Analysis of Checkpointing Schemes for Multiprocessor Systems

Avi Ziv*	Jehoshua Bruck
Information Systems Laboratory	IBM Almaden Research Center
Stanford University	650 Harry Road, K54/802
Stanford, CA 94305-4055	San Jose, CA 95120-6099
Phone: (415) 725-9696	Phone: (408) 927-1878
E-mail: avi@isl.stanford.edu	E-mail: bruck@almaden.ibm.com

Abstract

Parallel computing systems provide hardware redundancy that helps to achieve low cost fault-tolerance. Fault-tolerance is achieved, in those systems, by duplicating the task into more than one processor, and comparing the states of the processors at checkpoints. Many schemes that achieve fault tolerance exist, and most of them use checkpointing to reduce the time spent retrying a task. Performance evaluation for most of the schemes either relies on simulation results, or uses a simplified fault model.

This paper suggests a novel technique, based on a Markov Reward Model (MRM), for analyzing the performance of checkpointing schemes for fault-tolerance. 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 completion time and total work of four checkpointing schemes, TMR, DMR-B-2, DMR-F-1 and RFCS. We show that generally increasing the number of processors reduces the average completion time, but increases the total work done by the processors. Namely, the TMR scheme, which uses three processors, is the quickest but does the most work, while the DMR-B-2 scheme, which uses only two processors, is the slowest of the four schemes but does the least work. However, in cases where there is a big difference between the time it takes to perform different operations, those results can change. For example, when we assume that the schemes are implemented on workstations connected by a LAN and the time to move data between workstations is relatively long, the DMR-B-2 scheme can become quicker than the TMR scheme.

Key Words: parallel computing, fault-tolerance, checkpointing, Markov Reward Model.

*Partially supported by a grant from IBM.

1 Introduction

Parallel computing systems provide hardware redundancy that helps to achieve low cost fault-tolerance. Fault-tolerance is achieved, in those systems, by duplicating the task into more than a single processor, and comparing the states of the processors at checkpoints [9][11]. The usage of checkpoints reduces the time spent in retrying a task in the presence of failures, and hence reduces the average completion time of a task [4][15]. Reducing the task completion 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 [12] 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, hence 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, shortening fault recovery.

Implementation of a system which uses checkpointing can be done in various ways. Figure 1 illustrates a logical view of the system architecture. The system consists of a set of processing elements (PEs). Each PE has a processor and private memory, either its own memory in a distributed memory system or part of a shared memory system. All the PEs in the system are identical. The system has a stable storage, which can be accessed by all processing elements, that is used to store the states of the PEs at checkpoints. A special processor called the *Checkpoint Processor* is used to coordinate the execution of tasks on the PEs. The Checkpoint Processor assigns tasks to processors, it detects failures in the PEs by comparing the states of the processors at checkpoints, and coordinates the recovery according to the recovery scheme used.

Figure 1 illustrates an example for the architecture of the system, among many possible variations. For example, the Checkpoint Processor can be implemented as part of the processing elements or as a separate processor. The Stable Storage can be part of a shared memory, a separate part of a distributed memory, or stored in offline memory (disk). Different implementations of the architecture provide different execution times for operations in the fault recovery schemes and affect the performance.

The execution of a task is done in steps. Each step consists of a series of operations. The first operation in each step is executing 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 done, the Checkpoint Processor performs those operations necessary to achieve fault detection and recovery. The first is to store the states of the processors at the checkpoint in the stable storage, and to compare those states. Based on the result of the comparison and the scheme used, the Checkpoint Processor 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. Figure 2 shows the time notation for those operations, that is used in examples throughout the paper.

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.

Later schemes use checkpointing to avoid re-execution of an entire task. At each checkpoint the state of the task is stored into a stable memory. If a fault is detected and rollback is needed it can be done to the last stored checkpoint, not to the beginning of the task.

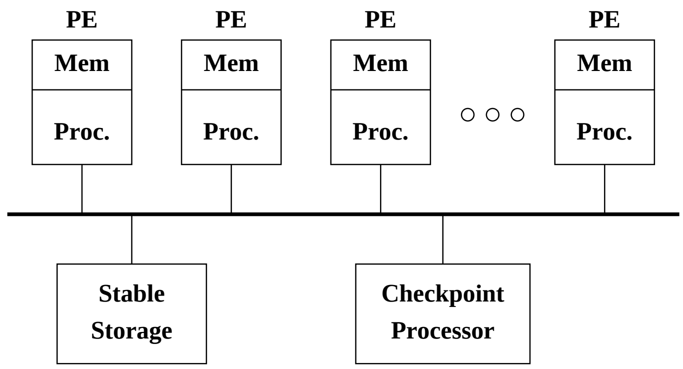


Figure 1: Logical system model

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

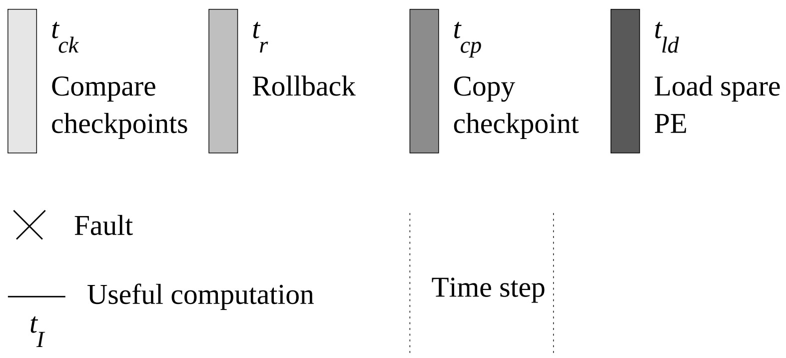


Figure 2: Time notation

Recently [9] and [11] proposed schemes that try to avoid rollback in the presence of a single fault by using spare processors to verify the state at the end of the last checkpoint, while the processors assigned to the task look ahead, beyond that checkpoint.

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 [11], or use a simplified fault model [9][11]. 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) [13]. 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.

Markov chains can be analyzed using transient analysis or steady-state analysis. The transient analysis is more accurate, but it is also more complicated. In this paper we concentrate on steady-state analysis, which is much simpler. We show, with simulation results, that the values achieved using steady-state analysis are close enough to the real values.

The proposed analysis technique is used to evaluate the average execution time and the CDF of the execution time for four checkpointing schemes, Triple Modular Redundant with checkpointing (TMR) [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) [11]. The results of the analysis are used to compare between the schemes. We evaluate two quantities, the average completion time of a task and the total work done to complete a task. The completion 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 tasks face hard deadlines. We show that the number of processors used to implement the scheme has a major effect on the average completion time, while the complexity of the scheme has only a minor effect.

Out of the four schemes examined in this paper, the TMR 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 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 completion time are not work efficient, and that the lowest work is done using schemes that use a small number of processors, and have higher completion time. The total work results of the four schemes examined here were the reverse of the completion time results. The DMR-B-2 scheme has the lowest total work, while the TMR 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 LAN only lightly is quicker than the TMR 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 the four schemes are described and analyzed. The analysis results are used to compare the average execution time and the total work of the schemes in Section 4. Section 5 concludes the paper.

2 Analysis Technique

The analysis of the schemes is based on the analysis of a discrete time *Markov Reward Model* (MRM) [13]. In a Markov Reward Model each state 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. In this paper we use a Markov Reward Model which is slightly different from the one used in [13]. In the model used here the rewards are assigned to the transitions of the Markov chain, not to the states.

Markov Reward Models are often used in evaluating the performance of computing systems.

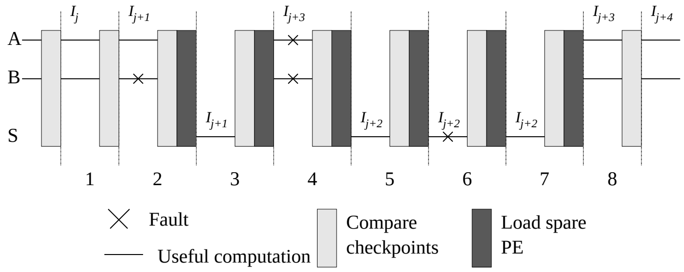


Figure 3: Example of execution with the DMR-B-1 scheme

Smith and Trivedi [13] describe the MRM and give examples of its use in evaluating reliability and performance of parallel computer, task completion time in faulty systems and properties of queueing systems. Others like [3], [6], [14], 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 a signature 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 matching states are found.

Figure 3 gives an example of execution of a task with the DMR-B-1 scheme. In the first step both processors that are assigned to the task execute the interval without faults, hence the comparison at the end of the interval succeeds, and the next interval is executed in the next step. During that step a fault occurred in one of the processors, and this causes the Checkpoint processor to start a third processor in the third step. This processor executes the same interval $(j+1)$. After it finishes, its state matches the state of processor A, hence the interval is verified, and normal execution can resume. In the forth step both processors have faults, and the spare processor has to produce two correct executions before fault recovery is achieved, and normal execution resumed. 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.

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 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 last state and the faults that occurred during the last 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 on Edge i. We measure the useful work as the number of intervals whose checkpoints were matched as a result of the event that caused the transition along the edge.

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.

The extended state-machine that describes the same scheme 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. Figure 4 shows the extended state-machine. 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 execution.

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).

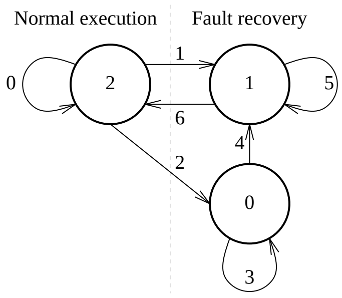


Figure 4: Extended state machine for the DMR-B-1 scheme

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. Table 2 gives the events for each transition edge.

In the example of Figure 3 the execution starts at State 2. In the first step there are no faults and a transition is made via Edge 0. In the second step there is a single fault which causes a transition to State 1 via Edge 1. In the next step a recovery is made and transition via Edge 6 takes place. In the forth step there are two faults which cause a transition via Edge 2 to State 0. In the next step there is one correct execution, hence there is a transition to State 1, and in step 7 the second correct execution is completed, which causes a transition back to State 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

Edge No.	Event	Interval completion (v_i)	Time to execute (t_i)	Transition Probability
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

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 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 even that it is a random process [8]. In this case the probabilities of transitions are functions of time or random processes.

The probabilities of transition 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 [11]. 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. This analysis can be done using computerized mathematics tools such as Mathematica [17].

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 $t \to \infty$. The limiting analysis is simpler than the transient analysis, but it is less accurate and does not always exist. A detailed discussion on analysis of Markov chains can be found in [7] and [16]. We concentrate on the steady-state analysis. Simulation results, given in Section 3, show that the steady-state results are close enough to the real results.

A discrete time Markov chain has limiting probabilities if it is irreducible, aperiodic and homogeneous [7]. 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

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

Probability Distribution of Execution Time

A precise calculation of the probability distribution of the execution time can be done with the transient analysis of the Markov chain. We give here an approximation to that distribution, based on the steady-state analysis of the Markov chain.

We approximate the number of steps it takes to complete n intervals as a negative binomial random variable with parameters V and n, where V is the average number of intervals completed in a single step, as given in Eq. (1). Hence the probability of completing the n intervals in k steps is

$$P_n(k) = \Pr\{n \text{ intervals completed in } k \text{ steps}\} = \binom{k-1}{n-1} V^n (1-V)^{k-n} \qquad k = n, n+1, \ldots$$

By approximating the time it takes to complete each step as T_s, the average time to complete a step, the probability of completing the task in time less than or equal to kT_s is

$$\Pr\{T \leq kT_s\} = \sum_{j=n}^{k} P_n(j).$$

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

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

and the steady-state probabilities are

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

which leads to the following quantities,

$$V = \frac{1-F}{1+F}, \tag{6}$$

$$S = \frac{1+F}{1-F}, \tag{7}$$

$$T_s = t_i + t_{ck} + \frac{4F - 3F^2 + F^3}{1+F} t_{ld}, \tag{8}$$

$$T_1 = \frac{(1+F)(t_i + t_{ck}) + (4F - 3F^2 + F^3) t_{ld}}{1-F}, \tag{9}$$

$$T_n = \frac{(1+F)(1 + nt_{ck}) + (4F - 3F^2 + F^3) nt_{ld}}{1-F}. \tag{10}$$

3 Schemes Analysis

In this section we describe and analyze four schemes that use parallel processors and checkpointing to achieve fault tolerance. The four schemes are Triple Modular Redundant with checkpointing

(TMR) [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) [11].

For each of the four schemes a short description of the scheme is given, with an execution example that uses the time notation of Figure 2. The description of the scheme is followed by the extended state-machine induced by the scheme with the rewards given to its edges. The state-machine is used to find the average execution time and the CDF of the execution time, using steady-state analysis of the Markov Reward Model. The analytical results are then compared with results measured using a simulation program.

In all the analysis done here the amount of useful computation in the task is 1. The task is divided into n intervals of length $\frac{1}{n}$. All time to perform any other operation is given as a fraction of the task length.

Fault Model

The fault model used in this section assumes that faults are time invariant, *i.e.*, the probabilities of faults do not change with time. This assumption is necessary for the steady-state analysis done here. Other fault models, like the one in [8] can be handled using the analysis technique given in the paper, but transient analysis of the Markov chain is needed.

In the fault model we use in this section the faults in different processors are independent of each other. The same model is used in [9] and [11]. We assume that faults occurred in each processor according to Poisson process with rate λ, *i.e.*, the probability that a processor will have a fault or faults while executing interval of length $t_I = \frac{1}{n}$ is

$$F = 1 - e^{-\frac{\lambda}{n}}. \tag{11}$$

Since faults in different processors are independent of each other, the probability of an event is the product of the probabilities of the sub events in each processor. For example the probability that a specific processor will not have a fault, while a single fault occur in one of two other processors is $(1-F) \cdot 2F(1-F) = 2F(1-F)^2$.

3.1 TMR

The simplest scheme described here is the TMR (Triple Modular Redundant) scheme. In this scheme the task is executed by three PEs, all of them executing the same interval. In this scheme

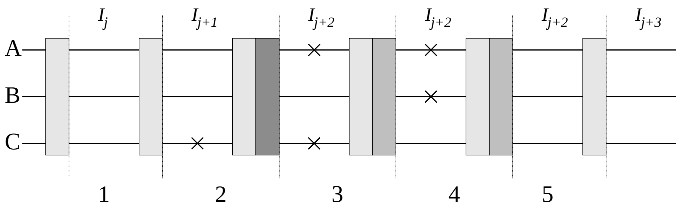
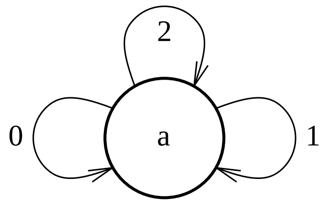


Figure 5: Time diagram for TMR

a fault in a single PE can be recovered without a rollback because two PEs with correct execution still agree on the checkpoint.

Figure 5 gives an example of execution of some intervals with the TMR scheme. If there are no faults during the execution of an interval, as in the first step in Figure 5, the next interval is executed immediately. In the second step there is a single fault in processor C, which is recovered immediately with the copy of the correct state from A or B to C. In the next two steps there are two faults, which cause rollback in both cases.

The Markov chain describing the scheme has only one state, with three edges for the three different cases of faults. Figure 6 gives the Markov chain for the scheme with description of the edges and their properties and the transition probability.



Edge No.	Event	Transition Probability	t_i	v_i
0	No faults	$(1-F)^3$	$t_I + t_{ck}$	1
1	Single fault	$3F(1-F)^2$	$t_I + t_{ck} + t_{cp}$	1
2	2 or 3 faults	$3F^2 - 2F^3$	$t_I + t_{ck} + t_r$	0

Figure 6: State machine for simple TMR scheme

The steady-state solution for the chain is easy to find. As there is only one state, the probability of being in that state is 1.

The steady-state edge probabilities are

$$\{e_0, e_1, e_2\} = \{(1-F)^3, 3F(1-F)^2, 3F^2 - 2F^3\}.$$

The average number of steps it takes to complete one interval is

$$S = \frac{1}{\sum_{i=0}^{2} e_i v_i} = \frac{1}{1-(3F^2-2F^3)}. \tag{12}$$

The average time to execute a step is

$$T_s = \sum_{i=0}^{2} e_i t_i = t_I + t_{ck} + (3F(1-F)^2)t_{cp} + (3F^2-2F^3)t_r. \tag{13}$$

The average time to complete a single interval and a task of n intervals are

$$T_1 = S \cdot T_s = \frac{t_I + t_{ck} + (3F(1-F)^2)t_{cp} + (3F^2-2F^3)t_r}{1-(3F^2-2F^3)},$$

$$T_n = nT_1 = \frac{1 + nt_{ck} + (3F(1-F)^2)nt_{cp} + (3F^2-2F^3)nt_r}{1-(3F^2-2F^3)}. \tag{14}$$

3.2 DMR-B-2

The DMR-B-2 scheme is described by Long *et al.* in [9]. This is a variation to the RAFT scheme [1]. In this scheme two processors execute the task. Whenever a fault occurred both processors are rolled back and executes the same interval again. The difference between this scheme and simple rollback schemes, like TMR, 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.

An example of execution of a task with the DMR-B-2 scheme is given in Figure 7. In the first step there are no faults, so the execution continues to the next interval in the second step. During the second and third steps there are faults, one in the second and two in the third, and there are no matching checkpoints, hence after both steps a rollback is done. During the forth step there is another fault, but this time a match between the checkpoint of processor B after this step and the checkpoint of processor A after step 2 is found and the next interval can be executed in the next step, after processor A is loaded with the correct checkpoint.

Figure 8 gives the Markov chain for the DMR-B-2 scheme. The chain has 2 states, 0 and 1, for the number of correct checkpoints for the interval found so far. For example at the second step in Figure 7 there is a single fault, hence one correct checkpoint is found and transition to state 1 is

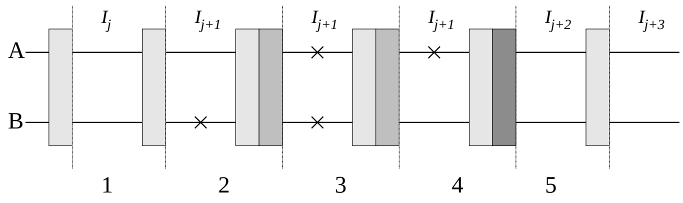
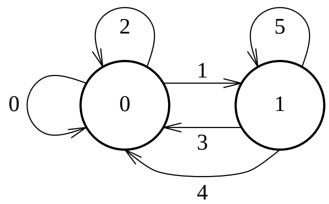


Figure 7: Time diagram for DMR-B-2



Edge No.	Event	Transition Probability	t_i	v_i
0	No faults	$(1-F)^2$	$t_I + t_{ck}$	1
1	Single fault	$2F(1-F)$	$t_I + t_{ck} + t_r$	0
2	Double fault	F^2	$t_I + t_{ck} + t_r$	0
3	No fault	$(1-F)^2$	$t_I + t_{ck}$	1
4	Single fault	$2F(1-F)$	$t_I + t_{ck} + t_{cp}$	1
5	Double fault	F^2	$t_I + t_{ck} + t_r$	0

Figure 8: State machine for DMR-B-2 scheme

made. In step 4 the second correct checkpoint is found, this cause a transition back to state 0, via edge 4.

The transition matrix for the Markov chain is

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

with steady-state probabilities of the states

$$\pi = \{\pi_0, \pi_1\} = \left\{ \frac{1+F}{1+3F}, \frac{2F}{1+3F} \right\}.$$

The average number of steps needed to complete one interval is

$$S = \frac{1}{\sum_{i=0}^{5} e_i v_i} = \frac{1+3F}{(1-F)(1+F)^2}, \tag{15}$$

and the average time to execute a step is

$$T_s = \sum_{i=0}^{5} e_i t_i = t_I + t_{ck} + \frac{4F^2(1-F)t_{cp} + (2F + F^2 + F^3)t_r}{(1+3F)}. \tag{16}$$

The average time to execute a single interval is

$$T_1 = ST_s = \frac{(1+3F)(t_I + t_{ck}) + 4F^2(1-F)t_{cp} + (2F + F^2 + F^3)t_r}{(1-F)(1+F)^2},$$

and the average time to execute the whole task is

$$T_n = nT_1 = \frac{(1+3F)(1 + nt_{ck}) + 4F^2(1-F)nt_{cp} + (2F + F^2 + F^3)nt_r}{(1-F)(1+F)^2}. \tag{17}$$

3.3 DMR-F-1

The next two schemes, DMR-F-1 described here, and RFCS described in the next subsection, use spare processors and the roll forward recovery technique in order to avoid roll back. 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 state of the two processors that are currently executing the task is 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 roll back is done. If the verification succeeds then no roll back is done and the processor with the correct checkpoint and the one that this checkpoint was copied to continue to execute the task.

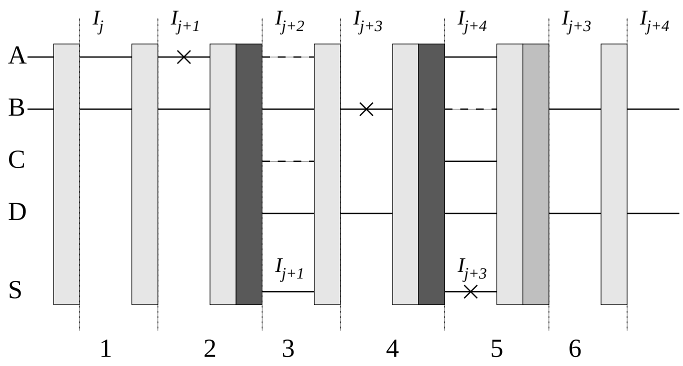


Figure 9: Time diagram for DMR-F-1

In Figure 9 processors A and B execute the task and processors C, D and S are the spares. In the second step there is a fault in processor A while executing interval $j+1$, so in the end of the step the state of processor A is copied to C, the state of B is copied to D and S is loaded with the state after interval j. In the third step S verifies the checkpoint of B and a full recovery is achieved, with processors B and D the new permanent processors for the task. In step 4 there is another fault in B and the recovery process begins again. This time S fails to verify where the fault was, so a rollback is done in the end of step 5, and interval $j+3$ is repeated in step 6.

Figure 10 shows the Markov chain for the DMR-F-1 scheme. The chain has three states, State 0 for normal execution, State 1 for fault recovery when single fault occurred, and recovery is possible, and State 2 for fault recovery when double faults occurred and recovery is impossible. The description of the edges in the chain is also given in Figure 10. In the fault recovery states processors A and B are the two processors in the non-faulty path. For example the execution of Figure 9 causes the following transitions (the number above the arrows are the edges that are used for the transitions)

$$0 \xrightarrow{0} 0 \xrightarrow{1} 1 \xrightarrow{3} 0 \xrightarrow{1} 1 \xrightarrow{4} 0 \xrightarrow{0} 0.$$

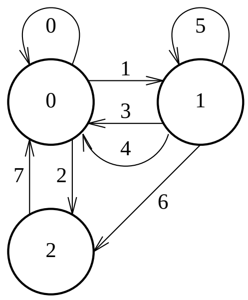
The transition matrix of the Markov chain is

$$P = \begin{bmatrix} p_0 & p_1 & p_2 \\ p_3 + p_4 & p_5 & p_6 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} (1-F)^2 & 2(1-F)F & F^2 \\ (1-F)^3 + F & 2(1-F)^2 F & (1-F)F^2 \\ 1 & 0 & 0 \end{bmatrix},$$

and the steady-state probabilities of the states are

$$\pi = \{\pi_0, \pi_1, \pi_2\} = \left\{ \frac{1 - 2F + 4F^2 - 2F^3}{1 + 3F^2 - 2F^3}, \frac{2(1-F)F}{1 + 3F^2 - 2F^3}, \frac{F^2}{1 + 3F^2 - 2F^3} \right\}.$$

Using these probabilities and the reward of the edges, v_i and t_i, we can find the average number



Edge No.	Event	Transition Probability	t_i	v_i
0	No faults in (A, B)	$(1-F)^2$	$t_I + t_{ck}$	1
1	Single fault in (A, B)	$2F(1-F)$	$t_I + t_{ck} + t_{ld}$	0
2	Double fault in (A, B)	F^2	$t_I + t_{ck} + t_{ld}$	0
3	No fault in (A, B, S)	$(1-F)^3$	$t_I + t_{ck}$	2
4	Fault in S, (A, B) don't care	F	$t_I + t_{ck} + t_r$	0
5	No fault in S, 1 fault in (A, B)	$2F(1-F)^2$	$t_I + t_{ck} + t_{ld}$	1
6	No fault in S, 2 faults in (A, B)	$F^2(1-F)$	$t_I + t_{ck} + t_{ld}$	1
7	Always	1	$t_I + t_{ck} + t_r$	0

Figure 10: State machine for DMR-F-1 scheme

of steps to complete a single interval as

$$S = \frac{1}{\sum_{i=0}^{7} v_i e_i} = \frac{1+3F^2-2F^3}{1-3F^2+2F^3}. \tag{18}$$

The average time to execute a step is

$$T_s = \sum_{i=0}^{7} t_i e_i = t_I + t_{ck} + \frac{(3F^2-2F^3)t_r + (2F-F^2)t_{ld}}{1+3F^2-2F^3}. \tag{19}$$

The average time to complete a single interval is

$$T_1 = ST_s = \frac{t_I + t_{ck} + (3F^2-2F^3)(t_I + t_{ck} + t_r) + (2F-F^2)t_{ld}}{1-3F^2+2F^3},$$

and the total average time to execute the task is

$$T_n = n \cdot T_1 = \frac{1 + nt_{ck} + (3F^2-2F^3)(1 + nt_{ck} + nt_r) + (2F-F^2)nt_{ld}}{1-3F^2+2F^3}. \tag{20}$$

3.4 RFCS

Pradhan and Vaidya [11] give 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 next 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 have only one correct execution. In Figure 11 the fault in processor B during step 2 if recovered in steps 3 and 4. In step 5 there is another fault in processor B. This time the recovery fails due to the fault in S in the next step, so rollback is done. After step 7 another fault recovery starts due to the fault in processor A while executing interval $j+4$. The first step of the recovery is successful but because of the fault in S at step 9 interval $j+5$ is not verified and roll back is needed.

Figure 12 gives the Markov chain for the RFCS scheme. State 0 is the normal execution state. States 1 and 2 are the first step in the recovery, with and without recovery chance. State 3 is the second step in the recovery. Description of the edges is given in Figure 12. The transitions of the execution in Figure 11 are

$$0 \xrightarrow{0} 0 \xrightarrow{1} 1 \xrightarrow{6} 3 \xrightarrow{9} 0 \xrightarrow{1} 1 \xrightarrow{4} 0 \xrightarrow{1} 1 \xrightarrow{6} 3 \xrightarrow{8} 0.$$

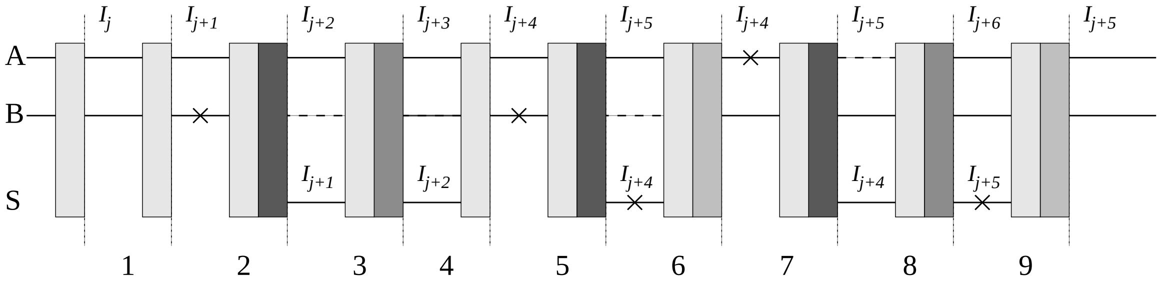
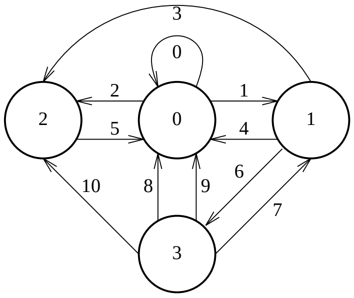


Figure 11: Time diagram for RFCS



Edge No.	Event	Transition Probability	t_i	v_i
0	No faults in (A, B)	$(1-F)^2$	$t_I + t_{ck}$	1
1	Single fault in (A, B) (assumed to be in A)	$2F(1-F)$	$t_I + t_{ck} + t_{ld}$	0
2	Double fault in (A, B)	F^2	$t_I + t_{ck} + t_{ld}$	0
3	Fault in B No fault in S	$F(1-F)$	$t_I + t_{ck} + t_{cp}$	1
4	Fault in S	F	$t_I + t_{ck} + t_r$	0
5	always	1	$t_I + t_{ck} + t_r$	0
6	No fault in (B, S)	$(1-F)^2$	$t_I + t_{ck} + t_{cp}$	1
7	Single fault in (A, B) No fault in S	$2F(1-F)^2$	$t_I + t_{ck}$	1
8	Fault in S	F	$t_I + t_{ck} + t_r$	0
9	No fault in (A, B, S)	$(1-F)^3$	$t_I + t_{ck}$	2
10	Double fault in (A, B) No fault in S	$F^2(1-F)$	$t_I + t_{ck}$	1

Figure 12: State machine for RFCS scheme

The transition matrix of the Markov chain is

$$P = \begin{bmatrix} p_0 & p_1 & p_2 & 0 \\ p_4 & 0 & p_3 & p_6 \\ 1 & 0 & 0 & 0 \\ p_8 + p_9 & p_7 & p_{10} & \end{bmatrix} = \begin{bmatrix} (1-F)^2 & 2F(1-F) & F^2 & 0 \\ F & 0 & F(1-F) & (1-F)^2 \\ 1 & 0 & 0 & 0 \\ (1-F)^3 + F & 2F(1-F)^2 & F^2(1-F) & 0 \end{bmatrix},$$

and the steady-state probabilities of the states are

$$\begin{aligned} \pi &= \{\pi_0, \pi_1, \pi_2 \pi_3\} \\ &= \left\{ \frac{1 - 2F + 8f^2 - 12F^3 + 8f^4 - 2F^5}{1 + 2F + 3F^2 - 10F^3 + 8f^4 - 2F^5}, \frac{2F(1-F)}{1 + 2F + 3F^2 - 10F^3 + 8f^4 - 2F^5}, \right. \\ &\quad \left. \frac{F^2(3 - 4F + 2F^2)}{1 + 2F + 3F^2 - 10F^3 + 8f^4 - 2F^5}, \frac{2F(1-F)^3}{1 + 2F + 3F^2 - 10F^3 + 8f^4 - 2F^5} \right\}. \end{aligned}$$

Using these probabilities and the reward of the edges, v_i and t_i, the average number of steps to complete a single interval is

$$S = \frac{1}{\sum_{i=0}^{10} v_i e_i} = \frac{1 + 2F + 3F^2 - 10F^3 + 8f^4 - 2F^5}{1 + 2F - 11F^2 + 14F^3 - 8F^4 + 2F^5}. \tag{21}$$

The average time to complete a step is

$$\begin{aligned} T_s &= \sum_{i=0}^{10} t_i e_i = \quad t_I + t_{ck} \\ &\quad + \frac{2F(1-F)^2}{1 + 2F + 3F^2 - 10F^3 + 8F^4 - 2F^5} t_{cp} \\ &\quad + \frac{2F - F^2}{1 + 2F + 3F^2 - 10F^3 + 8F^4 - 2F^5} t_{ld} \\ &\quad + \frac{7F^2 - 12F^3 + 8F^4 - 2F^5}{1 + 2F + 3F^2 - 10F^3 + 8F^4 - 2F^5} t_r. \end{aligned} \tag{22}$$

The average time to complete a single interval is

$$\begin{aligned} T_1 &= ST_s = \frac{1 + 2F + 3F^2 - 10F^3 + 8F^4 - 2F^5}{1 + 2F - 11F^2 + 14F^3 - 8F^4 + 2F^5}(t_I + t_{ck}) \\ &\quad + \frac{2F(1-F)^2}{1 + 2F - 11F^2 + 14F^3 - 8F^4 + 2F^5} t_{cp} \\ &\quad + \frac{2F - F^2}{1 + 2F - 11F^2 + 14F^3 - 8F^4 + 2F^5} t_{ld} \\ &\quad + \frac{7F^2 - 12F^3 + 8F^4 - 2F^5}{1 + 2F - 11F^2 + 14F^3 - 8F^4 + 2F^5} t_r, \end{aligned}$$

and the total average time to execute the task is

$$T_n = n \cdot T_1 = \frac{1 + 2F + 3F^2 - 10F^3 + 8F^4 - 2F^5}{1 + 2F - 11F^2 + 14F^3 - 8F^4 + 2F^5}(1 + nt_{ck})$$

$$
\begin{aligned}
&+ \frac{2F(1-F)^2}{1+2F-11F^2+14F^3-8F^4+2F^5} n t_{cp} \\
&+ \frac{2F-F^2}{1+2F-11F^2+14F^3-8F^4+2F^5} n t_{ld} \\
&+ \frac{7F^2-12F^3+8F^4-2F^5}{1+2F-11F^2+14F^3-8F^4+2F^5} n t_r . \qquad (23)
\end{aligned}
$$

3.5 Simulation Results

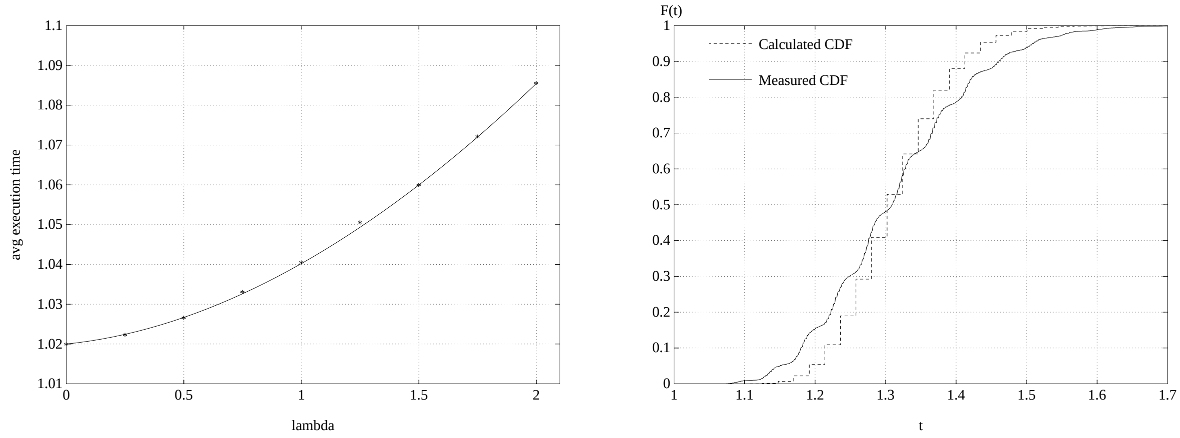
We compared the average execution time and the CDF derived in this section with values measured using a simulation program for all four schemes analyzed in this section. Figure 13 shows a comparison between the analytical values and those measured by the simulation program for the DMR-F-1 scheme. The results for the other three schemes where the same as the results for the DMR-F-1 scheme.

In Figure 13a the average execution time of the DMR-F-1 scheme, given by Eq. (20) is compared with the average execution time measured by a simulation program, for 20 checkpoints ($n = 20$), $t_{ck} = 0.001$, $t_r = 0.002$ and $t_{cp} = 0.002$. The solid line is the execution time given in Eq. (20) 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.

The CDF of the execution time as calculated by the technique shown in Section 2, with the values of S and T_s given in Eqs. (18) and (19), are compared in Figure 13b to the CDF measured by the simulation program for the case of $n = 100$, $\lambda = 10$, $t_{ck} = 0.001$, $t_r = 0.002$ and $t_{cp} = 0.002$. The calculated CDF is close to the measured one, but the slope of the graph is bigger in the calculated case. The difference in the distribution functions is caused by the fact that the negative binomial approximation is not accurate because a step can have more than single interval completed in it.

4 Scheme Comparison

In this section the results of Section 3 are used to compare between the the schemes described in that section. 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, rolling back, etc. To obtain general properties of the schemes, we will use a simpler model than the one used in Section 3. In this model the time to execute each step is $t_I + t_{oh}$, where t_{oh} is the overhead time



(a) Average execution time

(b) CDF of execution time

Figure 13: Comparison between analytical and simulation results for the DMR-F-1 scheme

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 3, 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 completion 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. In both cases we 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 (see [18]).

The average completion time of a task is important in real-time systems that have hard deadlines. We show here that the average completion time is affected mostly by the number of processors used by the scheme, and the complexity of the scheme has only a minor effect. In Section 4.1.1 we show that with enough processors event the simplest scheme can achieve the lowest possible average completion time.

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

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

and the average work is

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

For example, in the state machine of the DMR-B-1 scheme described in Figure 4, two processors are used during normal execution (State 2) and a single processor is used in 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.

4.1 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 3. 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

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

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

4.1.1 Lower Bounds

Before the four schemes, described in Section 3, are compared we give achievable lower bounds on both the average completion time and the average work. Later the performance of the schemes will be compared to these lower bounds.

Lemma 1 *The average time to complete a task of length 1, with overhead time of t_{oh} after each step, is greater than or equal to $1 + t_{oh}$.*

Proof: To execute the whole task at least one time unit is needed. The state of the task is compared at least once, hence the minimum time to complete the task is $1 + t_{oh}$. ∎

Lemma 2 *The lower bound on the execution time, given in Lemma 1, is achievable.*

Proof: We use a scheme, similar to the TMR scheme described in Section 3, with m processors instead of three. The task is executed with a single checkpoint in the end (the number of intervals is 1). The probability that the task will not be completed in a single step is the probability that at least $m - 1$ processors has faults. This probability is

$$P_f = F^m + m(1-F)F^{m-1} = F^{m-1}[F + m(1-F)] \leq mF^{m-1}.$$

The number of steps to complete the task is geometric random variable, hence the average time to complete the task is

$$\overline{T} = \frac{1 + t_{oh}}{1 - P_f} \leq \frac{1 + t_{oh}}{1 - mF^{m-1}},$$

and

$$\lim_{m \to \infty} \overline{T} = 1 + t_{oh}.$$

∎

Lemmas 1 and 2 show that given enough processors, we can bring the average time to complete a task as close as we want to the lower bound. This may be very important in real-time tasks, where tasks have to meet deadlines. The problem is that in order to achieve this bound we use a large number of processors, so that a lot of processor work is wasted.

On the other hand in transaction systems, where the important requirement from the system is availability we want to do as little work as possible to complete a task, even if it means slower execution time. Next we show that the work to complete a task can also be bounded.

Lemma 3 *The total average work, W, to complete a task of length 1 with overhead time of t_{oh} after each step, is bounded from below by*

$$\overline{W}_{\min} = \left(2 + \lambda t_{oh} + \sqrt{\lambda^2 + 4\lambda t_{oh}}\right)\ e^{\frac{2}{1+\sqrt{1+\frac{4}{\lambda t_{oh}}}}}. \tag{24}$$

Proof: The proof of the lemma uses the following propositions.

Proposition 4 *A scheme can be simulated by a single processor, with the same amount of work.*

Proof: A single processor can execute the intervals that are given to each of the processors assigned to the task in each step. As the failures are both time invariant and processor independent, the failure pattern is not affected by the simulation. ∎

Proposition 5 *The scheme that uses a single processor with the fastest average completion time has the lowest possible work.*

Proof: For a scheme that uses only one processor the completion time is also the total work of the scheme. As every scheme can be simulated by a single processor, there exists a scheme with the lowest possible work that uses a single processor. This scheme has the lowest average completion time. ∎

Proposition 6 *For a given number of checkpoints n, the best average completion time for a single processor is*

$$\overline{T}_{\min} = (2 + 2nt_{oh})\ e^{\frac{\lambda}{n}}. \tag{25}$$

Proof: The simple scheme, that executes each interval until two matching checkpoints are found, has the best average completion time because it never tries to execute an interval more than necessary. The probability of correct execution of an interval is $1 - F$, hence the average number of steps until a correct execution of an interval is $\frac{1}{1-F}$, the average number of steps to find two correct executions is $\frac{2}{1-F}$. The time to execute each step is $\frac{1}{n} + t_{oh}$ and the task has n intervals, so the average completion time is

$$\overline{T}_{\min} = \frac{2 + 2nt_{oh}}{1 - F}.$$

For the value of $F = 1 - e^{-\frac{\lambda}{n}}$ we get the minimum value of Eq. 25. ∎

From Proposition 5, the scheme described in Proposition 6 has the best work among all schemes with n intervals. To find the optimal work, we need to find the number of intervals that minimizes $\overline{T}_{\min}$. The value of n that achieves this minimum is

$$n_{\min} = \frac{\lambda + \sqrt{\lambda^2 + \frac{4\lambda}{t_{oh}}}}{2}$$

which leads to the value of optimal work given in Eq.24. ∎

Corollary 7 *The DMR-B-1 scheme with $n_{\min}$ intervals has optimal work.*

Proof: The scheme of Proposition 6 is the single processor simulation of DMR-B-1. ∎

4.1.2 Comparison of Average Completion Time

The average completion 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 completion time of the four schemes:

$$\begin{aligned}
\overline{T}_{\text{TMR}} &= \frac{1}{1-(3F^2-2F^3)} \cdot (1+nt_{oh}), \\
\overline{T}_{\text{DMR-B-2}} &= \frac{1+3F}{(1-F)(1+F)^2} \cdot (1+nt_{oh}), \\
\overline{T}_{\text{DMR-F-1}} &= \frac{1+3F^2-2F^3}{1-3F^2+2F^3} \cdot (1+nt_{oh}), \\
\overline{T}_{\text{RFCS}} &= \frac{1+2F+3F^2-10F^3+8F^4-2F^5}{1+2F-11F^2+14F^3-8F^4+2F^5} \cdot (1+nt_{oh}).
\end{aligned}$$

Figure 14 shows the average completion 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 completion time is minimized [18].

The figure shows that the TMR scheme, despite being the simplest of the four schemes, has the lowest completion time. The TMR scheme has better completion 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.

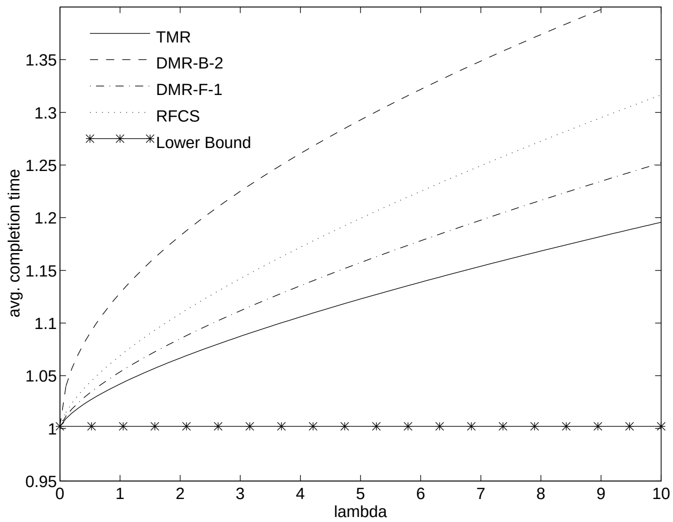


Figure 14: Average execution time with optimal checkpoints

4.1.3 Comparison of Average Work

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

$$\begin{aligned}
\overline{W}_{\text{TMR}} &= \frac{3}{1-(3F^2-2F^3)} \cdot (1+nt_{oh}), \\
\overline{W}_{\text{DMR-B-2}} &= \frac{2+6F}{(1-F)(1+F)^2} \cdot (1+nt_{oh}), \\
\overline{W}_{\text{DMR-F-1}} &= \frac{2+6F+3F^2-4F^3}{1-3F^2+2F^3} \cdot (1+nt_{oh}), \\
\overline{W}_{\text{RFCS}} &= \frac{2+8F+F^2-18F^3+16F^4-4F^5}{1+2F-11F^2+14F^3-8F^4+2F^5} \cdot (1+nt_{oh}).
\end{aligned}$$

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 15. The figure also shows the lower bound of the work, as given in Eq. (24).

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

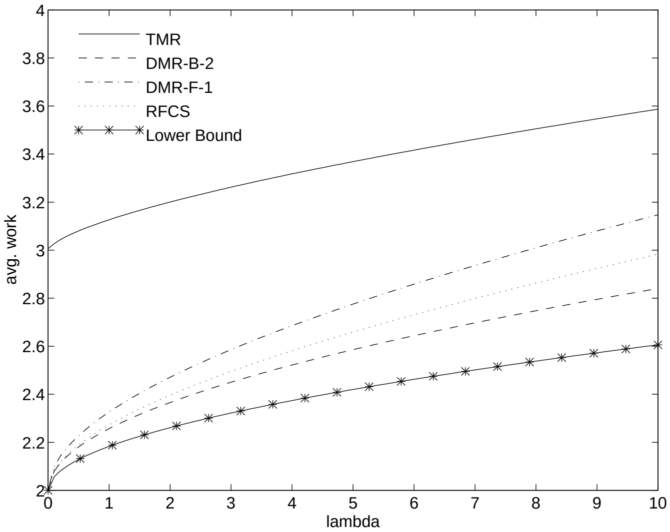


Figure 15: Average work with optimal checkpoints

4.2 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. Those cases can still be analyzed with the technique described in this paper, by using the execution time equations given in Section 3 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 16 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.

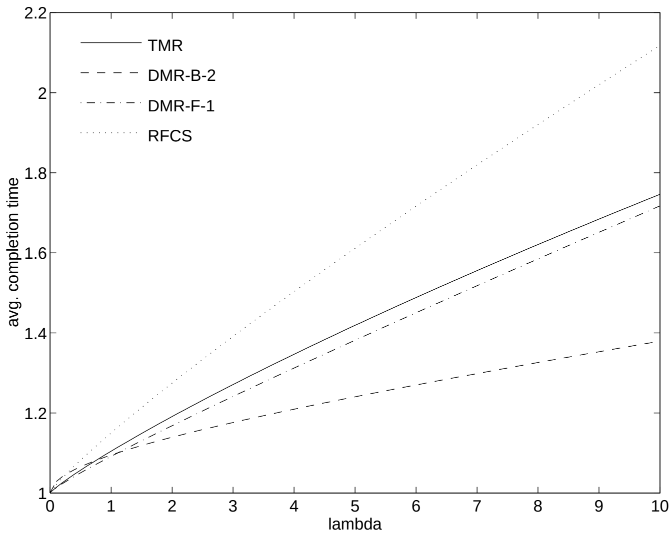


Figure 16: Average execution time for the workstations example

5 Conclusions

In this paper we 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 analyze the execution time of a task of four different schemes. We compared the average and the probability distribution function of the execution time, calculated using the proposed technique, with the real values, measured using a simulation program. Despite the fact that the analysis used approximations, such as the steady-state analysis of the Markov chain and the use of negative binomial as the distribution function, the difference between the analysis results and the simulation results is very small.

We compared the average execution time of a task and the total processor work done for four known checkpointing schemes. The 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 completion 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] L. Kleinrock. *Queueing Systems, Vol. I: Theory.* John Weiley, 1975.

[8] C. M. Krishna and A. D. Singh. Modeling correlated transient failures in fault-tolerant systems. In *The 20th IEEE International symposium on Fault-Tolerant Computing*, pages 244–251, June 1990.

[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 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.

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

[13] 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.

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

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

[16] K. S. Trivedi. *Probability and Statistics With Reliability, Queuing, and Computer Science Applications*. Prentice-Hall, 1982.

[17] S. Wolfram. *Mathematica A System for Doing Mathematics by Computer*. Addison-Wesley, 1988.

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