wang, W. X. andK. S. Trivedi, Performability analysis of clustered systems with rejuvenation under varying workload, Performance Evaluation 64 (2007) 247–265.

[54] S. Garg, Y. Huang, C. Kintala, K. S. Trivedi, Minimizing completion time of a program by checkpointing and rejuvenation, in: Proc. 1996 ACM SIGMETRICS Conf., 1996, pp. 252–261.

[55] A. Bobbio, S. Garg, M. Gribaudo, A. Horváth, M. Sereno, M. Telek, Modelling software systems with rejuvenation restoration and checkpointing through fluid stochastic Petri nets, in: Proc. 6th International Workshop on Petri Nets and Performance Models (PNPM'99), IEEE CS Press, 1999, pp. 82–91.

[56] A. Bobbio, S. Garg, M. Gribaudo, A. Horváth, M. Sereno, M. Telek, Compositional fluid stochastic Petri net model for operational software system performance, in: Supplemental Proceedings of 19th International Symposium on Software Reliability Engineering (ISSRE'08), 2008, p. 6 pages.

[57] H. Okamura, T. Dohi, Analysis of a software system with rejuvenation, restoration and checkpointing, in: T. Nanya, F. Maruyama, A. Pataricza, M. Malek (Eds.), Service Availability: 5th Int'l Service Availability Sympo.,

Lecture Notes in Computer Science, Vol. 5017, Springer-Verlag, 2008, pp. 110–128.

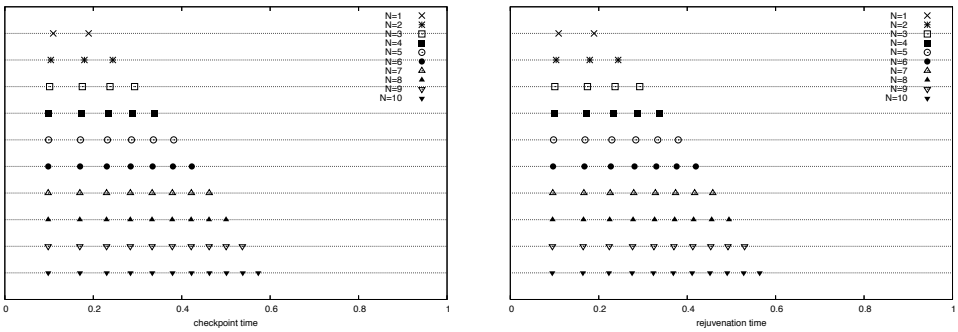
[58] H. Okamura, T. Dohi, Availability optimization in operational software system with aperiodic time-based rejuvenation scheme, in: Supplemental Proceedings of 19th International Symposium on Software Reliability Engineering (ISSRE'08), IEEE Computer Society Press, Los Alamitos, 2008, p. 6 pages.

[59] R. E. Barlow, F. Proschan, Mathematical Theory of Reliability, John Wiley & Sons, New York, 1965.

[60] R. Bellman, Dynamic Programming, Princeton University Press, 1957.

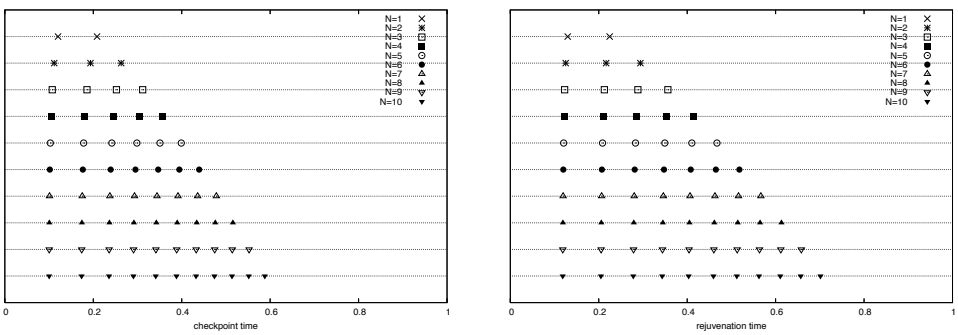
[61] M. Puterman, Markov Decision Processes, John Wiley & Sons, 1994.

[62] B. S. Gottfried, A stopping criterion for the golden-ratio search, Operations Research 23 (3) (1975) 553–555.



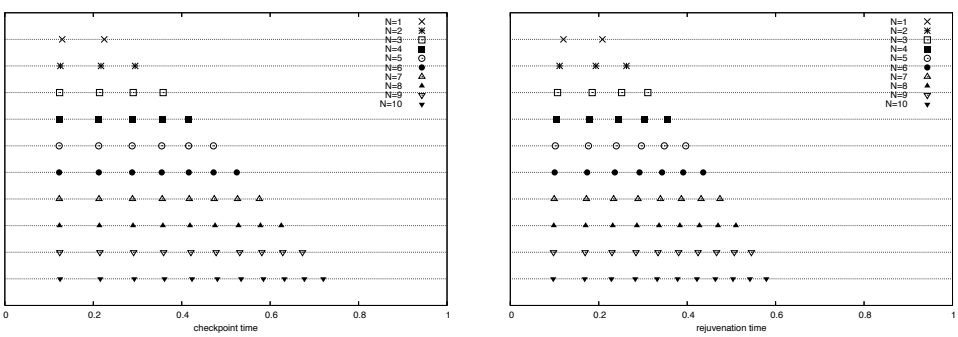
CPTR RPTC

Fig. 3. Optimal schedules ($\phi = 2.0$, $\beta = 0.0$, Case 1).



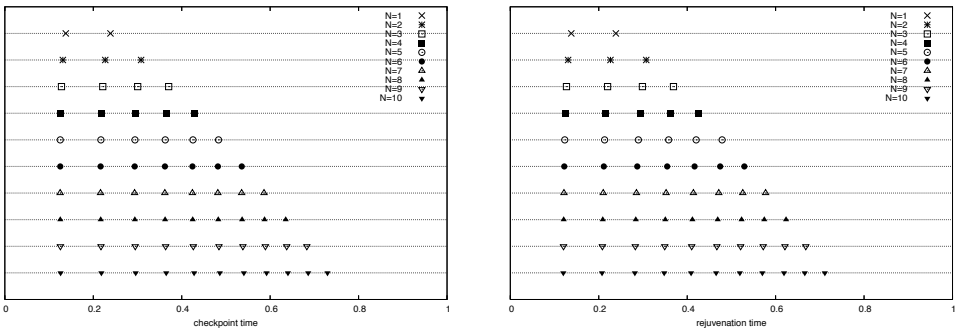
CPTR RPTC

Fig. 4. Optimal schedules ($\phi = 2.0$, $\beta = 0.0$, Case 2).



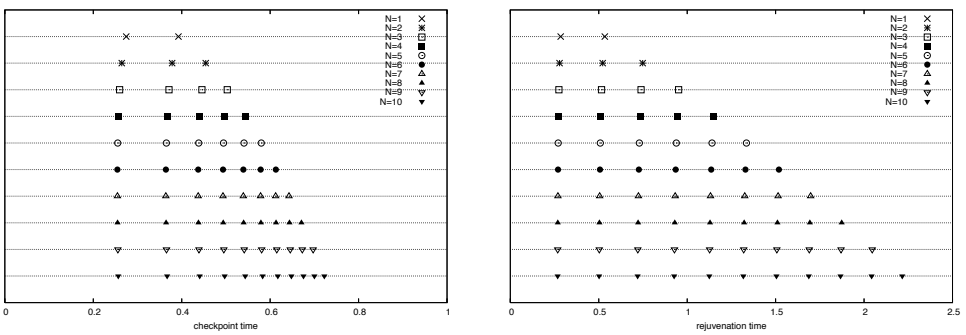
CPTR RPTC

Fig. 5. Optimal schedules ($\phi = 2.0$, $\beta = 0.0$, Case 3).



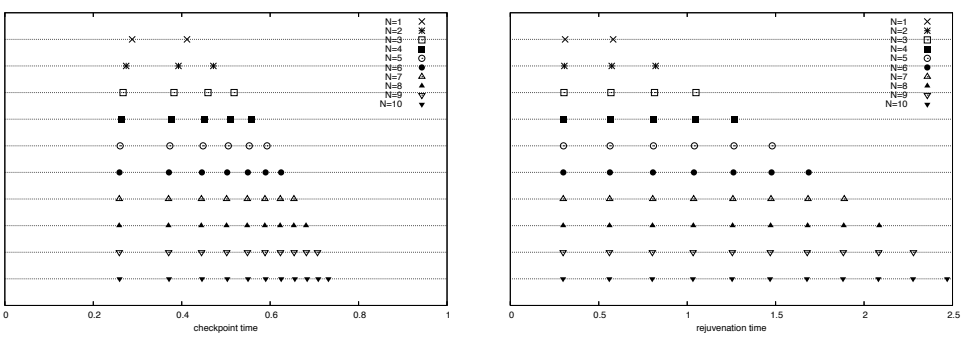
CPTR RPTC

Fig. 6. Optimal schedules ($\phi = 2.0$, $\beta = 0.0$, Case 4).



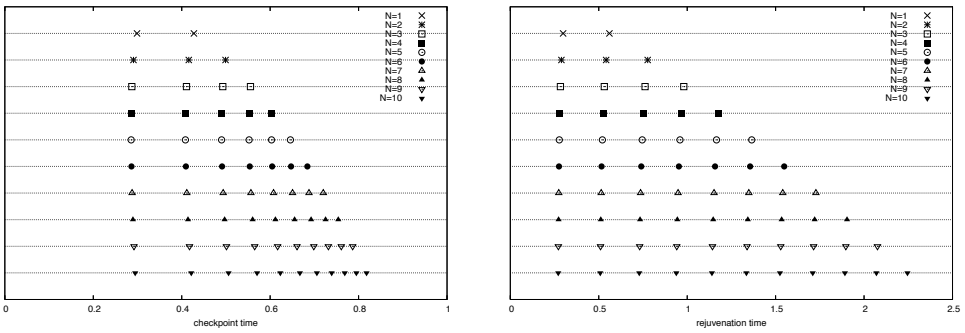
CPTR RPTC

Fig. 7. Optimal schedules ($\phi = 5.0$, $\beta = 0.0$, Case 1).



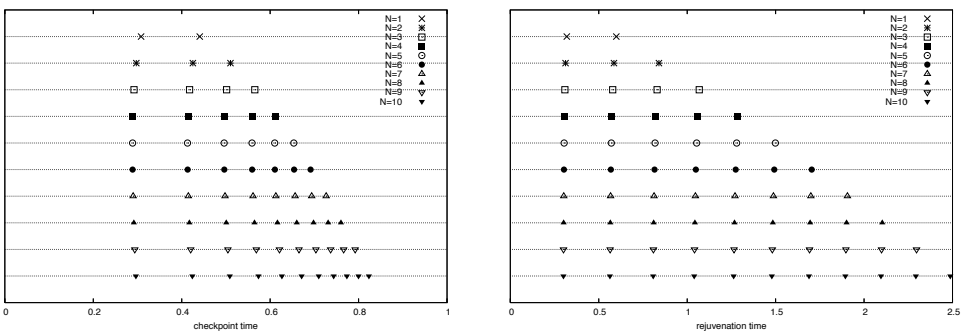
CPTR RPTC

Fig. 8. Optimal schedules ($\phi = 5.0$, $\beta = 0.0$, Case 2).



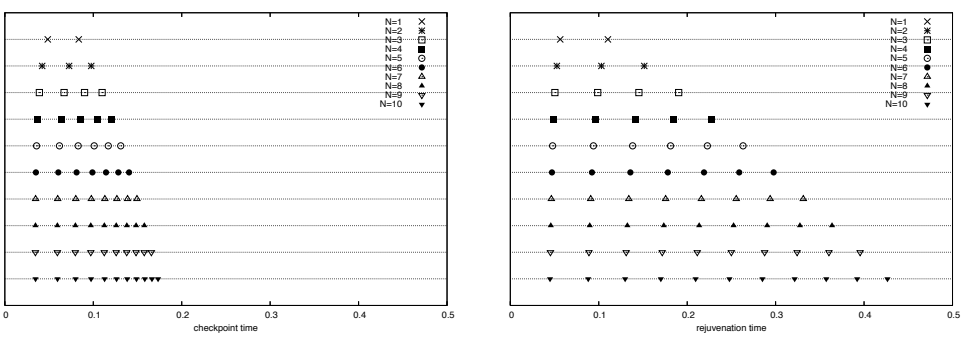
CPTR RPTC

Fig. 9. Optimal schedules ($\phi = 5.0$, $\beta = 0.0$, Case 3).



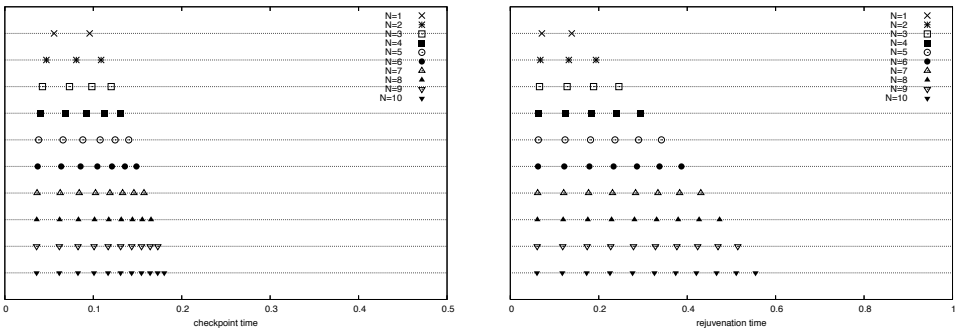
CPTR RPTC

Fig. 10. Optimal schedules ($\phi = 5.0$, $\beta = 0.0$, Case 4).



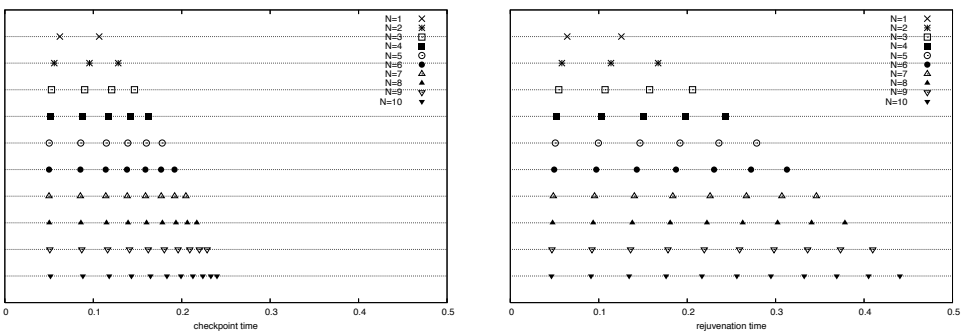
CPTR RPTC

Fig. 11. Optimal schedules ($\phi = 2.0$, $\beta = 0.5$, Case 1).



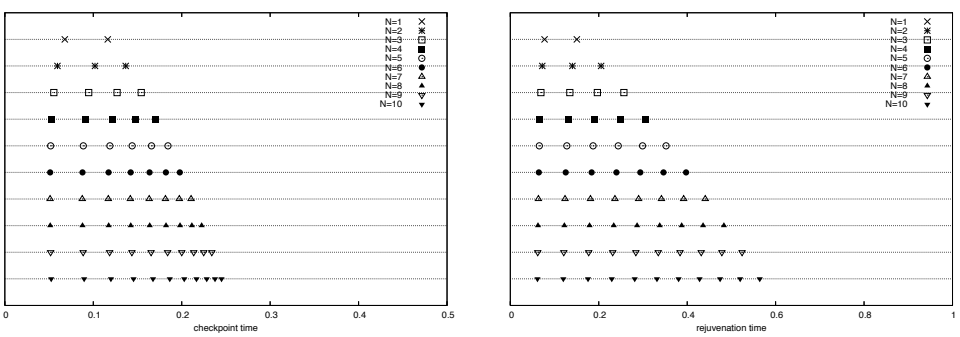
CPTR RPTC

Fig. 12. Optimal schedules ($\phi = 2.0$, $\beta = 0.5$, Case 2).



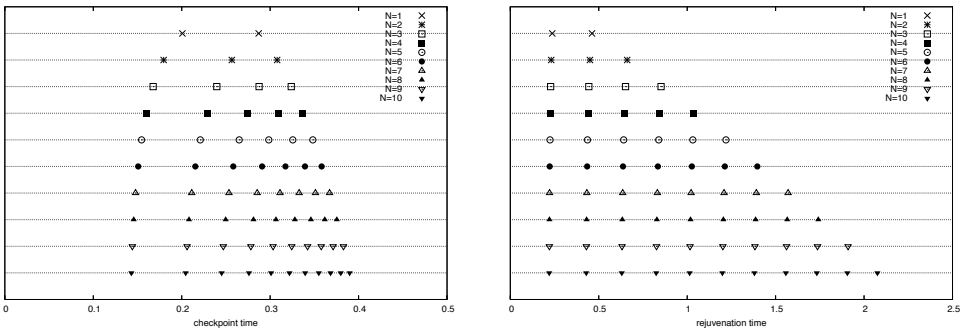
CPTR RPTC

Fig. 13. Optimal schedules ($\phi = 2.0$, $\beta = 0.5$, Case 3).



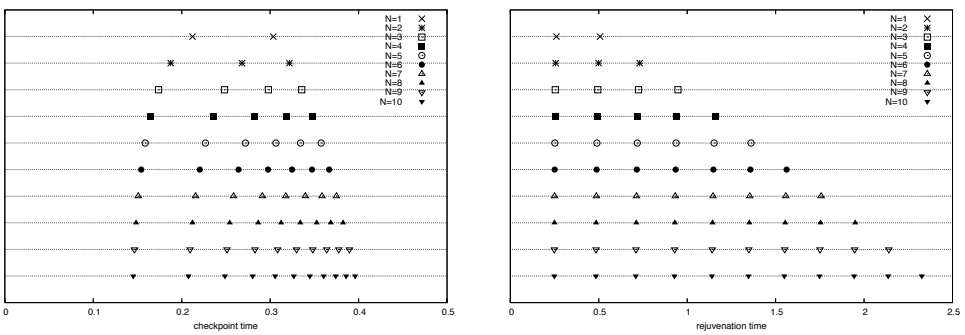
CPTR RPTC

Fig. 14. Optimal schedules ($\phi = 2.0$, $\beta = 0.5$, Case 4).



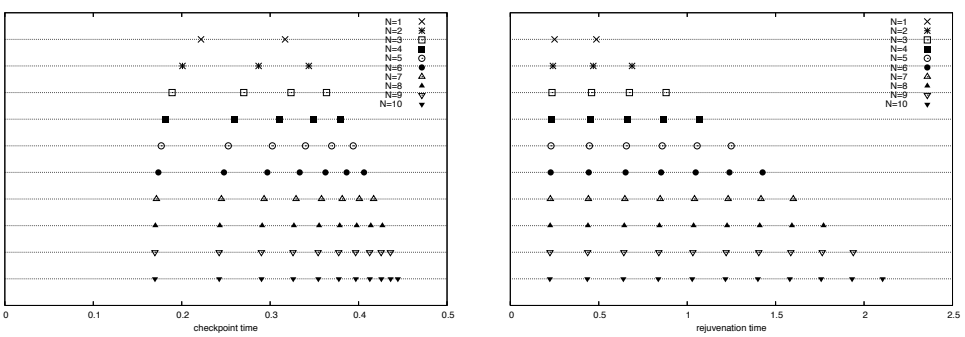
CPTR RPTC

Fig. 15. Optimal schedules ($\phi = 5.0$, $\beta = 0.5$, Case 1).



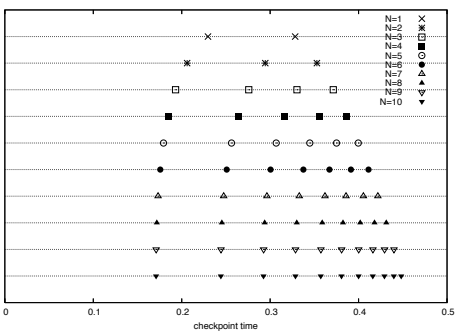
CPTR RPTC

Fig. 16. Optimal schedules ($\phi = 5.0$, $\beta = 0.5$, Case 2).

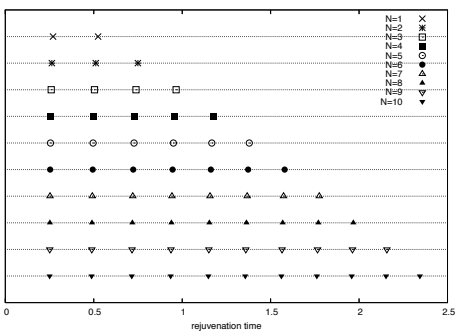


CPTR RPTC

Fig. 17. Optimal schedules ($\phi = 5.0$, $\beta = 0.5$, Case 3).

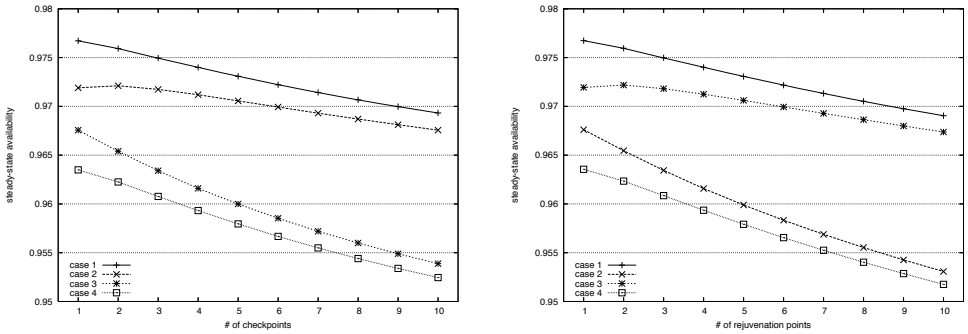


CPTR



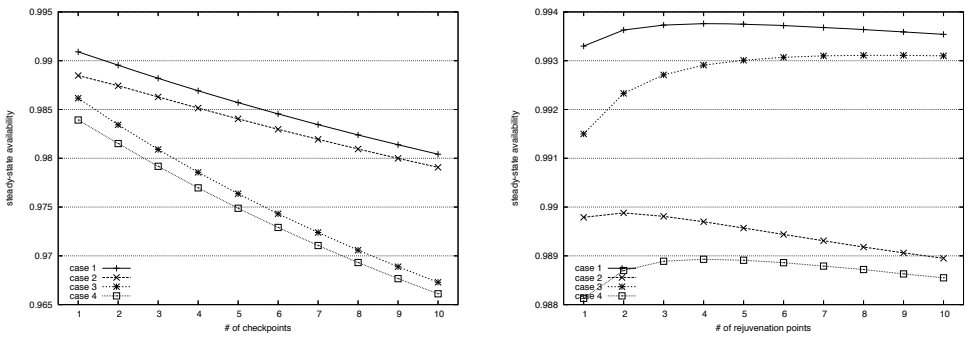
RPTC

Fig. 18. Optimal schedules ($\phi = 5.0$, $\beta = 0.5$, Case 4).



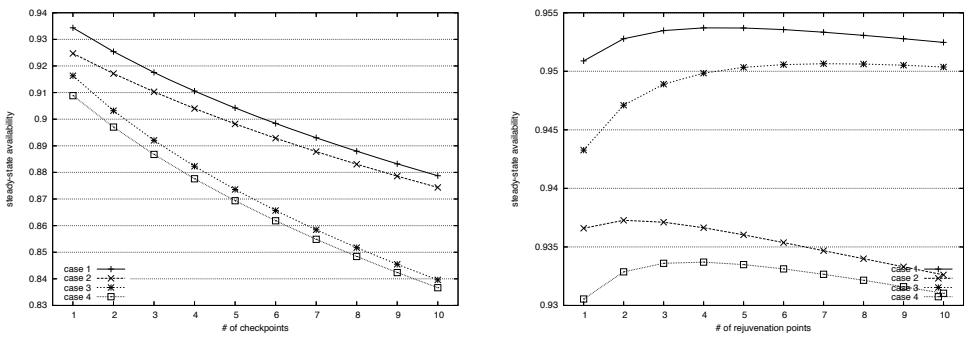
CPTR RPTC

Fig. 19. Maximum steady-state system availability ($\phi = 2.0$, $\beta = 0.0$).



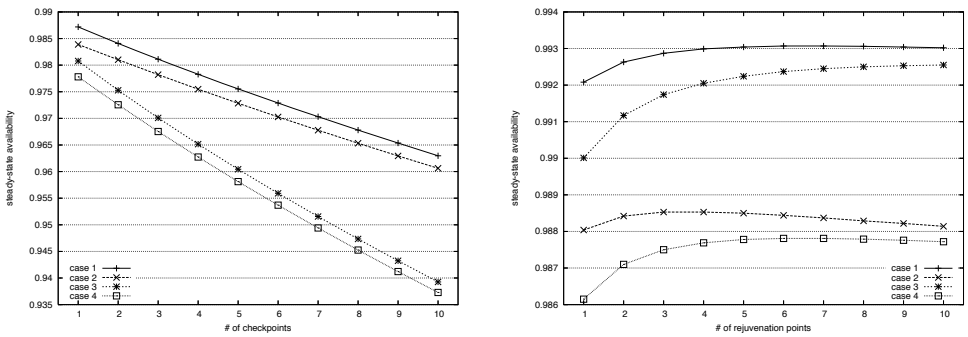
CPTR RPTC

Fig. 20. Maximum steady-state system availability ($\phi = 5.0$, $\beta = 0.0$).



CPTR RPTC

Fig. 21. Maximum steady-state system availability ($\phi = 2.0$, $\beta = 0.5$).



CPTR RPTC

Fig. 22. Maximum steady-state system availability ($\phi = 5.0$, $\beta = 0.5$).