A Case for Multi-Level Distributed Recovery Schemes

Nitin H. Vaidya
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112
E-mail: vaidya@cs.tamu.edu

Technical Report 94-043
May 1994

Abstract

Most of the distributed recovery schemes proposed in the literature are designed to tolerate arbitrary number of failures, with a few notable exceptions of schemes designed to tolerate single failures. In this report, we demonstrate that, it is often advantageous to use "multi-level" recovery schemes. A "multi-level" recovery scheme is one that can tolerate different number of faults at different costs, tolerance of larger number of failures requiring larger costs. The costs are incurred during failure-free operation as well as during recovery.

To demonstrate the advantages of multi-level recovery, we analyze a hypothetical 2-level recovery scheme that takes two different types of checkpoints, namely, 1-checkpoints and N-checkpoints. A single failure can be tolerated by rolling the system back to a 1-checkpoint, while multiple failure recovery is possible by rolling back to an N-checkpoint. The cost of a 1-checkpoint may be expected to be smaller than that of an N-checkpoint. For such a system, we demonstrate that to minimize the average overhead, it is often necessary to take *both* 1-checkpoints and N-checkpoints.

While the conclusions of this report are intuitive, the work on appropriate recovery schemes is lacking. The objective of this report is to motivate research into recovery schemes that can provide multiple levels of fault tolerance.

1 Introduction

A number of recovery schemes have been proposed in the past for tolerating failures in distributed systems. These schemes are also applicable to parallel applications executed in distributed environments or on multiprocessors such as nCube. The applications of interest here are of two types:

1. Long-running applications which do not need to *commit* an output to the *environment* until the completion of the task. This condition does not *per se* preclude the application from doing file (or disk) I/O. When the file system is *not* a part of the *environment*, the file output can be rolled back if necessary.

2. Long-running applications which may commit output during their execution, but the delay in output commit is not required to be small.

Many of the so called "grand-challenge" problems are of the above types.

Most of the recovery schemes proposed in the literature are designed to tolerate arbitrary number of failures (e.g., [2, 3, 8, 12, 16, 17, 18, 22]), with a few notable exceptions of schemes designed to tolerate single failures (e.g., [9]).

In this report, we demonstrate that, it is often advantageous to use "multi-level" recovery schemes. A "multi-level" recovery scheme is one that can tolerate different number of faults at different costs, tolerance of larger number of failures requiring larger overhead. The overhead is incurred during failure-free operation as well as during recovery. Although a large number of researchers have analyzed checkpointing and recovery [1, 4, 5, 6, 7, 10, 11, 13, 14, 15, 19, 20, 23, 24], to our knowledge, no analysis of multi-level recovery schemes has been attempted so far. Also much of this analysis treats the entire system as faulty or fault-free, and does not model failure of each individual processor separately. Recently, Wong and Franklin [23] presented analysis of distributed checkpointing schemes that allows each processor to fail independently, however, they do not address the issue of multi-level recovery.

To demonstrate the advantages of multi-level recovery, we analyze a hypothetical 2-level recovery scheme that takes two different types of checkpoints, namely, 1-checkpoints and N-checkpoints. A single failure can be tolerated by rolling the system back to a 1-checkpoint, while multiple failure recovery is possible by rolling back to an N-checkpoint.

The cost of a 1-checkpoint may be expected to be smaller than that of an N-checkpoint.[1] We show that for such a system, to minimize the average overhead, it is often necessary to take *both* 1-checkpoints and N-checkpoints.

While conclusions of this report are intuitive, the work on appropriate recovery schemes is lacking. The objective of this report is to motivate research into recovery schemes that can provide multiple (at least two) levels of fault tolerance. An obvious approach is to use two different schemes simultaneously, for example, one to tolerate a single failure, and another to tolerate arbitrary number of failures. While this approach may in fact be useful, it would be interesting to consider designs of a single recovery scheme that can provide both the levels of fault tolerance, at a smaller aggregate cost.

This report considers a simple system model that may be applicable to some recovery schemes. The goal here is to demonstrate the need for design of multiple levels of fault tolerance, and *not* a comprehensive analysis of all recovery schemes.

2 System Model

For a multi-level recovery scheme, the system model consists of two components:

- *Cost model*: Overhead is incurred during normal operation due to checkpointing, message logging, etc. Additionally, when a failure occurs, overhead is incurred due to the computation lost by failure, the time required to initiate recovery, etc. Actual overhead depends on the chosen recovery scheme.

  The *cost model* must model the overhead incurred by the recovery scheme during failure-free operation and during recovery.

- *Failure effect model*: The effect of failures on the system is a function of the recovery scheme used.

[1] This model approximates a coordinated checkpointing scheme that stores 1-checkpoints on volatile storage and N-checkpoints on stable storage. For example, the N processors executing an application may form a *chain*, i-th processor storing its checkpoint in the volatile storage of $(i+1)$-th processor, and the N-th processor storing its checkpoint on the volatile storage of an $(N+1)$-th processor that does not execute the application. This would constitute a 1-checkpoint, as failure of any one processor can be tolerated by using the checkpoints stored in volatile storage. An N-checkpoint would be taken by all processes storing their checkpoints on the stable storage. Presumably, the cost of storing in volatile storage would be smaller than that of storing in a stable storage.

For example, traditional coordinated checkpointing schemes roll the processes back to the previous checkpoint whenever a failure occurs. The same action is taken independent of how many failures occur. On the other hand, the hypothetical 2-level recovery scheme (briefly summarized earlier) rolls the system back to the most recent checkpoint (1- or N-checkpoint) when a single failure occurs, and to the most recent N-checkpoint when multiple failures occur. The 2-level recovery scheme is described below in more detail.

The effect failures have on the behavior of the recovery scheme affects the total overhead incurred by the recovery scheme. An appropriate *failure effect model* must be chosen for a given recovery scheme.

As noted earlier, in this report, we analyze a hypothetical 2-level recovery scheme. The system model used here is a simple generalization of a model used previously [1] for analysis of single processor systems.

The system is assumed to consist of N processors. (At this point, we do not differentiate between a process and a processor. We address this issue later.) Each processor is subject to transient failures, the inter-failure interval being governed by an exponential distribution with mean $1/\lambda$. Failures of the N processors are independent of each other. (This assumption may not always be true, as discussed later.)

Cost Model

It is assumed that the recovery scheme is capable of providing tolerance against single as well as multiple failures, possibly using different approaches. The processes take two types of checkpoints: *1-checkpoints* and *N-checkpoints*. The processes take checkpoints every T time units, every k-th checkpoint being an N-checkpoint ($k \geq 1$) and all others 1-checkpoints. Thus, the interval between two consecutive 1-checkpoints is T and the interval between every two consecutive N-checkpoints is kT (excluding the time required to take 1-checkpoints). Figure 1 illustrates this for $k = 3$. It is assumed that the execution time (length) of the task (application) is an integral multiple of T. However, length of the task is *not* necessarily an integral multiple of kT. Length of the task is denoted by Υ. Thus, $\Upsilon = \mu T$ for some integer μ.

We assume that no checkpoint needs to be taken at the beginning of the task, and an N-checkpoint is taken at the completion of the task. This implies that the first N-checkpoint may be taken after less than kT time units of computation, when the length of the task

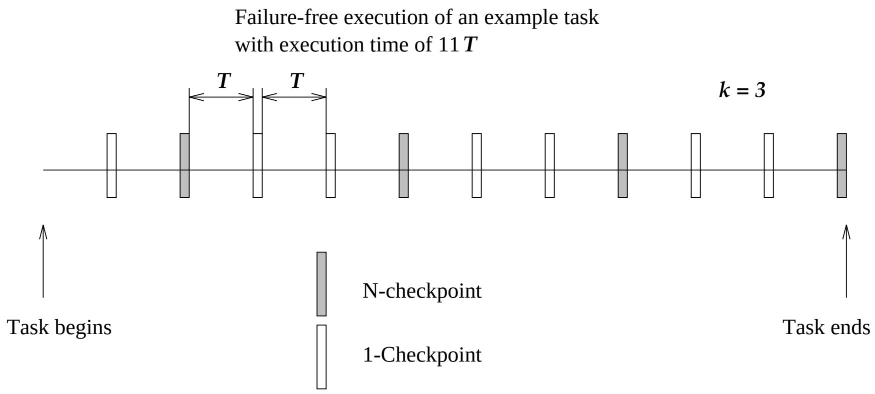

Figure 1: 1-checkpoints and N-checkpoints

is not an integral multiple of kT. For example, in Figure 1, length of the task is $11T$ and $k = 3$. Therefore, the computation time between adjacent N-checkpoints is $3T$. However, the computation time from the start of the task to the first N-checkpoint is $2T$.

The interval between any two consecutive checkpoints is called a 1-interval. The execution of the task is divided into certain number of *segments*, each segment terminating with an N-checkpoint. For example, in Figure 1, the task is divided into four segments.

Let the time required to take an N-checkpoint be C_N and the time required to take a 1-checkpoint be C_1. In a distributed system, various processors may take checkpoints at different times. However, our analysis assumes that all processors take checkpoints simultaneously. (Essentially, the cost of taking a checkpoint is bulked at the end of the interval.)

The time required to perform a rollback is assumed to be R. (This does not include the time required for re-execution). Here, the rollback time is assumed to be identical irrespective of whether the system rolls back to a 1-checkpoint or to an N-checkpoint or to the beginning of the task.

Consider a failure that can be detected by rolling back to a certain checkpoint CP. If the failure is detected when t time units of computation was performed after checkpoint CP, then it is assumed that t units of execution is required to re-do the lost computation (in absence of further failures). In the past, many researchers have assumed (e.g., [1]) that the time required to re-do the computation is αt for some constant α. Thus, we assume $\alpha = 1$ here. However, our analysis can be easily revised when $\alpha \neq 1$.

Failure Effect Model

The effect of failures is dependent on the failure recovery scheme used. We consider a failure effect model (named model B) that is *pessimistic* in the sense that the benefit of taking 1-checkpoints is likely to be more pronounced than indicated by our model. In spite of the *pessimistic* model, the need for taking 1-checkpoints as well as N-checkpoints is demonstrated by our results. (Accurate modeling needs exact knowledge of the recovery scheme used. More accurate models are expected to emphasize our conclusions further.)

Model B assumes that if at most one failure occurs during the execution of a *1-interval*, the failure can be tolerated by rolling back to the most recent checkpoint. (The most recent checkpoint may be a 1-checkpoint or an N-checkpoint.) If, however, a failure also occurs during the re-execution of the same 1-interval, system must be rolled back to the most recent N-checkpoint (or to the start of the task, if no N-checkpoint is taken before the failure). The second failure may or may not affect the same processor as the first failure – both cases require a rollback to the most recent N-checkpoint. The failure effect model is illustrated below with examples.

Figure 2(a) illustrates a scenario where a failure occurs during 1-interval I_2, and the system is rolled back to the most recent checkpoint ($CP1$). No failure occurs during the re-execution of I_2.

Figure 2(b) illustrates a scenario where a failure occurs during 1-interval I_2, and the system is rolled back to the previous checkpoint ($CP1$). Another failure occurs during the re-execution of I_2. Therefore, the system is rolled back to the most recent N-checkpoint (CP0).

Figure 2(c) illustrates a scenario similar to 2(a). In this case also a failure occurs during interval I_2 and no failure occurs during the re-execution of I_2. A failure occurring during interval I_3 is treated identical to the first failure during I_2. That is, the system rolls back to the most recent checkpoint $CP2$. Essentially, failures occurring during two different 1-intervals are treated independently.

This model is *pessimistic* in two ways:

- It requires a rollback to the most recent N-checkpoint even when the second failure affects the same processor as the first failure. It is conceivable that such a failure could be tolerated by rolling back to the most recent checkpoint (not necessarily N-checkpoint).

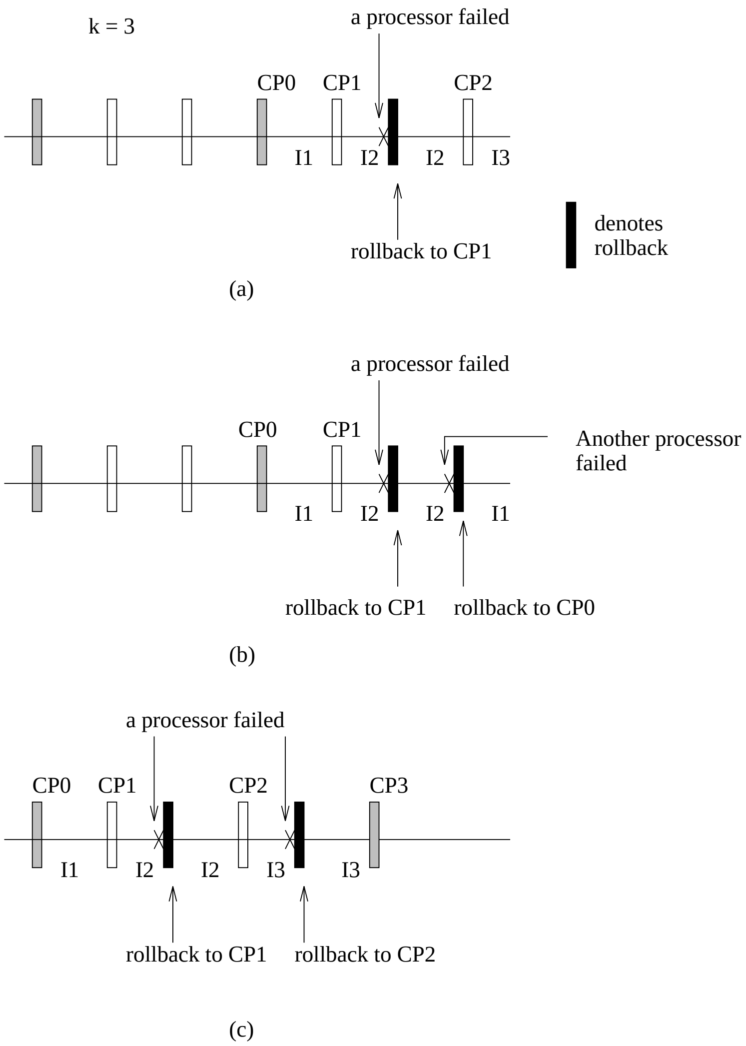

Figure 2: Illustration of fault effects

- It is conceivable that a recovery scheme may take an N-checkpoint as soon as a failure is detected. In this case, only multiple failures occurring before this N-checkpoint is taken will require a rollback to the previous N-checkpoint.

2.1 Other models

Many recovery schemes may not satisfy the model presented above. The analysis in this report can be repeated for other models, as well. As stated earlier, the primary goal here is to demonstrate the need for design of multi-level recovery schemes, and *not* a comprehensive analysis of all recovery schemes.

3 Performance Analysis

As noted in Section 1, the applications of interest here either do not need to commit any output until task completion or do not need to minimize output commit delays. Therefore, the figure of merit for such applications is the average time required to complete the task, or equivalently, the *average overhead* caused by the rollback recovery scheme. Let $E(\Gamma)$ be the expected time required to complete the task using the given recovery scheme. The average overhead is evaluated as a fraction of the execution time (Υ) required by the task. Specifically, average overhead is defined as

$$\frac{E(\Gamma)}{\Upsilon} - 1$$

This section presents an analysis of the average overhead. The results of the analysis have been verified using simulations.

3.1 Notation

Two superscripts are used in our notation, namely, $*$ and @. While the exact implications of the superscripts will be clearer as various notation is introduced, the two superscripts are intended to be used as defined below:

- A superscript $*$ denotes that the quantity is related to an interval that terminates with an N-checkpoint. Absence of the superscript $*$ generally implies (not always) that the quantity is related to an interval that terminates with a 1-checkpoint.

- A superscript @ denotes that the quantity is related to execution of a segment or an interval that is *not* initiated immediately following a failure. Absence of the superscript @ generally implies (not always) that the quantity is related to execution of a segment or an interval that is initiated immediately following a failure.

A quantity may have both, one or none of the two superscripts.

3.2 Analysis

Recall that each N-checkpoint terminates a *segment* of the task's execution. From the discussion above it is clear that multiple failures cause a rollback to the beginning of the segment during which the failures occur. Additionally, failures while executing one segment do not affect the time required to execute other segments. Therefore, the expected time required to complete the task can be obtained as the sum of expected time required to complete each segment of the task.

For a given k, the task is divided into $\lceil \frac{\mu}{k} \rceil$ segments. All segments, possibly except the first segment, includes a total of k checkpoint (of which $k-1$ are 1-checkpoints). The first segment may contain less than $(k-1)$ 1-checkpoints, as the task length Υ may not be an integral multiple of kT.

We first evaluate the expected time required to complete a single segment that includes c 1-checkpoints ($c \geq 0$) and one N-checkpoint, as shown in Figure 3. The c 1-checkpoints are labeled CP_1 through CP_c, and the N-checkpoint at the end of the segment is labeled CP_{c+1}. Observe that the segment consists of $(c+1)$ 1-intervals. Failures may occur while executing any of these intervals. If multiple failures occur while executing any one 1-interval, then the system must be rolled back to the start of the segment. The analysis below assumes that $c > 0$. The results for the case of $c = 0$ can be obtained similarly. Let:

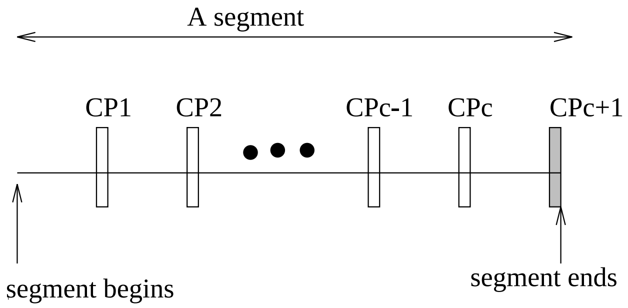

Figure 3: A segment

S_c	=	total time required to execute the above segment containing $(c+1)$ 1-intervals.
$P^@$	=	probability that a rollback will occur to the beginning of the segment, given that no previous rollback to the beginning of the segment has occurred.
P	=	probability that a rollback will occur to the beginning of the segment, given that a rollback to the beginning of the segment has already occurred.
ρ	=	number of times a rollback occurs to the beginning of the segment, *after* the first rollback to the beginning of the segment.
$F^@$	=	time lost due to a rollback to the beginning of the segment, given that this is the first rollback to the beginning of the segment.
F	=	time lost due to a rollback to the beginning of the segment, given that this is not the first rollback to the beginning of the segment.
$I^@$	=	time spent in executing a single 1-interval that terminates with a 1-checkpoint, given that at most a single failure occurs while executing the interval, and that a failure did not occur immediately before this interval started execution.
I	=	time spent in executing a single 1-interval that terminates with a 1-checkpoint, given that at most a single failure occurs while executing the interval, and that a failure occurred immediately before this interval started execution.
$I^{*@}$	=	time spent in executing a single 1-interval that terminates with an N-checkpoint, given that at most a single failure occurs while executing the interval, and that a failure did not occur immediately before this interval started execution.
$E(x)$	=	expected value of x.

For accurate analysis, it is necessary to distinguish between the first rollback to the beginning of a segment and the subsequent rollbacks. Figure 4 illustrates this. As shown in the figure, length of the first 1-interval of the segment before failure is $T + C_1$. However, after a rollback to the start of the segment occurs, due to the additional time required to

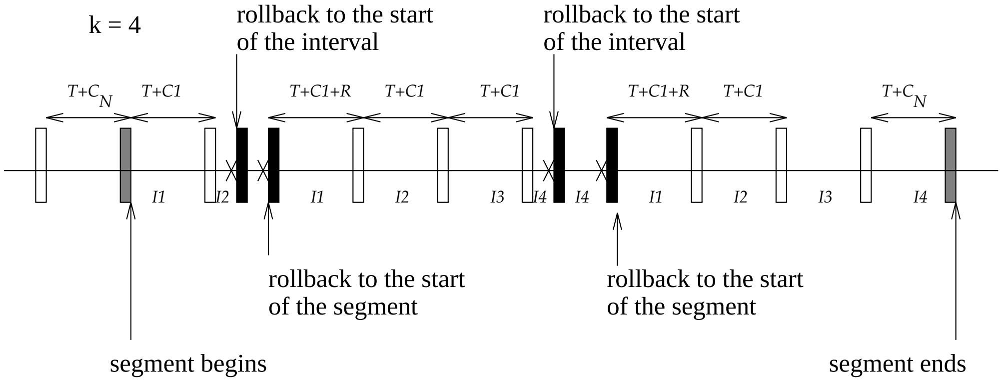

Figure 4: Rollback increases length of the first 1-interval

initiate recovery, the length of the first 1-interval in the segment is increased to $T + C_1 + R$. After each subsequent rollback, the length of the first 1-interval remains $T + C_1 + R$. The length of the 1-interval that terminates with an N-checkpoint is always $T + C_N$. (Recall that we are assuming $c > 0$). The length of all 1-intervals, except the first and last 1-intervals in the segment, is always $T + C_1$.

The execution of a segment consists of:

- Certain number of executions during which multiple failures occur that cause a rollback to the start of the segment. On the average, this requires $P^@ E(F^@) + P^@ E(\rho) E(F)$ units of time.

  Justification: $P^@$ is the probability that a rollback to the start of the segment will occur and $E(F^@)$ is the average cost of a *first* rollback to the start of the segment. Therefore, the first rollback contributes $P^@ E(F^@)$ to the average task completion time. $E(\rho)$ is the expected number of rollbacks to the start of the segment *after* the first such rollback. Therefore, the rollbacks to the start of the segment (excluding the first rollback) contribute $P^@ E(\rho) E(F)$ to the expected task completion time.

- An execution during which rollback to the start of the segment does not occur. On the average, this requires $(1 - P^@) E(I^@) + P^@ E(I) + (c - 1) E(I^@) + E(I^{*@})$ units of time.

  Justification: The expected time required to execute the first 1-interval after a rollback to the start of the segment is $E(I)$ and before such a rollback is $E(I^@)$. Therefore, the expected time required to complete the first 1-interval of the segment is $(1 - P^@) E(I^@) + P^@ E(I)$. The expected time required to complete the last 1-interval of the segment is $E(I^{*@})$ and the expected time required to complete the middle $(c-1)$ 1-intervals is $(c-1)E(I^{@})$.

Therefore,

$$\begin{aligned} E(S_c) &= P^{@}E(F^{@}) + P^{@}E(\rho)E(F) + (1-P^{@})E(I^{@}) + P^{@}E(I) + (c-1)E(I^{@}) + E(I^{*@}) \\ &= P^{@}E(F^{@}) + P^{@}E(\rho)E(F) + (c-P^{@})E(I^{@}) + P^{@}E(I) + E(I^{*@}) \qquad (1) \end{aligned}$$

We first evaluate each quantity on the right hand side of the above equation. The reader may skip sections 3.3, 3.4 and 3.5 without loss of continuity.

3.3 Evaluation of $P^{@}$, P and $E(\rho)$

Define

$p^{@}$ = probability that a rollback to the start of the segment occurs during a given 1-interval that terminates with a 1-checkpoint, given that a failure did not occur immediately before this interval started.

p = probability that a rollback to the start of the segment occurs during a given 1-interval that terminates with a 1-checkpoint, given that a failure occurred immediately before this interval started.

$p^{*@}$ = probability that a rollback to the start of the segment occurs during a given 1-interval that terminates with an N-checkpoint, given that a failure did not occur immediately before this interval started.

Then,

$$P^{@} = 1 - (1-p^{@})^{c}(1-p^{*@})$$

Evaluation of $p^{@}$ and $p^{*@}$

A rollback will occur during a 1-interval if a processor fails before completion of the interval, and a processor also fails while re-executing the interval (or while initiating the re-execution). Therefore,

$$p^{@} = (1-e^{-N\lambda(T+C_1)})(1-e^{-N\lambda(T+C_1+R)})$$

Similarly,

$$\begin{aligned} p^{*@} &= (1-e^{-N\lambda(T+C_N)})(1-e^{-N\lambda(T+C_N+R)}) \\ p &= (1-e^{-N\lambda(T+C_1+R)})(1-e^{-N\lambda(T+C_1+R)}) \end{aligned}$$

Knowing $p^{@}$ and $p^{*@}$, $P^{@}$ can be evaluated.

Evaluation of P and $E(\rho)$

When it is known that at least one rollback occurred to the beginning of the segment, the length of the first 1-interval in the segment becomes $R + T + C_1$. The length of other 1-intervals is unchanged. Therefore,

$$P = 1 - (1-p)(1-p^{@})^{c-1}(1-p^{*@}), \qquad c > 0$$

It follows that

$$E(\rho) = \frac{P}{1-P} = (1-p)^{-1}(1-p^{@})^{1-c}(1-p^{*@})^{-1} - 1, \qquad c > 0$$

3.4 Evaluation of $E(F)$

To be able to evaluate $E(F)$, we first need to evaluate $E(I^{@})$ and $E(I)$.

The definition of $I^{@}$ implies that a failure may occur while the 1-interval is executed, but no failure occurs when (and if) the 1-interval is re-executed. A rollback to the start of the 1-interval is required if a failure occurs any time during the T units of execution or while taking the 1-checkpoint at the end of the 1-interval. Thus, a failure during $T + C_1$ time units can cause a rollback to the start of the 1-interval. When a rollback occurs, R time units are spent in performing the rollback (i.e., initiating the re-execution). Therefore,

$$\begin{aligned} E(I^{@}) &= T + C_1 + \\ &\quad \frac{(1-e^{-N\lambda(T+C_1)})e^{-N\lambda(T+C_1+R)}}{e^{-N\lambda(T+C_1)} + (1-e^{-N\lambda(T+C_1)})e^{-N\lambda(T+C_1+R)}} \int_0^{T+C_1} (t+R)\frac{N\lambda\, e^{-N\lambda t}}{1-e^{-N\lambda(T+C_1)}}dt \\ &= T + C_1 + \\ &\quad \frac{(1-e^{-N\lambda(T+C_1)})e^{-N\lambda(T+C_1+R)}}{e^{-N\lambda(T+C_1)} + (1-e^{-N\lambda(T+C_1)})e^{-N\lambda(T+C_1+R)}} \left((R + (N\lambda)^{-1}) - \frac{(T+C_1)e^{-N\lambda(T+C_1)}}{1-e^{-N\lambda(T+C_1)}}\right) \end{aligned}$$

If a failure occurs immediately before the start of a 1-interval that terminates with a 1-checkpoint, then the length of that interval is $T + C_1 + R$. Therefore,

$$\begin{aligned} E(I) &= T + C_1 + R + \\ &\quad \frac{(1-e^{-N\lambda(T+C_1+R)})e^{-N\lambda(T+C_1+R)}}{e^{-N\lambda(T+C_1+R)} + (1-e^{-N\lambda(T+C_1+R)})e^{-N\lambda(T+C_1+R)}} \int_0^{T+C_1} (t)\frac{N\lambda\, e^{-N\lambda t}}{1-e^{-N\lambda(T+C_1+R)}}dt \end{aligned}$$

$$= T + C_1 + R + \frac{(1 - e^{-N\lambda(T+C_1+R)})e^{-N\lambda(T+C_1+R)}}{e^{-N\lambda(T+C_1+R)} + (1 - e^{-N\lambda(T+C_1+R)})e^{-N\lambda(T+C_1+R)}} \left((N\lambda)^{-1} - \frac{(T + C_1 + R)e^{-N\lambda(T+C_1+R)}}{1 - e^{-N\lambda(T+C_1+R)}} \right)$$

Note that the integral term above contains (t) unlike the integral term for $E(I^{@})$ which contains $(t + R)$. This is because, for $E(I)$, R is already included in the term outside the integral. $E(I^{*@})$ is obtained by replacing C_1 by C_N in the equation for $E(I^{@})$. Therefore,

$$E(I^{*@}) = T + C_N + \frac{(1 - e^{-N\lambda(T+C_N)})e^{-N\lambda(T+C_N+R)}}{e^{-N\lambda(T+C_N)} + (1 - e^{-N\lambda(T+C_N)})e^{-N\lambda(T+C_N+R)}} \left((R + (N\lambda)^{-1}) - \frac{(T + C_N)e^{-N\lambda(T+C_N)}}{1 - e^{-N\lambda(T+C_N)}} \right)$$

Evaluation of $E(F)$

Recall that F is defined as the time lost due to a rollback to the beginning of the segment, given that this is not the first rollback to the beginning of the segment. Evaluation of $E(F)$ is conditional on the fact that such a rollback indeed occurred. The rollback can occur during any one of the 1-intervals. Therefore,

$$E(F) = \sum_{i=1}^{c+1} Q_i \, E(F_i) \tag{2}$$

where, Q_i is the probability that a rollback to start of the segment occurred during interval i *given* that such a rollback occurred during the segment. F_i is the execution time lost because of such a rollback during interval i. Therefore, for $c > 0$,

$$Q_i = \begin{cases} \frac{p}{P}, & i = 1 \\ \frac{(1-p)(1-p^{@})^{i-2}p^{@}}{P}, & 1 < i \leq c \\ \frac{(1-p)(1-p^{@})^{c-1}p^{*@}}{P}, & i = c + 1 \end{cases}$$

(p, $p^{@}$, etc. were defined previously. It is easy to verify that $\sum_{i=1}^{c+1} Q_i = 1$.) Given that a rollback to the start of the interval occurred during interval i, for $1 < i \leq c$ and $c > 0$,

$$E(F_i) = E(I) + (i-2)E(I^{@}) + \int_0^{T+C_1} (t) \frac{N\lambda \, e^{-N\lambda t}}{1 - e^{-N\lambda(T+C_1)}} dt + \int_0^{T+C_1+R} (t) \frac{N\lambda \, e^{-N\lambda t}}{1 - e^{-N\lambda(T+C_1+R)}} dt$$

$$
\begin{aligned}
&= E(I) + (i-2)E(I^{@}) + (N\lambda)^{-1} - \frac{(T+C_1)e^{-N\lambda(T+C_1)}}{1-e^{-N\lambda(T+C_1)}} \\
&\quad + (N\lambda)^{-1} - \frac{(T+C_1+R)e^{-N\lambda(T+C_1+R)}}{1-e^{-N\lambda(T+C_1+R)}} \\
&= E(I) + (i-2)E(I^{@}) + 2(N\lambda)^{-1} - \frac{(T+C_1)e^{-N\lambda(T+C_1)}}{1-e^{-N\lambda(T+C_1)}} \\
&\quad - \frac{(T+C_1+R)e^{-N\lambda(T+C_1+R)}}{1-e^{-N\lambda(T+C_1+R)}} \qquad (3)
\end{aligned}
$$

Length of the first 1-interval in this case is $T + C_1 + R$. Therefore, $E(F_1)$ is obtained, similar to $E(F_i)$, as:

$$
\begin{aligned}
E(F_1) &= 2\int_0^{T+C_1+R} (t) \frac{N\lambda\, e^{-N\lambda t}}{1-e^{-N\lambda(T+C_1+R)}} dt \\
&= 2(N\lambda)^{-1} - 2\frac{(T+C_1+R)e^{-N\lambda(T+C_1+R)}}{1-e^{-N\lambda(T+C_1+R)}} \qquad (4)
\end{aligned}
$$

When $c > 0$, $E(F_{c+1})$ can be obtained by replacing C_1 by C_N and i by $c+1$ in Equation 3. $E(F)$ can now be evaluated using Equation 2 and the expressions for $E(F_i)$ and Q_i.

3.5 Evaluation of $E(F^{@})$

Recall that $F^{@}$ is defined as the time lost due to a rollback to the beginning of the segment, given that this is the first rollback to the beginning of the segment. Evaluation of $E(F^{@})$ is very similar to the evaluation of $E(F)$.

$$
E(F^{@}) = \sum_{i=1}^{c+1} Q_i^{@}\, E(F_i^{@}) \qquad (5)
$$

where, $Q_i^{@}$ is the probability that a rollback to start of the segment occurred during interval i *given* that such a rollback occurred during the segment and that the rollback is the first rollback in this segment. $F_i^{@}$ is the execution time lost because of such a rollback during interval i. Therefore, for $c > 0$,

$$
Q_i^{@} = \begin{cases} \frac{(1-p^{@})^{i-1}p^{@}}{P^{@}}, & 1 \le i \le c \\ \frac{(1-p^{@})^{c}p^{*@}}{P^{@}}, & i = c+1 \end{cases}
$$

($p^{*@}$ and $p^{@}$ were obtained previously. It is easy to verify that $\sum_{i=1}^{c+1} Q_i^{@} = 1$.) Given that a rollback to the start of the interval occurred during interval i, for $1 \leq i \leq c$ and $c > 0$,

$$\begin{aligned}
E(F_i^{@}) &= (i-1)E(I^{@}) + \int_0^{T+C_1} (t)\, \frac{N\lambda\, e^{-N\lambda t}}{1 - e^{-N\lambda(T+C_1)}} dt \\
&+ \int_0^{T+C_1+R} (t)\, \frac{N\lambda\, e^{-N\lambda t}}{1 - e^{-N\lambda(T+C_1+R)}} dt \\
&= (i-1)E(I^{@}) + 2(N\lambda)^{-1} - \frac{(T+C_1)e^{-N\lambda(T+C_1)}}{1 - e^{-N\lambda(T+C_1)}} \\
&- \frac{(T+C_1+R)e^{-N\lambda(T+C_1+R)}}{1 - e^{-N\lambda(T+C_1+R)}} \qquad (6)
\end{aligned}$$

$E(F_{c+1}^{@})$ can be obtained by replacing C_1 by C_N and i by $c+1$ in Equation 6. $E(F^{@})$ can now be evaluated using Equation 5 and the expressions for $E(F_i^{@})$ and $Q_i^{@}$.

3.6 Evaluation of expected task completion time

Using the expressions derived above, the value of $E(S_c)$ $(c > 0)$ can be obtained using Equation 1 repeated here:

$$E(S_c) = P^{@} E(F^{@}) + P^{@} E(\rho) E(F) + (c - P^{@}) E(I^{@}) + P^{@} E(I) + E(I^{*@})$$

$E(S_c)$ for $c = 0$ can also be obtained similarly. Recall that length of the task (Υ) is a multiple of T. Let $\Upsilon = \mu T$. Then, the task consists of $\lceil \mu/k \rceil$ segments, of which $\lceil \mu/k \rceil - 1$ segments contain k 1-intervals each and one segment contains $k^{\#} = \mu - k(\lceil \mu/k \rceil - 1)$ 1-intervals. Therefore, the expected task completion time $E(\Gamma)$ is obtained as

$$E(\Gamma) = (\lceil \mu/k \rceil - 1)\, E(S_{k-1}) + E(S_{k^{\#}-1}) \qquad (7)$$

As we know how to evaluate $E(S_c)$ for arbitrary c, the expected task completion time can now be evaluated.

3.7 Average Overhead

Two cases are possible:

1. The task is of finite size. In this case, the average overhead can be obtained as,

$$\begin{aligned} \text{average overhead} &= \frac{E(\Gamma)}{\Upsilon} - 1 \\ &= \frac{(\lceil \mu/k \rceil - 1)\, E(S_{k-1}) + E(S_{k^{\#}-1})}{\mu T} - 1 \end{aligned} \tag{8}$$

2. The task is of infinite size. In this case, the average overhead is obtained by taking a limit of Equation 8 as μ approaches ∞. To obtain the limit, observe that Equation 8 can be re-written as

$$\text{average overhead} = \frac{(\lceil \mu/k \rceil - 1)\, E(S_{k-1}) + E(S_{k^{\#}-1})}{((\lceil \mu/k \rceil - 1)k + k^{\#})T} - 1$$

   Thus, when $\mu \to \infty$, we obtain the average overhead as

$$\frac{E(S_{k-1})}{kT} - 1.$$

   Essentially, for large tasks, the average overhead is the same as the average overhead in executing a single segment.

Average *percentage* overhead is obtained by multiplying the average overhead by 100.

4 Numerical Results

In this section, we present numerical results to determine optimal values of k and μ for a given (finite) task size and a given λ. Significant effort has been devoted in the past for analytically determining optimal checkpoint intervals for checkpointing and rollback recovery schemes [1, 5, 15, 23, 24]. Due to the complexity of the expressions for the 2-level recovery scheme under consideration, an analytical approach for determining optimal k and μ is not very attractive. Instead, we choose to determine the optimal values numerically.

There are a number of parameters that affect system performance including C_1, C_N, λ, N and task length Υ. In this report, we are primarily interested in the effect of relative values of C_1 and C_N on the optimal operating point. (For a given task, an operating point is characterized by the chosen values of k and μ.)

We evaluate the average overhead for two hypothetical tasks for different values of C_1. For the first task $N\lambda\Upsilon$ is large and for the other task $N\lambda\Upsilon$ is small. Therefore, the probability that many failures occur is larger for the first task.

Task 1

Task 1 is characterized by the following parameters: $\lambda = 0.00001$, $\Upsilon = 200$, $N = 500$, $C_N = 1.0$, $R = 1.0$. Different values of C_1 are used in the following for different graphs. Note that for Task 1, $N\lambda\Upsilon = 1.0$ which is quite large.

The first interesting feature of the two-level recovery scheme is that the curves for average overhead do not always have a unique minimum. This is illustrated in Figures 5 and 6 which plot the average percentage overhead versus μ for $C_1 = 0.2$ and $k = 3$ and 10, respectively. Observe that the curve for $k = 3$ has many minimas while the curve for $k = 10$ has only two minimas. These curves are not convex, unlike the traditional checkpointing and rollback schemes (e.g., [1]).

The curve for $k = 1$ is shown in Figure 7. When $k = 1$, all the checkpoints are N-checkpoints, and the two-level recovery scheme reduces to traditional checkpointing and rollback schemes. Therefore, as shown previously [1], the curve for $k = 1$ is convex and has exactly one minimum.

Figures 8 through 11 plot the percentage overhead versus μ for various values of k and C_1. Table 1 lists the optimal values of k, μ and average percentage overhead for various values of C_1.

Note that when $C_1 = 1.0$, we have $C_1 = C_N$, i.e., taking 1-checkpoints is as expensive as N-checkpoints. As N-checkpoints provide more protection against failures, it is obvious that, to minimize the average overhead, all the checkpoints must be N-checkpoints (i.e., $k = 1$).

C_1	k	μ	average % overhead
0.2	14	27	7.1
0.4	6	18	9.0
0.6	3	14	10.3
1.0	1	10	11.2

Table 1: Task 1: Minimum average percentage overhead

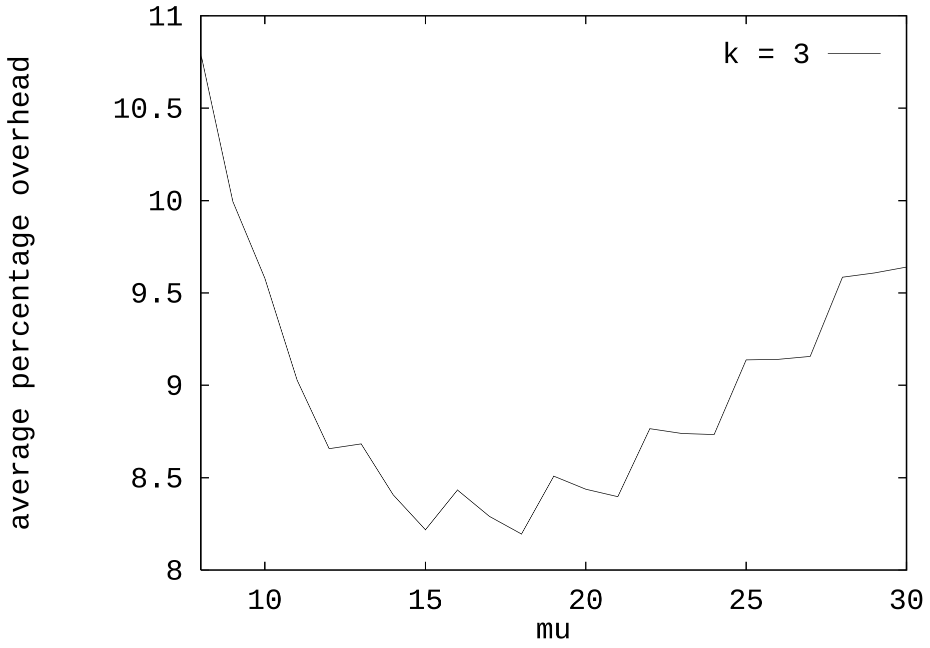

Figure 5: Task 1: $C_1 = 0.2$ and $k = 3$ – the curve is not convex

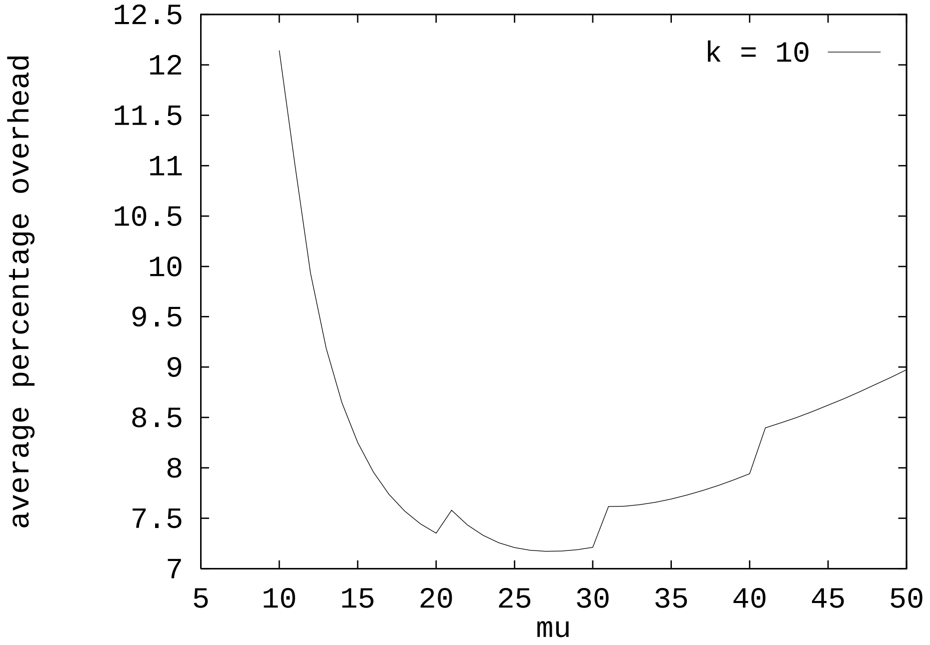

Figure 6: Task 1: $C_1 = 0.2$ and $k = 10$ – the curve is not convex

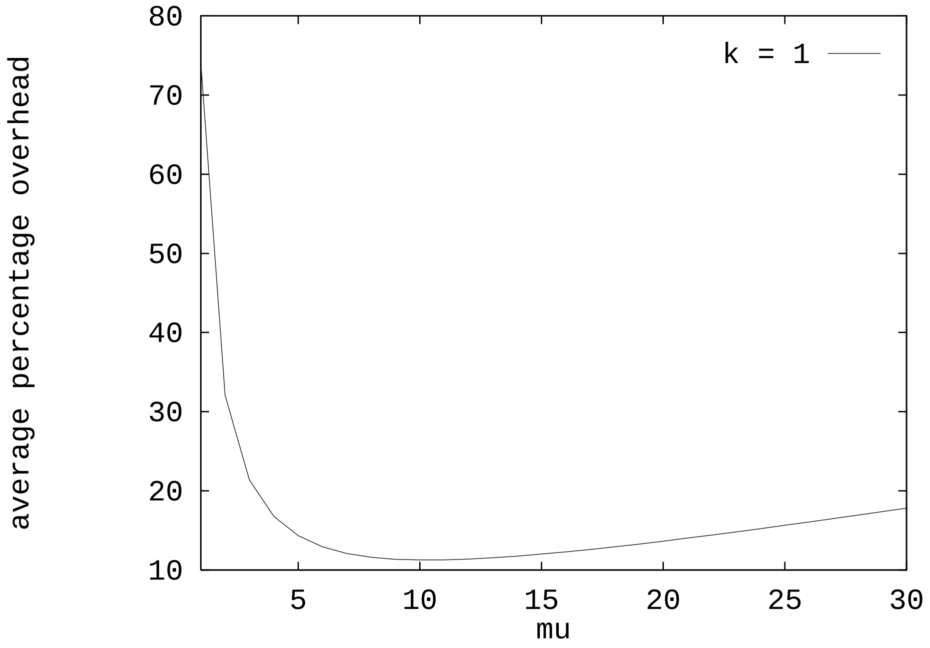

Figure 7: Task 1: $C_1 = 0.2$ and $k = 1$ – the curve is convex

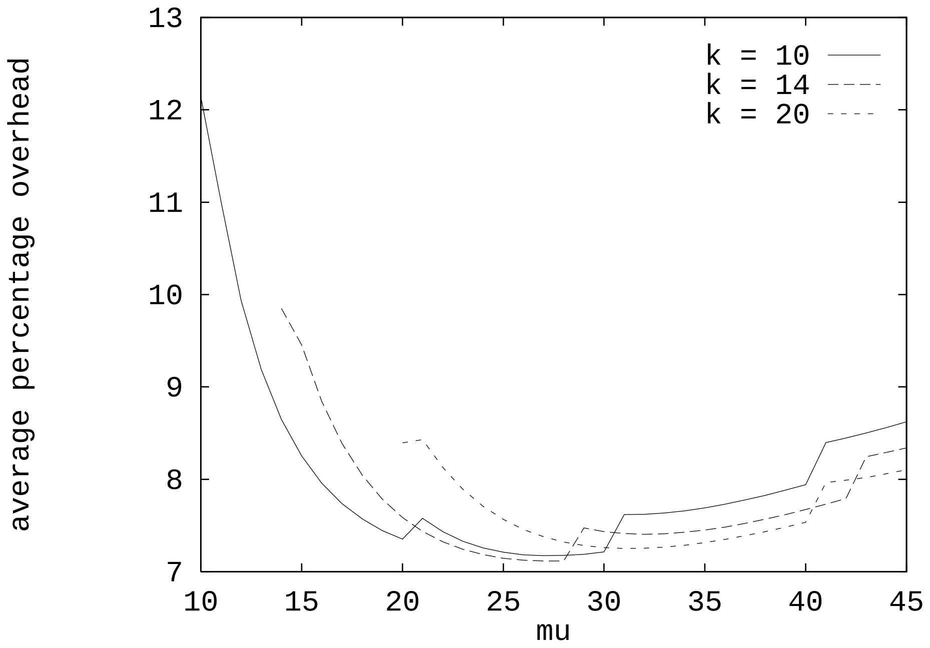

Figure 8: Task 1 – $C_1 = 0.2$

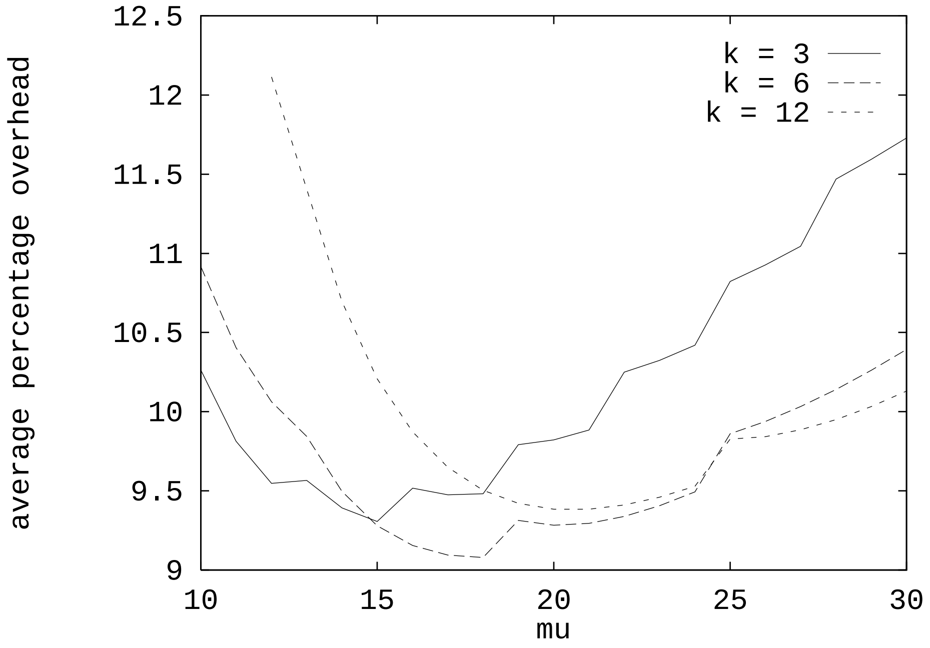

Figure 9: Task 1 – $C_1 = 0.4$

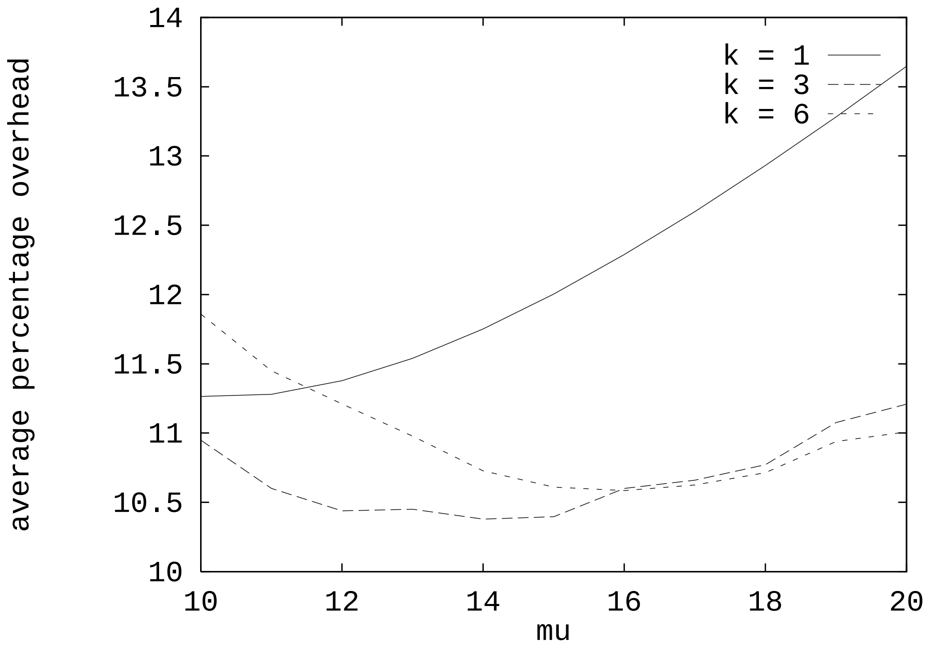

Figure 10: Task 1 – $C_1 = 0.6$

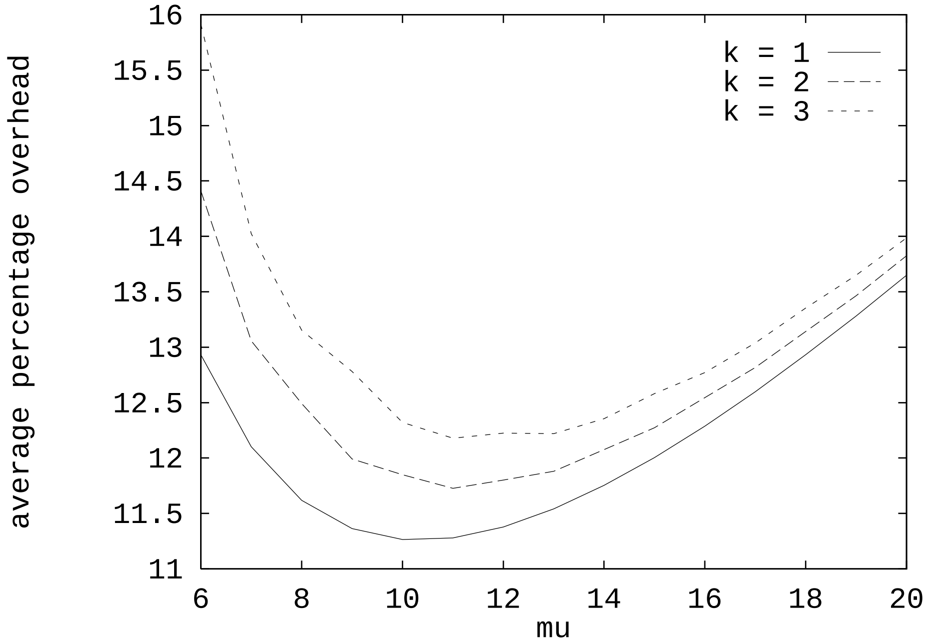

Figure 11: Task 1 – $C_1 = 1.0$

Observe that the values of μ and k at which the overhead is minimized are significantly affected by the changes in C_1. Additionally, the changes in C_1 also affect the minimum achievable overhead significantly. An overhead reduction of as small as 2% in a 500 processor system results in significant cost savings. Therefore, it is desirable to minimize C_1 as much as possible.

Task 2

Task 2 is characterized by the following parameters: $\lambda = 0.00001$, $\Upsilon = 200$, $N = 50$, $C_N = 1.0$, $R = 1.0$. Note that for Task 2, $N\lambda\Upsilon = 0.1$ which is an order of magnitude smaller that Task 1. Task 1 and 2 essentially differ in the number of processors they use.

For Task 2, Figures 12 through 15 plot the percentage overhead versus μ for various values of k and C_1. Table 2 lists the optimal values of k, μ and average percentage overhead for various values of C_1.

Observe that for Task 2, to optimize the overhead for values of C_1 not very close to C_N, μ needs to be equal to k, i.e. no N-checkpoints are taken during the task. This implies that when $N\lambda\Upsilon$ is small and C_1 is small compared to C_N, it is adequate to only take 1-checkpoints; N-checkpoints are not necessary at all to minimize the overhead. The actual

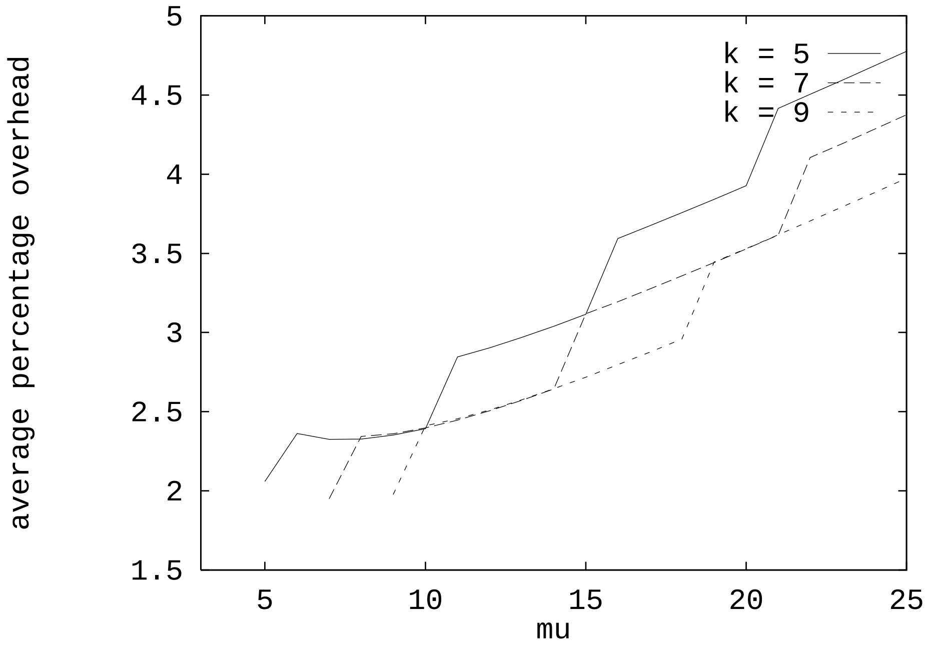

Figure 12: Task 2 – $C_1 = 0.2$

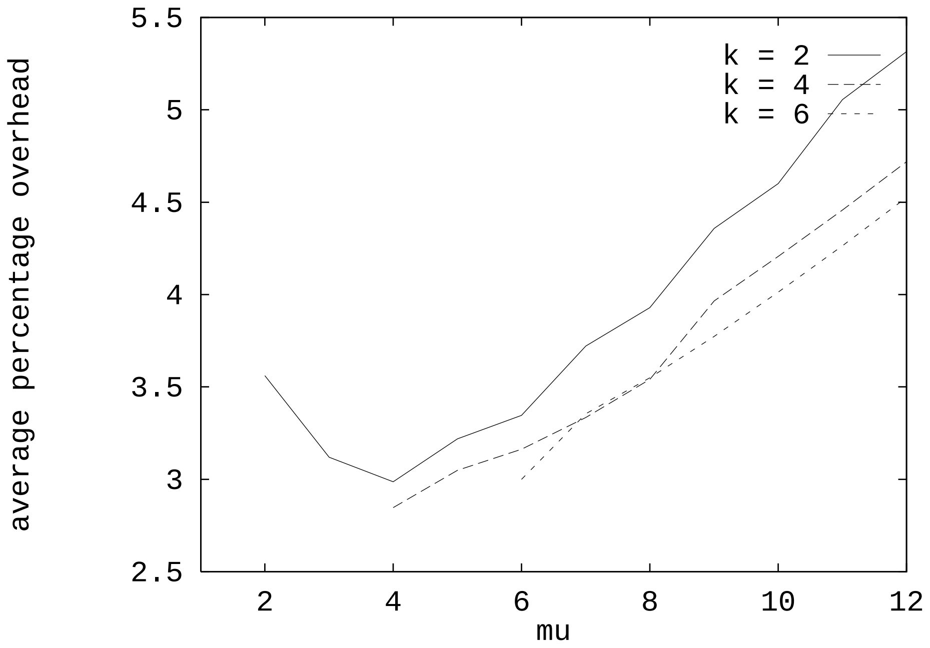

Figure 13: Task 2 – $C_1 = 0.6$

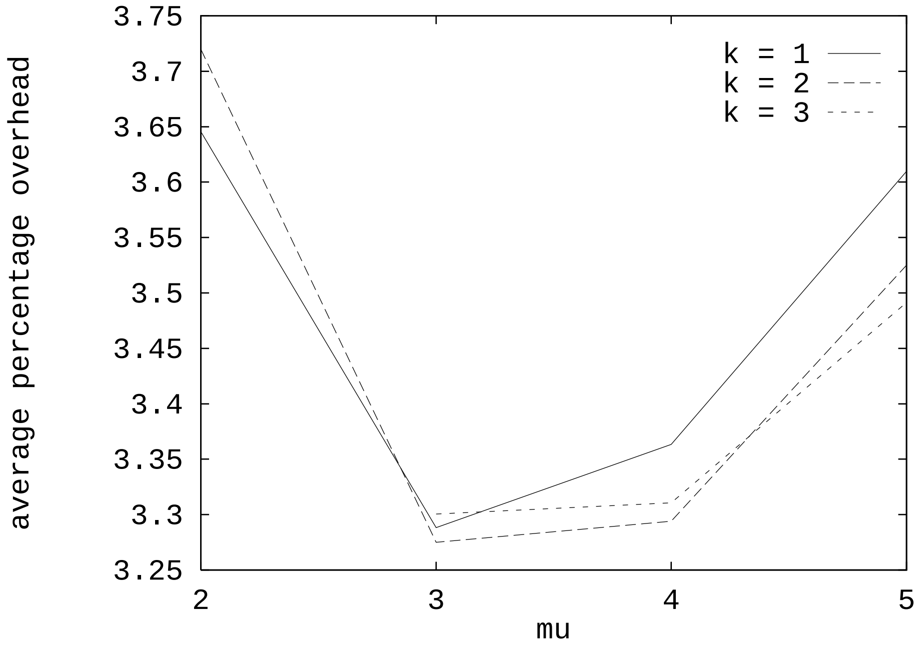

Figure 14: Task 2 – $C_1 = 0.9$

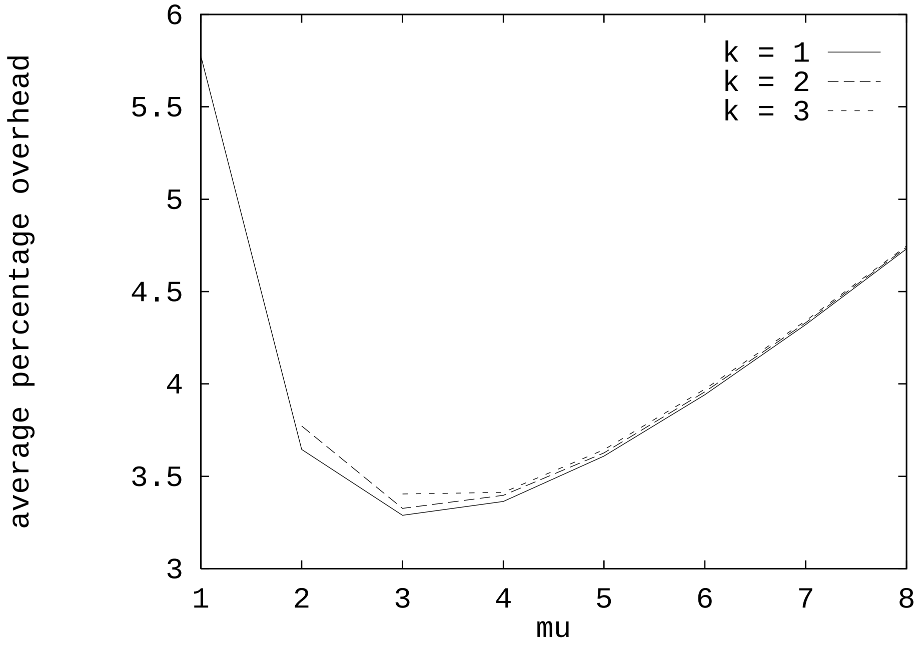

Figure 15: Task 2 – $C_1 = 1.0$

C_1	k	μ	average % overhead
0.2	7	7	1.94
0.6	4	4	2.84
0.9	2	3	3.27
1.0	1	3	3.28

Table 2: Task 2: Minimum average percentage overhead

value of μ at which the overhead is minimized decreases with an increase in C_1.

5 Accurate Modeling

The model used in this report at best approximates real systems. However, the conclusions drawn from the analysis presented above are expected to hold for most systems. An accurate system model must take into account the following:

- Recovery scheme: The actual recovery scheme affects both the *cost model* and the *failure effect model* described in Section 2.
- Application: The behavior of the application affects the overhead. For example, the variation in size of the checkpoint, number and size of messages sent by the processes can affect the *cost model*. For example, models similar to those presented in [4, 23] may be used to more accurately model an application's behavior.
- Failure model: In the analysis above, we assumed that the failures of different processors are independent. Additionally, we did not differentiate between a *processor* and a *process*.

  In practice, multiple processes may be scheduled on a single processor. This implies that a *single processor* failure could result in the failure of *multiple processes*.

  The *failure independence* assumption may not hold in practice. Often, multiple processors are packed onto a single printed circuit board. This may mean that the failure of one processor can cause the failure of other processors on the same board.

  The implication of the above observations is that it may often be necessary to design recovery schemes that logically partition the system into multiple *clusters*, each cluster containing some number of processes. The partitioning should be such that the failure of one process in the cluster is likely to cause (or be correlated with) failure of other processes in the cluster. In this context, the 2-level recovery scheme presented earlier can be redesigned to tolerate a *single cluster* failure at a low cost and *multiple cluster* failures at a higher cost. We previously proposed the *distributed recovery unit* abstraction [21], which can potentially be used to design such recovery schemes.

6 Discussion

The numerical results presented in Section 4 imply that sometimes neither 1-checkpoints nor N-checkpoints can be used exclusively to achieve optimal performance. For example, for Task 1 with $C_1 = 0.4$, the overhead is minimized when $k = 6$ and $\mu = 18$. A system that only takes N-checkpoints is equivalent to having $k = 1$. As the overhead is minimized when $k = 6$, it follows that taking only N-checkpoints is not optimal. Figure 16 shows the curve for $k = 1, 6$ and also for a system that takes only 1-checkpoints, i.e., all the μ checkpoints are 1-checkpoints. Such a system rolls back to the start of the task if multiple failures occur during any single 1-interval. This system does not achieve a smaller average overhead compared to $k = 6$ and $\mu = 18$. This implies that taking only 1-checkpoints is also not adequate. Essentially, this example illustrates that often it will be necessary to take both 1-checkpoints and N-checkpoints to minimize the average overhead. Also, our results indicate that, keeping everything else constant, the overhead is reduced significantly by reducing C_1.

The above observation is a motivating factor to design 2-level recovery schemes. Specifically, it seems necessary to investigate software techniques as well as hardware support to minimize the cost of taking 1-checkpoints (as compared to N-checkpoints).

7 Future Work

The following problems are a subject of ongoing and future research.

- Design of multi-level recovery schemes for message passing as well as shared memory systems.
- Accurate analysis of multi-level recovery schemes.

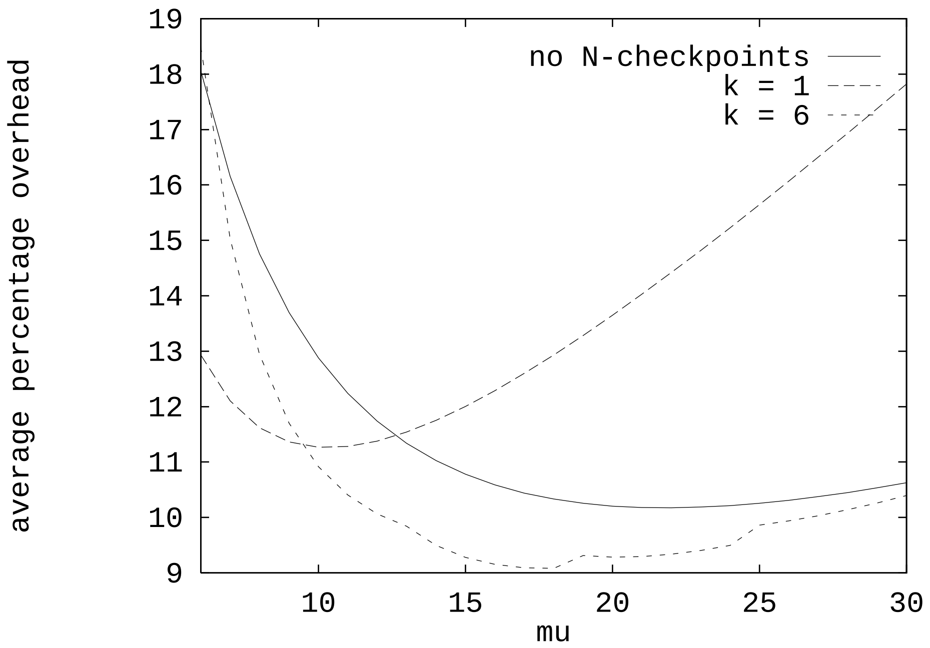

Figure 16: Task 1 – $C_1 = 0.4$

- Experimental evaluation of multi-level recovery schemes.
- Design of hardware support to aid the implementation of efficient multi-level recovery schemes.
- Correlated (or dependent) failures should be taken into account, where applicable, when performing the analysis.

As a starting point, we are looking at two-level recovery schemes.

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] E. N. Elnozahy and W. Zwaenepoel, "Manetho: Transparent rollback-recovery with low overhead, limited rollback, and fast output commit," *IEEE Trans. Computers*, vol. 41, May 1992.

[3] R. E. Gantenbein, "A bibliography of dependable distributed computing," *Operating Systems Review*, vol. 26, pp. 60–81, April 1992.

[4] S. Garg and K. F. Wong, "Analysis of an improved distributed checkpointing algorithm," Tech. Rep. WUCS-93-37, Dept. of Comp. Sc., Washington University, June 1993.

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

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

[7] V. Grassi, L. Donatiello, and S. Tucci, "On the optimal checkpoiting of critical tasks and transaction-oriented systems," *IEEE Trans. Softw. Eng.*, vol. 18, pp. 72–77, January 1992.

[8] D. B. Johnson, "Efficient transparent optimistic rollback recovery for distributed application programs," in *Symposium on Reliable Distributed Systems*, pp. 86–95, October 1993.

[9] D. B. Johnson and W. Zwaenepoel, "Sender-based message logging," in *Digest of papers: The* 17^{th} *Int. Symp. Fault-Tolerant Comp.*, pp. 14–19, June 1987.

[10] 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.

[11] 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.

[12] A. Lowry, J. R. Russell, and A. P. Goldberg, "Optimistic failure recovery for very large networks," in *Symposium on Reliable Distributed Systems*, pp. 66–75, 1991.

[13] V. F. Nicola and J. M. van Spanje, "Comparative analysis of different models of checkpointing and recovery," *IEEE Trans. Softw. Eng.*, vol. 16, pp. 807–821, August 1990.

[14] A. Reuter, "Performance analysis of recovery techniques," *ACM Trans. Database Systems*, vol. 9, pp. 526–559, December 1984.

[15] 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.

[16] A. P. Sistla and J. L. Welch, "Efficient distributed recovery using message logging," in *Proc. ACM Symp. on Principles of Distributed Computing*, pp. 223–238, August 1989.

[17] R. E. Strom, D. F. Bacon, and S. A. Yemini, "Volatile logging in n-fault-tolerant distributed systems," in *Digest of papers: The* 18^{th} *Int. Symp. Fault-Tolerant Comp.*, pp. 44–49, 1988.

[18] R. E. Strom and S. A. Yemini, "Optimistic recovery in distributed systems," *ACM Trans. Comp. Syst.*, vol. 3, pp. 204–226, August 1985.

[19] A. N. Tantawi, "Performance analysis of checkpointing strategies," *ACM Trans. Comp. Syst.*, vol. 2, pp. 123–144, May 1984.

[20] S. J. Upadhyaya and K. K. Saluja, "An experimental study to determine task size for rollback recovery schemes," *IEEE Trans. Computers*, vol. 37, pp. 872–877, July 1988.

[21] N. H. Vaidya, "Distributed recovery units: An approach for hybrid and adaptive distributed recovery," Tech. Rep. 93-052, Computer Science Department, Texas A&M University, College Station, November 1993.

[22] Y. Wang and W. K. Fuchs, "Lazy checkpoint coordination for bounding rollback propagation," in *Symposium on Reliable Distributed Systems*, pp. 78–85, October 1993.

[23] K. Wong and M. Franklin, "Distributed computing systems and checkpointing," in *Proc. 2nd Int. Symp. High Perf. Distr. Comp., Spokane, Washington*, pp. 224–233, July 1993.

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