On Checkpoint Latency

Nitin H. Vaidya
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112
E-mail: vaidya@cs.tamu.edu
Phone: (409) 845-0512
FAX: (409) 847-8578

Technical Report 95-015
March 1995*

Abstract

Checkpointing and rollback is a technique to minimize the loss of computation in the presence of failures. Two metrics can be used to characterize a checkpointing scheme: (i) checkpoint *overhead* (increase in the execution time of the application because of a checkpoint), and (ii) checkpoint *latency* (duration of time required to save the checkpoint). For many checkpointing methods, checkpoint latency is larger than checkpoint overhead. The contributions of this report are as follows:

1. The report evaluates the expression for "average overhead" of the checkpointing scheme as a function of checkpoint latency and overhead.
2. A mechanism that attempts to reduce checkpoint overhead usually causes an increase in the checkpoint latency. A decrease in checkpoint overhead can result in an increase or a decrease in the *average* overhead, depending on whether the latency is increased "too much" or not. This report determines a function g of checkpoint overhead C such that checkpoint latency should be less that $g(C)$ to achieve a *decrease* in the *average* overhead.
3. For equi-distant checkpoints, the optimal checkpoint interval is shown to be independent of the checkpoint latency (L).

*Revised April 14, 1995.

1 Introduction

Many applications (sequential and parallel) require large amount of time to complete. Such applications can encounter loss of a significant amount of computation if a failure occurs during the execution. *Checkpointing and rollback recovery* is a technique used to minimize the loss of computation in an environment subject to failures [1]. A *checkpoint* is a copy of the application's state stored on a *stable* storage – a *stable* storage is not subject to failures. The application periodically saves checkpoints; the application recovers from a failure by *rolling back* to a recent checkpoint. Checkpointing can be used for sequential as well as parallel (or distributed) applications. When the application consists of more than one process, a *consistent* checkpointing algorithm can be used to save a consistent state of the multi-process application [2, 6, 11].

Two metrics can be used to characterize a checkpointing scheme:

- Checkpoint overhead: Checkpoint *overhead* is the increase in the execution time of the application because of a checkpoint. We denote checkpoint overhead as C.
- Checkpoint latency: Checkpoint *latency* is the duration of time required to save the checkpoint. In many implementations, checkpoint *latency* is larger than the checkpoint *overhead*. (Section 2 illustrates this with an example.) We denote checkpoint latency as L.

In the past, a large number of researchers have analyzed the checkpointing and rollback recovery scheme (for instance, to determine the optimal checkpoint interval) [1, 4, 5, 7, 8, 12, 16]. This report evaluates the impact of checkpoint latency on the performance of a checkpointing scheme. The contributions of this report are as follows:

1. The report evaluates the *"average overhead"* of a checkpointing scheme, as a function of checkpoint latency (L) and checkpoint overhead (C). The *overhead* of a checkpointing scheme consists of two components: failure-free overhead due to checkpointing, and recovery overhead due to rollback.
2. A mechanism that attempts to reduce checkpoint overhead C usually causes an increase in the checkpoint latency L. It can be seen that, an increase in the checkpoint latency L, keeping C constant, results in an increase in the average overhead. Thus, a decrease in checkpoint overhead (C) can result in an increase or a decrease in the average overhead, depending on whether the latency is increased "too much" or not. This report determines a function g of checkpoint overhead C such that checkpoint latency L should be less that $g(C)$ to achieve a *decrease* in the average overhead.
3. For equi-distant checkpoints, the optimal checkpoint interval is shown to be independent of the checkpoint latency (L).

Related work: Plank et al. [11, 10, 9] present measurements of checkpoint latency and overhead for a few applications, however, they do not present any performance analysis. We also measured checkpoint latency and overhead for a few uni-process applications, and briefly analyzed the impact of checkpoint latency on performance of "two-level" recovery schemes [14]. This report presents results on the impact of checkpoint latency on traditional checkpointing and rollback schemes.

This report is organized as follows. Section 2 illustrates the difference between checkpoint *overhead* and *latency* by means of two checkpointing schemes. Section 3 discusses how to model latency. Performance analysis is presented in Sections 4 and 5. Section 6 discusses parallel and distributed applications. The report concludes with Section 7.

2 Checkpoint Latency

We limit the initial discussion and analysis to uni-process applications. We will address multi-process applications in Section 6.

For uni-process applications, checkpointing schemes have been proposed that achieve a low checkpoint overhead, while resulting in a large checkpoint latency [11, 3, 9]. In this section, we illustrate the distinction between checkpoint *latency* and checkpoint *overhead* with two examples of checkpointing schemes for uni-process applications.

Sequential checkpointing is an approach for which checkpoint *overhead* is identical to checkpoint *latency*. In this approach, when an application process wants to take a checkpoint, it pauses and saves its state on the stable storage [10]. The process continues execution only *after* the state is completely saved on the stable storage. Therefore, the time required to save the checkpoint (i.e., checkpoint latency) is practically identical to the increase in the execution time of the process (i.e., checkpoint overhead). Figure 1 illustrates this approach. The horizontal line represents processor execution, time increasing from left to right. The shaded box represents the checkpointing operation, the width of the box being equal to latency and overhead of sequential checkpointing. The sequential checkpointing approach achieves the *smallest* possible checkpoint *latency*. However, it results in a larger checkpoint *overhead* as compared to other approaches.

Forked checkpointing is an approach for which checkpoint *overhead* is usually much smaller than the checkpoint *latency*. In this approach, when a process wants to take a checkpoint, it *forks* a child process [10]. The state of the child process is identical to that of the parent process when `fork` is performed. After the `fork`, the parent process continues computation, while the child process saves its state on the stable storage. Figure 2(a) illustrates this approach. In this approach, computation is overlapped with stable storage access (i.e., overlapped with state saving), therefore the checkpoint *overhead* is smaller than the *sequential* checkpointing approach. Also, as the parent and the child execute in parallel, checkpoint *latency* is larger than the checkpoint

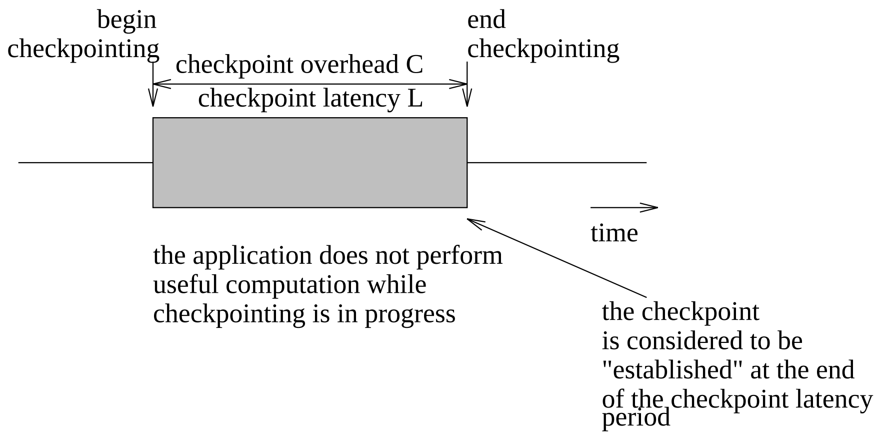


Figure 1: Sequential Checkpointing

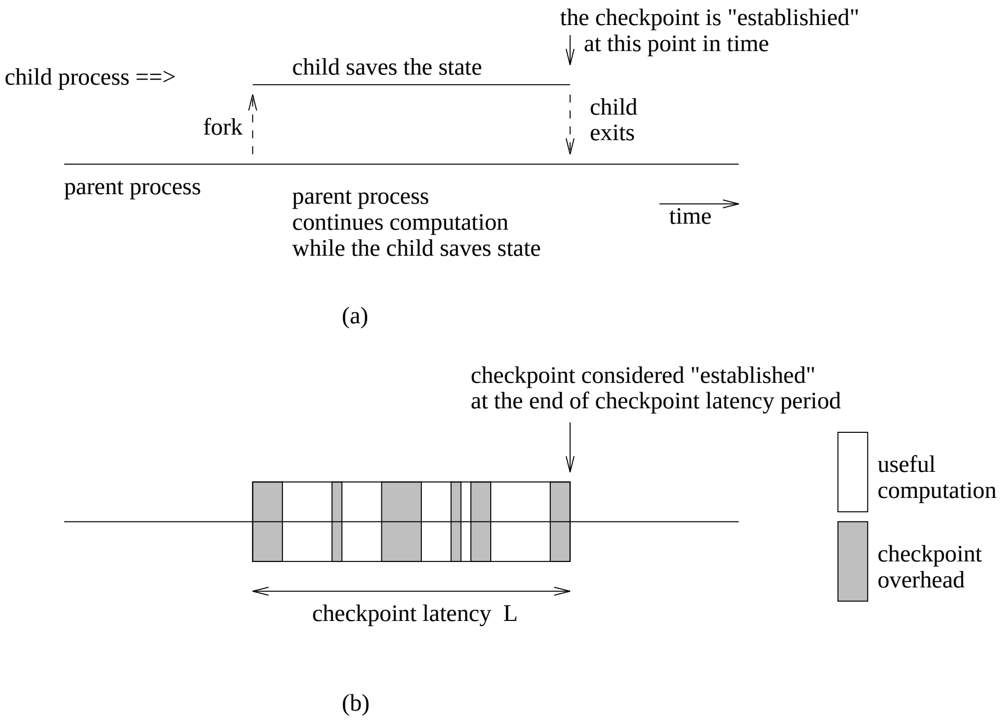


Figure 2: Forked Checkpointing

overhead. Figure 2(b) illustrates the interleaved execution of the child and parent processes. As shown in the figure, useful computation is interleaved with the checkpointing operation.

For future reference, note that, a checkpoint is said to have been *established* if a future failure can be tolerated by a rollback to this checkpoint. Thus, a checkpoint is not considered to be *established* until the end of the latency period. When the execution progresses past the end of the checkpoint latency period, the checkpoint is considered to have been established (refer Figures 1 and 2).

A checkpoint interval is defined as the duration between the *establishment* of two consecutive checkpoints. That is, an *interval* begins when one checkpoint is established, and the interval ends when the next checkpoint is established. We assume that the checkpoints are equi-distant, i.e., the amount of useful computation performed during each interval is identical. Let T denote the amount of useful computation performed in each interval.

3 A Simple Representation of Latency and Overhead

As discussed in the previous section, the checkpoint latency period is divided into two types of execution: (1) useful computation, and (2) execution necessary for checkpointing. The two types are usually interleaved in time. However, for modeling purposes, we can assume that the two types of executions are separated in time, as shown in Figure 3. As shown in the figure, the first C units of time during the checkpoint latency period is assumed to be used for saving the checkpoint. The remaining $(L - C)$ units of time is assumed to be spent on useful computation. Although the C units of overhead is modeled as being incurred at the beginning of the checkpoint latency period, the checkpoint is considered to have been *established* only at the end of the checkpoint latency period.

Although the above representation of checkpoint latency and overhead is simplified, we now demonstrate that it will lead to accurate analysis. Two distinct situations may occur when an interval is executed.

1. A failure does not occur while the interval is executed. In this case, the execution time from the beginning to the end of an interval is $T + C$. Of the $T + C$ units, T units are spent doing *useful* computation, while incurring an overhead of C time units. As shown in Figure 4(a), $(L - C)$ units of useful computation is performed during the checkpoint latency period. Now consider Figure 4(b). Similar to Figure 4(a), $L - C$ units of useful computation is performed during the latency period. Also, the execution time for the interval is $T + C$.

2. A failure occurs sometime during the interval. When a failure occurs, the task must be rolled back to the previous checkpoint, incurring an overhead of R time units. In Figure 5(a), the

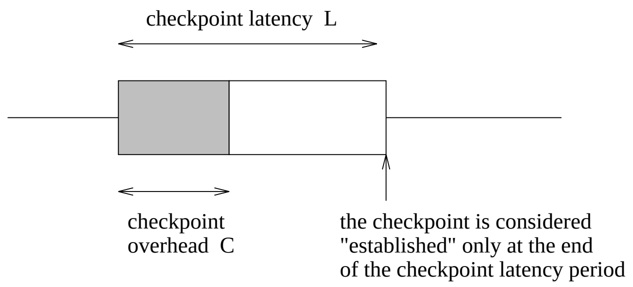


Figure 3: Modeling checkpoint latency and overhead

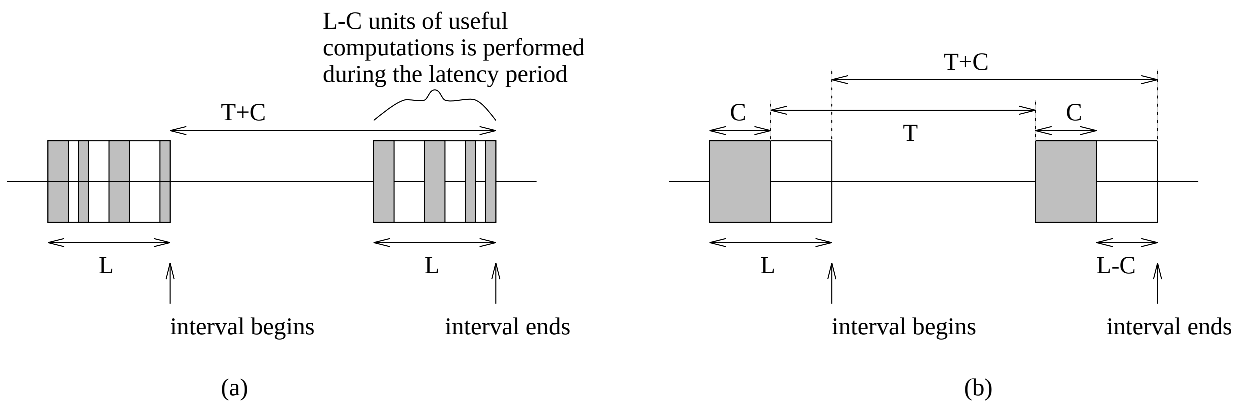


Figure 4: Situation 1

task is rolled back to checkpoint CP1. After the rollback, $L - C$ units of useful computation performed during the latency period of checkpoint CP1 must be performed again – this is necessary, because the state saved during checkpoint CP1 is the state at the *beginning* of the latency period for checkpoint CP1. In the absence of a further failure, additional $T + C$ units of execution is required before the completion of the interval. Thus, after a failure, $R + (L - C) + (T + C) = R + T + L$ units of execution is required before the completion of the interval, *provided* additional failures do not occur.

Now consider Figure 5(b). When the failure occurs, as shown in Figure 5(b), the system can be considered to have rolled back to the end of the "shaded portion" in the latency period for checkpoint CP1. (Note that no state change occurs during the "shaded portion".) Now it is apparent that, in the absence of further failure, $R + T + L$ units of execution is required to complete the interval. Thus, our representation of checkpoint latency and overhead yields the same conclusion as the more accurate representation in Figure 5(a).

The above discussion is also applicable if the failure occurs during the checkpoint latency period of checkpoint CP2. Such a failure will also require a rollback to checkpoint CP1, as checkpoint CP2 is *not* established when the failure occurred.[1]

The above cases imply that the simplified representation of checkpoint latency and overhead will yield the same results as the accurate representation.

4 Evaluating the Overhead

Note that the emphasis of this report is on understanding the impact of checkpoint latency on performance. The emphasis is *not* on presenting elaborate models for checkpointing schemes, as in many previous works. Therefore, this report uses a simple model that is adequate for our purpose. (For instance, we assume that C and L are constants for a given scheme. A more elaborate model may assume C and L to be some function of time.)

The fault model assumed for the analysis is as follows: Processor failures are governed by a Poisson process with rate λ. When a processor fails, its local state is corrupted. A processor can

[1] Chandy et al. [1] present an analysis of checkpointing schemes that does not take checkpoint latency into account. However, for sequential checkpointing (with $L = C$), our analysis is similar to theirs with one exception. An assumption made by Chandy et al. [1] implies that a failure that occurs while checkpoint CP2 (in Figure 5) is being saved, only requires re-initiation of the checkpointing operation. As per their assumption, computation during the interval preceding checkpoint CP2 need not be performed again even if a failure occurs while checkpoint CP2 is being established (i.e. the failure occurs after checkpointing is initiated but before it is completed). For many environments this assumption is not realistic. Therefore, we make the realistic assumption that a failure that occurs while a checkpoint (say, CP2 in Figure 5) is being established requires a rollback to the previous checkpoint (checkpoint CP1 in our example).

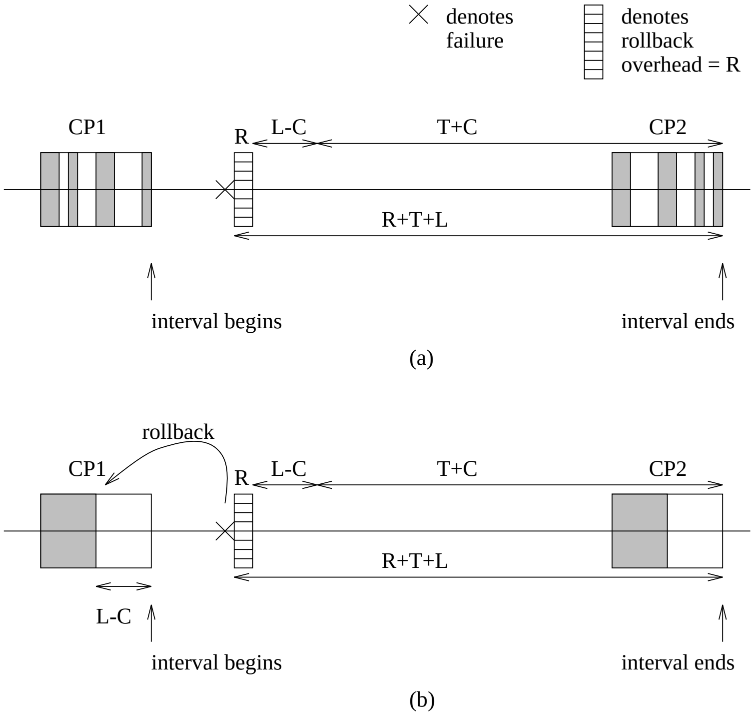


Figure 5: Situation 2

fail during normal operation, during checkpointing, as well as during rollback and recovery. The *stable* storage is not subject to failures. (The stable storage is used for storing checkpoints.)

Let $G(t)$ denote the expected (average) amount of execution time required to perform t units of *useful* computation. (*Useful* computation excludes the time spent on checkpointing and rollback recovery.) Then, we define *overhead ratio* (r) as:

$$\text{overhead ratio} \quad r \;=\; \lim_{t\to\infty} \frac{G(t)-t}{t} \;=\; \lim_{t\to\infty} \frac{G(t)}{t} \;-\; 1.$$

Note that $r \geq 0$, smaller r implying a smaller average overhead. In this report, we measure the average performance overhead of a recovery scheme in terms of its *overhead ratio*. In the remainder of this section, an expression for the overhead ratio is derived.

We assume that the system is executing an infinite task that takes *equi-distant* checkpoints, i.e. the amount of useful computation performed between consecutive checkpoints is fixed (the amount is denoted as T). The execution of the task can be considered to be a series of *intervals*, each interval *beginning* immediately after a checkpoint is established, and *ending* when the next checkpoint is established. Figure 6 illustrates intervals I1, I2 and I3. (The meaning of various *state*s in the figure will be explained shortly.) As shown in the figure, interval I1 begins after checkpoint CP0 is established, and ends when checkpoint CP1 is established. It is possible that during an interval some failures occur. As shown in the figure, interval I2 begins after checkpoint CP1 is established. Before checkpoint CP2 is established, two failures occur, each requiring a rollback to checkpoint CP1. Subsequently, the computation progresses without failure and checkpoint CP2 is established. Interval I2 is completed when checkpoint CP2 is established.

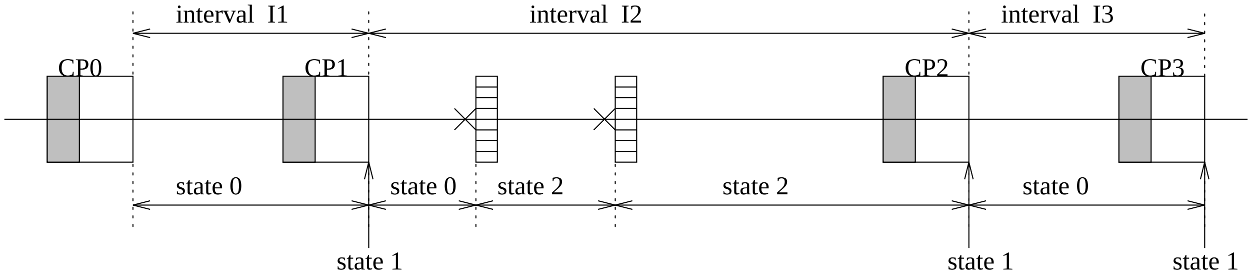


Figure 6: Intervals

Observe that T units of *useful* computation is performed during each interval. Provided no failures occur during the interval, the total time required to execute an interval is $T+C$. If one or more failure occurs while executing an interval, then the execution time is longer than $T+C$. Let Γ denote the expected (average) execution time of an interval. Then, it is easy to see that,

$$\begin{aligned} \text{overhead ratio} \quad r &= \lim_{t\to\infty} \frac{G(t)}{t} - 1 \\ &= \frac{\Gamma}{T} - 1 \end{aligned}$$

Expected execution time Γ of a single interval can be evaluated using the 3-state discrete Markov chain [13, 17] presented in Figure 7. State 0 is the initial state, when an interval starts execution. A transition from state 0 to state 1 occurs if the interval is completed without a failure. If a failure occurs while executing the interval, then a transition is made from state 0 to state 2. After state 2 is entered, a transition occurs to state 1 if no further failure occurs before the next checkpoint is established. If, however, another failure occurs after entering state 2 and before the next checkpoint is established, then a transition is made from state 2 back to state 2. When state 1 is entered, the interval has completed execution. Therefore, state 1 is a sink state – there are no transitions out of state 1. Figure 6 illustrates the various states for an example execution. As shown in Figure 6, during interval I1, the task is in state 0, and enters state 1 when the interval completes without a failure. During interval I2, the task is initially in state 0, and enters state 2 when a failure occurs. When another failure occurs, the task remains in state 2. Subsequent to the second failure, interval I2 completes without any further failures. State 1 is entered at the end of the interval.

Each transition (X, Y), from state X to state Y in the Markov chain, has an associated *transition probability* P_{XY} and a *cost* K_{XY}. *Cost* K_{XY} of a transition (X, Y) is the expected (average) time spent in state X before making the transition to state Y.

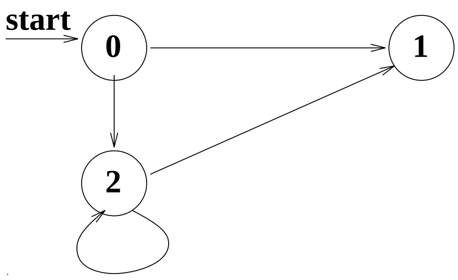


Figure 7: Markov chain

State 0 is the initial state, when an interval starts execution. A transition from state 0 to state 1 occurs if the interval is completed without a failure. If this transition is made, then $T+C$ units of time is spent in state 0 (while executing the interval) before the transition occurs. Therefore, the transition probability of transition (0,1) is

$$P_{01} = e^{-\lambda(T+C)}$$

and the cost is

$$K_{01} = T + C.$$

If a failure occurs while executing the interval, then a transition is made from state 0 to state 2. The probability of this transition is

$$P_{02} = 1 - P_{01} = 1 - e^{-\lambda(T+C)}$$

The cost of this transition is the expected duration, from the beginning of the interval until the time when the failure occurred, given that a failure occurs before the end of the interval. Therefore,

$$K_{02} = \int_0^{T+C} (t) \frac{\lambda\, e^{-\lambda t}}{1 - e^{-\lambda(T+C)}} dt = \lambda^{-1} - \frac{(T+C)e^{-\lambda(T+C)}}{1 - e^{-\lambda(T+C)}}$$

After state 2 is entered, a transition is made to state 1 if no further failure occurs before the next checkpoint is established. As discussed earlier, the execution time required for the next checkpoint to be established after a failure is $R + T + L$. Therefore,

$$P_{21} = e^{-\lambda(R+T+L)}$$

and

$$K_{21} = R + T + L.$$

If, however, another failure occurs after entering state 2 and before the next checkpoint is established, then a transition is made from state 2 back to state 2. For this transition, the transition probability is given by

$$P_{22} = 1 - P_{21} = 1 - e^{-\lambda(R+T+L)}.$$

The cost of this transition can be evaluated similar to K_{02}, as

$$K_{22} \;=\; \int_0^{R+T+L} (t) \frac{\lambda\, e^{-\lambda t}}{1 - e^{-\lambda(R+T+L)}} dt \;=\; \lambda^{-1} - \frac{(R+T+L)e^{-\lambda(R+T+L)}}{1 - e^{-\lambda(R+T+L)}}$$

When state 1 is entered, the interval has completed execution. Therefore, there are no transitions out of state 1.

In the above, we have described various transitions in the Markov chain, and derived the transition probability and cost of each transition. The expected time, Γ, required to execute one interval is the expected *cost* of a path from state 0 to state 1. It follows that,

$$\Gamma \;=\; P_{01}K_{01} + P_{02}\left(K_{02} + \frac{P_{22}}{1 - P_{22}}K_{22} + K_{21}\right)$$

Substituting the expressions for various costs and transition probabilities, into the above expression for Γ, yields the following:

$$\begin{aligned}\Gamma \;=\; & e^{-\lambda(T+C)}(T+C) + \\ & (1 - e^{-\lambda(T+C)}) \times \\ & \left[\lambda^{-1} - \frac{(T+C)e^{-\lambda(T+C)}}{1 - e^{-\lambda(T+C)}} + \frac{1 - e^{-\lambda(R+T+L)}}{e^{-\lambda(R+T+L)}}\left(\lambda^{-1} - \frac{(R+T+L)e^{-\lambda(R+T+L)}}{1 - e^{-\lambda(R+T+L)}}\right)\right. \\ & \left. \quad + R + T + L \quad \right]\end{aligned}$$

On simplification, the above expression reduces to the following:

$$\Gamma \;=\; \lambda^{-1}(1 - e^{-\lambda(T+C)})e^{\lambda(R+T+L)} \;=\; \lambda^{-1}e^{\lambda(L-C+R)}(e^{\lambda(T+C)} - 1) \tag{1}$$

It follows that, the overhead ratio r is given by

$$r \;=\; \frac{\Gamma}{T} - 1 \;=\; \frac{\lambda^{-1}e^{\lambda(L-C+R)}(e^{\lambda(T+C)} - 1)}{T} - 1 \tag{2}$$

In practice, checkpoint overhead is always non-zero, therefore, we assume $C > 0$.

4.1 Minimizing the Overhead Ratio

Consider a checkpointing scheme that achieves a certain overhead C and checkpoint latency L. For this checkpointing scheme, the objective now is to choose an appropriate value of T so as to minimize the overhead ratio r. The optimal value of T must satisfy the following equation:

$$\frac{\partial r}{\partial T} = 0 \tag{3}$$

$$\Longrightarrow \frac{\partial}{\partial T}\left[\frac{\lambda^{-1}e^{\lambda(L-C+R)}(e^{\lambda(T+C)}-1)}{T} - 1\right] = 0$$

$$\Longrightarrow \frac{\partial}{\partial T}\left(\frac{e^{\lambda(T+C)}-1}{T}\right) = 0$$

$$\Longrightarrow \frac{e^{\lambda(T+C)}(\lambda T - 1) + 1}{T^2} = 0$$

$$\Longrightarrow e^{\lambda(T+C)}\,(1-\lambda T) = 1 \qquad \text{for } T \neq 0 \tag{4}$$

As shown in the Appendix, there exists only one positive value of T that satisfies the above equality, and r is minimized at this value of T.

As Equation 4 does not include L or R, the optimal value of T is *not* dependent on L and R – the optimal T, however, depends on C. Thus, to evaluate the optimal checkpoint interval for a given checkpointing scheme, it is adequate to know the value of C (the value of L is not necessary). However, to evaluate the overhead ratio with the optimal T, L must also be known.

5 Inter-Dependence Between L and C

Checkpoint latency and checkpoint overhead are dependent on each other. An attempt to reduce the checkpoint overhead typically causes an increase in the checkpoint latency. Figure 8 illustrates a hypothetical example of three approaches for taking a checkpoint, resulting in three possible combinations of latency and overhead. Point 1 in the figure corresponds to *sequential* checkpointing – for this, the overhead and latency are identical. Points 2 and 3 correspond to two other approaches that result in a decrease in the overhead, while increasing the latency. (In general, decrease in checkpoint overhead is *not* necessarily equal to the increase in latency.) Now, from Equation 2,

$$\frac{\partial r}{\partial L} = \frac{e^{\lambda(L-C+R)}(e^{\lambda(T+C)}-1)}{T} > 0 \tag{5}$$

Observation 1: Above implies that, as L increases, while C and T are fixed, overhead ratio increases.

For instance, Figure 9(a) plots the overhead ratio as a function of L, for $C = R = 10$, $\lambda = 10^{-5}$ and $T = 1000$. As shown, overhead ratio increases monotonically as L increases. (Note: We do not imply that all the L-C pairs used to plot the graph are achievable. The graph simply demonstrates the dependence of r on L.)

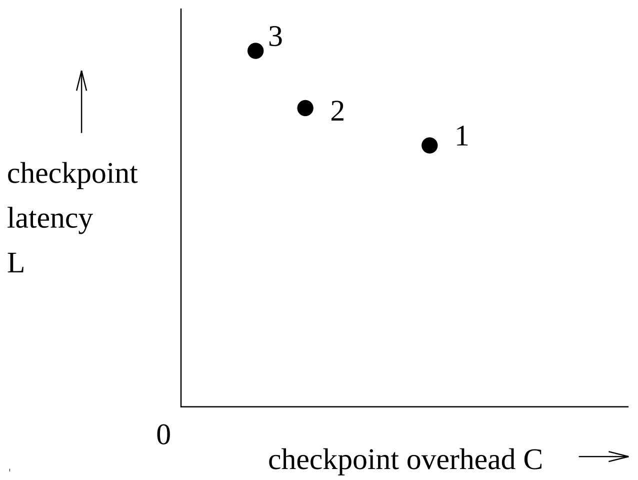
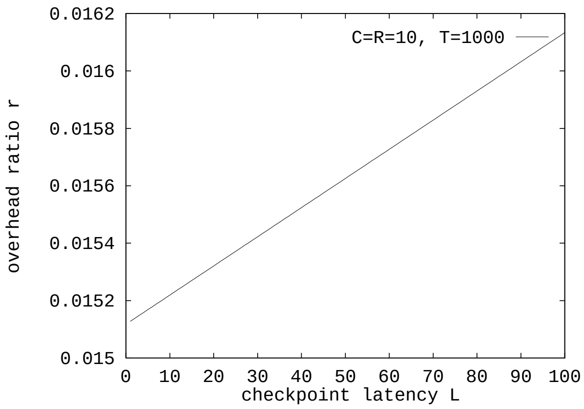
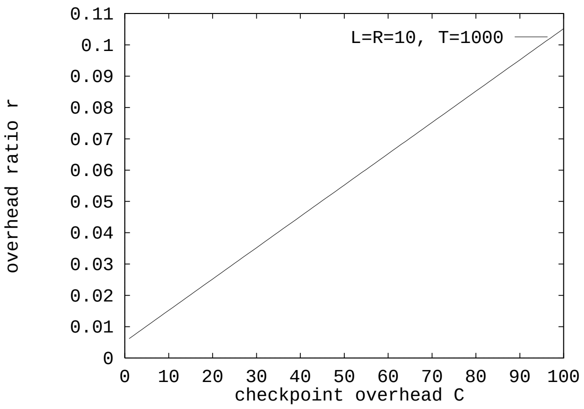


Figure 8: Inter-dependence between L and C



(a) r versus L: other parameters fixed



(b) r versus C: other parameters fixed

Figure 9: Dependence of overhead ratio on L and C

Now observe that

$$\frac{\partial r}{\partial C} \;=\; \frac{e^{\lambda(L-C+R)}}{T} \;>\; 0 \tag{6}$$

This implies that, as C increases, while L and T are fixed, the overhead also increases. Conversely:

Observation 2: As C decreases, while L and T are fixed, the overhead ratio decreases.

For instance, Figure 9(b) plots the overhead ratio as a function of C, for $L = R = 10$, $\lambda = 10^{-5}$ and $T = 1000$. As shown, overhead ratio decreases monotonically as C decreases. Note the the overhead ratio r is more sensitive to the changes in C, as compared to the changes in L (refer Figure 9). This is also evident from Equations 5 and 6. Observe that

$$\frac{\partial r}{\partial L} = (e^{\lambda(T+C)} - 1)\ \frac{\partial r}{\partial C}.$$

Also observe that $(e^{\lambda(T+C)} - 1)$ is likely to be much smaller than 1 for realistic values of λ, T and C. Thus, $\frac{\partial r}{\partial L}$ will typically be much smaller than $\frac{\partial r}{\partial C}$ – this implies that r is more sensitive to the changes in C, as compared to changes in L.

In practice, if some checkpointing scheme increases L and also results in an increase in C, then one will not use that checkpointing scheme.

Observation 3: In practice, an increase in latency L is accompanied by a decrease in C.

For sequential checkpointing, checkpoint overhead and latency are identical, say C_{max}. A recovery scheme that attempts to achieve a smaller checkpoint overhead (C) than sequential checkpointing will achieve a latency (L) larger than sequential checkpointing. One would not use such a scheme, unless it resulted in a lower overhead ratio r as compared to sequential checkpointing. Observations 1 and 2 imply that a recovery scheme that achieves a smaller checkpoint overhead and larger latency, as compared to sequential checkpointing, can achieve a smaller overhead ratio than sequential checkpointing, *provided that* the latency is not "too much" larger than sequential checkpointing.

It is our objective here to determine when the latency is not "too much" larger than sequential checkpointing. More precisely, the objective is to determine a function g of C such that, for any $C < C_{max}$, the overhead ratio r is smaller than the sequential checkpointing scheme if $L < g(C)$. Essentially, function $g(C)$ is such that, if $L = g(C)$ then the overhead ratio r is constant independent of the value of C (specifically, r is identical to that of the sequential checkpointing scheme).[2]

Let $r(L, C)$ denote the overhead ratio for a particular value of L and C. $r(L, C)$ is given by Equation 2. For sequential checkpointing, latency and overhead are both C_{max}. Therefore, from the above discussion, we have

$$r(C_{max}, C_{max}) = r(g(C), C)$$

That is, the overhead ratio of sequential checkpointing is equal to that of a scheme with checkpoint overhead C and latency $g(C)$. To proceed further, we need to determine which value of T is to be used for the two schemes. It is fair to use that value of T which minimizes overhead ratio r with the given checkpoint overhead C. Let T_m denote the optimal value of T for sequential checkpointing, i.e., when $C = C_{max}$. In general, let T_c denote the optimal value of T that minimizes[3] the overhead ratio for a scheme whose checkpoint overhead is C. (T_c is the solution of Equation 4.) Thus, we will use $T = T_m$ when evaluating $r(C_{max}, C_{max})$ and $T = T_c$ when evaluating $r(g(C), C)$.

[2] In the discussion here, we assume that R is a constant, independent of C. When this assumption is not valid, it is more convenient to replace $L + R$ in Equation 2 by another variable, say Z ($Z = L + R$), and model Z as a function f of C, i.e., $Z = f(C)$. Also, in this case, we can consider r to be a function $r(Z, C)$ of Z and C. Analysis similar to that presented in this section can determine function $f(C)$, similar to function $g(C)$.

[3] Recall that optimal T is not dependent on L.

Now, from Equation 2,

$$r(C_{max}, C_{max}) = \frac{\lambda^{-1} e^{\lambda R}(e^{\lambda(T_m + C_{max})} - 1)}{T_m} - 1$$

and

$$r(g(C), C) = \frac{\lambda^{-1} e^{\lambda(g(C) - C + R)}(e^{\lambda(T_c + C)} - 1)}{T_c} - 1$$

From the above three equations, it follows that, the right-hand-sides of the above two equations are equal. That is,

$$\begin{aligned} \frac{\lambda^{-1} e^{\lambda R}(e^{\lambda(T_m + C_{max})} - 1)}{T_m} - 1 &= \frac{\lambda^{-1} e^{\lambda(g(C) - C + R)}(e^{\lambda(T_c + C)} - 1)}{T_c} - 1 \\ \Longrightarrow \quad e^{\lambda g(C)} &= e^{\lambda C} \, \frac{e^{\lambda(T_m + C_{max})} - 1}{T_m} \, \frac{T_c}{e^{\lambda(T_c + C)} - 1} \qquad (7) \end{aligned}$$

As T_c is the optimal value of T for checkpoint overhead C, T_c satisfies the equality in Equation 4. From Equation 4 it follows that

$$\begin{aligned} e^{\lambda(T_c + C)} &= \frac{1}{1 - \lambda T_c} \\ \Longrightarrow \frac{T_c}{e^{\lambda(T_c + C)} - 1} &= \frac{T_c}{\frac{1}{1 - \lambda T_c} - 1} \\ \Longrightarrow \frac{T_c}{e^{\lambda(T_c + C)} - 1} &= \frac{1 - \lambda T_c}{\lambda} \qquad (8) \\ \text{Similarly,} \quad \frac{T_m}{e^{\lambda(T_m + C_{max})} - 1} &= \frac{1 - \lambda T_m}{\lambda} \qquad (9) \end{aligned}$$

From Equations 7, 8 and 9, we get

$$\begin{aligned} e^{\lambda g(C)} &= e^{\lambda C} \, \frac{\lambda}{1 - \lambda T_m} \, \frac{1 - \lambda T_c}{\lambda} = e^{\lambda C} \, \frac{1 - \lambda T_c}{1 - \lambda T_m} \\ \Longrightarrow \quad g(C) &= C + \lambda^{-1} \ln \frac{1 - \lambda T_c}{1 - \lambda T_m} \qquad (10) \end{aligned}$$

Clearly, as one would expect, $g(C_{max}) = C_{max}$. (Recall that, when $C = C_{max}$, $T_c = T_m$.)

Having determined $g(C)$, we would like to illustrate the function numerically. In Figure 10, we plot $g(C)$ for $\lambda = 10^{-6}$ and 10^{-4}. (The values of T_c and T_m necessary for evaluating $g(C)$, can be obtained by solving Equation 4.[4])

[4] It can be verified that, when C is small as compared to $1/\lambda$, $T_c \approx \sqrt{2 * C/\lambda}$. We used this approximation when plotting $g(C)$. Young [16] previously obtained this approximate expression for T_c by a somewhat different analysis.

Consider the $g(C)$ curve for $C_{max} = 25$ in Figure 10(a). The definition of $g(C)$ implies that, if a checkpointing scheme achieves overhead and latency corresponding to a point "below" the $g(C)$ curve for $C_{max} = 25$, then this scheme achieves a smaller overhead ratio than the corresponding sequential checkpointing scheme (with checkpoint overhead 25). For instance, if some scheme reduces C from 25 to 10, then it can achieve a smaller overhead ratio r than the sequential checkpointing scheme, even if it increases the latency from 25 to as large as 2000.

Comparison of Figures 10(a) and 10(b) indicates that, for the same C_{max}, as λ increases, $g(C)$ decreases. This is intuitive, because with larger λ, it is necessary to keep checkpoint latency smaller (to avoid an increase in the overhead ratio).

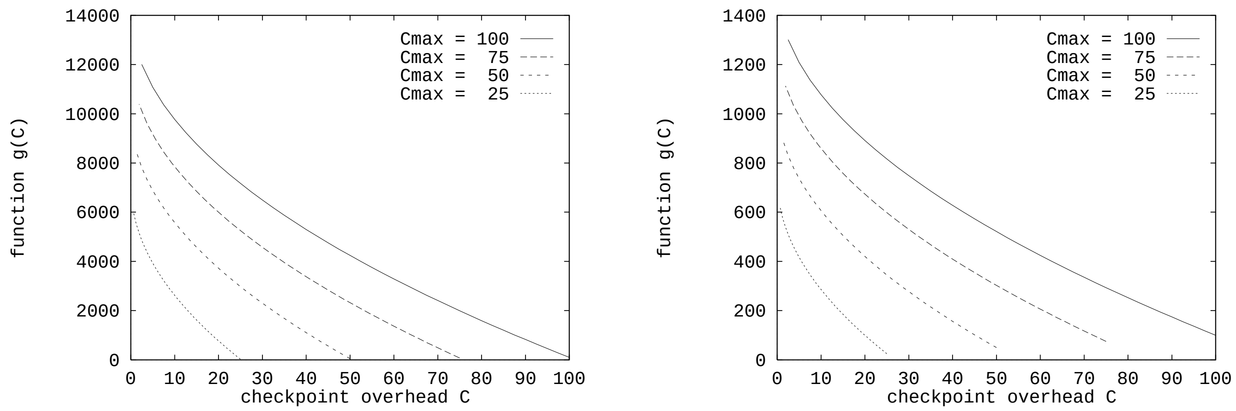
The "measured L" curve in Figure 11 plots checkpoint overhead and latency measured for a merge sort program using four different checkpointing schemes – the data is borrowed from Li et al. [9]. One of the four schemes is sequential checkpointing with overhead $C_{max} = 31$ seconds. For comparison, Figure 11 also plots $g(C)$ for three different values of λ. *(Note that the vertical axis in Figure 11 is a log scale – the vertical axes in all other graphs are linear.)* Observe that, even when λ is as large as 10^{-4} per second, the measured checkpoint latency is well below the $g(C)$ curve. This indicates that, the checkpointing techniques used in practice can achieve a significantly smaller overhead ratio as compared to the sequential checkpointing scheme.

The $g(C)$ curve derived above can be used to determine when a checkpointing scheme will perform better than the sequential checkpointing scheme.

6 Multi-process Applications

The above discussion considered a uni-process application. When an application consists of multiple processes in a distributed environment, we assume that that application periodically takes *consistent* checkpoints [2, 11, 15]. A *consistent* checkpoint consists of one checkpoint per process, and possibly a few messages logged on the stable storage. Because the system is distributed, it is possible that different processes checkpoint their state at different times – also, it is possible that different processes complete the consistent checkpointing algorithm at different times. In other words, in such a system, different processes may have different overhead and latency. Figure 12 illustrates an application that uses the Chandy-Lamport algorithm [2] for consistent checkpointing. (The "marker" messages used by the algorithm are not shown in the figure.)

As shown in the figure, let us assume that the application consists of three processes P1, P2 and P3 that are located on three processors in a distributed system. To establish a "consistent" checkpoint of the distributed application, each process records its own state on the stable storage. Each process may use any approach to record the state – here, we assume that each process uses



(a) $\lambda = 10^{-6}$ (b) $\lambda = 10^{-4}$

Figure 10: $g(C)$ for various values of C_{max}

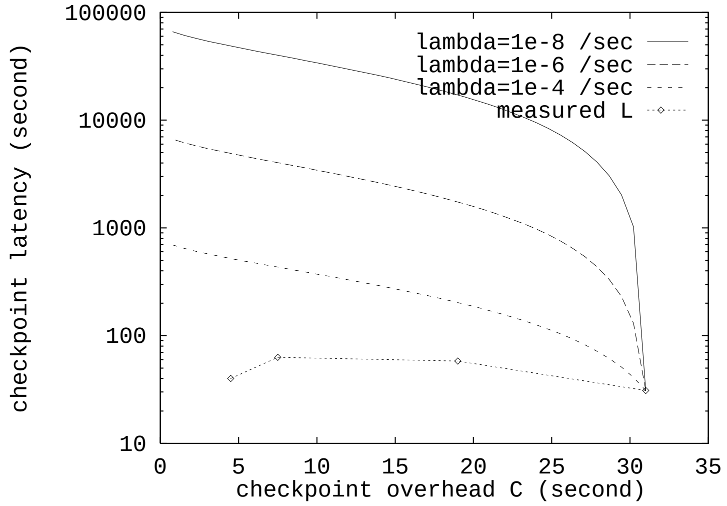


Figure 11: Comparison of measured data and $g(C)$

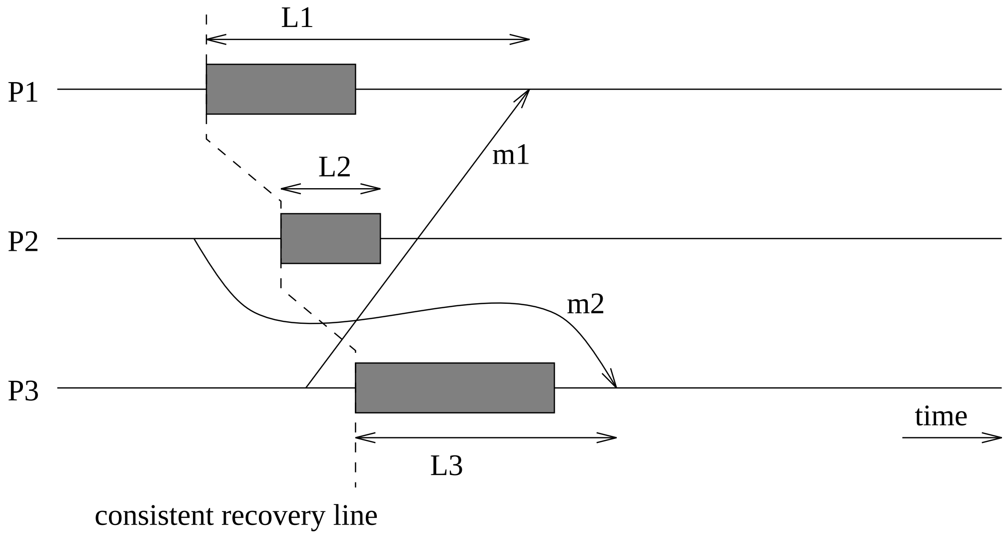


Figure 12: Different processes may have different overhead and latency

the sequential checkpointing approach. The states recorded by the processes form a "consistent recovery line". Any messages that cross the recovery line are also logged (in stable storage) by their corresponding receiver processes. In our example, processes P1 and P3 log messages m1 and m2, respectively, in the stable storage. Process P2 does not need to log any messages, because it does not receive any message that crosses the recovery line. When a process has recorded its state and logged messages (if necessary), its participation in the consistent checkpointing algorithm has completed. As shown in the figure, different processes may complete the algorithm at different times (depending on message arrival delays, and on size of each process state). Therefore, different processes observe different checkpoint latencies – latencies L1, L2 and L3 are experienced by processes P1, P2 and P3 in our example. The overheads experienced by the three processes can also be different, depending on the size of their state, and the size of messages they have to save during the consistent checkpointing algorithm.

In spite of the variations in overhead and latency encountered by different processes, we suggest that our analysis in the previous section can be applied to multi-process applications to estimate the overhead ratio. There are two possibilities:

- To obtain a pessimistic estimate of the overhead ratio (i.e. an upper bound), the values of parameters, such as latency L, used in the analysis can be obtained as the *maximum* over all processes. For instance, in our example, we will choose $L = L1$.
- To obtain a more accurate estimate of the overhead ratio, the values of parameters can be obtained as the *average* over all processes.

Just as there are different approaches for checkpointing uni-process applications, there are different schemes for consistent checkpointing of multi-process applications [2, 3, 6, 11, 15], that achieve different overhead and latency. The analysis presented in the previous section can be used to determine which approach for consistent checkpointing is superior.

7 Conclusions

This report evaluates an expression for the *overhead ratio* of a checkpointing scheme, as a function of checkpoint *latency* (L) and checkpoint *overhead* (C). Our analysis shows that, for an equi-distant checkpointing strategy, the optimal checkpoint interval is not dependent on the value of L – though it depends on the value of C. It is also observed that the *overhead ratio* is much more sensitive to the changes in C, as compared to changes in L.

In practice, a mechanism that attempts to reduce checkpoint overhead usually causes an increase in the checkpoint latency. A decrease in checkpoint overhead C, keeping L constant, results in a decrease in the *overhead ratio*. Similarly, increasing checkpoint latency L, keeping C constant, results in increase in *overhead ratio*. A decrease in C can result in an increase or a decrease in the *overhead ratio*, depending on whether the latency is increased "too much" or not. This report determines a function g of checkpoint overhead C such that checkpoint latency L should be less that $g(C)$ to achieve a *decrease* in the average overhead.

The report presents an example to illustrate that in distributed systems different processes are likely to encounter different checkpoint latencies. However, the analysis presented in the report can be used to obtain an approximate estimate of performance overhead of distributed *consistent* checkpointing schemes.

Appendix

The analysis presented below is similar to [1].

Consider the overhead ratio r as a function of T. To determine the maxima and minima of r, we must solve the equation

$$\frac{\partial r}{\partial T} = 0.$$

Now, from Equation 2,

$$\begin{aligned}
\frac{\partial r}{\partial T} &= 0 \\
\Longrightarrow \quad \frac{\partial}{\partial T}\left[\frac{\lambda^{-1}e^{\lambda(L-C+R)}(e^{\lambda(T+C)}-1)}{T} - 1\right] &= 0 \\
\Longrightarrow \quad \frac{\partial}{\partial T}\left[\frac{e^{\lambda(T+C)}-1}{T}\right] &= 0
\end{aligned}$$

$$
\begin{aligned}
\Longrightarrow \frac{e^{\lambda(T+C)}(\lambda T - 1) + 1}{T^2} &= 0 \\
\Longrightarrow e^{\lambda(T+C)}(1 - \lambda T) &= 1 \qquad \text{for } T \neq 0 \qquad (11)
\end{aligned}
$$

Thus, the positive values of T where r is minimized or maximized must satisfy the above equation. (Note: $T = 0$ is not of interest.)

Let us evaluate $\frac{\partial^2 r}{\partial T^2}$ at the extremas. From Equation 2,

$$
\begin{aligned}
\frac{\partial^2 r}{\partial T^2} &= \lambda^{-1} e^{\lambda(L-C+R)} \left[\frac{(e^{\lambda(T+C)} - 1) \times 2}{T^3} - \frac{2\lambda e^{\lambda(T+C)}}{T^2} + \frac{\lambda^2 e^{\lambda(T+C)}}{T} \right] \\
&= \lambda^{-1} e^{\lambda(L-C+R)} \frac{(1 - \lambda T)^2 e^{\lambda(T+C)} + e^{\lambda(T+C)} - 2}{T^3}
\end{aligned}
$$

We want to evaluate $\frac{\partial^2 r}{\partial T^2}$ at those (positive) T that maximize or minimize r. Such values of T must satisfy Equation 11. Multiplying both sides of Equation 11 by $(1-\lambda T)$, we have $(1-\lambda T)^2 e^{\lambda(T+C)} = (1 - \lambda T)$. Substituting this into the above equation, we get

$$
\begin{aligned}
\frac{\partial^2 r}{\partial T^2} &= \lambda^{-1} e^{\lambda(L-C+R)} \frac{(1 - \lambda T) + e^{\lambda(T+C)} - 2}{T^3} \\
&= \lambda^{-1} e^{\lambda(L-C+R)} \frac{e^{\lambda(T+C)} - 1 - \lambda T}{T^3}
\end{aligned}
$$

Now,[5] $e^{\lambda(T+C)} - 1 - \lambda T > 0$ for $T > 0$ and $C > 0$. (Note: In practice, $C > 0$.) Therefore, for positive T that satisfy Equation 11, $\frac{\partial^2 r}{\partial T^2} > 0$. Therefore, function r has only minimas (for $T > 0$). As r is not a constant function of T, if it has no maxima (at $T > 0$), it cannot have more than one minima at a positive T. Therefore, r has only one minimum (for $T > 0$) – the minimum is achieved at T obtained as the solution of Equation 11.

References

[1] K. M. Chandy, J. C. Browne, C. W. Dissly, and W. R. Uhrig, "Analytic models for rollback and recovery strategies in data base systems," *IEEE Trans. Softw. Eng.*, vol. 1, pp. 100–110, March 1975.

[2] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states in distributed systems," *ACM Trans. Comp. Syst.*, vol. 3, pp. 63–75, February 1985.

[5] $e^{\lambda(T+C)} = 1 + \lambda(T + C) + \lambda^2 (T + C)^2/2 + \cdots$.

[3] E. N. Elnozahy, D. B. Johnson, and W. Zwaenepoel, "The performance of consistent checkpointing," in *Symposium on Reliable Distributed Systems*, 1992.

[4] E. Gelenbe and D. Derochette, "Performance of rollback recovery systems under intermittent failures," *Comm. ACM*, vol. 21, pp. 493–499, June 1978.

[5] E. Gelenbe, "On the optimum checkpointing interval," *J. ACM*, vol. 2, pp. 259–270, April 1979.

[6] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," *IEEE Trans. Softw. Eng.*, vol. 13, pp. 23–31, January 1987.

[7] V. G. Kulkarni, V. F. Nicola, and K. S. Trivedi, "Effects of checkpointing and queueing on program performance," *Commun. Statist.-Stochastic Models*, vol. 4, no. 6, pp. 615–648, 1990.

[8] P. L'Ecuyer and J. Malenfant, "Computing optimal checkpointing strategies for rollback and recovery systems," *IEEE Trans. Computers*, vol. 37, pp. 491–496, April 1988.

[9] K. Li, J. F. Naughton, and J. S. Plank, "Low-latency, concurrent checkpointing for parallel programs," *IEEE Trans. Par. Distr. Syst.*, vol. 5, pp. 874–879, August 1994.

[10] J. S. Plank, M. Beck, G. Kingsley, and K. Li, "Libckpt: Transparent checkpointing under Unix," in *Usenix Winter 1995 Technical Conference, New Orleans*, January 1995.

[11] J. S. Plank, *Efficient Checkpointing on MIMD Architectures*. PhD thesis, Dept. of Computer Science, Princeton University, June 1993.

[12] K. Shin, T.-H. Lin, and Y.-H. Lee, "Optimal checkpointing of real-time tasks," *IEEE Trans. Computers*, vol. 36, pp. 1328–1341, November 1987.

[13] K. S. Trivedi, *Probability and Statistics with Reliability, Queueing and Computer Science Applications*. Prentice-Hall, 1988.

[14] N. H. Vaidya, "Another *two-level* failure recovery scheme: Performance impact of checkpoint placement and checkpoint latency," Tech. Rep. 94-068, Computer Science Department, Texas A&M University, College Station, December 1994. Available via anonymous ftp from ftp.cs.tamu.edu in directory /pub/vaidya.

[15] N. H. Vaidya, "Consistent logical checkpointing," Tech. Rep. 94-051, Computer Science Department, Texas A&M University, College Station, July 1994. Available via anonymous ftp from ftp.cs.tamu.edu in directory /pub/vaidya.

[16] J. W. Young, "A first order approximation to the optimum checkpoint interval," *Comm. ACM*, vol. 17, pp. 530–531, September 1974.

[17] A. Ziv and J. Bruck, "Analysis of checkpointing schemes for multiprocessor systems," Tech. Rep. RJ 9593, IBM Almaden Research Center, November 1993.