Minimizing Completion Time of a Program by Checkpointing and Rejuvenation

Sachin Garg[1]*
sgarg@ee.duke.edu

Yennun Huang[2]
yen@research.att.com

Chandra Kintala[2]
cmk@research.att.com

Kishor S. Trivedi[1]
kst@ee.duke.edu

[1]Center for Adv. Comp. and Comm.
Department of Elec. & Comp. Engg.
Duke University
Durham, NC 27705

[2]AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974

Abstract

Checkpointing with rollback-recovery is a well known technique to reduce the completion time of a program in the presence of failures. While checkpointing is corrective in nature, rejuvenation refers to preventive maintenance of software aimed to reduce unexpected failures mostly resulting from the "aging" phenomenon. In this paper, we show how both these techniques may be used together to further reduce the expected completion time of a program. The idea of using checkpoints to reduce the amount of rollback upon a failure is taken a step further by combining it with rejuvenation. We derive the equations for expected completion time of a program with finite failure free running time for the following three cases when; (a) neither checkpointing nor rejuvenation is employed, (b) only checkpointing is employed, and finally (c) both checkpointing and rejuvenation are employed.

We also present numerical results for Weibull failure time distribution for the above three cases and discuss optimal checkpointing and rejuvenation that minimizes the expected completion time. Using the numerical results, some interesting conclusions are drawn about benefits of these techniques in relation to the nature of failure distribution.

*Supported in part by an IBM fellowship and by an AT&T Bell laboratories summer internship

1 Introduction

Checkpointing with rollback recovery is a well known technique. It involves occasional saving of the program state on stable storage. Upon a failure, the software/program does not need to be restarted from the very beginning but can be restarted from the last saved checkpoint (rollback recovery). For a program with finite failure free running time, this technique substantially reduces its completion time.

In earlier work on the analysis of checkpointing, failures were assumed to be caused mostly by hardware faults, independent of the program/software running on them, and for the most part the assumption of Poisson failure process was adequate. As hardware technology has constantly improved in terms of performance and reliability, it has been observed [8] that most of the failures are caused due to defects in the software. Evolution of large and complex software has added to this effect, emphasizing the need for software fault-tolerance. Recovery blocks [9], N-version programming [10] and N self-checking programming [2] are some of the prominent techniques for tolerating software failures. Based on the principle of design diversity, these techniques are reactive in nature, i.e., they provide the means of dealing with a failure after it has occurred. Another reactive approach based on data diversity has been proposed in [13].

Behavior of a program is determined by three components; the volatile state, the persistent state and the OS environment [5]. The volatile state consists of the program stack and static and dynamic data segments. Persistent state refers to all the user files related to a program's execution while the OS environment refers to resources that the program must access through the operating system, such as swap space, file systems, communication channels, keyboard, monitors, time etc. [5].

In recent studies [1, 6], it has been observed that a large percentage of software failures are transient in nature, i.e., they may not occur again if the program were to be re-executed. Lee [12] also observed that more than 70% of software failures in Tandem's system software are manifestations of transient faults such as race conditions, timing problems, etc. Such failures occur because of an undesirable faulty state reached in the OS environment of the program. It is also observed in [4] that owing to the presence of subtle software bugs called "Heisenbugs" [7] and due to interactions for sharing the hardware and operating system resources, such undesirable states in the OS environment accrue with time causing the software to "age". This phenomenon of software aging eventually manifests as a transient failure. Such transient failures are likely to disappear if the program is re-executed after a certain amount of clean-up and re-initialization of the OS environment which counteracts the "aging" phenomenon. This observation calls for a fault-tolerant technique based on environment diversity [11] and has led to an approach which is preventive in nature. Huang et. al. [4] call it *Software Rejuvenation* and it consists of occasionally stopping the running program and cleaning its internal state to remove the accrued transient conditions in the OS environment that might lead to failure. Flushing buffer queues maintained by file server, freeing unused and wrongly allocated memory, reinitializing the internal kernel tables, cleaning up the file system etc. are some physical examples of what rejuvenation might involve. A commonly known and very simple way of rejuvenating is the "reboot" of a computer.

In this paper, we combine checkpointing with rejuvenation. The goal is to minimize the completion time of a program which has a finite execution time in the absence of failures. Checkpointing by itself reduces the completion time of a program which can fail. We show that further reduction in the completion time is possible by incorporating rejuvenation. Checkpointing involves saving the volatile state on stable storage[1]. The OS environment, however, is not saved as part of a checkpoint.

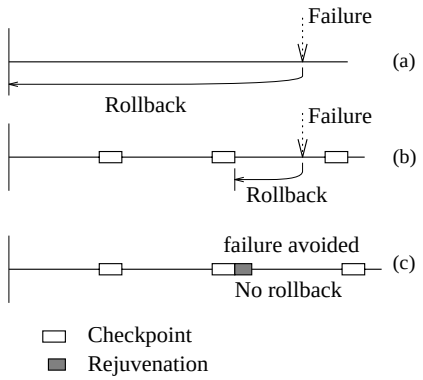

Figure 1: Effect of checkpointing and rejuvenation

In the presence of failures, the benefit of checkpointing comes from reduction in the amount of rollback. This idea is taken a step further by incorporating rejuvenation and is explained in Figure 1 as follows. The program fails at an arbitrary instant and as seen in Figure 1(a), involves a large rollback to the beginning of the program in the absence of checkpointing. Figure 1(b) shows that due to checkpointing, the same failure results in a smaller rollback, only to the last saved checkpoint. Assume that the failure had occurred because through gradual deterioration, an undesirable state was reached in the OS environment of the program. In Figure 1(c), the program is rejuvenated immediately after a checkpoint, thus removing the degradation in the OS state occurred so far. This "renewal" of the program prevents (or at least postpones) an unexpected failure. The planned stopping, cleaning up and restarting (as opposed to failure at an arbitrary time) right after a checkpoint results in no rollback. Although, the probability of unexpected failures is still non-zero it reduces to a large extent for such transient failures. This leaves us with the question of when and how often should the program be rejuvenated ?

The rest of the paper deals with the analysis of above three scenarios under a particular checkpointing and rejuvenation model and is organized as follows. In

[1] sometimes, depending on the application, the persistent state may also be saved. See [5] for details.

Section 2, we outline and compare previous work done on checkpointing analysis and state the contributions of this paper. In Section 3, we present a very simple checkpointing and rejuvenation model and precisely state the problem. In Section 4, expressions for the expected completion time for the three possible cases are derived. In Section 5, we provide a numerical example with Weibull failure time distribution to illustrate the benefit of combining the two techniques. Finally, we conclude the paper in Section 6 with possible implications of this work and future directions.

The words "software" and "program" are used interchangeably in this paper.

2 Previous Work

Software rejuvenation is a fairly new concept and its analysis has only recently been carried out. A typical problem is that of finding the rejuvenation policy which optimizes the measure of interest given a particular software system. The formulation of the problem will depend on the system specifics.

Checkpointing, on the other hand, has been well analyzed both for infinitely running transaction oriented software and programs with a finite failure free completion time. In this paper, we restrict ourselves to the latter. Such analyses differ in two main respects; the performance measure evaluated/optimized and the set of assumptions made regarding the system behavior. Whereas the case of minimizing the expected completion time is the most common [17, 15, 19, 18], maximizing the probability of completion of a task by a given deadline [14] or in a finite specified number of retries [20] has also been evaluated. Another important distinguishing feature is the set of assumptions made on the distribution and renewal of the failure process. As most of the earlier work regarded checkpointing as a way of tolerating hardware failures, most common assumption is that the time to failure distribution is exponential [18, 17, 14]. This restriction was removed in [19, 15] by considering general distributions.

Figure 2 illustrates the difference in assumptions made on the renewal of the failure process. Figure 2(a) plots the time to failure distribution (denoted by $F_X(t)$). Figures 2(b)-(d) plot the execution of the program (the vertical bars represent the program undergoing a checkpoint) superimposed by the time to failure distribution $F_X(t)$. An early assumption of no failures during checkpoints and the failure process getting renewed at each checkpoint (shown in Figure 2(b)) was made in [17]. The former restriction was removed in [14, 19] by allowing the failure process to continue through checkpointing. It is, however, still renewed upon completion of each checkpoint (Figure 2(c)). Figure 2(d) shows another generalization where the failure process does not renew after each checkpoint [18, 15]. In [18], however, it is assumed that the time to failure is exponentially distributed, which by it memoryless property is equivalent to the case shown in Figure 2(c). Leung and Choo [15] assume arbitrary distribution and have solved the problem in the most general setting. They also allow for failures to occur during rollback recovery.

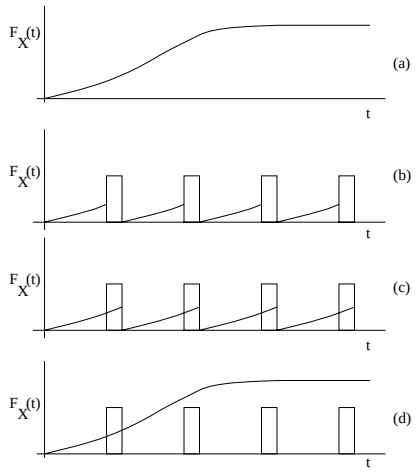

Figure 2: Assumptions on time to failure distribution

Our assumptions closely follow those of [15] and are shown in Figure 2(d). The failure process continues through checkpointing and the time to failure has an arbitrary distribution. The program, however, is assumed not to fail while in recovery. Failure process is renewed only if the program fails and is restarted or after rejuvenation is performed. State degradation of the program with respect to its OS environment, which causes the transient failure continues during the checkpointing process when it is done at the application level. As an example, `libft`[2] provides library routines to perform application level checkpointing [3].

[2] `libft` is a registered trademark of AT&T Bell Laboratories.

2.1 Our Contribution

We show how rejuvenation and checkpointing can be used to minimize the completion time of a program. To the best of our knowledge, checkpointing is combined with rejuvenation for the first time in this paper as a two-dimensional optimization problem. Total number of equidistant checkpoints and the number of rejuvenations to be performed during the execution of the program constitute the two dimensions along which the expected completion time is minimized. Coffman and Gilbert, in one of their models [16] considered preventive maintenance along with checkpointing, but assumed that the system underwent some maintenance with every checkpoint. The joint operation of checkpointing and preventive maintenance was called a "save". The combination essentially resulted in the failure process getting renewed at every checkpoint yielding a failure model of Figure 2(c) and still remaining a one-dimensional optimization problem.

3 Problem Formulation

Assume that a given program requires w units of execution time to complete in the absence of failures, checkpointing or rejuvenation. Further, assume that w is a constant and shall be referred to as the "work-requirement" of the program. Time to failure of the program is denoted by the random variable X with a given distribution $F_X(\cdot)$. Time to complete a checkpoint is assumed to be constant and is denoted by α. Upon a crash failure, some cleanup is performed and the program is restarted by reloading the last saved checkpoint from stable storage. This is assumed to take a constant time γ_f. Rejuvenation is performed right after the program has successfully completed a checkpoint. It includes stopping the program, cleaning up, and reloading and is also assumed to take a constant time γ_r. Since recovery from a failure and rejuvenation, both involve the same procedure, γ_f is typically equal to γ_r. A simple (yet common) example of restarting after a "crash" failure occurs is "reboot". Rejuvenation may also simply involve occasional, planned reboot as it cleans internal tables, reinitializes and frees memory. The assumption of constant checkpointing time is reasonable as the time to save the volatile state is very small compared to w. Thus, slight variation in checkpointing time may be ignored. Similar reasoning applies to justifying assumptions of constant recovery and rejuvenation times. Given a program and a system it runs on, none of the above parameters can be controlled.

Checkpointing Model: We assume that the program needs to complete a total of N checkpoints to finish execution. N is an integer constant whose value can be varied and constitutes the first dimension of the optimization. The checkpoints are equidistant with each failure free inter-checkpoint interval being w/N. [3] The work requirement of each of these segments (including the checkpoint) is therefore $w/N + \alpha$ denoted as β. The total work requirement of the program including checkpointing time is thus $N\beta$.

Rejuvenation Model: Our rejuvenation model is simply to perform it after every kth checkpoint. We shall refer to k as the *rejuvenation distance*. In our model, k is an integer constant whose value may be controlled and it constitutes the second dimension along which the expected completion time is to be minimized.

Our goal is to first evaluate the expected completion time of a program with given N and k and then to find values of N and k for which the expected completion time is minimal. We shall denote these optimal values by N^* and k^* respectively.

4 Expected Completion Time

Let $T(w)$ denote the completion time of the program when neither checkpointing nor rejuvenation is employed and w is the "work-requirement". Clearly, $T(w)$ is a random variable due to randomness in the failure process. Note that in the absence of any failures, $T(w) = w$. Similarly, let the random variable $T_c(N\beta, N)$ denote the completion time of the same program when only checkpointing is employed ($N\beta$ represents the work requirement in this case) and finally let the random variable $T_{rc}(N\beta, N, k)$ denote the completion time when both checkpointing and rejuvenation are employed. It is straight forward to see that the case of

[3] For simplicity, we have assumed that the program after completing w units of execution takes the Nth checkpoint and then exits.

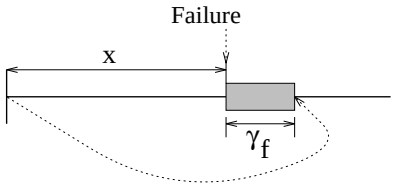

Figure 3: Program with no Checkpointing or Rejuvenation

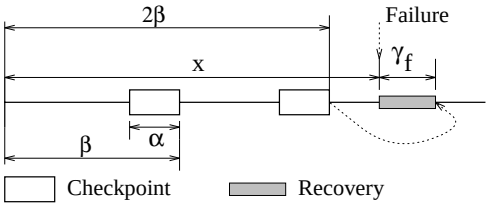

Figure 4: Program with checkpointing only

only employing rejuvenation is meaningless under the deterministic policy of rejuvenating every $r < w$ time units as there is no saved state to restart the program from an intermediate point. Restart from the beginning after rejuvenating every r time units would never allow the program to complete.

We now proceed to derive expressions for the expected values of these three random variables viz. $E[T(w)]$, $E[T_c(N\beta, N)]$ and $E[T_{rc}(N\beta, N, k)]$.

4.1 No Checkpointing or Rejuvenation

Let $\overline{F}_X(w) = 1 - F_X(w)$, then the expected completion time is given by the following theorem.

Theorem 1:

$$E\left[T(w)\right] = w + \frac{\gamma_f F_X(w)}{\overline{F}_X(w)} + \frac{\int_0^w x\, dF_X(x)}{\overline{F}_X(w)}$$

Proof: We proceed by conditioning on the time to first failure $X = x$ (See Figure 3). If $x > w$, i.e., if the failure does not occur within w units, the program completes. On the other hand, if a failure does occur before completion ($x < w$), the program completion time is given by the sum of three distinct components. First, the time used so far (x), second, the time to recover and restart (γ_f) and last, the expected completion time for the remaining work-requirement. As the program is started from the very beginning (shown by the dotted line in Figure 3), the remaining work requirement is still w and its expected completion time is $E[T(w)]$. Formally, the conditional expected completion time is written as:

$$E\left[T(w)|X = x\right] = \begin{cases} w, & \text{if } x \geq w \\ x + \gamma_f + E\left[T(w)\right], & \text{if } x < w \end{cases}$$

By the law of total expectation, $E[T(w)] = \int_0^w E[T(w)|X = x]\, dF_X(x)$. Therefore, $E[T(w)]$

$$= w\,\overline{F}_X(w) + \int_0^w \left(x + \gamma_f + E[T(w)]\right) dF_X(x)$$

$$= w\,\overline{F}_X(w) + \left(\gamma_f + E[T(w)]\right) F_X(w) + \int_0^w x\, dF_X(x)$$

Rearranging with respect to $E[T(w)]$, we get

$$E\left[T(w)\right] = w + \frac{\gamma_f F_X(w)}{\overline{F}_X(w)} + \frac{\int_0^w x\, dF_X(x)}{\overline{F}_X(w)}$$

□

4.2 Checkpointing Only

The program execution is shown in Figure 4. Under our model, N equidistant checkpoints are taken dividing the program into N segments, each with a work requirement β (including the checkpointing time). The expected completion time is given by the following recurrence relation.

Theorem 2: $E\left[T_c(N\beta, N)\right]\, \overline{F}_X(\beta) =$

$$N\beta\,\overline{F}_X(N\beta) + \gamma_f\, F_X(N\beta) + \int_0^{N\beta} x\, dF_X(x) +$$

$$\sum_{i=1}^{N-1} E[T_c((N-i)\beta, N-i)]\left[F_X((i+1)\beta) - F_X(i\beta)\right]$$

Proof: Again, we proceed by conditioning on the time to first failure $X = x$. If $x > N\beta$, i.e., the failure does not occur within the work requirement, the program completion time is $N\beta$. If however, the failure occurs in any of the N segments(say the ith segment), i.e., if

$(i-1)\beta \le x < i\beta$, for i from 1 to N, the program completion time is the summation of three distinct components. First, the time spent so far, x, second, the restart time γ_f and last, the expected completion time with the remaining work requirement. As $(i-1)$ checkpoints have already been completed, remaining work requirement is $(N-i+1)\beta$, the completion time of which is denoted by random variable $T_c((N-i+1)\beta,(N-i+1))$. Formally, the conditional expected completion time may be written as; $E[T_c(N\beta,N)|X=x] =$

$$\begin{cases} x+\gamma_f+E[T_c(N\beta,N)], & 0 \le x < \beta \\ x+\gamma_f+E[T_c((N-1)\beta,N-1)], & \beta \le x < 2\beta \\ \quad \cdots & \\ x+\gamma_f+E[T_c(2\beta,2)], & (N-2)\beta \le x < (N-1)\beta \\ x+\gamma_f+E[T_c(\beta,1)], & (N-1)\beta \le x < N\beta \\ N\beta, & x > N\beta \end{cases}$$

By the law of total expectation, $E[T_c(N\beta,N)]$

$$\begin{aligned} &= \int_0^{\infty} E[T_c(N\beta,N)|X=x]\, dF_X(x) \\ &= \int_0^{\beta} (x+\gamma_f+E[T_c(N\beta,N)])\, dF_X(x) + \\ &\quad \int_{\beta}^{2\beta} (x+\gamma_f+E[T_c((N-1)\beta,N-1)])\, dF_X(x) \\ &\quad + \cdots + \\ &\quad \int_{(N-2)\beta}^{(N-1)\beta} (x+\gamma_f+E[T_c(2\beta,2)])\, dF_X(x) + \\ &\quad \int_{(N-1)\beta}^{N\beta} (x+\gamma_f+E[T_c(\beta,1)])\, dF_X(x) + \\ &\quad \int_{N\beta}^{\infty} N\beta\, dF_X(x). \end{aligned}$$

Combining the integrals with x and γ_f terms, we get $E[T_c(N\beta,N)] =$

$$\gamma_f\, F_X(N\beta) + N\beta\, \overline{F}_X(N\beta) + \int_0^{N\beta} x\, dF_X(x) + \sum_{i=0}^{N-1} \int_{i\beta}^{(i+1)\beta} E[T_c((N-i)\beta,(N-i))]\, dF_X(x)$$

Combining $E[T_c(N\beta,N)]$ on both sides, evaluating the integral and rearranging, we get

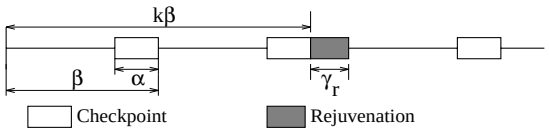

Figure 5: Program with checkpointing and rejuvenation

$$E\,[T_c(N\beta,N)]\ \overline{F}_X(\beta) =$$

$$N\beta\,\overline{F}_X(N\beta) + \gamma_f\, F_X(N\beta) + \int_0^{N\beta} x\, dF_X(x) +$$

$$\sum_{i=1}^{N-1} E[T_c((N-i)\beta,N-i)]\,[F_X((i+1)\beta) - F_X(i\beta)]$$

□

The above expression for $E[T_c(N\beta,N)]$ is a recurrence relation which involves a weighted sum of $E[T_c(i\beta,i)]$ forall $1 \le i < N$ and does not have a simple closed form solution. However, a numerical iterative or recursive solution is straight forward.

4.3 Checkpointing and Rejuvenation Combined

Under our checkpointing and rejuvenation model, the program takes a total of N checkpoints and rejuvenates every kth checkpoint, with $1 \le k \le N-1$ (no rejuvenation is performed after the program completes the last checkpoint). This divides the program execution in $\lceil N/k \rceil$ segments separated by rejuvenation. Two cases arise: if $N/k = \lceil N/k \rceil$, then each of the N/k segments have work requirement $k\beta$. However, If $N/k \neq \lceil N/k \rceil$, then each of the first $\lfloor N/k \rfloor$ segments have work requirement $k\beta$ and the last segment consisting of $k' = N - k\lfloor N/k \rfloor$ checkpoints has a work requirement $k'\beta$. The expected completion time of the program is given by the following theorem:

Theorem 3: $E[T_{rc}(N\beta,N,k)] =$

$$\begin{cases} \dfrac{N}{k}\,E[T_c(k\beta,k)] + \left(\dfrac{N}{k}-1\right)\gamma_r, & \text{if } \dfrac{N}{k} = \left\lceil \dfrac{N}{k} \right\rceil \\ \left\lfloor \dfrac{N}{k} \right\rfloor (E[T_c(k\beta,k)] + \gamma_r) + E[T_c(k'\beta,k')], & \text{otherwise} \end{cases}$$

Proof: Let the completion time of the ith segment be denoted by the random variable Z_i, where $1 \le i \le$

$\lceil N/k \rceil$. The expected completion time of the whole program is equal to the expectation of the summation of these random variables plus the expected time spent in rejuvenation. The total number of rejuvenations performed if $N/k = \lceil N/k \rceil$ is $N/k - 1$ else it is $\lfloor N/k \rfloor$. Therefore, $E[T_{rc}(N\beta, N, k)] =$

$$\begin{cases} E\left[\sum_{i=1}^{N/k} Z_i\right] + \left(\frac{N}{k} - 1\right)\gamma_r, & \text{if } \frac{N}{k} = \left\lceil \frac{N}{k} \right\rceil \\ E\left[\sum_{i=1}^{\lceil N/k \rceil} Z_i\right] + \lfloor N/k \rfloor \gamma_r, & \text{otherwise} \end{cases}$$

which by linearity of expectation becomes $E[T_{rc}(N\beta, N, k)] =$

$$\begin{cases} \sum_{i=1}^{N/k} E[Z_i] + \left(\frac{N}{k} - 1\right)\gamma_r, & \text{if } \frac{N}{k} = \lceil \frac{N}{k} \rceil \\ \sum_{i=1}^{\lceil N/k \rceil} E[Z_i] + \lfloor N/k \rfloor \gamma_r, & \text{otherwise} \end{cases}$$

After rejuvenation, as the failure process gets renewed, Z_i's are independent random variables. For i between 1 and $\lfloor N/k \rfloor$, Z_i's are also identical as work requirement for each of these is $k\beta$. The work requirement for the last random variable, however, is $k^{'}\beta$. Finally, each of these segments is an instance of the case where only checkpointing is employed. Therefore, theorem 2 can be applied to calculate their expected completion times.

$$E[Z_i] = \begin{cases} E[T_c(k\beta, k)] & 1 \le i \le \lfloor N/k \rfloor \\ E[T_c(k^{'}\beta, k^{'})] & i = \lceil N/k \rceil \end{cases}$$

Substituting in the equation for $E[T_{rc}(N\beta, N, k)]$ yields the desired result. □

5 Results

So far in our expressions, we have kept the failure distribution as any general distribution $F_X(\cdot)$. To obtain numerical results, we now assume it to be Weibull. This particular choice was made as it enables us to model an increasing failure rate, typical of the aging phenomenon. The distribution function is given by

$$F_X(t) = 1 - e^{-\lambda t^{\theta}}$$

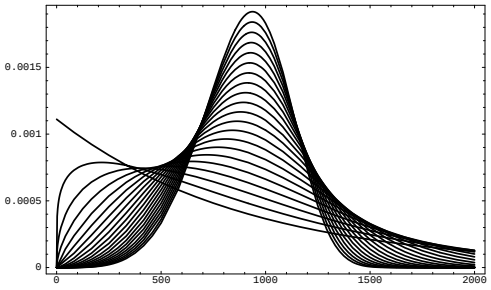

Figure 6: Density function for Weibull (MTTF = 900)

with λ and θ being the two parameters. Versatility of Weibull lies in the choice of θ to vary the failure rate. If $\theta < 1$, failure rate decreases with time, and if $\theta > 1$, the failure rate increases with time. For $\theta = 1$, Weibull reduces to the exponential distribution. Furthermore, for a given mean of the distribution, higher values of θ imply peakier nature in the density function where larger probabilities of failure are concentrated in a small time region. The mean time to failure is given by [21]

$$MTTF = \left(\frac{1}{\lambda}\right)^{1/\theta} \Gamma(1 + 1/\theta) \qquad (1)$$

Since the actual value of θ is not known, we obtain the results for many different values of θ. It can be seen from the numerical results that for the same mean time to failure, the benefit from rejuvenation increases for peakier failure densities (larger values of θ).

We use the following values for various parameters. The failure free execution time $w = 1200$ minutes. Checkpointing takes $\alpha = 4$ minutes. Restarting from a failure and rejuvenating, both take $\gamma_r = \gamma_f = 5$ minutes. The mean time to failure in the absence of rejuvenation $MTTF = 900$ minutes. θ takes values from 1.0, to 5.0. Given the MTTF and θ, λ is calculated using equation 1. Figure 6 plots the Weibull density function for different θ values, MTTF being fixed at 900. Larger the value of θ, peakier is the density curve. The rejuvenation and the restart times are close to the time it takes to reboot a large computer. Checkpointing time is close to the time it takes to save critical data associated with a large scientific application program on hard-disk.

Given all the above parameters, $E[T(w)]$ results in a numeric value (for a specific value of θ). $E[T_c(N\beta, N)]$ remains a function of N, and $E[T_{rc}(N\beta, N, k)]$ remains

θ	$E[T(w)]$	$E[T_c(N^*\beta, N^*)]$	N^*	$E[T_{rc}(N^*\beta, N^*, k^*)]$	N^*	k^*
1.0	2528.27	1328.01	15	1333.01	15	8-14
1.2	2653.83	1321.38	14	1321.20	14	7
1.4	2792.16	1326.80	14	1310.39	12	4
1.6	2945.42	1313.41	13	1299.83	11	3
1.8	3116.06	1311.00	13	1290.10	10	2
2.0	3306.92	1309.30	13	1281.57	8	2
2.2	3521.31	1308.11	13	1274.55	8	2
2.4	3763.15	1307.28	13	1268.16	6	1
2.6	4037.14	1306.68	12	1262.59	5	1
3.4	5586.91	1305.96	12	1247.76	4	1
4.4	9461.95	1306.35	12	1236.68	4	1

Table 1: Optimal expected completion times

a function of N and k. In our calculations, N was varied from 1 to 50. and k was varied from 1 to $N-1$. The values which yielded the minimum expected completion times are the optimal values of total number of checkpoints and the rejuvenation distance denoted by N^* and k^* respectively. Table 1 lists these optimal results.

As can be seen from the table, the benefit from rejuvenation depends on the nature of the density function. In the case of exponentially distributed time to failure ($\theta = 1.0$), the expected completion time with rejuvenation is in fact more than when only checkpointing is employed. Due to its memoryless property, the exponential distribution renews every instant and the deliberate renewal by rejuvenation does not buy any additional benefit. Indeed, it simply results in an overhead of γ_r every time it is performed. From the table, for $\theta = 1.0$, for a total of 15 checkpoints, if the rejuvenation distance takes a value from 8 to 14, only one rejuvenation is performed costing 5 units. Therefore, $E[T_{rc}(N^*\beta, N^*, k^*)] = E[T_c(N^*\beta, N^*)] + 5$. The memoryless property of exponential distribution is contradictory to the phenomenon of "aging" and does not represent the time to failure distribution of software with transient faults. Henceforth, we shall discuss the results for $\theta > 1.0$ only.

For larger values of θ, the memoryless property is no longer valid and we can see that the benefit from rejuvenation and checkpointing as opposed to only checkpointing starts to increase. The optimal expected completion time is plotted in Figure 7 against θ which is self-explanatory.

Table 1 also shows the values of N^* and k^*. Three important observations can be made; First, the optimal number of checkpoints decreases with θ for the only checkpointing case as well as for checkpointing with rejuvenation combined. Second, for the same θ, N^* for checkpointing and rejuvenation is less than or equal to

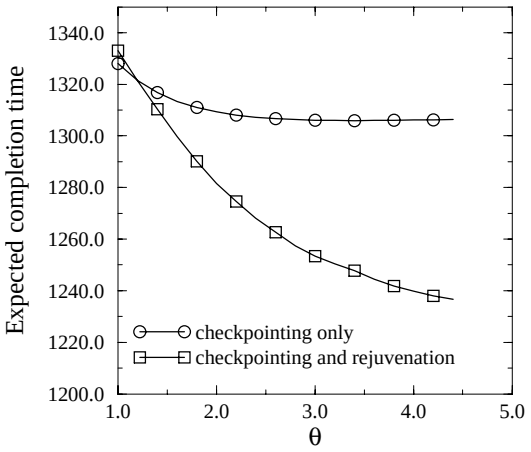

Figure 7: Effect of θ on completion time

N^* for the only checkpointing case. The effect of rejuvenation is to reduce the mean number of failures that may occur in a given period which results in using lesser number of checkpoints. Last, the optimal rejuvenation distance decreases with increase in θ. Note that k is in units of per checkpoint and should not be mistaken for time to rejuvenate. The variation in time to rejuvenate with θ depends on actual parameter values and an intuitive explanation is not obvious.

Figure 8 shows the expected completion time plotted against the number of checkpoints for a particular value of θ (2.2 in this case) and illustrates the tradeoffs of the cost of checkpointing and rejuvenation. First, each of the six curves attains an optimum (minimum) value for a certain number of checkpoints. Given a rejuvenation distance (or no rejuvenation) If the checkpointing is infrequent, the amount of rollback dominates the extra time required because of failures. If the checkpointing is too frequent, the cost involved in performing these operations starts to dominate.

Second, Figure 8 also shows that as the rejuvenation distance increases, the number of checkpoints required to minimize the expected completion time also increases. Note that the value of this minima is not the same for different values of k.

6 Conclusions and Future Work

We have shown in this paper that rejuvenation can be combined with checkpointing to reduce the completion time of a program. We derived equations for expected

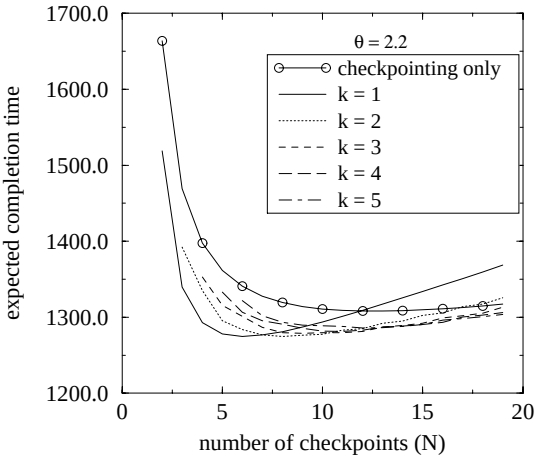

Figure 8: Effect of rejuvenation on completion time

completion times for the three possible cases and compared the results numerically. One natural extension of this work is to derive the distribution of the completion time for these cases. This will enable us to evaluate other performance measures such as the probability of completion by a given deadline which apply to real time systems. Another extension is to formulate checkpointing and rejuvenation as a two-dimensional dynamic optimization problem. [19] is an example in which dunamic programming is used to find the optimal checkpointing policy. Using this approach, the equidistance assumed in checkpointing and rejuvenation models is not necessary.

Checkpointing has also been used to maximize availability of transaction based database systems and a wide body of literature exists in its analysis. Again, finding the optimal checkpointing strategy has been a problem of interest. Rejuvenation can be combined with checkpointing and analyzed as a two-dimensional optimization problem for such transaction based systems.

In computer systems, as the cause of failures shifts from hardware to software and as the transient nature of software failures is accentuated, alternative fault tolerant techniques such as rejuvenation prove useful. Combining other corrective techniques with this preventive technique may also be beneficial and should be explored. We hope that this paper will spur further research in this area.

References

[1] M. Sullivan and R. Chillarege, "Software defects and their impact on system availability - A study of field failures in operating systems", in *Proc. IEEE Fault-Tolerant Computing Symposium*, pp. 2-9, 1991.

[2] J-C. Laprie, J. Arlat, C. Béounes and K. Kanoun, "Architectural issues in software fault-tolerance", *Software Fault Tolerance*, Ed. M. R. Lyu, John, Wiley & sons. ltd., pp. 47-80, 1995.

[3] Y. Huang and C. Kintala, "Software fault-tolerance in the application layer", *Software Fault Tolerance*, Ed. M. R. Lyu, John, Wiley & sons. ltd., pp. 231-248, 1995.

[4] Y. Huang, C. Kintala, N. Koletis, N. D. Fulton, "Software Rejuvenation- design, implementation and analysis", Proc. of *Fault-tolerant Computing Symposium*, Pasadena, CA, June 1995.

[5] Y-M Wang, Y. Huang, P. Vo, P-Y Chung and C. Kintala, "Checkpointing and its applications", In Proc. of *Symposium on Fault Tolerant Computer Systems*, Pasadena, California, 1995.

[6] J. Gray, "A census of tandem system availability between 1985 and 1990", *IEEE Trans. on Reliability*, Vol. 39, pp. 409-418, Oct. 1990.

[7] J. Gray, "Why do computers stop and what can be done about it?", *Proc. of 5th Symp. on Reliability in Distributed Software and Database Systems*, pp. 3-12, January 1986.

[8] J. Gray and D. P. Siewiorek, "High-availability computer systems", *IEEE Computer Mag.*, pp. 39-48, Sept. 1991.

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

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

[11] P. Jalote, Y. Huang and C. Kintala, "A framework for understanding and handling transient failures", In *Proc. of 2nd ISSAT Intnl. Conf. on Reliability and Quality in Design*, March 8-10, 1995, Orlando, Florida, pp.231-237.

[12] Inhwan Lee, "Software dependability in the operational phase", *Ph.D. Thesis*, Dept. of Electrical and Computer Engineering, Univ. of Illinois, Urbana-Champaign, 1995.

[13] P. E. Ammann and J. C. Knight, "Data-diversity: an approach to software fault-tolerance", *Proc. of 17th Intnl. Symp. on Fault Tolerant Computing*, pp. 122-126, June 1987.

[14] K. G. Shin, T. Lin and Y. Lee, "Optimal checkpointing of real-time tasks", *IEEE Transactions on Computers*, Vol. C-36, No. 11, November 1987.

[15] C. H. C. Leung and Q. H. Choo, "On the execution of large batch programs in unreliable computing systems", *IEEE Trans. on Software Engg.*, Vol. SE-10, No. 4, July 1984, pp. 444-450.

[16] E. G. Coffman and E. N. Gilbert, "Optimal strategies for scheduling checkpoints and preventive maintenance" *IEEE Trans. on Reliability*, Vol. 39, No. 1, April 1990, pp. 9-18.

[17] A. Duda, "The effects of checkpointing on program execution time", *Information Processing Letters*, Vol. 16, pp. 221-229, 1983.

[18] V.G. Kulkarni, V.F. Nicola and K. S. Trivedi, "Effects of checkpointing and queuing on program performance", *Communications on Statistics-Stochastic Models*, 6(4), 615-648, 1990.

[19] S. Toueg and O. Babouglu, "On the optimum checkpoint selection problem", *SIAM Journal on Computing*, Vol. 13, No. 3, pp. 630-649, August 1984.

[20] R. Geist, R. Reynolds and J. Westall, "Selection of a checkpoint interval in a critical-task environment", *IEEE Trans. on Reliability*, 37(4), 395-400, October 1988.

[21] K. S. Trivedi, "Probability and Statistics with reliability, queuing and computer science applications", *Prentice-Hall*, 1982.