The Average Availability of Parallel Checkpointing Systems and Its Importance in Selecting Runtime Parameters

James S. Plank

Michael G. Thomason

Department of Computer Science
University of Tennessee
Knoxville, TN 37996
[plank,thomason]@cs.utk.edu

Appearing in the 29th International Symposium on Fault-Tolerant Computing (FTCS-29),
Madison, WI, June 18, 1999.

Available via ftp to cs.utk.edu in pub/plank/papers/FTCS29.ps.Z
Or on the web at http://www.cs.utk.edu/~plank/plank/papers/FTCS29.html

The Average Availability of Parallel Checkpointing Systems and Its Importance in Selecting Runtime Parameters

James S. Plank Michael G. Thomason
Department of Computer Science, University of Tennessee
[plank,thomason]@cs.utk.edu

Abstract

Performance prediction of checkpointing systems in the presence of failures is a well-studied research area. While the literature abounds with performance models of checkpointing systems, none address the issue of selecting runtime parameters other than the optimal checkpointing interval. In particular, the issue of processor allocation is typically ignored. In this paper, we briefly present a performance model for long-running parallel computations that execute with checkpointing enabled. We then discuss how it is relevant to today's parallel computing environments and software, and present case studies of using the model to select runtime parameters.

1 Introduction

Performance prediction of checkpointing systems is a well-studied area. Most work in this area revolves around selecting an *optimal checkpoint interval*. This is the frequency of checkpointing that minimizes the expected execution of an application in the presence of failures. For uniprocessor systems, selection of such an interval is for the most part a solved problem [19, 26]. There has been important research in parallel systems [12, 25, 28], but the results are less unified.

To date, most checkpointing systems for long-running distributed memory computations (e.g. [4, 5, 13, 22, 24]) are based on *coordinated checkpointing* [8]. At each checkpoint, the global state of all the processors is defined and stored to a highly available stable storage. If any processor fails, then a replacement processor is selected to take the place of the failed processor, and then all processors restore the saved state of the computation from the checkpoint.

When a user must execute a long-running application on a distributed memory computing system, he or she is typically faced with an important decision: How many processors should the application use? Most programs for such environments require the user to choose such a value before the computation begins, and once underway, the value may not change. On a system with no checkpointing, the application typically employs as many as are available for the most parallelism and the shortest running time. However, when a system is enabled with checkpointing, then the answer is less clear. If all processors are used for the application and one fails, then the application may not continue until that processor is repaired and the whole system may recover. If fewer processors are used for the application, then the application may take longer to complete in the absence of failures, but if a processor fails, then there may be a *spare* processor standing by to be an immediate replacement. The application will spend less time down due to failures. Consequently, selecting the number of processors on which to run the application is an important decision.

In this paper, we model the performance of coordinated checkpointing systems where the number of processors dedicated to the application (termed a for "active") and the checkpoint interval (termed I) are selected by the user before running the program. We use the model to determine the *average availability* of the program in the presence of failures, and we show how average availability can be used to select values of a and I that minimize the expected running time of the program. We then give examples of parameter selection using parallel benchmarks and failure data from a variety of parallel workstation environments.

The significance of this work is that it addresses an important runtime parameter selection problem that has not been addressed heretofore.

2 The System Model

We are running a parallel application on a distributed memory system with N total processors. Processors are interchangeable. The application uses exactly $a \leq N$ processors, a being chosen by the user. Processors may fail and be repaired. We term a processor as *functional* when it can be used to execute the application. Otherwise it is *failed and*

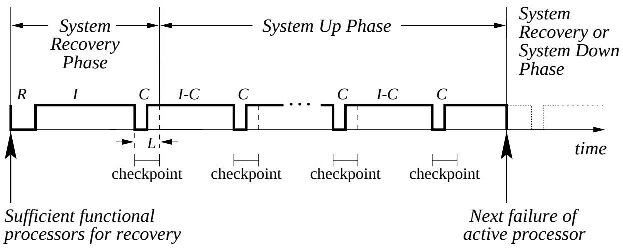


Figure 1. The sequence of time between the recovery of an application from a failure, and the failure of an active processor.

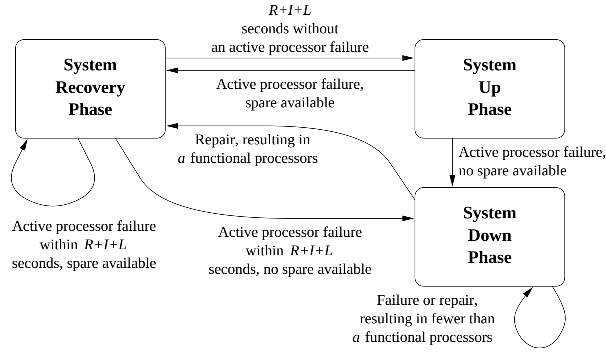


Figure 2. Phase transition diagram.

under repair. We assume that interoccurrence times of failures for each processor are independent and identically distributed (*iid*) as exponential random variables with the same failure rate $\lambda > 0$. Likewise, repairs are *iid* as exponential random variables with repair rate θ. Occurrences of failures or repairs at exactly the same instant have probability 0 for the exponential probability laws.

When the user initiates the application, it may start running as soon as there are a functional processors. If, after I seconds, none of the a processors has failed, a checkpoint is initiated. This checkpoint takes L seconds to complete, and once completed it may be used for recovery. L is termed the "checkpoint latency." The checkpoint adds C seconds of overhead to the running time of the program. C is termed the "checkpoint overhead." Many checkpointing systems use optimizations such as "copy-on-write" so that $C \ll L$, which improves performance significantly [26]. While there are no failures among the a processors, checkpoints are initiated every I seconds. I must be greater than or equal to L so that the system is never attempting to store multiple checkpoints simultaneously.

When an active processor fails, the application is halted and a replacement is sought. If there are no replacements, then the application must stand idle until there are again a functional processors. When there are a functional processors, the application is restarted from the most recently completed checkpoint. This takes R seconds (termed the "recovery time"), and when recovery is finished, execution begins at the same point as when the checkpoint was initiated. I seconds after recovery is complete, checkpointing begins anew. This process continues until the program completes. To illustrate the system model, see Figure 1, which depicts a segment of time between the recovery of an application and the failure of an active processor.

While the application is running, the S=N-a processors not being employed by the application are termed "spares." Their failure and subsequent repair does not affect the running of the application while the active processors are functional. It is only when an active fails that the status of the spares is important (i.e. that the number of non-failed active and spare processors numbers at least a, so that recovery may begin immediately).

To help in the explanation of the performance model, we partition the execution of a checkpointing system into three phases. They are depicted in Figure 1.

System Recovery Phase: This phase is initiated by recovery from a checkpoint. It ends either upon the successful completion of the first checkpoint following recovery (i.e. if no active processor fails in $R + I + L$ seconds), or when an active processor fails within $R + I + L$ seconds of the phase's inception.

System Up Phase: This phase is initiated by the completion of the first checkpoint after recovery. It ends when an active processor fails.

System Down Phase: This phase occurs whenever there are fewer than a functional processors. The application cannot execute during this phase. It ends as soon as a processors are functional again.

The phase transition diagram for this system is depicted in Figure 2. In this diagram, the only failures that cause transitions are failures to active processors. The failure and subsequent repair of spare processors is only important when an active processor fails. The status of the spares then determines whether the next phase is a System Recovery or System Down phase.

3 Calculating Availability

In the following sections, we introduce a discrete-parameter, finite-state Markov chain [10, 16] $\mathcal{M}$ to study the *availability* of the distributed memory checkpointing system described above. *Availability* is defined to be the fraction of time that the system spends performing *useful work*, where useful work is time spent performing computation on the application that will never be redone due to a failure. In other words, this is the time spent executing the application before a checkpoint completes. If time is spent executing the application, but an active processor fails before the next checkpoint completes, then that part of the application must be re-executed, and is therefore not useful. Likewise, recovery time, checkpoint overhead, and time spent in the System Down Phase also do not contribute to useful work.

Suppose that the running time of an application with checkpointing is $U + D$ seconds. This is the sum of time spent performing useful work (U) and time spent not performing useful work (D). The availability of the system during that time is: $A = U/(U + D)$.

Given the parameters N, a, C, L, R, I, λ and θ, we use $\mathcal{M}$ to determine the average availability A of the parallel system. A is an asymptotic value for the availability of a program whose running time approaches infinity. A can be used to approximate the availability of executing a program with a long running time, or of many executions of a program with a shorter running time.

The determination of availability is useful in the following way. The user of a parallel checkpointing system is confronted with an important question: What values of a and I minimize the expected running time of the application? Using large values of a can lower the running time of the program due to more parallelism. However, it also exposes the program to a greater risk of not being able to run due to too few functional processors. Similarly, increasing I improves the performance of the program when there are no failures, since checkpointing overhead is minimized. However, it also exposes the program to a greater recomputing penalty following a failure. Thus, we look for an optimal combination of a and I to minimize the expected running time of a program in the presence of failures and repairs.

Suppose the user can estimate the failure-free running time RT_a of his or her program when employing a active processors and no checkpointing. Moreover, suppose the user can estimate C_a, L_a and R_a. Additionally, suppose that λ and θ are known. Then the user can select any value of a and I, and compute the average availability $A_{a,I}$ of the system. The value $RT_a/A_{a,I}$ is then an estimate of the program's expected running time in the presence of failures. Thus, the user's question may be answered by choosing values of a and I that minimize $RT_a/A_{a,I}$.

In Section 6, we show nine examples of this kind of parameter selection.

4 Realism of the Model

This calculation is only useful if the underlying model has basis in reality. The model of the checkpointing system with parameters C, L, R and I mirrors most coordinated checkpointing systems that store their checkpoints to a centralized storage. Examples of these are the public-domain checkpointers MIST [4], CoCheck [22, 24], and Fail-Safe PVM [13], as well as several unnamed checkpointers that have been employed for research projects [9, 17].

A priori selection of I and a is a requirement all all the above systems. Moreover, parallel programs such as the NAS Parallel benchmarks [1], and all programs based on the MPI standard [15] have been written so that the user selects a fixed number of processors a on which to execute.

The modeling of failures and repairs as *iid* exponential random variables has less grounding in reality. Although such random variables have been used in many research papers on the performance of uniprocessor and multiprocessor checkpointing systems (see [19, 26] for citations), the studies that observe processor failures have shown that the time-to-failure and time-to-repair intervals are extremely unlikely to belong to an exponential distribution [19].

Nonetheless, there are three reasons why performance evaluations based on exponential random variables have utility. First, when failures and repairs are rare, independent events, their counts may be approximated by Poisson processes [2]. Poisson counts are equivalent to exponential interoccurrence times [10], meaning that if failures and repairs are rare (with respect to I, C, R, L, etc), their TTF distributions may be approximated by an exponential. Second, if the true failure distribution has an increasing failure rate (like the workstation failure data in [14]) rather than the constant failure rate of the exponential distribution, then the results of this paper provide a conservative (i.e. lower bound) approximation of the availability. Third, simulation results on real failure data [19] have shown in the uniprocessor case that the determination of the optimal value of I using an exponential failure rate gives a good first-order approximation of the optimal value of I determined by the simulation.

Thus, in the absence of any other information besides a mean time to failure and a mean time to recovery for processors, the availability calculation in this paper can be a reasonable indicator for selecting optimal values of a and I.

5 The Markov Chain $\mathcal{M}$

In this following sections, we define a discrete-parameter, finite-state Markov chain [10, 16] $\mathcal{M}$ to study the availability of parallel checkpointing systems. A more detailed description of $\mathcal{M}$ (with examples) is in [21].

Given values of N and a (and $S = N - a$), $\mathcal{M}$ consists of $N + S + 2$ states, partitioned into three groups based on the three phases defined above. States are entered and exited when any of the events depicted in Figure 2 occur.

System Recovery States: There are $S + 1$ System Recovery States, labeled $[R : s]$ for $0 \leq s \leq S$. Each state $[R : s]$ is entered following a failure which leaves a functional processors to perform the application and s spares.

State $[R:0]$ may also be entered from the System Down State $[D:a-1]$ when a processors become functional. Once a System Recovery State is entered, it is not exited until either $R+I+L$ seconds have passed with no active processor failure, or an active processor fails before $R+I+L$ seconds have passed. The number of functional spares during this time is immaterial. It is only at the instant that the state is exited that the number of functional spares is important. Note that if $N>a$, there are no transitions into state $[R:S]$, and it may be omitted. If N equals a, then there is one System Recovery State: $[R:0]$.

System Up States: There are $S+1$ System Up States, labeled $[U:s]$ for $0 \leq s \leq S$. Each state $[U:s]$ is entered from a System Recovery State when $R+I+L$ seconds have passed with no active processor failures. The value of s depends on the number of functional spare processors at the time the state is entered. System Up States are exited when an active processor fails. At that time, the total number of functional processors p determines the next state. If $p \geq a$, then System Recovery State $[R:p-a]$ is entered. If $p=a-1$ (no functional spares at the time of failure), then System Down State $[D:a-1]$ is entered.

System Down States: There are a System Down States, labeled $[D:p]$ for $0 \leq p < a$. State $[D:p]$ is entered whenever a failure or repair leaves the system with exactly p functional processors. No computation may be performed in a System Down State, since there are not enough processors. System Down States are exited whenever there is a processor failure or repair. If the resulting total number of functional processors p' is less than a, then the transition is to $[D:p']$. Otherwise, $p'=a$, and the transition is to $[R:0]$.

5.1 Birth-Death Markov Chain $\mathcal{S}^{s,\tau}$

In order to define the transition probabilities out of the System Recovery and System Up states, we need to have some notion of the number of functional spares at the time of the transition. For this determination, we employ a second Markov chain $\mathcal{S}^{s,\tau}$.

The solution of Markov chain $\mathcal{S}^{s,\tau}$ yields a $(s+1)\times(s+1)$ matrix $Q^{s,\tau}$ of probabilities. Suppose that there are s processors, and at a certain time, exactly i of them are functional. Entry $q_{i,j}^{s,\tau}$ of $Q^{s,\tau}$ is the probability that exactly j of those s processors are functional τ seconds later. Obviously $\sum_{j=0}^{s} q_{i,j}^{s,\tau}=1$ for each i. We use $Q^{s,\tau}$ to define the transition probabilities from the System Recovery and System Up states.

For brevity, we do not give an exact description of $\mathcal{S}^{s,\tau}$. Such a description, complete with examples, may be found in [21]. In general Markov chain theory, $\mathcal{S}^{s,\tau}$ is a *continuous-parameter, finite-state, birth-death Markov chain* [7, 16], and $Q^{s,\tau}$ is easy to calculate with standard matrix operations.

We use three values of τ in the calculations below:

- τ_1: the mean time to the first failure (*MTTF*) among a active processors with *iid* exponential failures: $\tau_1=\frac{1}{a\lambda}$.
- τ_2: the length of time during which there must be no failure in order to leave the System Recovery Phase successfully: $\tau_2=R+I+L$.
- τ_3: the conditional *MTTF*, given a failure within the first $R+I+L$ seconds in the System Recovery Phase:

$$\tau_3=\frac{1}{a\lambda}-(\tau_2)\frac{e^{-a\lambda(\tau_2)}}{1-e^{-a\lambda(\tau_2)}}=\tau_1-\tau_2\frac{e^{-a\lambda\tau_2}}{1-e^{-a\lambda\tau_2}}.$$

5.2 Transition Probabilities

In this section, we define the transition probabilities between states of $\mathcal{M}$. The sum of all probabilities emanating from a state must equal one.

System Recovery States: Transitions out of a System Recovery State $[R:i]$ are based on the time $\tau_2=R+I+L$. The probability of the event "no active processor failure during interval τ_2" is $e^{-a\lambda\tau_2}$. Thus, the probability of a transition to a System Up State is $e^{-a\lambda\tau_2}$. The specific System Up State depends on the number of functional spares at the end of the interval. This probability is given by Q^{S,τ_2}. In particular, the probability of a transition from $[R:i]$ to $[U:j]$ is $(e^{-a\lambda\tau_2})(q_{i,j}^{S,\tau_2})$.

The probability of an active processor failure during the interval τ_2 is $1-e^{-a\lambda\tau_2}$. Such a failure causes a transition either to a System Recovery State or to System Down State $[D:a-1]$. Again, the exact state depends on the number of spares at the time of the failure. We calculate the transition probabilities with Q^{S,τ_3}, based on the the conditional *MTTF* given a failure in the interval τ_2. The probability of a transition to state $[R:j]$ is $(1-e^{-a\lambda\tau_2})(q_{i,j+1}^{S,\tau_3})$. The probability of a transition to state $[D:a-1]$ is $(1-e^{-a\lambda\tau_2})(q_{i,0}^{S,\tau_3})$.

System Up States: Transitions out of a System Up State $[U:i]$ are based on τ_1, the *MTTF* of the first processor in a set of a processors[1]. This failure causes a transition either to $[D:a-1]$ (when there are no spares at the time of failure), or to $[R:j]$ (when there are $j+1$ spares). The transition probabilities are defined by Q^{S,τ_1}. The probability of a transition to state $[D:a-1]$ is $(q_{i,0}^{S,\tau_1})$. The probability of a transition to state $[R:j]$ is $(q_{i,j+1}^{S,\tau_1})$.

System Down States: Transitions out of a System Down State occur whenever there is a failure or repair. In state $[D:$

[1]Note that the "memoryless" property of *iid* exponentials means that the *MTTF* is independent of how long the processors have already been functional. Therefore, even though at the beginning of state $[U:i]$, the processors have already been functional for $R+I+L$ seconds, their *MTTF* remains τ_1.

$p]$, there are $p < a$ functional processors that are subject to failure rate λ, and $N - p$ failed processors that are subject to repair rate θ. Their cumulative distribution function is $F(t) = 1 - e^{-(p\lambda+(N-p)\theta)t}$. A property of this form of the exponential *cdf* is that whenever an event does occur, the probability that it is a repair is $(N-p)\theta/(p\lambda + (N-p)\theta)$ and that it is a failure is $p\lambda/(p\lambda+(N-p)\theta)$ [7]. These two ratios are independent of the time the event occurs. Thus, the transition probability to state $[D : p+1]$ (or to state $[R : 0]$ if $p = a-1$) is $(N-p)\theta/(p\lambda+(N-p)\theta)$, and the transition probability to state $[D : p-1]$ is $p\lambda/(p\lambda+(N-p)\theta)$.

5.3 Transition Weightings

We label each transition $\mathcal{T}$ with two weightings, $U_{\mathcal{T}}$ and $D_{\mathcal{T}}$. $U_{\mathcal{T}}$ is the average amount of useful work performed in the state which the transition is leaving, and $D_{\mathcal{T}}$ is the average amount of non-useful work. Our description is based on the states which the transitions are leaving:

System Recovery States: A transition $\mathcal{T}_{R\rightarrow R}$ from state $[R : i]$ to $[R : j]$ indicates that a failure has occurred before the first checkpoint completes. Therefore, $U_{\mathcal{T}_{R\rightarrow R}} = 0$, and $D_{\mathcal{T}_{R\rightarrow R}} = \tau_3$. The transitions from $[R : i]$ to $[D : a-1]$ have the same weightings. A transition $\mathcal{T}_{R\rightarrow U}$ from state $[R : i]$ to $[U : j]$ indicates that no failure has occurred in the first $R + I + L$ seconds. Therefore $U_{\mathcal{T}_{R\rightarrow U}} = I$, and $D_{\mathcal{T}_{R\rightarrow U}} = R + L$.

System Up States: Let $\mathcal{T}_U$ be any transition from a System Up State. The values of $U_{\mathcal{T}_U}$ and $D_{\mathcal{T}_U}$ are computed with reference to the checkpoint interval I. The probability of the event "no active processor failure in an interval I" is $e^{-a\lambda I}$ and the probability of its complement is $1-e^{-a\lambda I}$. These two events are the outcomes of a Bernoulli trial [10] for which the mean number of trials until a failure is $M = \frac{e^{-a\lambda I}}{1-e^{-a\lambda I}}$. In other words, M is the mean number of intervals I completed until the first active processor failure occurs.

Therefore, $U_{\mathcal{T}_U} = M(I-C)$. The amount of non-useful work is $D_{\mathcal{T}_U} = MC + (\tau_1 - IM)$. This includes MC for the overhead of all the successful checkpoints plus the mean duration of the last, unsuccessful interval.

System Down States: Let $\mathcal{T}_{[D:p]}$ be any transition from System Down State $[D : p]$. Obviously, $U_{\mathcal{T}_{[D:p]}} = 0$. $D_{\mathcal{T}_{[D:p]}}$ is the mean time of occupancy in state $[D : p]$: $1/(p\lambda + (N-p)\theta)$.

5.4 Calculating $A_{a,I}$

The transition probabilities of $\mathcal{M}$ may be represented in a square matrix **P**. Each state of $\mathcal{M}$ is given a row of **P** such that P_{ij} is the probability of the transition from state i to state j. Similarly, the weightings may be represented in the matrices **U** and **D**. We use the long-run properties of $\mathcal{M}$ to compute A. $\mathcal{M}$ is a recurrent chain with well-defined, asymptotic properties [11, 16]. In particular, the long-run, unconditional probability of occupancy of state i in terms of number of transitions is entry π_i in the unique solution of the matrix equation $\Pi = \Pi\mathbf{P}$ where $\sum_i \pi_i = 1, \pi_i > 0$.

Once Π is obtained, the availablility $A_{a,I}$ may be calculated as the ratio of the mean useful time per transition to the mean total time per transition:

$$A_{a,I} = \frac{\sum_{i,j} U_{ij}\pi_i P_{ij}}{\sum_{i,j}(U_{ij} + D_{ij})\pi_i P_{ij}}.$$

$A_{a,I}$ may then be used to obtain optimal values of a and I as detailed in Section 3. Greater detail on this process, complete with example calculations, is available in [21]. We have encapsulated the process in the form of Matlab scripts, which are available on the web at **http://www.cs.utk.edu/˜plank/plank/avail/**.

6 Case Studies

In the following sections, we detail nine case studies of parameter selection in checkpointing systems. We selected three long-running parallel applications from the NASA Ames NAS Parallel Benchmarks [1]. These are the BT (block tridiagonal solver), LU (linear equation solver), and EP (random number generator) applications.

Name	r	z
BT	$(\text{Matrix Size})^3$	$(\text{Matrix Size})^2$
LU	$(\text{Matrix Size})^3$	$(\text{Matrix Size})^2$
EP	$\frac{\#\text{ random numbers}}{2^{26}}$	1 (constant)

Table 1. Basic application data

For the purposes of parameter selection, RT_a, C_a, L_a, and R_a must be functions of a. Amdahl's law has been shown to characterize the NAS Benchmarks very well according to number of processors a and a performance metric r based on the input size [23]. Thus, we calculate RT_a using a slightly enhanced statement of Amdahl's law:

$$RT_a = \frac{b_1 r}{a} + \frac{b_2}{a} + b_3 r + b_4.$$

We assume that C_a, L_a, and R_a are proportional to the total global checkpoint size CS_a, and that the global checkpoint is composed of global data partitioned among all the processors (such as the matrix in BT and LU), and replicated/private data for each processor. Thus, CS_a is a function of a and a size metric z:

$$CS_a = c_1 za + c_2 a + c_3 z + c_4.$$

The first two terms are for the replicated/private data and the second two are for the shared data. The BT, LU and EP applications have clear definitions of r, and z which are included in Table 1.

For each application, we used timing and checkpoint size data from a performance study of the NAS benchmarks on a network of Sparc Ultra workstations [3]. From these, we used Matlab's regression tools to calculate the coefficients b_i and c_i. These are listed in Table 2.

Coef.	BT	LU	EP
b_1	1.551e-02	9.400e-03	1.059e+02
b_2	-3.788e+01	-3.441e+01	1.980e+02
b_3	3.643e-04	1.560e-04	5.767e+00
b_4	-6.425e-01	-6.989e+00	-4.122e+01
c_1	1.875e-04	5.650e-04	0
c_2	1.952e+00	4.594e-01	1.700e+00
c_3	8.345e-02	1.882e-02	0
c_4	-2.790e+01	-1.838e+01	0

Table 2. The coefficients b_i and c_i.

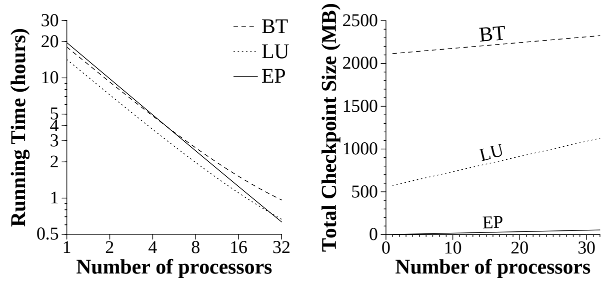
We constructed three processing environments for our case studies. All three are based on published checkpointing and failure/repair data. We assume that all are composed of 32 processors and exhibit the same processing capacity as the Ultra Sparc network in [3]. However, they differ in failure rate, repair rate and checkpointing performance. The environments are detailed in Table 3 and below.

HIGH is a high-performance environment characterized by low failure rates and excellent checkpointing performance. The failure and repair rates come from the **PRINCETON** data set in [19], where failures are infrequent, and the checkpointing performance data comes from CLIP [5], a checkpointer for the Intel Paragon, which has an extremely fast file system. In **HIGH**, C, L and R are equal because CLIP cannot implement the copy-on-write optimization.

MEDIUM is a medium-performance workstation network such as the Ultra Sparc network from [3]. We use workstation failure data from a study on workstation failures on the Internet [14], and checkpointing performance data from a PVM checkpointer on a similar workstation network [17].

Environment	λ	θ	$\frac{C_a}{CS_a}$	$\frac{L_a, R_a}{CS_a}$
HIGH	$\frac{1}{32.7 \text{ days}}$	$\frac{1}{1.30 \text{ days}}$	$\frac{1 \text{ sec}}{24.8 \text{ MB}}$	$\frac{1 \text{ sec}}{24.8 \text{ MB}}$
MIDDLE	$\frac{1}{13.0 \text{ days}}$	$\frac{1}{2.02 \text{ days}}$	$\frac{1 \text{ sec}}{2.04 \text{ MB}}$	$\frac{1 \text{ sec}}{0.120 \text{ MB}}$
LOW	$\frac{1}{70 \text{ min}}$	$\frac{1}{75 \text{ min}}$	$\frac{1 \text{ sec}}{1.00 \text{ MB}}$	$\frac{1 \text{ sec}}{0.200 \text{ MB}}$

Table 3. Failure, repair and checkpointing data for the three processing environments.



(a) (b)

Figure 3. (a): Running time (RT_a) of the applications as a function of the number of processors. (b): Checkpoint size (CS_a) as a function of the number of processors.

Finally, **LOW** is based on an idle-workstation environment such as the one supported by CosMiC [6], where workstations are available for computations only when they are not in use by their owners. Failure and repair data was obtained by the authors of [6], and the checkpointing performance data was gleaned from performance results of CosMiC's transparent checkpointer **libckp** [27]. It is assumed that the copy-on-write optimization yields an 80% improvement in checkpoint overhead [18]. The failure rate of **LOW** is extremely high, which is typical of these environments.

For each application, we selected a problem size that causes the computation to run between 14 and 20 hours on a single workstation with no checkpointing or failures. These are matrix sizes of 160 and 175 for BT and LU respectively, and 2^{35} random numbers for EP. We then calculate values of RT_a for $1 \leq a \leq 32$. These are plotted in Figure 3(a) (using a log-log plot, meaning perfect speedup is a straight line). As displayed by this graph, EP shows the best scalability as a increases. BT and LU scale in a roughly equal manner. In these instances, BT takes a little longer than LU. We assume that the programming substrate recognizes processor failures (as does PVM).

The total checkpoint size CS_a for each application and value of a is calculated using the data in Table 2, and then plotted in Figure 3(b). BT has very large checkpoints (over 2 GB). The checkpoints in LU are smaller, but grow faster with a. EP's checkpoints are very small (1.7 MB per processor).

7 Experiment

For each value of a from 1 to 32, we determine the value I_{opt} of I that minimizes $A_{a,I}$. This is done using Matlab, with a straightforward parameter sweep and iterative refinement of values for I_{opt}, making sure that $I_{opt} \geq L_a$. We

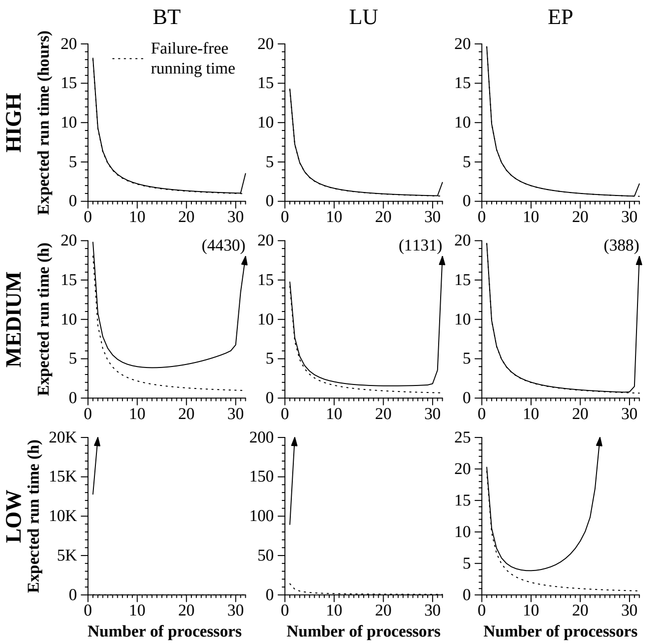


Figure 4. Optimal expected running times of all case studies in the presence of failures as a function of a.

then calculate $RT_a/A_{a,I_{opt}}$, which is the optimal expected running time of the application in the presence of failures. These are plotted using the solid lines in Figure 4. Arrows indicate when these values go beyond the extent of the Y-axes. In the **MEDIUM** cases, the values for $a = 32$ are noted. Dotted lines plot RT_a to compare $RT_a/A_{a,I_{opt}}$ to the failure-free running times. The optimal values of a and I are shown in Table 4.

The first thing to note about Figure 4 and Table 4 is that the optimal value of a varies widely over all cases. In the **HIGH** processing environment, the optimal a in all cases is 31, meaning that it is best to always have a spare processor available in case of failure. If no spare is available, then the application spends a significant amount of idle time waiting for failed processors to be repaired.

In the **MEDIUM** processing environment, the optimal a ranges from 13 to 29. The optimal a is smaller than in **HIGH** because of more frequent failures and much larger latencies, overheads, and recovery times. Of the applications, EP has the highest value of a_{opt} and the best running times. This is mainly because of its smaller checkpoints.

In the **LOW** processing environment, BT and LU have poor expected running times. The optimal values of a are one, and the expected running times are 12791 hours (533 days) and 89 hours (3.7 days) respectively. The reason for these large running times is that $R + L$ is 5.9 hours for BT and 1.6 hours for LU. Both of these are larger than the single processor *MTTF* of 1.2 hours. Thus, even when I equals L, most of time of these applications is spent executing code that will not be checkpointed. The EP application has much smaller checkpoints (its largest $R + L$ value is 0.15 hours), and therefore spends more time performing useful work. It achieves an acceptable optimal running time of 3.85 hours with $a_{opt} = 10$.

As shown by the dotted lines in Figure 4 and the rightmost column of Table 4, checkpointing and failures add very little overhead in the **HIGH** processing environment. In the **MEDIUM** environment, the smaller checkpoints of EP lead to good performance in the presence of failures, while LU and BT perform less well. In the **LOW** environment, BT is basically unrunnable. Given the nature of the environment and the size of the application, LU's performance is barely passable, and EP's is decent. It is worth noting that although checkpointing and process migration environments have been built for idle workstation environments [4, 6, 22], this is the first piece of work that attempts to characterize the performance of large parallel applications on such environments.

8 Related Work

As stated above, there has been much work on checkpointing performance prediction in the presence of failures for uniprocessor and multi-processor (again, see [19, 26] for citations). However, this is the first paper that considers the use of spare processors to take the place of failed active processors. Of note is the work of Wong and Franklin [28], which assumes that the program may reconfigure itself during execution to use a variable number of processors. However, as stated in Section 4, the majority of parallel programs and checkpointing environments do not allow reconfiguration (e.g. [1, 4, 9, 13, 17, 20, 22, 24]).

9 Conclusion

We have presented a method for estimating the average running time of a long-running parallel program, enabled with coordinated checkpointing, in the presence of failures and repairs. This method allows a user to perform an optimal selection of the checkpointing interval and number of active processors. We have shown case studies of three applications from the NAS parallel benchmarks executing on three different but realistic parallel processing environments. Our results show that the optimal number of active processors can vary widely, and that the selection of the number of active processors can have a significant effect on the average running time. We expect this method to be useful for those executing long-running programs on parallel processing environments that are prone to failure.

Application	Processing Environment	a_{opt}	I_{opt} (hours)	$A_{a_{opt},I_{opt}}$	$RT_{a_{opt}}$ (hours)	$RT_{a_{opt}}/A_{a_{opt},I_{opt}}$ (hours)	Overhead of failures and checkpointing
BT	**HIGH**	31	1.16	0.947	0.98	1.04	6.1%
BT	**MEDIUM**	13	5.07	0.458	1.77	3.87	119%
BT	**LOW**	1	2.94	0.00141	18.1	12791	70756%
LU	**HIGH**	31	0.80	0.961	0.68	0.71	4.4%
LU	**MEDIUM**	22	2.19	0.557	0.87	1.56	80%
LU	**LOW**	1	0.80	0.159	14.2	89.4	529%
EP	**HIGH**	31	0.17	0.986	0.65	0.66	1.5%
EP	**MEDIUM**	29	0.33	0.923	0.70	0.75	7.1%
EP	**LOW**	10	0.033	0.515	1.98	3.85	94%

Table 4. Optimal a and I for all tests.

There are three directions in which to extend this work. First, we can attempt to illuminate the method with stochastic simulation based on *iid* exponential failure and repair intervals. This can both validate the model, as in [12], and point to interesting areas of research. Second, we can explore the impact of the assumption of *iid* exponential failures and repairs, by performing simulation based on real failure data, as in [19]. Third, we can attempt to study a wider variety of checkpointing systems, such as two-level [25] and diskless checkpointing systems [20].

10 Acknowledgements

This material is based upon work supported by the National Science Foundation under grants CDA-9529459 and CCR-9703390. The authors thank Emerald Chung, Darrell Long and Richard Golding, for providing failure and repair data, Kim Buckner and Erich Strohmaier for help in obtaining performance data, and Henri Casanova for discussions of the model.

References

[1] D. Bailey *et al*. The NAS parallel benchmarks 2.0. Tech. Rep. NAS-95-20, NASA Ames Research Center, Dec. 1995.

[2] A. D. Barbour, L. Holst, and S. Janson. *Poisson Approximation*. Clarendon Press (Oxford University), Oxford, UK, 1992.

[3] K. Buckner. Timings and memory usage for the NAS parallel benchmarks on a network of Ultra workstations. Tech. Rep. CS-98-408, Univ. of Tenn., 1998.

[4] J. Casas *et al*. MIST: PVM with transparent migration and checkpointing. In *3rd Annual PVM Users' Group Meeting*, Pittsburgh, PA, May 1995.

[5] Y. Chen *et al*. CLIP: A checkpointing tool for message-passing parallel programs. In *SC97: High Perf. Networking and Comp.*, Nov. 1997.

[6] P. E. Chung *et al*, Checkpointing in CosMiC: a user-level process migration environment. In *Pac. Rim Int. Symp. on Fault-Tol. Systems*, Dec. 1997.

[7] D. R. Cox and H. D. Miller. *The Theory of Stochastic Processes*. Chapman and Hall Ltd., London, UK, 1972.

[8] E. N. Elnozahy, D. B. Johnson, and Y. M. Wang. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, to appear.

[9] E. N. Elnozahy, D. B. Johnson, and W. Zwaenepoel. The performance of consistent checkpointing. In *11th Symp. on Rel. Dist. Sys.*, pp. 39–47, Oct. 1992.

[10] W. Feller. *An Introduction to Probability Theory and Its Applications (Third Edition)*. John Wiley & Sons, Inc., NY, 1968.

[11] J. G. Kemeny and J. L. Snell. *Finite Markov Chains*. Van Nostrand, Princeton, NJ, 1960. Republished by Springer-Verlag, NY, 1976.

[12] G. P. Kavanaugh and W. H. Sanders. Perf. analysis of two time-based coordinated checkpointing protocols. In *Pac. Rim Int. Symp. on Fault-Tol. Sys.*, 1997.

[13] J. León, *et al*. Fail-safe PVM: A portable package for distributed programming with transparent recovery. Tech. Rep. CS-93-124, Carn. Mel. Univ., 1993.

[14] D. Long, A. Muir, and R. Golding. A longitudinal survey of internet host reliability. In *14th Symp. on Rel. Dist. Sys.*, pp. 2–9, Sep. 1995.

[15] Message Passing Interface Forum. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications*, 8(3/4), 1994.

[16] E. Parzen. *Stochastic Processes*. Holden-Day, San Francisco, CA, 1962.

[17] J. S. Plank. Improving the performance of coordinated checkpointers on networks of workstations using RAID techniques. In *15th Symposium on Reliable Distributed Systems*, pp. 76–85, October 1996.

[18] J. S. Plank, M. Beck, G. Kingsley, and K. Li. **Libckpt**: Transparent checkpointing under Unix. In *Usenix Winter Tech. Conf.*, pp. 213–223, Jan. 1995.

[19] J. S. Plank and W. R. Elwasif. Experimental assessment of workstation failures and their impact on checkpointing systems. In *28th Int. Symp. on Fault-Tol. Comp.*, pp. 48–57, June 1998.

[20] J. S. Plank, K. Li, and M. A. Puening. Diskless checkpointing. *IEEE Transactions on Parallel and Distributed Systems*, 9(10):972–986, October 1998.

[21] J. S. Plank and M. G. Thomason. The average availability of multiprocessor checkpointing systems. Tec. Rep. CS-98-403, Univ. of Tenn., Nov. 1998.

[22] J. Pruyne and M. Livny. Managing checkpoints for parallel programs. In *Workshop on Job Scheduling Strategies for Parallel Processing (IPPS '96)*, 1996.

[23] H. D. Simon and E. Strohmaier. Amdahl's law and the statistical content of the NAS Parallel Benchmarks. *Supercomputer*, 11(4):75–88, 1995.

[24] G. Stellner. CoCheck: Checkpointing and process migration for MPI. In *10th International Parallel Processing Symposium*, pp. 526–531. Apr. 1996.

[25] N. H. Vaidya. A case for two-level distributed recovery schemes. In *ACM SIGMETRICS Conf. on Meas. and Modeling of Computer Sys.*, May 1995.

[26] N. H. Vaidya. Impact of checkpoint latency on overhead ratio of a checkpointing scheme. *IEEE Transactions on Computers*, 46(8):942–947, August 1997.

[27] Y-M. Wang *et al*. Checkpointing and its applications. In *25th Int. Symp. on Fault-Tol. Comp.*, pp. 22–31, June 1995.

[28] K. F. Wong and M. Franklin. Checkpointing in distributed systems. *Journal of Parallel & Distributed Systems*, 35(1):67–75, May 1996.