Journal of Computer Science 2 (6): 505-512, 2006
ISSN 1549-3636
© 2006 Science Publications

Optimal Rejuvenation Scheduling of Distributed Computation Based on Dynamic Programming

Hiroyuki Okamura, Kazuki Iwamoto and Tadashi Dohi
Department of Information Engineering, Hiroshima University
1-4-1 Kagamiyama, Higashi-Hiroshima 739-8527, Japan

Abstract: Recently, a complementary approach to handle transient software failures, called software rejuvenation, is becoming popular as a proactive fault management technique in operational software systems. In this study, we develop the optimal scheduling algorithms to trigger software rejuvenation in distributed computation circumstance. In particular, we focus on two different computation circumstances in terms of detection of failures. Based on the dynamic programming, we derive the optimal software rejuvenation schedule which minimizes the expected total time of computation. In numerical examples, we examine the sensitivity of model parameters characterizing the failure phenomenon to the resulting optimal rejuvenation schedule.

Key words: Software aging, software rejuvenation, distributed computation, dynamic programming

INTRODUCTION

It has been recognized for a long time that software system never deteriorates in operational phases. However, the phenomenon called *software aging*[1,2], is often observed in actual software like operating systems and widely-used applications. The software aging affects adversely the software performance and eventually causes a system failure. Huang *et al.*[3] report this phenomenon in a telecommunication billing application where over time the application experiences a crash or a hang failure. The aging phenomenon in a telecommunication switching software is observed in Avritzer and Weyuker[4], where the effect manifests as gradual performance degradation. Garg *et al.*[5], Shereshevsky *et al.*[6] and Vaidyanathan *et al.*[7] carry out empirical researches to measure memory resource exhaustion and its associated aging phenomena in software systems. In fact, the software aging occurs by unexpected software faults and causes the performance degradation such as memory leaks, heap corruption and fragmentation.

Our common experience suggests that almost all the software failures caused by the software aging are transient in nature[8]. Since transient failures are not recaptured even if the same operation is retried later, it is quite difficult to detect the software fault causing the transient failure and the software aging. Therefore, the software aging phenomenon and its related transient software failures have to be tolerated in operational phases. Usual strategies to deal with transient failures are passive in nature; they consist of actions taken after failures. A complementary approach to handle transient software failures, called *software rejuvenation*, is becoming popular[9]. Software rejuvenation is 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, *etc*. An extreme, but well-known example of rejuvenation is a hardware reboot. Apart from being used in an ad-hoc manner by almost all computer users, the software rejuvenation has been used in high availability and mission critical systems[10]. The most vivid example of aging in safety critical systems is the Patriot's software[3], where the accumulated roundoff errors lead to the failure that resulted in loss of human life.

Software rejuvenation typically takes an overhead and the system cannot provide the service during the operation of rejuvenation. However, in general, the overhead cost caused by a scheduled downtime is expected to be much lower than that caused by an unexpected downtime. This is true on the overhead costs of failure and rejuvenation. In terms of cost performance, it is important to perform the occasional software rejuvenation for preventing more severe failures. From the above point of view, many authors consider the rejuvenation scheduling problems under a variety of dependability measures. Huang *et al.*[9] propose a continuous-time Markov chain model with random software rejuvenation. Rinsaka and Dohi[11] extend Huang's *et al.* model[9] to a fault-tolerant software system with a redundant component. Dohi *et al.*[12,13], Suzuki *et al.*[14,15] generalize the same model to semi-Markov models with different dependability

Corresponding Author : Hiroyuki Okamura, Department of Information Engineering, Hiroshima University, Japan

measures and develop the computation methods of the optimal software rejuvenation schedule. As an alternative modeling approach, Garg *et al.*[16] model a transaction-based software system involving arrival and service processes and evaluate the effect of transient failures on the time-based rejuvenation schedule. Recently, this model is extended by Okamura *et al.*[17] to a communication network system with burst arrival and rejuvenation.

In this study, we consider a software rejuvenation scheduling problem in a distributed computation circumstance. First we suppose that a distributed software system is composed of a number of processes that can communicate with exchange of messages. It is assumed that all the messages are delivered without time loss and that the communication is synchronous. That is, any process always receives a positive or negative acknowledgment message after sending a message to another process. In such a circumstance, the transient failure is defined as a process deadlock which is caused by competition of resources. In the synchronous communication, the process deadlock can be detected by timeout of acknowledgments. That is, if a process does not receive an acknowledgment in a specified time period, the process can detect the failure. Also, in order to prevent the process deadlock, it would be useful to execute an occasional software rejuvenation such as garbage collection and re-scheduling of transmitted messages. In such a distributed computation circumstance, we propose rejuvenation scheduling algorithms based on the dynamic programming (DP) and minimize the expected total time of computation. The similar but somewhat different problem is considered by Garg *et al.*[18], where the authors consider a minimization problem of the job completion time in a non-distributed computation with checkpointing and rejuvenation.

NOTATION

C : computation time (r.v.),
$G(t) = \Pr\{X \le t\}$: probability distribution function for computation time *C*,
X : transient failure time (r.v.),
$F(t) = \Pr\{X \le t\}$: probability distribution function for transient failure time *X*,
$\pi = \{t_1, \ldots, t_N\}$: rejuvenation schedule,
t_i : the *i*-th rejuvenation time,
μ_R : expected overhead time of rejuvenation,
$\rho(t)$: expected time for recovery operation,
T : deadline of computation,
N : the number of scheduled rejuvenation points,
$CT_i(t_{i+1})$: the expected total time to computation after the *i*-th rejuvenation, provided that the (*i*+1)-st rejuvenation time is t_{i+1},
v_i^* : the minimum expected total time to computation after the *i*-th rejuvenation.

ASSUMPTIONS

* A distributed computation is completed at time *C*.
* The computation fails at time *X*.
* The failed computation has to be retried after a recovery operation with time $\rho(t)$.
* *N* rejuvenations are performed at scheduled time sequence $\pi = \{t_1, \ldots, t_N\}$.
* The computation has to be retried with a recovery operation just after deadline *T*.
* In Model I, the transient failure is immediately detected.
* In Model II, the detection of failure is executed at the same time as rejuvenation.

MODEL DESCRIPTION

In the distributed computation circumstance, we suppose that a process starts a distributed computation at *t*=0 and completes at a random time *C*. The computation time *C* has the probability distribution function *G*(*t*) (*t*>0). The computation occasionally fails due to a process deadlock, but in general, the failed computation can be recovered by restarting the computation, *i.e.*, the failure is transient. In this study, we focus on such a transient failure, which occurs at a random time *X* having the probability distribution function *F*(*t*) (*t*>0). After detecting the transient failure, the process executes recovery operation; for example, it broadcasts messages of retry to the other communicative processes. Then, the computation undergoes the recovery operation and retry, where the expected time required for recovery operation is given by $\rho(t)$ (*t*>0) and *t* means the total time length of computation just before the failure. In many situations, the recovery operation takes a long overhead time. Thus, to prevent the transient failure, the process checks the condition of computation. This operation corresponds to the software rejuvenation. In our model, the rejuvenation is triggered at scheduled times $\pi = \{t_1, t_2 \ldots\}$. At each rejuvenation time t_j (*j*=1, 2, …), for example, the messages of rejuvenation are delivered to the other processes. The expected time length of rejuvenation is assumed to be μ_R (>0). After completing the rejuvenation, the rate of transient failure becomes as good as one at the beginning of computation. If the computation does not complete by the deadline of computation T, then the process has to execute both recovery operation and retry immediately.

Generally speaking, the detection of transient failure is a difficult problem in the distributed computation circumstance. In this study, we consider two models with different failure-detection procedures. In the first model, the process is always monitoring an acknowledgment of passing messages. Thus the failure can be detected immediately at the same time as its occurrence. In the second model, the failure detection is executed at the rejuvenation time. This implies that the process is not always monitoring the acknowledgment, but it tries to check the computation and acknowledgments from the other processes periodically. In practice, the difference between the first and second models is essentially whether the process waits for an acknowledgment from the other processes or not. That is, the first model corresponds to the system with sequential tasks. The second model represents the system with completely parallel and distributed tasks. We call the first and second models as Model I and Model II, respectively.

Figures 1 and 2 illustrate the possible realizations of distributed computation with rejuvenation for respective models. The arrows located at the upper parts in both figures indicate non-failed computations. At each rejuvenation time marked by a dot, the rate of transient failure is renewed. The time required by rejuvenation operation is the expected overhead time μ_R. After the failures which are pointed by stars, the computations are retried from the beginnings of computation (the arrows located at the lower parts) with the recovery overheads $\rho(t_2 + X)$ and $\rho(t_3)$. Note that the detection of failure is delayed in Model II. Finally, on the lower arrows, the computations are completed at the time pointed by boxes.

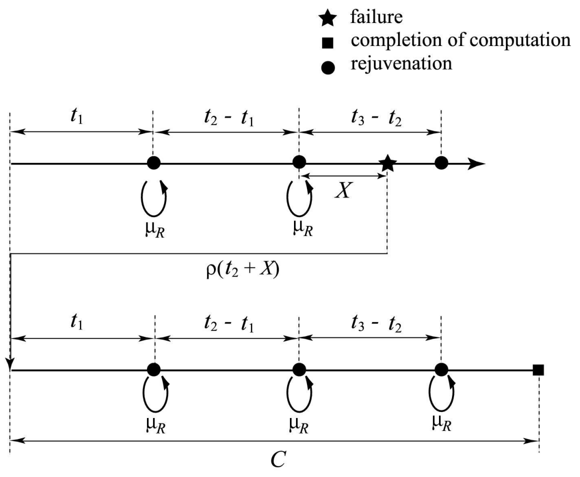

Fig. 1: Possible realization of distributed computation with rejuvenation (Model I)

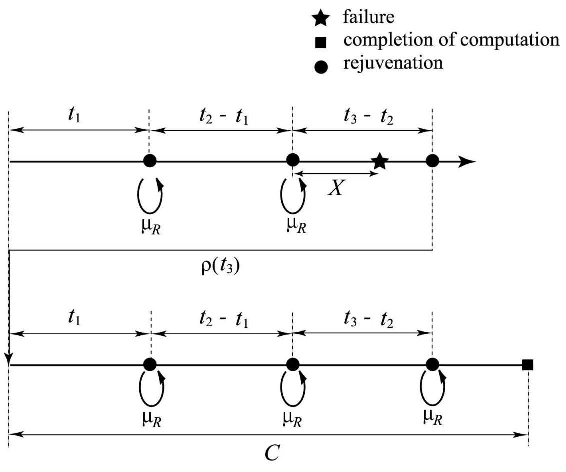

Fig. 2: Possible realization of distributed computation with rejuvenation (Model II)

OPTIMAL REJUVENATION SCHEDULING

Consider the following two software rejuvenation schedules:

Periodic rejuvenation: The time intervals of successive rejuvenations are given by a constant τ (>0), *i.e.*, the rejuvenation schedule is denoted by $\pi = \{\tau, 2\tau, 3\tau, \ldots\}$.

Non-periodic rejuvenation: The rejuvenation is triggered at non-constant time sequence but the number of rejuvenations until the completion of computation is fixed as N (≥ 1). Then the software rejuvenation schedule is given by $\pi = \{t_1, t_2, \ldots, t_N\}$.

Under these two policies, we discuss the optimal scheduling algorithms for software rejuvenation to minimize the expected total time to computation.

Model I

Periodic rejuvenation policy: Let $CT_i(\tau)$ denote the expected total computation time from the i-th rejuvenation under the periodic rejuvenation schedule. Since the transient failure causes retry of the computation, we have

$$\begin{aligned} CT_i(\tau) = & \\ & \int_0^\tau \int_0^s \{u + \rho(i\tau + u) + CT_0(\tau)\} dF(u) dG(s|i\tau) \\ & + \int_0^\tau \int_s^\infty s dF(u) dG(s|i\tau) \\ & + \int_\tau^\infty \int_0^\tau \{u + \rho(i\tau + u) + CT_0(\tau)\} dF(u) dG(s|i\tau) \\ & + \int_\tau^\infty \int_\tau^\infty \{\tau + \mu_R + CT_{i+1}(\tau)\} dF(u) dG(s|i\tau), \\ & i = 0, \ldots, N-1, \end{aligned} \quad (1)$$

$$
\begin{aligned}
CT_N(\tau) = &\int_0^{T-N\tau}\int_0^s \{u+\rho(N\tau+u) \\
&+CT_0(\tau)\}dF(u)dG(s|N\tau) \\
&+\int_0^{T-N\tau}\int_s^\infty sdF(u)dG(s|N\tau) \\
&+\int_{T-N\tau}^\infty\int_0^{T-N\tau}\{u+\rho(N\tau+u) \\
&+CT_0(\tau)\}dF(u)dG(s|N\tau) \\
&+\int_{T-N\tau}^\infty\int_{T-N\tau}^\infty\{T-N\tau+\rho(T) \\
&+CT_0(\tau)\}dF(u)dG(s|N\tau),
\end{aligned} \qquad (2)
$$

where N is the maximum integer which satisfies $T > n\tau$ and in general $\overline{\psi}(\cdot) = 1-\psi(\cdot)$. Also, $G(\cdot|\cdot)$ is the conditional probability distribution:

$$G(s\,|\,x) = 1-\overline{G}(x+s)/\overline{G}(x) \qquad (3)$$

In both Eqs. (1) and (2), the first term corresponds to the event where a failure occurs before the completion of computation and where the completion precedes the operation of rejuvenation. The second term indicates that the computation is completed before occurance of failure and execution of rejuvenation. The third and fourth terms mean that a failure occurs before both completion of computation and rejuvenation and that the rejuvenation is executed, respectively. Then the problem is to find the optimal time interval τ^* which minimizes $CT_0(\tau)$. Since the function $CT_0(\tau)$ is a non-linear function of τ , we can apply any numerical optimization method such as the Newton's method.

Non-periodic rejuvenation policy: Next consider the optimal non-periodic software rejuvenation schedule $\pi^* = \{t_1^*,\dots,t_N^*\}$ which minimizes the expected total time to computation. Define

$CT_i(t_{i+1})$: expected total time to computation after the i-th rejuvenation, provided that only the (i+1)-st rejuvenation can be chosen as $t_i^* \le t_{i+1} \le t_{i+2}^*$ and the others are strictly scheduled on the optimal time sequence of rejuvenations, *i.e.*, π^*.

v_i^* : minimum expected total time to computation after the i-th rejuvenation.

From the principle of optimality, we have the following optimality equations.

$$
v_i^* = \min_{t_i^*\le t_{i+1}\le t_{i+2}^*} CT_i(t_{i+1}), \quad i=0,1,\dots,N-1, \qquad (3)
$$

$$
v_N^* = CT_N(T), \qquad (4)
$$

and

$$
\begin{aligned}
CT_i(t_{i+1}) = &\\
&\int_0^{t_{i+1}-t_i}\int_0^s\{u+\rho(t_i+u)+v_0^*\}dF(u)dG(s|t_i) \\
&+\int_0^{t_{i+1}-t_i}\int_s^\infty sdF(u)dG(s|t_i) \\
&+\int_{t_{i+1}-t_i}^\infty\int_0^{t_{i+1}-t_i}\{u+\rho(t_i+u)+v_0^*\} \\
&\times dF(u)dG(s|t_i) \\
&+\int_{t_{i+1}-t_i}^\infty\int_{t_{i+1}-t_i}^\infty\{t_{i+1}-t_i+\mu_R+v_{i+1}^*\} \\
&\times dF(u)dG(s|t_i), \\
&i=0,\dots,N-1,
\end{aligned} \qquad (6)
$$

$$
\begin{aligned}
CT_N(T) = &\\
&\int_0^{T-t_N}\int_0^s\{u+\rho(t_N+u)+v_0^*\}dF(u)dG(s|t_N) \\
&+\int_0^{T-t_N}\int_s^\infty sdF(u)dG(s|t_N) \\
&+\int_{T-t_N}^\infty\int_0^{T-t_N}\{u+\rho(t_N+u)+v_0^*\} \\
&\times dF(u)dG(s|t_N) \\
&+\int_{T-t_N}^\infty\int_{T-t_N}^\infty\{T-t_N+\rho(T)+v_0^*\} \\
&\times dF(u)dG(s|t_N),
\end{aligned} \qquad (7)
$$

where $t_0 = 0$. In particular, when $F(t)$ and $G(t)$ are absolutely continuous probability distribution functions, Eqs. (6) and (7) can be simplified in the following forms:

$$
\begin{aligned}
CT_i(t_{i+1}) = &\int_0^{t_{i+1}-t_i}\overline{F}(s)\overline{G}(s|t_i) \\
&\times\{1+\rho(t_i+s)\lambda(s)+v_0^*\lambda(s)\}ds \\
&+(\mu_R+v_{i+1}^*)\overline{F}(t_{i+1}-t_i)\overline{G}(t_{i+1}-t_i|t_i), \\
&i=0,1,\dots,N-1,
\end{aligned} \qquad (8)
$$

$$
\begin{aligned}
CT_N(T) = &\int_0^{T-t_N}\overline{F}(s)\overline{G}(s|t_N) \\
&\times\{1+\rho(t_N+s)\lambda(s)+v_0^*\lambda(s)\}ds \\
&+(\rho(T)+v_0^*)\overline{F}(T-t_N)\overline{G}(T-t_N|t_N),
\end{aligned} \qquad (9)
$$

where

$$\lambda(s) = \frac{dF(s)/ds}{\overline{F}(s)}. \qquad (10)$$

By solving the above optimality equations, we can derive the optimal rejuvenation schedule π^*.

To develop an algorithm to compute the optimal rejuvenation schedule, we rewrite Eqs. (8) and (9) as functions of t_i , t_{i+1} and v_{i+1}^* , namely,

$$
WT_i(t|t_i,t_{i+2},v_{i+2}) = CT_i\Big(t_i,t,CT_{i+1}(t,t_{i+2},v_{i+2})\Big), \quad i=0,1,\dots,N-2 \qquad (11)
$$

and

$$WT_{N-1}(t|t_{N-1},T,v_0) = CT_{N-1}\Big(t_{N-1},t,CT_N(t,T,v_0)\Big). \tag{12}$$

Then we derive the following DP algorithm to compute the optimal rejuvenation schedule for Model I.

DP-based algorithm for scheduling rejuvenation:
Step 1: Let $k = 0$.
Step 2: Give the initial values

$$t_0^{(0)} := 0, \tag{13}$$
$$\pi^{(0)} := (t_1^{(0)},\dots,t_N^{(0)}) \tag{14}$$

and the expected times to computation $w_i^{(0)}$, $i = 1, \dots, N$.
Step 3: Calculate

$$t_0^{(k+1)} := 0, \tag{15}$$
$$w_i^{(k+1)} := \min_{t_i^{(k)}\le t\le t_{i+2}^{(k)}} WT_i(t|t_i^{(k)},t_{i+2}^{(k)},w_{i+2}^{(k)}), \quad \text{for } i=0,1,\dots,N-2, \tag{16}$$
$$t_{i+1}^{(k+1)} := \operatorname*{argmin}_{t_i^{(k)}\le t\le t_{i+2}^{(k)}} WT_i(t|t_i^{(k)},t_{i+2}^{(k)},w_{i+2}^{(k)}), \quad \text{for } i=0,1,\dots,N-2, \tag{17}$$
$$w_{N-1}^{(k+1)} := \min_{t_{N-1}^{(k)}\le t\le T} WT_{N-1}(t|t_{N-1}^{(k)},T,w_0^{(k)}), \tag{18}$$
$$t_N^{(k+1)} := \operatorname*{argmin}_{t_{N-1}^{(k)}\le t\le T} WT_{N-1}(t|t_{N-1}^{(k)},T,w_0^{(k)}), \tag{19}$$
$$w_N^{(k+1)} := CT_N(t_N^{(k)},T,w_0^{(k)}). \tag{20}$$

Step 4: For all $i = 1,\dots N$, if $|t_i^{(k+1)} - t_i^{(k)}| < \varepsilon$, stop the algorithm, where ε is an error tolerance, otherwise, let $k := k+1$ and go to Step 3.

Model II
Periodic rejuvenation policy: In Model II, the function $CT_i(\tau)$ can be rewritten as follows:

$$\begin{aligned}
CT_i(\tau) = & \int_0^{\tau}\int_0^{s}\{s+\rho(i\tau+s)+CT_0(\tau)\} \\
\times\ & dF(u)dG(s|i\tau) \\
+ & \int_0^{\tau}\int_s^{\infty} sdF(u)dG(s|i\tau) \\
+ & \int_{\tau}^{\infty}\int_0^{\tau}\{\tau+\rho((i+1)\tau)+CT_0(\tau)\}dF(u)dG(s|i\tau) \\
+ & \int_{\tau}^{\infty}\int_{\tau}^{\infty}\{\tau+\mu_R+CT_{i+1}(\tau)\}dF(u)dG(s|i\tau), \\
& i=0,\dots,N-1,
\end{aligned} \tag{21}$$

$$\begin{aligned}
CT_N(\tau) = & \int_0^{T-N\tau}\int_0^{s}\{s+\rho(N\tau+s)+CT_0(\tau)\} \\
\times\ & dF(u)dG(s|N\tau) \\
+ & \int_0^{T-N\tau}\int_s^{\infty} sdF(u)dG(s|N\tau) \\
+ & \int_{T-N\tau}^{\infty}\{T-N\tau+\rho(T)+CT_0(\tau)\}dG(s|N\tau).
\end{aligned} \tag{22}$$

The problem is to find the optimal time interval τ^* which minimizes $CT_0(\tau)$ and can be solved by the bisection method or the Newton's method.

Non-periodic rejuvenation policy: Consider the optimization problem in Model II. In this case, using $CT_i(t_{i+1})$ and v_i^*, we have the optimality equations for the rejuvenation schedule:

$$v_i^* = \min_{t_i^*\le t_{i+1}\le t_{i+2}^*} CT_i(t_{i+1}), \quad i=0,1,\dots,N-1, \tag{23}$$
$$v_N^* = CT_N(T), \tag{24}$$

where

$$\begin{aligned}
CT_i(t_{i+1}) = & \int_0^{t_{i+1}-t_i}\int_0^{s}\{s+\rho(t_i+s)+v_0^*\} \\
\times\ & dF(u)dG(s|t_i) \\
+ & \int_0^{t_{i+1}-t_i}\int_s^{\infty} sdF(u)dG(s|t_i) \\
+ & \int_{t_{i+1}-t_i}^{\infty}\int_0^{t_{i+1}-t_i}\{t_{i+1}-t_i+\rho(t_{i+1})+v_0^*\} \\
\times\ & dF(u)dG(s|t_i) \\
+ & \int_{t_{i+1}-t_i}^{\infty}\int_{t_{i+1}-t_i}^{\infty}\{t_{i+1}-t_i+\mu_R+v_{i+1}^*\} \\
\times\ & dF(u)dG(s|t_i), \\
& i=0,\dots,N-1,
\end{aligned} \tag{25}$$

$$\begin{aligned}
CT_N(T) = & \int_0^{T-t_N}\int_0^{s}\{s+\rho(t_i+s)+v_0^*\} \\
\times\ & dF(u)dG(s|t_N) \\
+ & \int_0^{T-t_N}\int_s^{\infty} sdF(u)dG(s|t_N) \\
+ & \int_{T-t_N}^{\infty}\{T-t_N+\rho(T)+v_0^*\}dG(s|t_N).
\end{aligned} \tag{26}$$

Similar to Eqs. (8) and (9), when *F*(*t*) and *G*(*t*) are continuous probability distribution functions, Eqs. (25) and (26) can be simplified in the following forms:

$$\begin{aligned}
CT_i(t_{i+1}) = & \int_0^{t_{i+1}-t_i}\overline{G}(s|t_i)\{1+(\rho(t_i+s)+v_0^*) \\
\times\ & F(s)r(t_i+s)\}ds \\
& +\{\rho(t_{i+1})+v_0^*\}F(t_{i+1}-t_i)\overline{G}(t_{i+1}-t_i|t_i) \\
& +(\mu_R+v_{i+1}^*)\overline{F}(t_{i+1}-t_i)\overline{G}(t_{i+1}-t_i|t_i), \\
& i=0,1,\dots,N-1,
\end{aligned} \tag{27}$$

$$\begin{aligned}
CT_N(T) = & \int_0^{T-t_N}\overline{G}(s|t_N)\{1+(\rho(s)+v_0^*) \\
\times\ & F(s)r(t_N+s)\}ds \\
& +\{\rho(T)+v_0^*\}\overline{G}(T-t_N|t_N),
\end{aligned} \tag{28}$$

where $r(\cdot)$ is the failure rate function of $G(\cdot)$:

$$r(s) = \frac{dG(s)/ds}{\overline{G}(s)}. \tag{29}$$

In a fashion similar to Model I, we can define the function $WT_i(t \mid t_i, t_{i+2}, v_{i+2})$ corresponding to Eqs. (11) and (12):

$$WT_i(t|t_i, t_{i+2}, v_{i+2}) = CT_i\Big(t_i, t, CT_{i+1}(t, t_{i+2}, v_{i+2})\Big), \quad i = 0, 1, \ldots, N-2 \qquad (30)$$

$$WT_{N-1}(t|t_{N-1}, T, v_0) = CT_{N-1}\Big(t_{N-1}, t, CT_N(t, T, v_0)\Big). \qquad (31)$$

The DP-based algorithm for Model II is the similar scheme to Model I. That is, substituting the above functions WT_i into the DP-algorithm for Model I yields a DP-based algorithm for the optimal rejuvenation scheduling in Model II.

NUMERICAL EXAMPLES

In this section, we present numerical examples for the non-periodic software rejuvenation schedule and investigate the dependence of the optimal software rejuvenation schedule on failure time distribution $F(\cdot)$, computation time distribution $G(\cdot)$ and rejuvenation overhead μ_R.

Suppose that the failure time distribution and the computation time distribution are given by the Weibull distributions:

$$F(t) = 1 - \exp\{-\eta_f t^{m_f}\}, \qquad (32)$$

$$G(t) = 1 - \exp\{-\eta_g t^{m_g}\}, \qquad (33)$$

respectively, where m_f (>0) and m_g (>0) are shape parameters which characterize the aging properties of distributions. Also, the parameters η_f (>0) and η_g (>0) are scale parameters. If the shape parameter and its mean value in the Weibull distribution are given, then the scale parameter is uniquely determined from the shape parameter and the mean value. Thus we give shape parameters and mean values in the following three different cases.

Case 1: $m_f = 2$, $\mathrm{E}[X] = 15$, $m_g = 2$, $\mathrm{E}[C] = 10$,

Case 2: $m_f = 2$, $\mathrm{E}[X] = 15$, $m_g = 5$, $\mathrm{E}[C] = 10$,

Case 3: $m_f = 2$, $\mathrm{E}[X] = 5$, $m_g = 2$, $\mathrm{E}[C] = 10$.

Case 1 is defined as the basic parameter set. Compared with Case 1, the variance of computation time in Case 2 decreases. On the other hand, Case 3 corresponds to a failure-prone case. To investigate the effect of failure and computation times, the other model parameters are fixed as $\mu_R = 0$ and $\rho(t) = 0.1t + 1.0$. In these cases, we calculate the optimal rejuvenation schedules with $N = 1,2,5,20$.

Figures 3 to 5 show the results on the optimal non-periodic rejuvenation policy. In the bottom parts of figures, we illustrate the optimal rejuvenation schedules with N = 1, 2, 5, 20, where each dot indicates the scheduled rejuvenation time. In addition, each arrow indicates its associated minimum expected time to computation. Also, in the upper parts of figures, we depict the failure time density functions and the computation time density functions. Comparing Fig.3 with Fig. 4, we can see that the time intervals of rejuvenation in Case 2 become shorter than those in Case 1. On the other hand, there is no remarkable difference between Case 1 and Case 3 in terms of the rejuvenation schedules. These results tell us that the optimal schedule of software rejuvenation is strongly affected by the computation time probability density rather than the failure time probability density. Next, we focus on the minimum expected time to computation. It is found that the expected time to computation in Case 3

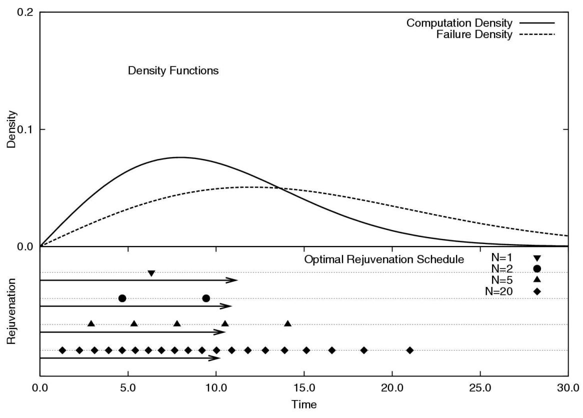

Fig. 3: Optimal rejuvenation scheduling in Model I (Case 1)

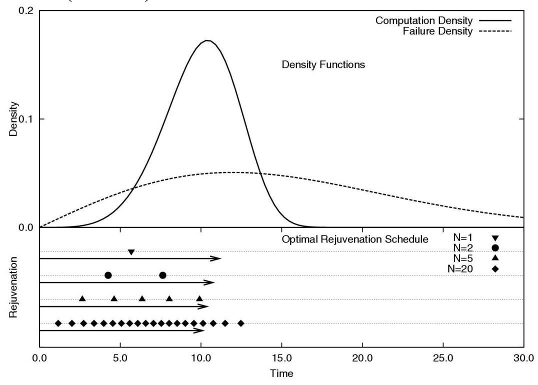

Fig. 4: Optimal rejuvenation scheduling in Model I (Case 2)

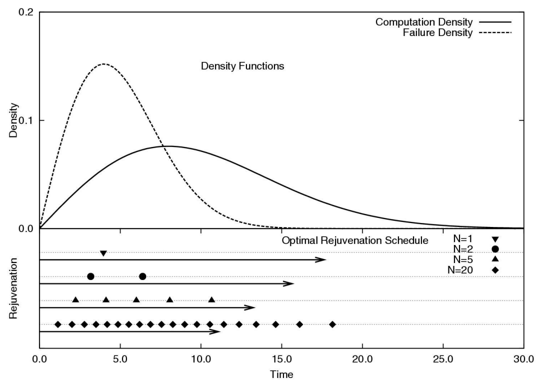

Fig. 5: Optimal rejuvenation scheduling in Model I (Case 3)

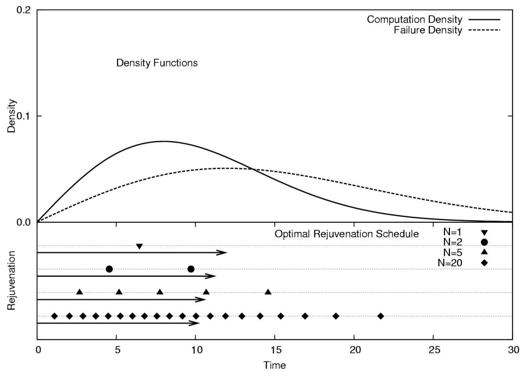

Fig. 6: Optimal rejuvenation in Model II (Case 1)

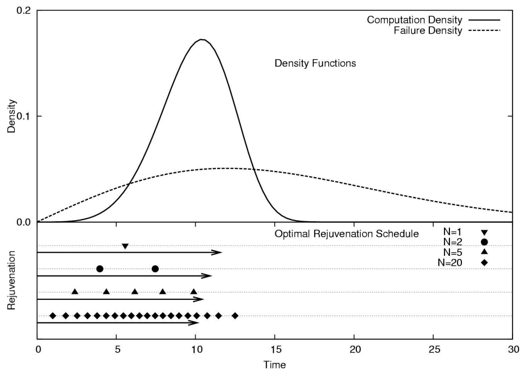

Fig. 7: Optimal rejuvenation in Model II (Case 2)

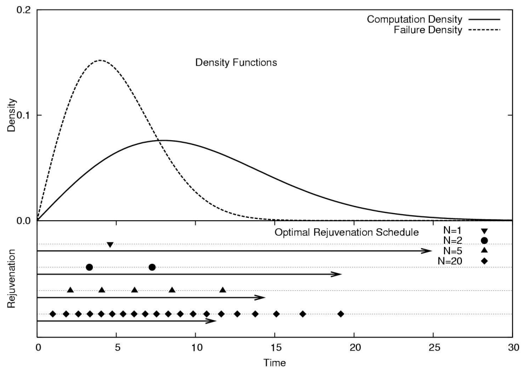

Fig. 8: Optimal rejuvenation scheduling in Model II (Case 3)

is relatively longer than those in the other cases, because the probability of failure is higher. However, the minimum expected completion time to computation is close to E[C]=10 as the total number of rejuvenation points increases. Therefore, we observe that the software rejuvenation is quite effective even in such case. Figures 6 to 8 illustrate the results on the optimal non-periodic rejuvenation policy in Model II. Comparing the results of Model I and Model II, there is no remarkable difference in Cases 1 and 2. However, the rejuvenation timings of Model II in Case 3 are slightly shorter than those of Model I. In particular, the expected minimum expected time to computation becomes longer in Model II. This is because the

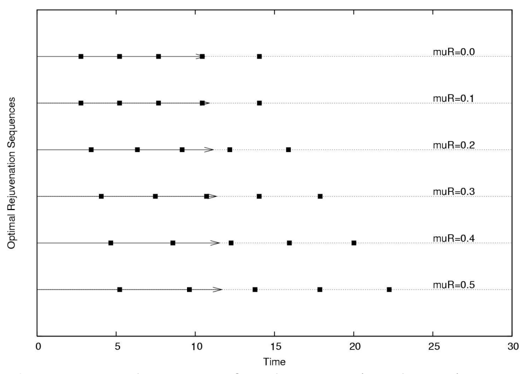

Fig. 9: Dependence of the optimal rejuvenation schedule on rejuvenation overheads in Model I (Case 1)

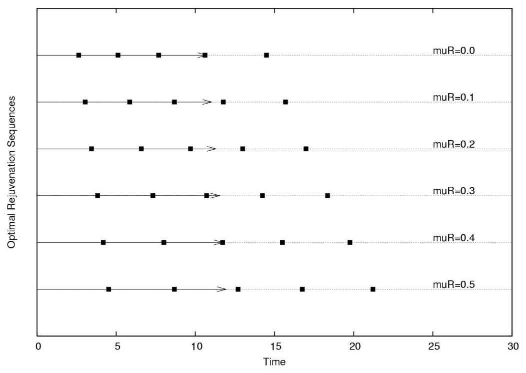

Fig. 10: Dependence of the optimal rejuvenation schedule on rejuvenation overheads in Model II (Case 1)

detection of failure is delayed in Model II, so that the computation times of Model II tend to be longer than those of Model I.

Next we investigate the dependence of the optimal rejuvenation schedule on rejuvenation overheads. Figures 9 and 10 illustrate the optimal rejuvenation sequences in Model I and Model II with varying rejuvenation overheads, $\mu_R = 0.0, \ldots, 0.5$, respectively, where N=5 and the other model parameters are the same as Case 1. From these results, in both models, time intervals of the optimal rejuvenation sequences become longer as the rejuvenation overheads increase. The expected minimum times to computation are also longer as the overheads increase, but the increments of the expected minimum times to computation are not so large. Comparing the result of Model I with that of Model II, we can observe that the increments of the time intervals in Model I are larger than those in Model II. That is, the optimal rejuvenation sequences are more sensitive to the rejuvenation overheads in the case of the immediate fault-detection circumstance.

CONCLUSION

This study has considered the optimal scheduling problems with software rejuvenation in the distributed computation. Based on the dynamic programming, we have developed the iterative algorithms to calculate the optimal software rejuvenation schedule minimizing the expected time to computation. In numerical examples, the dependence of the optimal rejuvenation schedule on the failure time and computation time probability distributions has been investigated. As a result, it has been concluded that the optimal software rejuvenation schedule is strongly affected by the computation time distribution rather than the failure time distribution. In future, we will develop an online algorithm to estimate the optimal software rejuvenation sequence based on the Bayesian estimation. Furthermore, we will apply the Monte Carlo simulation to evaluate the effectiveness of the rejuvenation algorithms in a distributed computation.

ACKNOWLEDGMENTS

This research was partially supported by the Ministry of Education, Science, Sports and Culture: Grant-in-Aid for Young Scientists (B), Grant No. 15700060 (2003-2004) and Exploratory Research, Grant No. 15651076 (2003-2005).

REFERENCES

1. Adams, E., 1984. Optimizing preventive service of the software products. IBM J. Res. and Development, 28: 2-14.
2. Castelli, V., R.E. Harper, P. Heidelberger, S.W. Hunter, K.S. Trivedi, K. Vaidyanathan and W.P. Zeggert, 2001. Proactive management of software aging. IBM J. Res. and Development, 45:311-332.
3. Marshall, E., 1992. Fatal error: How Patriot overlooked a scud. Science, 3: 1347.
4. Avritzer, A. and E.J. Weyuker, 1997. Monitoring smoothly degrading systems for increased dependability. Empirical Software Eng., 2: 59-77.
5. Garg, S., A. Van Moorsel, K. Vaidyanathan and K. S. Trivedi, 1998. A methodology for detection and estimation of software aging. Proc. 9th Intl. Symp. on Software Reliab. Eng., pp: 282-292.
6. Shereshevsky, M., B. Cukic, J. Crowel and V. Candikota, 2003. Software aging and multifractality of memory resources. Proc. Intl. Conf. on Dependable Systems and Networks, pp: 721-730.
7. Vaidyanathan, K. and K.S. Trivedi, 1999. A measurement-based model for estimation of resource exhaustion in operational software systems. Proc. 10th Int'l Symp. on Software Reliab. Eng., pp: 84-93.
8. Gray, J. and D.P. Siewiorek, 1991. High-availability computer systems, IEEE Computer, 24: 39-48.
9. Huang, Y., C. Kintala, N. Kolettis and N.D. Fulton, 1995. Software rejuvenation: Analysis, module and applications. Proc. 25th Intl. Symp. on Fault Tolerant Computing, pp: 381-390.
10. Tai, A.T., L. Alkalai and S.N. Chau, 1999. On-board preventive maintenance: A design-oriented analytic study for long-life applications. Performance Evaluation, 35: 215-232.
11. Rinsaka, K. and T. Dohi, 2005. Behavioral analysis of a fault-torellant software system with rejuvenation. Proc. 7th Intl. Symp. on Autonomous Decentralized Systems, pp: 159-166.
12. Dohi, T., K. Goseva-Popstojanova and K.S. Trivedi, 2001. Estimating software rejuvenation schedule in high assurance systems. The Computer J., 47: 473-485.
13. Dohi, T., H. Suzuki and K.S. Trivedi, 2004. Comparing software rejuvenation policies under different dependability measures. IEICE Trans. on Information and Systems (D), E87-D: 2078-2085.
14. Suzuki, H., T. Dohi, K. Goseva-Popstojanova and K.S. Trivedi, 2002. Analysis of multistep failure models with periodic software rejuvenation. In Advances in Stochastic Modelling (J.R. Artalejo and A. Krishnamoorthy, Eds.), Notable Publications, Inc., pp: 85-108.
15. Suzuki, H., T. Dohi, N. Kaio and K.S. Trivedi, 2003. Maximizing interval reliability in operational software system with rejuvenation. Proc. 14th Intl. Symp. on Software Reliab. Eng., pp: 246-256.
16. Garg, S., S. Pfening, A. Puliafito, M. Telek and K.S. Trivedi, 1998. Analysis of preventive maintenance in transactions based software systems. IEEE Trans. on Comput., 47: 96-107.
17. Okamura, H., S. Miyahara and T. Dohi, 2005. Effect of preventive rejuvenation in communication network system with burst arrival. Proc. 7th Intl. Symp. on Autonomous Decentralized Systems, pp: 151-158.
18. Garg, S., Y. Huang, C. Kintala and K.S. Trivedi, 1996. Minimizing completion time of a program by checkpointing and rejuvenation. Proc. ACM SIGMETRICS Conf., pp: 252-261.