Analysis of Checkpointing Schemes for Multiprocessor Systems*

Avi Ziv

Information Systems Laboratory
Stanford University
Stanford, CA 94305-4055
E-mail: avi@isl.stanford.edu

Jehoshua Bruck

California Institute of Technology
Mail Code 116-81
Pasadena, CA 91125
E-mail: bruck@systems.caltech.edu

Abstract

Parallel computing systems provide hardware redundancy that helps to achieve low cost fault-tolerance, by duplicating the task into more than a single processor, and comparing the states of the processors at checkpoints. This paper suggests a novel technique, based on a Markov Reward Model (MRM), for analyzing the performance of checkpointing schemes with task duplication. We show how this technique can be used to derive the average execution time of a task and other important parameters related to the performance of checkpointing schemes. Our analytical results match well the values we obtained using a simulation program. We compare the average task execution time and total work of four checkpointing schemes, and show that generally increasing the number of processors reduces the average execution time, but increases the total work done by the processors. However, in cases where there is a big difference between the time it takes to perform different operations, those results can change.

1 Introduction

Parallel computing systems provide hardware redundancy that helps to achieve low cost fault-tolerance, by duplicating the task into more than a single processor, and comparing the states of the processors at checkpoints [11]. The usage of checkpoints reduces the time spent in retrying a task in the presence of failures, and hence reduces the average execution time of a task [4][16]. Reducing the task execution time is very important in many applications like real-time systems with hard deadlines, and transactions systems, where high availability is required. Examples of systems that use checkpointing for fault recovery are Sequoia [2], Eternity from Tolerant Transactions Systems [13] and NonStop from Tandem Computers [5].

In checkpointing schemes the task is divided into n intervals. At the end of each interval a checkpoint is added, either by the programmer [4] or by the compiler [10]. In the systems considered here, the checkpoints serve two purposes: detecting faults that occurred during the execution of a task, and reducing the time spent in recovering from faults. Fault detection is achieved by duplicating the task into two or more processors, and comparing the states of the processors at the checkpoints. We assume that the probability of two faults resulting in identical states is very small, so that two matching states indicate a correct execution. By saving the state of the task at each checkpoint, we avoid the need to restart the task after each fault. Instead, the task can be rolled back to the last correct checkpoint and execution resumed from there, thereby shortening fault recovery.

The execution of a task is done in steps, Each consisting of a series of operations. The first operation is to execute one interval of the task by all the processors that are assigned to it. Note that it is not necessary that all the processors execute the same interval at the same step. After the execution of each interval is completed, the system performs the operations necessary to achieve fault detection and recovery. The first operation is to store the states of the processors in the stable storage and to compare those states. Based on the result of the comparison and the scheme used, the system decides what further action should take place. If no fault occurred, then the execution of the task is resumed with the next interval in the next step. Otherwise the Checkpoint processor performs operations to recover from the fault. Table 1 presents a list of possible operations that are used in this paper, and the time it takes to perform each of them.

*This research was partially supported by the IBM Almaden Research Center, San Jose, California, and partially supported by NSF Young Investigator Award CCR-9457811.

Operation	Time
Execute one task interval	t_I
Store and compare checkpoint states	t_{ck}
Roll back to last verified checkpoint	t_r
Copy state from one processor to another	t_{cp}
Load spare processor with task and start it	t_{ld}

Table 1: List of possible operations

Agrawal [1] describes a fault tolerance scheme, called RAFT (Recursive Algorithm for Fault Tolerance), which achieves fault tolerance by duplicating the computation of a task on two processors. If the results of the two executions do not match, the task is executed again in another processor until a pair of processors produces identical results. The RAFT scheme does not use checkpoints, and every time a fault is detected the task has to be started from its beginning. More recent schemes use checkpointing to avoid re-execution of an entire task [11]. At each checkpoint, the state of the task is stored into a stable memory. If a fault is detected and a rollback is needed, it can be done to the last stored checkpoint, not to the beginning of the task. Different recovery techniques are used by the schemes to shorten the fault recovery time. Examples of such techniques are rollback with look-back [9] and roll-forward recovery [9][12].

Performance analysis is very important when trying to evaluate and compare different schemes, or check if a scheme achieves its goals in a certain system. Most authors rely on simulations for performance evaluation [12], or use a simplified fault model [9][12]. The use of simulation leads to long and time consuming evaluation, and does not allow examination of many cases. The simplified fault model provides only approximate results.

In this paper we describe an analysis technique for studying the performance of checkpointing schemes for fault-tolerance. The technique provides means to evaluate important parameters in the performance of a scheme. It provides a way to compare various schemes and select optimal values for some parameters of the scheme, like the number of checkpoints [18].

The analysis of a scheme is based on the analysis of a discrete time *Markov Reward Model* (MRM) [7]. The analysis is done in three steps. In the first step, the analyzed scheme is modeled as a state-machine. The transition edges of the state-machine are assigned with values that correspond to the properties that need to be evaluated, like the useful work done by the transition, the time to execute the transition etc. In the second step, the edges of the state-machine are assigned transition probabilities according to the events that cause the transition and the fault model used. In the last step, the Markov chain, created by the first two steps, is analyzed, and values for the properties of interest are derived.

The proposed analysis technique is used to compare four checkpointing schemes: Triple Modular Redundant with checkpointing (TMR-F) [9], Double Modular Redundant with backward recovery and two recovery processors (DMR-B-2) [9], Double Modular Redundant with forward recovery and one recovery processor (DMR-F-1) [9], and Roll-Forward Checkpointing Scheme (RFCS) [12]. We evaluate two quantities, the average execution time of a task and the total work done to complete a task. The execution time of a task is defined as the total elapsed time from the beginning of the execution of the task, until the last checkpoint is compared correctly. This parameter is important in real-time systems, where fast response is desired. We show that the number of processors used to implement the scheme has a major effect on the average execution time, while the complexity of the scheme has only a minor effect. Out of the four schemes examined in this paper, the TMR-F scheme, which uses three processors and a simple recovery technique, is the quickest. The DMR-F-1 and RFCS schemes, which use two processors during normal execution and add spare processors during fault recovery, are slower than TMR-F but quicker than the DMR-B-2 scheme, that always uses two processors.

The total work to complete a task depends on not only the time to complete the task but also the number of processors used. It is defined as the sum of the time each of the processors is used by the scheme. This parameter is important in transactions systems, where high availability is important. In these types of systems, reducing the total work to complete a task means increasing the total throughput of the system. We show that schemes with low execution time are not work efficient, and that the lowest work is done using schemes that use a small number of processors, and have higher execution time. The total work results of the four schemes examined here were the reverse of the execution time results. The DMR-B-2 scheme has the lowest total work, while the TMR-F scheme has the highest total work.

There are some cases where a big difference in the time it takes to perform various operations can cause the schemes to behave differently than described above. Those cases can still be analyzed with the technique described in this paper. For example, when workstations connected by a LAN are used to implement the schemes, operations that involve more than one workstation, and need the LAN, take longer time to execute than operations that can be done locally. In this case the DMR-B-2 scheme that uses the network only lightly is quicker than the TMR-F scheme.

The rest of the paper is organized as follows. Section 2 describes the analysis technique, using Double Modular Redundant scheme with backward recovery and a single recovery processor (DMR-B-1) [9] as an example. In Section 3 we compare the average execution time and the total work of four checkpointing schemes. Section 4 concludes the paper.

2 Analysis Technique

The analysis of the schemes is based on the analysis of a discrete time *Markov Reward Model* (MRM) [7]. In the Markov Reward Model used in this paper, each transition edge of the Markov chain has a reward level associated with it. The properties of the reward of the Markov chain are used to evaluate the measures of interest. Markov Reward Models are often used in evaluating the performance of computing systems. Smith and Trivedi [14] give examples of the use of MRM in evaluating reliability and performance of parallel computer, task completion time in faulty systems and properties of queueing systems. Others like [3], [6], [15], use MRM to evaluate various aspects of computer system performance.

The analysis of the schemes is done in three steps: building the *extended* state-machine of the scheme, assigning probabilities to the transitions of the machine according to the fault models, and solving the Markov chain created by the the first two steps to get the desired analysis. Next we describe the three steps in more detail, using as an example the DMR-B-1 scheme [9].

In the DMR-B-1 scheme the task is executed by two processors in parallel. At the end of each interval the states of both processors (or signatures of them) are compared. If they match then a correct execution is assumed, and the execution of the next interval starts. In case the states do not match, a new processor executes the interval, and its state is compared to all the states of the previous executions of the interval until two identical states are found.

Figure 1 gives an example of execution of a task with the DMR-B-1 scheme. In the figure, the horizontal lines represent execution of the task code by the processors assigned to it (PE A and B) or the spare processor, with the interval number indicated by the $I_{\dots}$ above the horizontal lines. The boxes represent operations done by the system to achieve fault-tolerance. In step 2 of the execution, while executing interval I_{j+1}, a fault occurred in processor B, so in the step 3 a spare processor repeats the execution of the same interval. After it finishes, its state matches the state of processor A, hence the interval is verified, and normal execution can be resumed. During step 4, both processors have faults, and the spare processor has to produce two correct executions before fault recovery is achieved, and normal execution can be resumed. As the spare processor has a fault during the recovery process (in step 6), it takes 3 steps to complete the recovery. We assume that at each step a new spare processor is used, in order to avoid a match between two faulty states that were caused by the same permanent fault. Because of that, a processor has to be loaded every step during fault recovery.

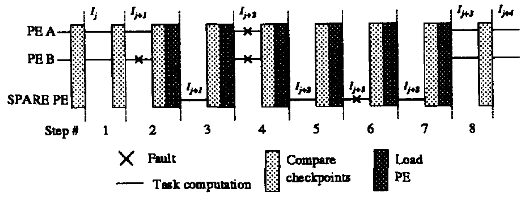

Figure 1: Example of execution with the DMR-B-1 scheme

2.1 Building The State-Machine

The first step in analyzing a fault recovery scheme is to build the *extended* state-machine that describes the operation of the scheme. The extended state-machine describes the behavior of the scheme in the eyes of an external viewer, who can observe the faults that occurred during a step. Two fault patterns that are not distinguishable in the scheme, but might later cause different actions, cause transitions to different states in the extended state-machine. For example, when two processors execute the same interval, and their states do not match at the end, the scheme can not tell if the fault occurred in one of the processors or in both. The number of faults might affect the ability of the scheme to recover from the faults, and thus should cause transitions to different states in the extended state-machine.

Each transition in the state-machine represents one step, and a transition is done at the end of each step. Because of the way the state-machine is constructed, the transition is determined only by the current state and the faults that occur during the current step (Markov property).

Each transition has associated with it a set of properties, called rewards. The rewards are used to evaluate the measures of interest related to the scheme. In this paper we are interested in the execution time of the schemes, and use two rewards for that. Other measures, such as the number of checkpoints stored in the stable storage and the number of processors used, can also be viewed as rewards and analyzed using the technique described here. The two quantities we use for execution time analysis are:

v_i — The amount of useful work that is done during the transition. We measure the useful work as the number of intervals whose checkpoints were matched as a result of the event that caused the transition.

t_i — The time it takes to complete the step that corresponds to the transition. As defined earlier a step starts when the processor(s) start to execute an interval, and ends the next time an interval is ready to be executed.

The time it takes to complete a step is the time to perform all the operations of that step. Each step includes at least the execution of the interval, denoted as t_I, and the comparison of the states at the end of the interval, denoted as t_{ck}. Some steps may include other operations that appears in Table 1.

Note that the first step in the analysis depends only on the scheme and is totally independent of the fault model.

In the DMR-B-1 scheme the operation has two basic modes. The first mode is the normal operation mode, where two processors are executing the task in parallel. The second mode is the fault recovery mode, where a single processor tries to find a match to an unverified checkpoint.

Figure 2 shows the extended state-machine for the DMR-B-1 scheme. The state-machine has two different fault recovery states, the first state has no correct execution of the current interval so far, and the second state has a single correct execution. State 2 in the machine is the normal execution state, and states 0 and 1 are the fault recovery states with the respective number of correct executions. Table 2 gives all the possible transitions in the state-machine with their properties. The first two columns in the table describe for each possible transition the event that causes it. The rest of the columns are explained later in the section.

The execution of the scheme starts at state 2, and if no faults occur it remains there, or in other words a transition is made via edge 0. If a mismatch between the states of the processors is found a transition to a fault recovery state is made. As the external observer knows how many faults occurred, it knows if it has to move along edge 2 to state 0 (faults in both processors), or along edge 1 to state 1 (one fault only).

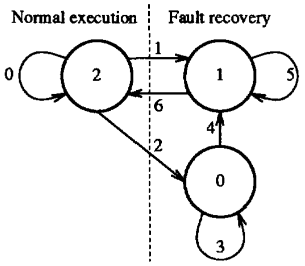

Figure 2: Extended state-machine for the DMR-B-1 scheme

In the fault recovery states the recovery processor executes the task again, and every time it fails it remains in the same state (transition via edge 3 or 5). When a correct execution is completed a transition to the next state is made.

For example, the execution of Figure 1 causes the following transitions (the number above the arrows are the edges that are used for the transitions)

$$2 \xrightarrow{0} 2 \xrightarrow{1} 1 \xrightarrow{6} 2 \xrightarrow{2} 0 \xrightarrow{4} 1 \xrightarrow{5} 1 \xrightarrow{6} 2 \xrightarrow{0} 2.$$

After the state-machine is built, rewards are assigned to each of its edges. The third and fourth columns in Table 2 show the values of the two rewards of interest, v_i and t_i.

In DMR-B-1 there are two transitions that complete the execution of an interval, and hence do useful work. The first transition is edge 0, where no fault occurred during normal execution. The second one is the transition out of the recovery mode, edge 6. The value of v_i for those two edges is 1. All other transitions do not do any useful work, and thus their value of v_i is 0.

The time to complete any step in the DMR-B-1 scheme includes the time to execute the interval and compare the checkpoints at the end. We assume that a spare processor is loaded before every step in the fault recovery mode, and the main processors are loaded

Edge No.	Event	Interval completion (v_i)	Time to execute (t_i)	Transition Probability (p_i)
0	No faults	1	$t_I + t_{ck}$	$(1-F)^2$
1	One fault	0	$t_I + t_{ck} + t_{ld}$	$2F(1-F)$
2	Two faults	0	$t_I + t_{ck} + t_{ld}$	F^2
3	Fault	0	$t_I + t_{ck} + t_{ld}$	F
4	No fault	0	$t_I + t_{ck} + t_{ld}$	$(1-F)$
5	Fault	0	$t_I + t_{ck} + t_{ld}$	F
6	No fault	1	$t_I + t_{ck} + t_{ld}$	$(1-F)$

Table 2: Transition description for the DMR-B-1 extended state-machine

when the recovery is completed. Hence all the edges have execution time of $t_I + t_{ck} + t_{ld}$, except edge 0 that has execution time of $t_0 = t_I + t_{ck}$.

2.2 Creating the Markov Chain

The second step in the analysis is assigning probabilities to each of the transitions in the state-machine constructed in the first step. Each edge i is assigned a probability p_i, which is the probability that the event that causes the transition via that edge will occur.

The probabilities assigned to the edges are determined by the fault model. In the simplest case it is assumed that the fault patterns do not change with time, and thus the transition probabilities are constants. More complex models assume that the fault pattern changes with time, or is a random process. In this case the probabilities of transitions are functions of time or random processes.

The transition probabilities out of a state do not depend on the way this state was reached. Hence the state-machine with the transition probabilities corresponds to a Markov chain. Together with the properties of the transitions, or the rewards, described earlier a Markov Reward Model is created. The analysis of this MRM provides results related to the fault recovery scheme.

In the example here, we assume that the fault pattern does not change with time, and thus the transition probabilities are constants. We also assume that the faults in different processors are independent of each other. This fault model is used in [9] and [12]. In this model F is the probability that a processor will have a fault while executing an interval. The probabilities of the transitions using this fault model appear in the fifth column of Table 2.

2.3 Analyzing the Scheme Using the MRM

After constructing the MRM induced by the fault recovery scheme and the fault model, its analysis provides the required results. The first step in solving the MRM is constructing the transition matrix of the Markov chain. In the transition matrix, called P, each entry $p_{i,j}$ is the probability of transition from state i to state j. If the transitions are not time dependent the Markov chain is called ***homogeneous*** and its transition matrix is constant. Otherwise the Markov chain is ***non-homogeneous***, and the transition matrix at time t is denoted by $P(t)$.

There are two ways to analyze a Markov chain, transient analysis and steady-state or limiting analysis. In the transient analysis we look at the state probabilities at each step, and from those probabilities get the desired quantities. In limiting analysis we look at the state probabilities in the limit as $t \rightarrow \infty$. A detailed discussion on analysis of Markov chains can be found in [8]. In this paper we use the steady-state analysis. At the end of the section we show that the steady-state analysis results match well simulation results.

A discrete time Markov chain has limiting probabilities if it is irreducible, aperiodic and homogeneous [8]. The limiting, or steady-state probabilities π can be found by solving the system of equations

$$\begin{aligned} \pi &= \pi \cdot P, \\ \sum_i \pi_i &= 1. \end{aligned}$$

The steady-state probability e_i of transition via edge i, from state u to state v is

$$e_i = \pi_u \cdot p_i,$$

where p_i is the transition probability of the edge. The average reward R for edge reward vector r is

$$R = \sum_i r_i e_i.$$

We now show how π, e and R can be used to perform time analysis of a checkpointing scheme of a task with n intervals.

Average Execution Time

The first thing we can get from the steady-state analysis of the Markov chain is the average number of intervals completed in a step, or the amount of useful work done during a step. This quantity is the average reward for the reward v and is given by

$$V = \sum_i v_i e_i. \tag{1}$$

The average number of steps it takes to complete a single interval is

$$S = \frac{1}{V} = \frac{1}{\sum_i v_i e_i}. \tag{2}$$

The average time it takes to complete a single step is the average reward for the reward vector t and is given by

$$T_s = \sum_i t_i e_i \tag{3}$$

From Eqs. (2) and (3) the average time to complete one interval and the whole task of n intervals are

$$\begin{aligned} T_1 &= S \cdot T_s = \frac{\sum_i t_i e_i}{\sum_i v_i e_i}, & (4) \\ T_n &= n \cdot T_1 = n \frac{\sum_i t_i e_i}{\sum_i v_i e_i}. & (5) \end{aligned}$$

Analysis of DMR-B-1

We now apply the results to the DMR-B-1 scheme with the fault model described earlier. The transition matrix of the scheme is

$$P = \begin{bmatrix} p_3 & p_4 & 0 \\ 0 & p_5 & p_6 \\ p_2 & p_1 & p_0 \end{bmatrix} = \begin{bmatrix} F & (1-F) & 0 \\ 0 & F & (1-F) \\ F^2 & 2F(1-F) & (1-F)^2 \end{bmatrix},$$

the steady-state probabilities are

$$\pi = \left\{ \frac{F^2}{1+F}, \frac{(2-F)F}{1+F}, \frac{1-F}{1+F} \right\},$$

and the average execution time of a task is

$$T_n = \frac{(1+F)(nt_I + nt_{ck}) + (4F - 3F^2 + F^3)nt_{ld}}{1-F}. \tag{6}$$

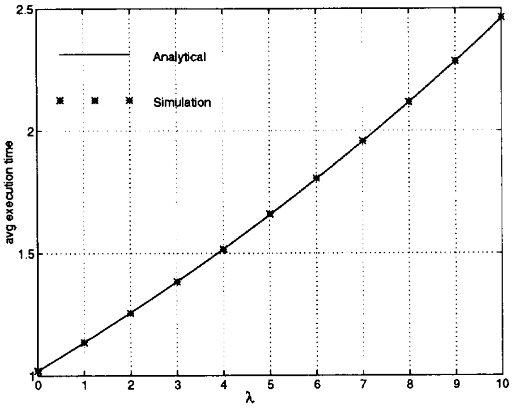

Figure 3: Comparison between analytical and simulation results of the average execution time for the DMR-B-1 scheme

Simulation Results

We compared the average execution time derived in this section with values measured using a simulation program. Figure 3 shows a comparison between the analytical values and those measured by the simulation program when faults occur according to a Poisson random process with rate λ, i.e., the probability of a fault in a processor during the execution of an interval is $F = 1 - e^{-\frac{\lambda}{n}}$. The comparison was made for a task of length 1 with 20 checkpoints ($n = 20$, $t_I = .05$), $t_{ck} = 0.001$ and $t_{ld} = 0.003$. The solid line is the execution time given in Eq. (6) and the asterisks are the average execution time measured using the simulation program. We can see that the simulation points fall on the line of analytical plot. In [17] we compared the analytical results with simulation results for other schemes. In all the schemes we examined the analytical and simulation results match well.

3 Scheme Comparison

In this section the analysis technique is used to compare between four existing checkpointing schemes. The schemes we compare are Triple Modular Redundant with checkpointing (TMR-F) [9], Double Modular Redundant with backward recovery and two recovery processors (DMR-B-2) [9], Double Modular Redundant with forward recovery and one recovery processor (DMR-F-1) [9], and Roll-Forward Checkpointing Scheme (RFCS) [12]. A short description of the schemes is given here. A more detailed description and the analysis of those schemes can be found in [17].

The simplest scheme is the TMR-F scheme [9]. In this scheme the task is executed by three processors, all of them executing the same interval. A fault in a single processor can be recovered without a rollback because two processors with correct execution still agree on the checkpoint. If faults occur in more than one processor all the processors are rolled back and execute the same interval again.

The DMR-B-2 scheme is described by Long *et al.* in [9]. In this scheme two processors execute the task. Whenever a fault occurs both processors are rolled back and execute the same interval again. The difference between this scheme and simple rollback schemes, like TMR-F, is that all the unverified checkpoints are stored and compared, not just the checkpoints of the last step. Hence two steps with a single fault are enough to verify an interval.

The next two schemes, DMR-F-1 and RFCS, use spare processors and the roll-forward recovery technique in order to avoid rollback [11]. In the DMR-F-1 scheme, suggested by Long *et al.* in [9], two processors are used during fault free steps. Three additional spare processors are added for a single step after each fault to try to recover without a rollback. The states of the two processors that are currently executing the task are copied to two of the spare processors. The third spare processor is loaded with the last verified checkpoint and tries to verify the faulty checkpoint. If it fails, either because it had a fault or because both processors had faults in the previous step, a rollback is done. If the verification succeeds then no rollback is done and the processor with the correct checkpoint and the one that this checkpoint was copied to continue to execute the task.

Pradhan and Vaidya [12] describe another roll-forward scheme called Roll-Forward Checkpointing Scheme (RFCS). In this scheme, as in DMR-F-1, a spare processor is used in fault recovery in order to avoid rollback. The difference between the schemes is that RFCS uses only one spare processor and the recovery takes two steps instead of one step in DMR-F-1. In the first step of fault recovery the spare processor is loaded with the last verified checkpoint and it tries to verify the current checkpoint, while the two regular processors continue with the normal execution. If the spare processor succeeds in verifying the first checkpoint the state of the correct processor is copied to the faulty processor. In the next step the spare processor tries to verify the next checkpoint, that has only one correct execution.

The behavior of the schemes is greatly affected by their exact implementation and the architecture of the parallel computer. Those parameters affect the time it takes to execute the operations that are needed at the end of each step, like comparing checkpoints and rolling back. To obtain general properties of the schemes, we will use a simpler model than the one used in Section 2. In this model the time to execute each step is $t_I + t_{oh}$, where t_{oh} is the overhead time required by the scheme. This overhead time is the same for all the transitions of the state-machine of the scheme. It is also assumed to be the same for all schemes.

The results of the simplified model are still valid when a more precise model, like the one used in Section 2, is used, for a large range of scheme parameters. However, in cases where there is a big difference between the time it takes to perform different operations, those results can change. Later in the section we give an example of such a case, where we assume that the schemes are implemented on workstations connected by a LAN. This implementation causes the slowest scheme in the general case, the DMR-B-2 scheme, to become the quickest scheme.

We compare here two properties of the schemes. The first property is the average execution time of a task using the scheme. The second property is the average work used to complete the execution of a task using the scheme. We assume that faults occur according to a Poisson random process with rate λ, i.e., the probability of a fault in a processor during the execution of an interval is $F = 1 - e^{-\frac{\lambda}{n}}$. We also assume that the number of checkpoints in the task is chosen such that the best possible result is achieved, given the scheme and the fault rate λ.

The average execution time of a task is important in real-time systems where fast response is desired. We show here that the average execution time is affected mostly by the number of processors used by the scheme, and the complexity of the scheme has only a minor effect.

The total work to complete the execution of a task is the sum, over all processors used by the scheme to complete the task, of the time they were in use. As the number of processors does not change during a step, the work W can be defined as

$$W = \sum_{\text{step } i} t_i \cdot c_i,$$

where t_i is the length of the step and c_i is the number of processors used in that step.

The average work of a scheme can be found by using the analysis technique described in Section 2. The number of processors used in every transition edge of the state-machine of the scheme is used as a reward C. The average number of processors used in a step is given by

$$\overline{C} = \sum_{\text{edges } i} e_i c_i,$$

and the average work is

$$\overline{W} = \overline{C} \cdot \overline{T},$$

where $\overline{T}$ is the average execution time of the task. For example, in the state-machine of the DMR-B-1 scheme described in Figure 2, two processors are used during normal execution (state 2) and a single processor is used in the fault recovery mode (states 0 and 1). The reward vector of the number of processors for the scheme is $\{2, 2, 2, 1, 1, 1, 1\}$. The average number of processors is given by

$$\overline{C} = 2 - \frac{2F}{1+F}.$$

The work is important in transaction systems, where high availability of the system is required, and thus the system should use as few resources as possible. We show here that the best work is achieved when a small number of processors is used, and again the complexity of the scheme has only a minor effect.

3.1 Simplified Model

To obtain general properties of the schemes without the influence of a specific implementation, we use a simpler model than the one used in Section 2. Using this model, we can also prove achievable lower bounds on both the average execution time and the total work. In the simplified model the time to execute each step is $t_I + t_{oh}$, where t_{oh} is the overhead time required by the scheme. This overhead time is the same for all the transitions of the state-machine of the scheme. It is also assumed to be the same for all schemes. Using this simplified model, the average execution time and the total work of a task with n intervals ($t_I = \frac{1}{n}$) are simplified to

$$\begin{aligned} \overline{T} &= n \cdot S \cdot (t_I + t_{oh}) = S \cdot (1 + nt_{oh}), \\ \overline{W} &= \overline{C} \cdot \overline{T} = \overline{C} \cdot S \cdot (1 + nt_{oh}), \end{aligned}$$

where S is the average number of steps to complete an interval.

3.1.1 Lower Bounds

Before the four schemes are compared, we give achievable lower bounds on both the average execution time and the average work. Later the performance of the schemes will be compared to these lower bounds. The lower bound for the average execution time is

$$\overline{T}_{\min} = 1 + t_{oh},$$

and the lower bound for the average work is

$$\overline{W}_{\min} = \left(2 + \lambda t_{oh} + \sqrt{\lambda^2 + 4\lambda t_{oh}}\right) \; e^{\frac{2}{1+\sqrt{1+\frac{4}{\lambda t_{oh}}}}}. \tag{7}$$

The DMR-B-1 scheme with optimal number of checkpoints achieves the lower bound for the average work. Proofs for both lower bounds can be found in [17].

3.1.2 Average Execution Time

The average execution time of a task with n checkpoints is

$$\overline{T} = n \cdot S \cdot (t_I + t_{oh}) = S \cdot (1 + nt_{oh}),$$

where S is the average number of steps it takes to complete an interval. Using the analysis technique described in Section 2, we calculated the average execution time of the four schemes considered in this section [17]:

$$\overline{T}_{\text{TMR-F}} = \tfrac{1}{1-(3F^2-2F^3)} \cdot (1 + nt_{oh}),$$

$$\overline{T}_{\text{DMR-B-2}} = \tfrac{1+3F}{(1-F)(1+F)^2} \cdot (1 + nt_{oh}),$$

$$\overline{T}_{\text{DMR-F-1}} = \tfrac{1+3F^2-2F^3}{1-3F^2+2F^3} \cdot (1 + nt_{oh}),$$

$$\overline{T}_{\text{RFCS}} = \tfrac{1+2F+3F^2-10F^3+8F^4-2F^5}{1+2F-11F^2+14F^3-8F^4+2F^5} \cdot (1 + nt_{oh}).$$

Figure 4 shows the average execution time of a task using each of the four schemes, with overhead time of $t_{oh} = 0.002$ for each step. The number of checkpoints for each scheme is chosen such that its average execution time is minimized [18].

The figure shows that the TMR-F scheme, despite being the simplest of the four schemes, has the lowest execution time. The TMR-F scheme has better execution time because it is using more processors than the other schemes, and thus has a much lower probability of failing to find two matching checkpoints. The DMR-B-2 scheme is the worst because it uses only two processors, and does not use spare processors to try to overcome the failure. The RFCS and DMR-F-1 schemes use spare processors during fault recovery, and thus have better performance than DMR-B-2.

3.1.3 Average Work

Applying the analysis technique to the four schemes gives the following average work:

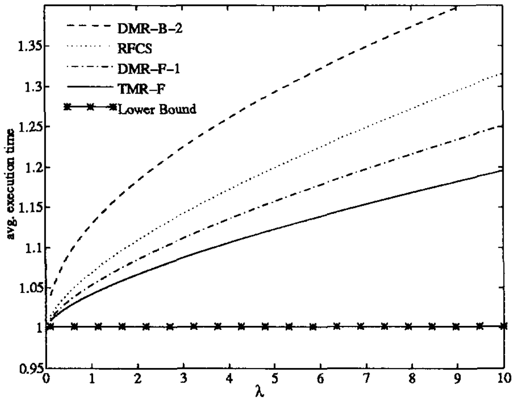

Figure 4: Average execution time with optimal checkpoints

$$\overline{W}_{\text{TMR-F}} = \tfrac{3}{1-(3F^2-2F^3)} \cdot (1 + nt_{oh}),$$

$$\overline{W}_{\text{DMR-B-2}} = \tfrac{2+6F}{(1-F)(1+F)^2} \cdot (1 + nt_{oh}),$$

$$\overline{W}_{\text{DMR-F-1}} = \tfrac{2+6F+3F^2-4F^3}{1-3F^2+2F^3} \cdot (1 + nt_{oh}),$$

$$\overline{W}_{\text{RFCS}} = \tfrac{2+8F+F^2-18F^3+16F^4-4F^5}{1+2F-11F^2+14F^3-8F^4+2F^5} \cdot (1 + nt_{oh}).$$

The average work of a task of length 1 with overhead time of $t_{oh} = 0.002$ for the four schemes is shown in Figure 5. The figure also shows the lower bound of the work, as given in Eq. (7).

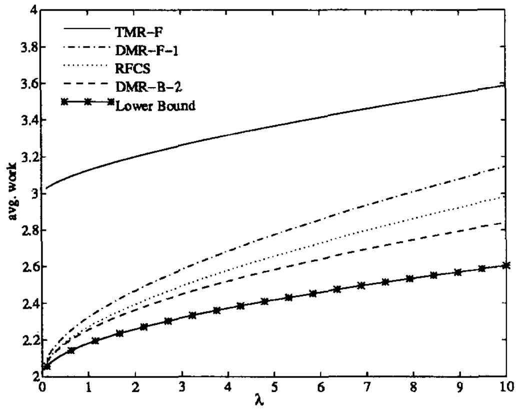

Figure 5: Average work with optimal checkpoints

The results here are the reverse of the results in the average execution time. The best scheme here is the DMR-B-2, which always uses only two processors. The RFCS and DMR-F-1, which use 2 processors during normal execution and add spare processors during fault recovery, require more work. The TMR-F scheme, which uses 3 processors, is the worst scheme.

3.2 Precise Model

When a more precise model is used, in which the time to perform each operation is used (as in the analysis done in Section 2), the results shown here for the simplified model are still valid for a large range of scheme parameters. There are some cases where a big difference in scheme parameters can cause different behaviors of the schemes than those described for the simplified model. These cases can still be analyzed with the technique described in this paper, by using the execution time equations given in [17] instead of the equation used in the simplified model.

For example, consider the following case: workstations connected by a LAN are used to implement the schemes. Each workstation saves its own checkpoint states, and sends only a short signature of them to the other workstations for comparison. In this implementation, operations that are done within a workstation can be completed relatively quickly, while operations that involve more than one workstation, and need the LAN, take much longer to execute. In this case schemes that do not use the network heavily have lower execution time than those which do. Specifically, the slowest scheme under the general model, the DMR-B-2 scheme, which uses the network only for state comparison can become the quickest scheme under these conditions. Figure 6 shows the execution time of a task when $t_{ck} = 0.001$, $t_r = .001$, $t_{cp} = 0.03$ and $t_{ld} = 0.03$, with optimal checkpoints. It can be seen that the DMR-B-2 scheme is the quickest after the failure rate, λ, reaches some critical value that require the other schemes to use the network heavily.

4 Conclusions

In this paper we have proposed a novel technique to analyze the performance of checkpointing schemes. The proposed technique is based on modeling the schemes under a given fault model with a Markov Reward Model, and evaluating the required measures by analyzing the MRM.

We used the proposed technique to compare the average execution time of a task and the total processor work done for four known checkpointing schemes. The

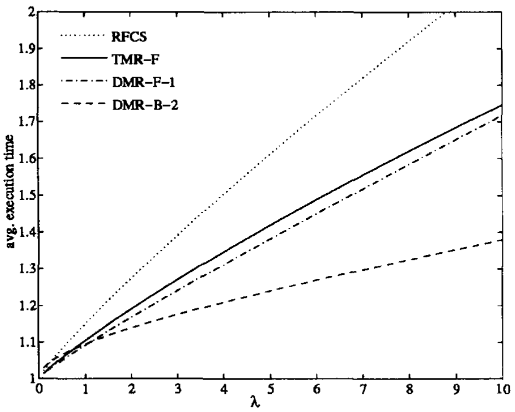

Figure 6: Average execution time for the workstations example

comparison shows that generally the number of processors has a major effect on both quantities. When a scheme uses more processors, its execution time decreases, while the total work increases. The complexity of the scheme has only a minor effect on its performance. In some cases, when there is a big difference between the time it takes to perform different operations, the general comparison results are no longer true. However, the proposed technique can still handle these cases and give correct results for them.

The proposed technique is not limited to the schemes described in this paper, or to the fault model used here. It can be used to analyze any checkpointing fault-tolerance scheme, with various fault models. The proposed technique can be also used to provide analytical answers to problems that haven't been dealt with before or were handled by a simulation study. Examples of such problems are deriving the number of checkpoints that minimizes the average execution time and computing the probability of meeting a given deadline.

References

[1] P. Agrawal. Fault tolerance in multiprocessor systems without dedicated redundancy. *IEEE Transactions on Computers*, 37:358–362, March 1988.

[2] P. A. Bernstein. Sequoia: A fault-tolerant tightly coupled multiprocessor for transaction processing. *Computer*, 21:37–45, February 1988.

[3] A. Bobbio. A multi-reward stochastic model for the completion time of parallel tasks. In *Proceedings of the Thirteenth International Teletraffic Congress*, pages 577–582, 1991.

[4] K. M. Chandy and C. V. Ramamoorthy. Rollback and recovery strategies for computer programs. *IEEE Transactions on Computers*, 21:546–556, June 1972.

[5] C. I. Dimmer. The tandem nonstop system. In T. Anderson, editor, *Resilient Computing Systems*, pages 178–196. John Wiley, 1985.

[6] L. Donatiello and V. Grassi. On evaluating the cumulative performance distribution of fault-tolerant computer systems. *IEEE Transactions on Computers*, 40:1301–1307, November 1991.

[7] R. A. Howard. *Dynamic Probabilistic Systems Vol II: Semi Markov and Decision Processes.* John Wiley, 1971.

[8] L. Kleinrock. *Queueing Systems, Vol. I: Theory.* John Wiley, 1975.

[9] J. Long, W. K. Fuchs, and J. A. Abraham. Forward recovery using checkpointing in parallel systems. In *The 19th International Conference on Parallel Processing*, pages 272–275, August 1990.

[10] J. Long, W. K. Fuchs, and J. A. Abraham. Compiler-assisted static checkpoint insertion. In *The 22nd IEEE International Symposium on Fault-Tolerant Computing*, pages 58–65, July 1992.

[11] D. K. Pradhan. Redundancy schemes for recovery. TR-89-cse-16, ECE Department, University of Massachusetts, Amherst, 1989.

[12] D. K. Pradhan and N. H. Vaidya. Roll-forward checkpointing scheme: Concurrent retry with nondedicated spares. In *IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, pages 166–174, July 1992.

[13] O. Serlin. Fault-tolerant systems in commercial applications. *Computer*, 17:19–30, August 1984.

[14] R. M. Smith and K. S. Trivedi. The analysis of computer systems using Markov reward processes. In H. Takagi, editor, *Stochastic Analysis of Computer and Communication Systems*, pages 589–629. North-Holland, 1990.

[15] D. Tang and R. K. Iyer. Dependability measurement and modeling of a multicomputer system. *IEEE Transactions on Computers*, 42:62–75, January 1993.

[16] S. Toueg and Ö. Babaoğlu. On the optimum checkpoint selection problem. *SIAM Journal on Computing*, 13:630–649, August 1984.

[17] A. Ziv and J. Bruck. Analysis of checkpointing schemes for multiprocessor systems. IBM Research Report RJ 9593, November 1993.

[18] A. Ziv and J. Bruck. Optimal number of checkpoints in checkpointing schemes. Manuscript, 1993.