Processor Allocation and Checkpoint Interval Selection in Cluster Computing Systems

James S. Plank Michael G. Thomason*

Appearing in *Journal of Parallel and Distributed Computing* in 2001. A condensed version of this paper appeared in the *29th Internation Symposium on Fault-Tolerant Computing*, reference [28].

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 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.

Keywords: Checkpointing, performance prediction, parameter selection, parallel computation, Markov chain, exponential failure and repair distributions.

1 Introduction

To tolerate failures in cluster and parallel computing systems, parallel applications typically instrument themselves with the ability to checkpoint their computation state to stable storage. When one or more processors fail, the application may be restarted from the most recent checkpoint, thereby reducing the amount of recomputation that must be performed.

To date, most checkpointing systems for long-running distributed memory computations (e.g. [1, 5, 6, 18, 26, 29, 32]) are based on *coordinated checkpointing* [11]. At each checkpoint, the global state of all the processors is defined and stored to a highly available stable storage. If any processor fails, 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. If no replacement processor is available, then the application is halted until one becomes available.

When a user must execute a long-running application on a cluster computing system, he or she is typically faced with two important decisions: How many processors should the application use, and how frequently should they checkpoint? Most programs for cluster systems require the user to choose the number of processors before the computation begins,

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

and once underway, this number may not change. On a system with no checkpointing, processor allocation is simple: the application should allocate as many processors as are available for the most parallelism and the shortest running time. However, when a system is enabled with checkpointing, the processor allocation is less clear. If all processors are used for the application and one fails, then the application cannot continue until that processor is repaired and the whole system may recover. If some subset of the processors is used for the application, the application may take longer to complete in the absence of failures, but if one or more processors fail, then there may be *spare* processors standing by to be immediate replacements. 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.

Selecting the frequency of checkpointing is also important. If the application checkpoints too frequently, then its performance will be penalized by absorbing too much checkpointing overhead. On the other hand, if it checkpoints too infrequently, then it may be penalized by too much recomputation overhead following a failure. Deciding upon the optimal checkpointing frequency is called the *optimal checkpoint interval* problem, and is a well-studied research area. For uniprocessor systems, selection of such an interval is for the most part a solved problem [25, 34]. There has been important research in parallel systems [16, 33, 36], but the results are less unified. No previous work has addressed the issue of processor availability following a failure in cluster computing systems.

In this paper, we model the performance of coordinated checkpointing systems where the number of processors dedicated to the application (denoted a for "active") and the checkpoint interval (denoted I) are selected by the user before running the program. We use the model to determine the *availability* of the parallel system over the long-term to do useful computation in the presence of random failures, and we show how availability can be used to select values of a and I that minimize the expected execution time of a long-running program in the presence of failures. We then give examples of parameter selection using standard 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 of processor allocation), 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 under repair*, and in this model all failed processors are under repair simultaneously. 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 $\theta > 0$. 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 starts running as soon as there are a functional processors. If, after I

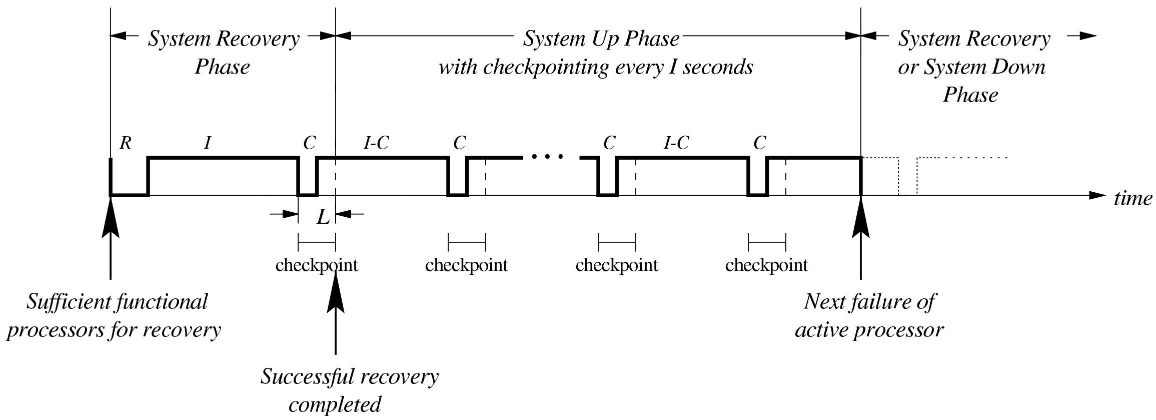

Figure 1: The sequence of time between the recovery of an application from a failure, and the failure of an active processor.

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 [34]. I must be greater than or equal to L so that the system never attempts to store multiple checkpoints simultaneously.

Checkpoints are initiated every I seconds when there are no failures among the a active processors. When an active processor does fail, the application is halted and a replacement processor is sought. If there are no replacements, the application must stand idle until there are again a functional processors. As soon as there are a active processors again, the application is restarted from the most recently completed checkpoint. This takes R seconds (termed the "recovery time"). If no active processor fails during these R seconds, execution resumes at the same point as when the checkpoint was initiated and I seconds later, checkpointing begins anew. This process continues until the program completes. See Figure 1, which depicts a segment of time between the recovery of an application and the next 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 do not affect the running of the application while all a active processors are functional; but when an active processor fails, the status of the spares is important because the total number of non-failed processors must be at least a in order to attempt a recovery.

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

- **System Recovery Phase**: This phase is initiated by recovery from a checkpoint. The phase ends either successfully (upon the completion of the first checkpoint when none of the a active processors fails in a span of $R + I + L$ seconds) or unsuccessfully (when an active processor fails within $R + I + L$ seconds and no spares are functional

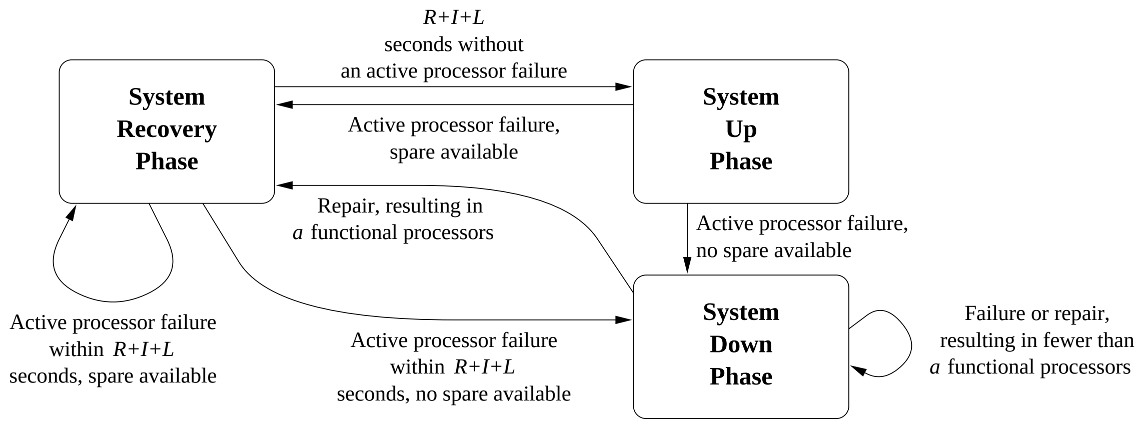

Figure 2: Phase transition diagram.

at that time). If an active processor fails and at least one spare is functional, the system prolongs this phase by changing one spare to active as a replacement for the failed processor and trying again to recover.

- **System Up Phase**: This phase is initiated by the completion of the first checkpoint during recovery (i.e. at the end of a successful 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 of active processors. The failure and subsequent repair of spare processors impacts the parallel application when it is running only if an active processor fails in System Up or System Recovery Phases; then the status of the spares determines whether the next phase is System Recovery or System Down.

3 Calculating Availability

In the following sections, we introduce a discrete-parameter, finite-state Markov chain [14, 22] $\mathcal{M}$ 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 R, checkpoint overhead C, and time spent in the System Down Phase 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 ("uptime U") and time spent not performing useful work ("downtime D"). The availability of the system during the running time is the fraction

$$\frac{U}{U+D}. \tag{1}$$

Given the parameters N, a, C, L, R, I, λ and θ, we use $\mathcal{M}$ to determine the *availability A of the parallel system as an average over the long-term*. 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.

4 Utility

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 a value of a and I for $1 \leq a \leq N$ and $I \geq L$, 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 7, we show nine examples of this kind of parameter selection.

5 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 [5], CoCheck [29, 31, 32], and Fail-Safe PVM [18], as well as several other checkpointers that have been employed for research projects [1, 12, 13, 23, 27].

A priori selection of the checkpoint interval is inherent in all the above systems. Selection of the parameter a is is also a requirement of all these systems, since they cannot perform reconfiguration during execution. Most parallel programs, for example the well-known ScaLAPACK suite [7], the NAS Parallel benchmarks [2], and all programs based on the MPI standard [21] 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 (for example [15, 16, 34, 36, 37]), the few studies that observe the nature of processor failures have shown that the time-to-failure and time-to-repair intervals are extremely unlikely to belong to an exponential distribution [10, 20, 25].

Nonetheless, there are three reasons why performance evaluations based on exponential random variables have utility. First, when failures and repairs are rare and (stochastically) independent, their counts may be approximated by Poisson processes [3]. Poisson counts are equivalent to exponential interoccurrence times [14], meaning that 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 [20]) rather than the constant failure rate of the exponential distribution, then the results of this paper provide a conservative (lower bound) approximation of the availability. Third, simulation results on real failure data [25] 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. Although published values exist for many of the parameters for this model, it must be stated that they can be difficult to obtain for individual clusters and applications.

6 The Markov Chain $\mathcal{M}$

In this section, we define a *finite-state, discrete-parameter Markov chain* [14, 22] $\mathcal{M}$ to study the availability of parallel checkpointing systems.

6.1 State Definition

Given values of N and a (and $S = N - a$), $\mathcal{M}$ has $N + S + 1$ states partitioned into three groups for the three phases defined above. States are entered and exited when any of the events depicted in Figure 2 occur.

- **System Recovery States**: If $a < N$, there are S System Recovery states, labeled $[R : s]$ for $0 \leq s \leq S - 1$. If $a = N$, the single System Recovery state is $[R : 0]$. The state $[R : s]$ is entered from System Up or System Recovery upon the event "there is failure of an active processor and $s+1$ spares are functional." Upon occurrence of this event, the system replaces the failed processor by one functional spare, thereby having a functional processors to perform the application and s functional spares at the start of the recovery. State $[R : 0]$ is also entered from the System Down state $[D : a - 1]$ upon the event "repair of a failed processor" because this event marks the time when the system has exactly a functional processors again. When a System Recovery state is entered, it is not exited until either $R + I + L$ seconds have passed with no active processor failure (i.e. a successful recovery) or an active processor fails before $R + I + L$ seconds have passed. As long as no active processor fails, variation in the number of functional spares does not impact the progress of the recovery. The number of functional spares is important when a System Recovery state is exited.

- **System Up States**: There are $S+1$ System Up states, labeled $[U:s]$ for $0 \leq s \leq S$. A state $[U:s]$ is entered from a System Recovery state after a span of $R+I+L$ seconds has passed with no active processor failures. The value of s is the number of functional spare processors at the time the state is entered. As long as no active processor fails, variation in the number of functional spares does not impact System Up, but when an active processor fails, the next state is determined by the total number of functional processors p in the parallel system. If $p \geq a$, then System Recovery state $[R:p-a]$ is entered; if $p = a-1$ (i.e. there are no functional spares), then System Down state $[D:a-1]$ is entered.

- **System Down States**: There are a System Down states, labeled $[D:p]$ for $0 \leq p \leq a-1$. State $[D:p]$ is entered whenever a failure or repair leaves exactly p processors functional out of the N processors in the system. No execution of the application can take place in System Down states, since there are not enough functional 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 $[D:p']$ and System Down continues; otherwise, $p' = a$ and the transition is to $[R:0]$ to attempt a recovery.

6.2 Birth-Death Markov Chain $\mathcal{S}^\tau$

The transition probabilities out of the System Up states, and some of those out of System Recovery states, are based on the event "an active processor fails and j spares are functional." To define these transition probabilities, we employ a second Markov chain $\mathcal{S}^\tau$ to describe the number of functional spares at time τ. For brevity, numerical details of $\mathcal{S}^\tau$ are in the Appendix.

The number of functional spares is always between 0 and S. $\mathcal{S}^\tau$ is a *finite-state, continuous-parameter, birth-death Markov chain* [9, 22] with $S+1$ states. For indexing, the states are numbered from 1 (the state for no spares failed and all S functional) through $S+1$ (the state for S failed spares and none functional). The $(S+1) \times (S+1)$ matrix $\mathbf{Q}^{S,\tau} = [q_{k\ell}^{S,\tau}]$ is found by standard matrix operations such that $q_{S-i+1,S-j+1}^{S,\tau}$ is the conditional probability that there are j functional spares τ seconds after starting with i functional spares. Thus, $\sum_{\ell=1}^{S+1} q_{k\ell}^{S,\tau} = 1$ for each k and for all $\tau \geq 0$.

We use $\mathbf{Q}^{S,\tau}$ in three evaluations which are referred to in the next section:

- Let $\varrho = R+I+L$ denote the length of time during which there must be no failure of an active processor in order to exit the System Recovery Phase successfully. The entries in each row of $\mathbf{Q}^{S,\varrho}$ are the conditional probabilities for S through 0 functional spares, given an initial number of functional spares and the passage of $\tau = \varrho$ seconds.

- If there is a failure of an active processor during the first ϱ seconds in a System Recovery state, the TTF random variable τ is in the interval $[0,\varrho)$ and its probability density is the exponential $a\lambda e^{-a\lambda\tau}$ renormalized for $0 \leq \tau < \varrho$

(i.e. its conditional density function is $a\lambda e^{-a\lambda\tau}/(1-e^{-a\lambda\varrho})$ for $0 \le \tau < \varrho$ and 0 for $\tau \ge \varrho$). As described in the Appendix, the matrix $[q_{k\ell}^{\text{Rec},S}]$ of likelihoods is the integral of $\mathbf{Q}^{S,\tau}$ times this density function for τ.

- Measured from the start of the System Up Phase, the TTF random variable τ is in the interval [0,∞) and its density function is $a\lambda e^{-a\lambda\tau}$. Computation of the matrix $[q_{k\ell}^{\text{Up},S}]$ for this distribution of τ is also described in the Appendix.

6.3 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 defined by reference to the time span $\varrho = R + I + L$. The probability of the event "no active processor failure during interval [0,ϱ)" is $e^{-a\lambda\varrho}$, and this is the probability of a transition to a System Up state. The specific System Up state depends on the number of functional spares after ϱ seconds, the probabilities of which are given by $\mathbf{Q}^{S,\varrho}$. In particular, the probability of a transition from $[R : i]$ to $[U : j]$ is $(e^{-a\lambda\varrho})(q_{S-i+1,S-j+1}^{S,\varrho})$.

  The probability of an active processor failure during the interval [0,ϱ) is $1-e^{-a\lambda\varrho}$. 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. We compute likelihoods based on the probability density $a\lambda e^{-a\lambda\tau}$ conditioned on $0 \le \tau < \varrho$. The matrix $[q_{k\ell}^{\text{Rec},S}]$ of these likelihoods is described in the Appendix. The probability of a transition from $[R : i]$ to $[R : j]$ is $(1-e^{-a\lambda\varrho})(q_{S-i+1,S-j}^{\text{Rec},S})$ and from $[R : i]$ to $[D : a-1]$ is $(1-e^{-a\lambda\varrho})(q_{S-i+1,S+1}^{\text{Rec},S})$.

- **System Up States**: Transitions out of a System Up state $[U : i]$ are also based on the TTF random variable for a active processors, but in this case τ is an exponentially distributed random variable without an upper limit[1]. A failure causes a transition either to $[D : a-1]$ (if there are no functional spares at the time of failure), or to $[R : s]$ (if there are $s+1$ functional spares, one of which immediately becomes an active processor, leaving s functional spares). The transition probabilities are defined by matrix $[q_{k\ell}^{\text{Up},S}]$ computed for likelihoods as described in the Appendix. The probability of a transition to state $[D : a-1]$ is $q_{S-i+1,S+1}^{\text{Up},S}$. The probability of a transition to state $[R : s]$ is $q_{S-i+1,S-s}^{\text{Up},S}$.

- **System Down States**: Transitions out of a System Down state occur whenever there is a failure or repair. In state $[D : p]$, 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 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 $p\lambda/(p\lambda+(N-p)\theta)$ [9]. These two ratios are independent of the

[1] Note that the "memoryless" property of *iid* exponentials means that the τ does not depend on 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 $1/(a\lambda)$.

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 $p\lambda/(p\lambda + (N-p)\theta)$.

6.4 Transition Weightings

We label each transition $\mathcal{T}$ with two weightings, $U_{\mathcal{T}}$ (for "uptime") and $D_{\mathcal{T}}$ (for "downtime"). $U_{\mathcal{T}}$ is the average amount of useful time devoted to the application while in the state which the transition is leaving, and $D_{\mathcal{T}}$ is the average amount of non-useful time. Our description is based on the states which the transitions are leaving:

- **System Recovery States**: A transition $\mathcal{T}_{R\to R}$ from state $[R : i]$ to $[R : j]$ indicates that a failure has occurred before the first checkpoint completes. Therefore, $U_{\mathcal{T}_{R\to R}} = 0$ and

  $$D_{\mathcal{T}_{R\to R}} = \frac{1}{a\lambda} - \varrho\frac{e^{-a\lambda\varrho}}{1-e^{-a\lambda\varrho}} \tag{2}$$

  is the MTTF conditioned on failure of an active processor before ϱ seconds. The transitions from $[R : i]$ to $[D : a-1]$ have the same weightings. A transition $\mathcal{T}_{R\to U}$ from state $[R : i]$ to $[U : j]$ indicates that no failure has occurred in a span of ϱ seconds while in state $[R : i]$; therefore, $U_{\mathcal{T}_{R\to U}} = I$ and $D_{\mathcal{T}_{R\to U}} = R + L$.

- **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 [14] for which the mean number of trials until a failure is

  $$M = \frac{e^{-a\lambda I}}{1-e^{-a\lambda I}}. \tag{3}$$

  In other words, M is the mean number of intervals of length I completed until the first active processor failure occurs.

  Therefore, $U_{\mathcal{T}_U} = M(I - C)$. The non-useful time spent is $D_{\mathcal{T}_U} = MC + (1/(a\lambda) - IM)$ where the MTTF is $1/(a\lambda)$. 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 per visit to state $[D : p]$, namely, $1/(p\lambda + (N-p)\theta)$.

6.5 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 [17, 22]. 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 $\pi_i > 0$ for each state i and $\sum_i \pi_i = 1$.

The probabilities in row vector Π are the limiting relative frequencies of the states with respect to the number of transitions and are independent of the starting state of the Markov chain. If $\mathcal{M}$ is taken through n transitions randomized according to the transition probabilities, and if n_i is the number of occurrences of state i during those transitions, then

$$\pi_i = \lim_{n\to\infty} \frac{n_i}{n+1}. \tag{4}$$

Since each visit to state i is followed by probabilistic selection of an exit transition, the limiting relative frequency of occurrence of transition $i \to j$ is the joint probability $\pi_i P_{ij}$. Associated with transition $i \to j$ are two random variables for useful time and non-useful time, the conditional mean values of which are the weightings U_{ij} and D_{ij} respectively. For a long-running task, $U_{ij}\pi_i P_{ij}$ is the expected contribution of useful time due to the relative frequency of transition $i \to j$; similarly, $D_{ij}\pi_i P_{ij}$ is that transition's expected contribution of non-useful time. The availability $A_{a,I}$ is the ratio of the *mean useful time per transition* to the *mean total time per transition*:

$$A_{a,I} = \frac{\sum_{i,j} U_{ij}\pi_i P_{ij}}{\sum_{i,j}(U_{ij}+D_{ij})\pi_i P_{ij}}. \tag{5}$$

6.6 Putting it all together

Therefore, the calculation of availability goes as follows. Given values of N, λ and θ, the user selects values of a and I. From these, it is assumed that the user may calculate or estimate C_a, R_a and L_a. From these, the user builds $\mathcal{M}$, in the form of matrices **P**, **U** and **D**. From **P**, the probability vector Π is calculated, and finally $A_{a,I}$ is calculated from **P**, **U**, **D** and Π.

We have encapsulated this process in the form of Matlab scripts, which are available on the web at **http://www.cs.utk.-edu/~plank/plank/avail/**.[2]

7 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 [2]. These are the BT (block tridiagonal solver), LU (linear equation solver), and EP (random number generator) applications.

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 [30]. Thus, we calculate RT_a using a slightly enhanced statement of Amdahl's law:

$$RT_a = \frac{b_1 r}{a} + \frac{b_2}{a} + b_3 r + b_4. \tag{6}$$

[2] If desired, these can be put into an appendix, or if JPDC has an archival web site, they can be placed there.

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

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:

$$CS_a = c_1 z a + c_2 a + c_3 z + c_4. \tag{7}$$

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 cluster of Sparc Ultra workstations [4]. 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 cluster in [4]. 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 cluster characterized by low failure rates and excellent checkpointing performance. The failure and repair rates come from the **PRINCETON** data set in [25], where failures are infrequent, and the checkpointing performance data comes from CLIP [6], 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 cluster such as the Ultra Sparc cluster from [4]. We use workstation failure data from a study on workstation failures on the Internet [20], and checkpointing performance data from a PVM checkpointer on a similar workstation cluster [23].

Environment	λ	θ	C_a/CS_a	L_a/CS_a, R_a/CS_a
HIGH	1/32.7 days	1/1.30 days	24.8 MB/sec	24.8 MB/sec
MIDDLE	1/13.0 days	1/2.02 days	2.04 MB/sec	0.120 MB/sec
LOW	1/70 min	1/75 min	1.00 MB/sec	0.200 MB/sec

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

Finally, **LOW** is based on an idle-workstation environment such as the ones supported by Condor [19] and CosMiC [8], 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 [8], and the checkpointing performance data was gleaned from performance results of CosMiC's transparent checkpointer **libckp** [35]. It is assumed that the copy-on-write optimization yields an 80 percent improvement in checkpoint overhead [24].

The failure rate of **LOW** is extremely high, which is typical of these environments, and as the data later show, they are not particularly conducive to this kind of parallel computing. This will be discussed later.

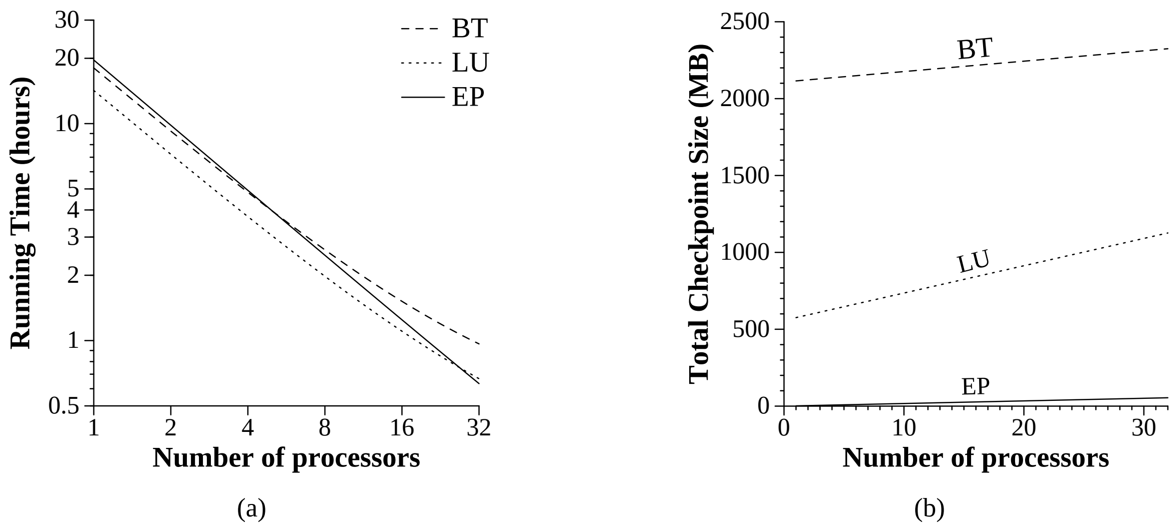

Figure 3: (a): Running time (RT_a) of the applications as a function of the number of processors. (b): Checkpoint size (CS_a) as a function of the number of processors.

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

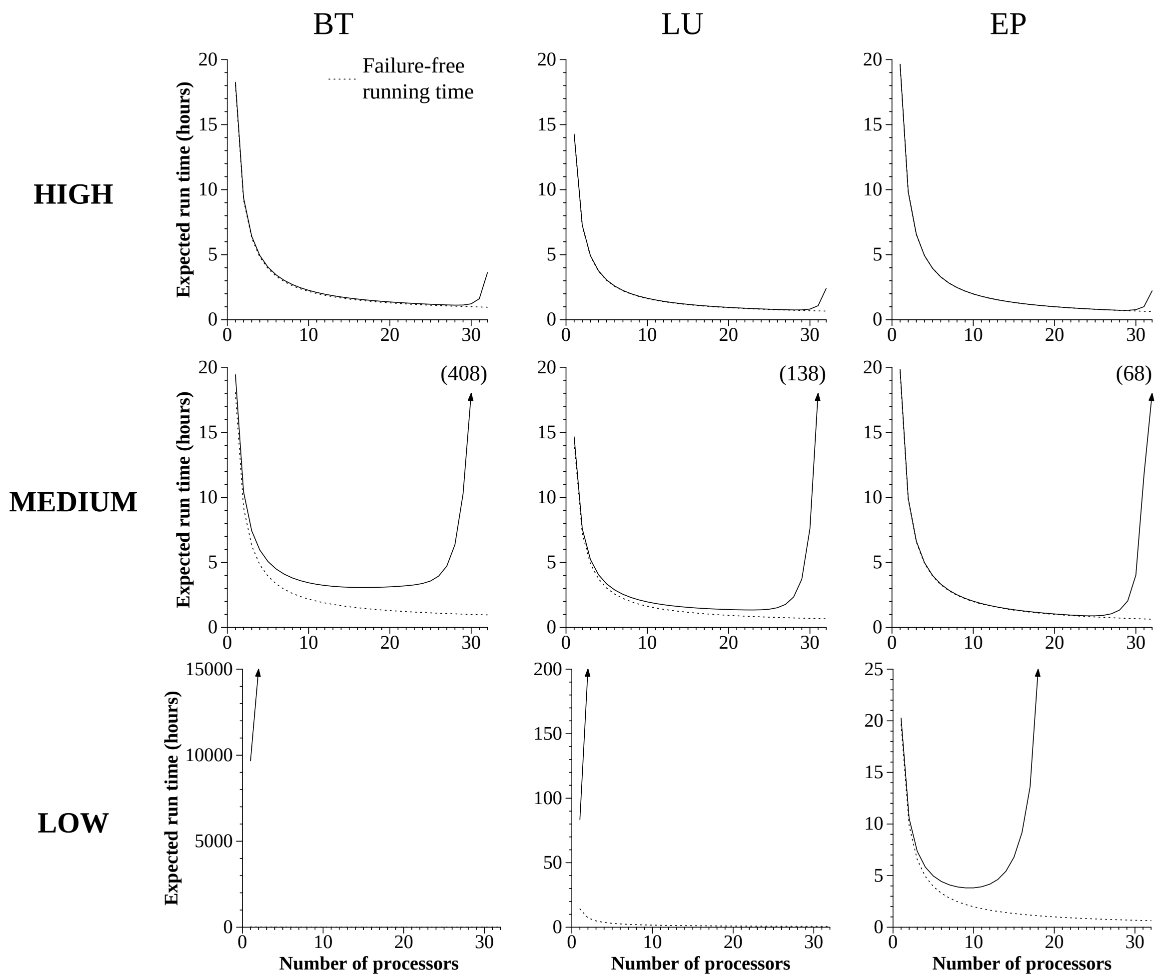

Figure 4: Optimal expected running times of all case studies in the presence of failures as a function of a.

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).

8 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 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

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**	28	1.82	0.928	1.04	1.12	7.7%
BT	**MEDIUM**	17	5.13	0.473	1.45	3.07	111%
BT	**LOW**	1	2.94	0.0019	18.1	9671	53476%
LU	**HIGH**	28	0.82	0.964	0.73	0.75	3.7%
LU	**MEDIUM**	23	2.23	0.624	0.84	1.34	60%
LU	**LOW**	1	0.80	0.170	14.2	83.4	487%
EP	**HIGH**	29	0.17	0.961	0.70	0.71	4.1%
EP	**MEDIUM**	25	0.55	0.903	0.81	0.89	11%
EP	**LOW**	9	0.036	0.577	2.2	3.8	73.18%

Table 4: Optimal a and I for all tests.

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 is 28 for BT and LU, and 29 for EP. This shows that even when checkpointing is quick, it pays to have a few spare processors so that the system does not sit idle often. For example, the LU application spends 0.68% of its time in Down states with 28 processors, 3.3% with 29 processors, and 12% with 30 processors.

In the **MEDIUM** processing environment, the optimal a ranges from 17 to 25. 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 $a_{opt} = 10$.

In the rightmost column of Table 4, checkpointing and failures add very little overhead in the **HIGH** processing cluster. In the **MEDIUM** cluster, the smaller checkpoints of EP lead to good performance in the presence of failures, while LU and BT perform less well. unrunnable. Given the nature of the environment and the size of the application, LU's performance is barely passable, and EP's is at least a speedup over the application's uniprocessor performance.

It is worth noting that although checkpointing and process migration environments have been built for idle workstation clusters [5, 8, 29], this is the first piece of work that attempts to characterize the performance of large parallel applications on these environments. One conclusion that may be drawn from this work is that unless the latency of checkpointing is significantly smaller than the environment's aggregate MTTF, then instrumenting a parallel program with coordinated checkpointing will not be an effective way of utilizing the environment. A more effective strategy may be to replicate processors and have them perform duplicate work so that it is the failure of the *last* processor in each replica group, rather than the failure of the *first* processor, that requires the application to roll back to a checkpoint.

9 Simulation

To gain a second measure of confidence concerning the Markov model, we implemented a stochastic simulator for parallel checkpointing systems. The simulator is a C program that takes the same input parameters as the model, generates failures and repairs using the `randlib()` random number generator from **www.netlib.gov**, and then calculates the availability for a given number of simulated days. For each value of a, we calculated the optimal interval and availability using a simulation of one million days, and for each of these values, the difference between the model's availability and the simulation's availability was a maximum of 2.7 percent.

10 Related Work

As stated above, there has been much work on checkpointing performance prediction in the presence of failures for uniprocessor and multi-processor systems [15, 16, 34, 36, 37]). 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 [36], which assumes that the program may reconfigure itself during execution to employ a variable number of processors. However, as stated above, the majority of long-running scientific programs use message passing libraries like PVM and MPI that either do not allow reconfiguration (MPI), or do not provide a good degree of support for it (PVM). Moreover, most checkpointing libraries for PVM and MPI do not allow reconfiguration [5, 12, 13, 18, 23, 27, 29, 31, 32].

11 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.

There are three directions in which to extend this work. First, we can explore the impact of the assumption of *iid* exponential failures and repairs, by performing simulation based on real failure data, as in [25]. Second, we can attempt to study a wider variety of checkpointing systems, such as two-level checkpointing systems [33] and diskless checkpointing systems [27]. Third, we can explore replication strategies as mentioned above in reference to idle workstation environments to derive more effective programming methodologies for these environments.

The Matlab scripts implementing $\mathcal{M}$ given values of N, a, I, C, L, R, λ and θ are available on the web at **http://www.cs.utk.edu/˜plank/plank/avail/**.

12 Acknowledgements

This material is based upon work supported by the National Science Foundation under grants CDA-9529459, CCR-9703390 and ACI-9876895. 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 the anonymous referees for their insightful suggestions.

References

[1] A. Agbaria and R. Friedman. Starfish: Fault-tolerant dynamic MPI programs on clusters of workstations. In *8th IEEE International Symposium on High Performance Distributed Computing*,. `http://dsl.cs.technion.ac.il/Starfish`, 1999.

[2] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. The NAS parallel benchmarks 2.0. Technical Report NAS-95-20, NASA Ames Research Center, December 1995.

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

[4] K. Buckner. Timings and memory usage for the NAS parallel benchmarks on a network of Sun Ultra workstations. Technical Report CS-98-408, University of Tennessee, November 1998.

[5] J. Casas, D. L. Clark, P. S. Galbiati, R. Konuru, S. W. Otto, R. M. Prouty, and J. Walpole. MIST: PVM with transparent migration and checkpointing. In *3rd Annual PVM Users' Group Meeting*, Pittsburgh, PA, May 1995.

[6] Y. Chen, J. S. Plank, and K. Li. CLIP: A checkpointing tool for message-passing parallel programs. In *SC97: High Performance Networking and Computing*, San Jose, November 1997.

[7] J. Choi, J. Dongarra, R. Pozo, and D. Walker. ScaLAPACK: A scalable linear algebra library for distributed memory concurrent computers. In *Proceedings of the Fourth Symposium on the Frontiers of Massively Parallel Computation*, pages 120–127. IEEE Publishers, 1992.

[8] P. E. Chung, Y. Huang, S. Yajnik, G. Fowler, K. P. Vo, and Y. M. Wang. Checkpointing in CosMiC: a user-level process migration environment. In *Pacific Rim International Symposium on Fault-Tolerant Systems*, December 1997.

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

[10] L. H. Crow and N. D. Singpurwalla. An empirically developed fourier series model for describing software failures. *IEEE Transactions on Reliability*, R-33:176–183, June 1984.

[11] E. N. Elnozahy, L. Alvisi, Y-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. Technical Report CMU-CS-99-148, Carnegie Mellon University, June 1999.

[12] E. N. Elnozahy, D. B. Johnson, and W. Zwaenepoel. The performance of consistent checkpointing. In *11th Symposium on Reliable Distributed Systems*, pages 39–47, October 1992.

[13] E. N. Elnozahy and W. Zwaenepoel. On the use and implementation of message logging. In *24th International Symposium on Fault-Tolerant Computing*, pages 298–307, Austin, TX, June 1994.

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

[15] E. Gelenbe. On the optimum checkpoint interval. *Journal of the ACM*, 26:259–270, April 1979.

[16] G. P. Kavanaugh and W. H. Sanders. Performance analysis of two time-based coordinated checkpointing protocols. In *Pacific Rim International Symposium on Fault-Tolerant Systems*, December 1997.

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

[18] J. León, A. L. Fisher, and P. Steenkiste. Fail-safe PVM: A portable package for distributed programming with transparent recovery. Technical Report CMU-CS-93-124, Carnegie Mellon University, February 1993.

[19] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor – a hunter of idle workstations. In *Proceedings of 8th International Conference on Distributed Computing Systems*, 1988.

[20] D. Long, A. Muir, and R. Golding. A longitudinal survey of internet host reliability. In *14th Symposium on Reliable Distributed Systems*, pages 2–9, Bad Neuenahr, September 1995. IEEE.

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

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

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

[24] J. S. Plank, M. Beck, G. Kingsley, and K. Li. **Libckpt**: Transparent checkpointing under Unix. In *Usenix Winter Technical Conference*, pages 213–223, January 1995.

[25] J. S. Plank and W. R. Elwasif. Experimental assessment of workstation failures and their impact on checkpointing systems. In *28th International Symposium on Fault-Tolerant Computing*, pages 48–57, Munich, June 1998.

[26] J. S. Plank and K. Li. Ickp — a consistent checkpointer for multicomputers. *IEEE Parallel & Distributed Technology*, 2(2):62–67, Summer 1994.

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

[28] J. S. Plank and M. G. Thomason. The average availability of parallel checkpointing systems and its importance in selecting runtime parameters. In *29th International Symposium on Fault-Tolerant Computing*, pages 250–259, Madison, WI,, June 1999.

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

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

[31] G. Stellner. Consistent checkpoints of PVM applications. In *First European PVM User Group Meeting*, Rome, 1994.

[32] G. Stellner. CoCheck: Checkpointing and process migration for MPI. In *10th International Parallel Processing Symposium*, pages 526–531. IEEE Computer Society, April 1996.

[33] N. H. Vaidya. A case for two-level distributed recovery schemes. In *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Ottawa, May 1995.

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

[35] Y-M. Wang, Y. Huang, K-P. Vo, P-Y. Chung, and C. Kintala. Checkpointing and its applications. In *25th International Symposium on Fault-Tolerant Computing*, pages 22–31, Pasadena, CA, June 1995.

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

[37] J. S. Young. A first order approximation to the optimum checkpoint interval. *Communications of the ACM*, 17(9):530–531, September 1974.

Appendix: Birth-Death Markov Chain $\mathcal{S}^\tau$

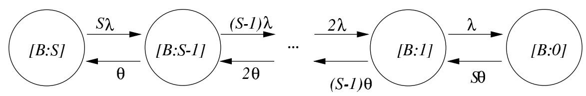

Figure 5: Birth-Death Markov Chain $\mathcal{S}^\tau$ with failure rate λ and repair rate θ.

This Appendix describes the calculations with Markov chain $\mathcal{S}^\tau$ referred to in Sections 6.2 and 6.3. Given S spare processors, i of which are currently functional, the chain $\mathcal{S}^\tau$ lets us find the probability of exactly j functional spares, $0 \leq j \leq S$, after τ seconds have passed.

$\mathcal{S}^\tau$ has $S+1$ states, denoted $[B:u]$ for $0 \leq u \leq S$. Each state $[B:u]$ corresponds to u processors among the S being functional, and $S-u$ being failed and under repair. Transitions from $[B:u]$ occur on failure or repair of a single processor, and thus may only be to $[B:u-1]$ and $[B:u+1]$. By convention [9, 22], $\mathcal{S}^\tau$ is drawn as a chain of states with the parameters of the exponential probability laws noted on the arcs as in Figure 5.

For indexing, the states in Figure 5 are numbered 1 through $S+1$ from left to right; thus, state k corresponds to $k-1$ failed spares and $S-k+1$ functional spares. The square matrix **R** of instantaneous transition rates is defined as follows:

$$\mathbf{R} = \begin{bmatrix} -S\lambda & S\lambda & 0 & 0 & \ldots & 0 & 0 & 0 \\ \theta & -((S-1)\lambda+\theta) & (S-1)\lambda & 0 & \ldots & 0 & 0 & 0 \\ 0 & 2\theta & -((S-2)\lambda+2\theta) & (S-2)\lambda & \ldots & 0 & 0 & 0 \\ & & & & \ldots & & & \\ 0 & 0 & 0 & 0 & \ldots & (S-1)\theta & -(\lambda+(S-1)\theta) & \lambda \\ 0 & 0 & 0 & 0 & \ldots & 0 & -S\theta & S\theta \end{bmatrix} \tag{8}$$

By standard computation for this kind of process [9], the conditional probability $q_{k\ell}^{S,\tau}$ that $\mathcal{S}^\tau$ is in state ℓ, given starting in state k and the passage of τ seconds, is the $(k\ell)^{th}$ entry in the matrix

$$\begin{aligned} \mathbf{Q}^{S,\tau} &= \text{expm}(\mathbf{R}\tau) \\ &= \sum_{k=0}^{\infty} \mathbf{R}^k \frac{\tau^k}{k!} \end{aligned} \tag{9}$$

where the function $\text{expm}(\mathbf{R}\tau)$ is the matrix exponential of $\mathbf{R}\tau$. One of our computations is $\mathbf{Q}^{S,\varrho}$ where $\tau = \varrho = R+I+L$ is the reference time for success in the System Recovery Phase.

The probabilities in matrix $\mathbf{Q}^{S,\tau}$ are functions of the TTF random variable τ, but τ's probability density is known to be an exponential form f_τ for a active processors. We can compute the likelihood of being in state ℓ, given starting in state k, as

$$q_{k\ell}^{S} = \int_t q_{k\ell}^{S,t} f_\tau(t) dt. \tag{10}$$

In order to find the $(S+1) \times (S+1)$ matrix

$$[q_{k\ell}^{S}] = \int_t \mathbf{Q}^{S,t} f_\tau(t) dt, \tag{11}$$

it is convenient to write a diagonalization of the rate matrix **R** as

$$\mathbf{R} = \mathbf{V}\mathbf{E}\mathbf{V}^{-1} \tag{12}$$

where

$$\mathbf{E} = \begin{bmatrix} d_1 & 0 & 0 & \cdots & 0 \\ 0 & d_2 & 0 & \cdots & 0 \\ 0 & 0 & d_3 & \cdots & 0 \\ & & & \cdots & \\ 0 & 0 & 0 & \cdots & d_{S+1} \end{bmatrix} \tag{13}$$

is the diagonal matrix with $\mathbf{R}$'s eigenvalues $d_1, d_2, \ldots, d_{S+1}$ on the main diagonal and $\mathbf{V}$ is a matrix of eigenvectors of $\mathbf{R}$ [14]; then

$$\begin{aligned} \mathbf{Q}^{S,\tau} &= \mathbf{V}\text{expm}(\mathbf{E}\tau)\mathbf{V}^{-1} && (14) \\ &= \mathbf{V} \begin{bmatrix} e^{d_1\tau} & 0 & 0 & \cdots & 0 \\ 0 & e^{d_2\tau} & 0 & \cdots & 0 \\ 0 & 0 & e^{d_3\tau} & \cdots & 0 \\ & & & \cdots & \\ 0 & 0 & 0 & \cdots & e^{d_{S+1}\tau} \end{bmatrix} \mathbf{V}^{-1} && (15) \end{aligned}$$

and

$$[q_{k\ell}^{S}] = \mathbf{V} \left(\int_t \text{expm}(\mathbf{E}t) f_\tau(t) dt \right) \mathbf{V}^{-1}. \tag{16}$$

All the eigenvalues of **R** are nonpositive, real numbers and one of them is 0.

Two intervals of values of the TTF random variable τ are used to compute two different matrices of probabilities. These matrices are denoted $[q_{k\ell}^{\text{Up},S}]$ for the System Up Phase and $[q_{k\ell}^{\text{Rec},S}]$ for the System Recovery Phase. For the System Up Phase, τ is in the interval [0,∞) and its density function is $f_\tau(t) = a\lambda e^{-a\lambda t}$. We compute

$$\int_0^\infty e^{d_i t} a\lambda e^{-a\lambda t} dt = -\frac{a\lambda}{d_i - a\lambda} \tag{17}$$

for each eigenvalue d_i. Then the matrix of likelihoods $[q_{k\ell}^{\mathrm{Up},S}]$ used as transition probabilities out of the System Up states is

$$[q_{k\ell}^{\mathrm{Up},S}] = \mathbf{V} \begin{bmatrix} -\frac{a\lambda}{d_1-a\lambda} & 0 & 0 & \cdots & 0 \\ 0 & -\frac{a\lambda}{d_2-a\lambda} & 0 & \cdots & 0 \\ & & & \cdots & \\ 0 & 0 & 0 & \cdots & -\frac{a\lambda}{d_{S+1}-a\lambda} \end{bmatrix} \mathbf{V}^{-1}. \tag{18}$$

If there is failure of an active processor while the system is in a System Recovery state, then τ is known to be less than $\varrho = R + I + L$ seconds. The TTF probability density function is conditioned on τ being in the interval $[0,\varrho)$, and we compute

$$\frac{1}{1 - e^{-a\lambda\varrho}} \int_0^{\varrho} e^{d_i t} a\lambda e^{-a\lambda t} dt = \frac{a\lambda(e^{(d_i - a\lambda)\varrho} - 1)}{(d_i - a\lambda)(1 - e^{-a\lambda\varrho})} \tag{19}$$

for each eigenvalue d_i. The matrix of likelihoods $[q_{k\ell}^{\mathrm{Rec},S}]$ used in computing the probabilities of transitions out of the System Recovery states is then obtained by placing the above values on the main diagonal of a diagonal matrix, multiplying on the left by $\mathbf{V}$, and multiplying on the right by $\mathbf{V}^{-1}$.