The Average Availability of Uniprocessor Checkpointing Systems, Revisited

James S. Plank Michael G. Thomason

Technical Report UT-CS-98-400
Department of Computer Science
University of Tennessee
August 25, 1998

See `http://www.cs.utk.edu/ plank/plank/papers/CS-98-400.html` for other information about this paper.

Abstract

Performance prediction of checkpointing systems in the presence of failures is a well-studied research area. This paper makes three small contributions to this research area. First, we show how to apply the concept of *availability* from reliability theory as a useful metric for checkpointing systems. Second, we study the average availability of uniprocessor checkpointing systems, using the **libckpt** checkpointer as a model. This is a slight deviation from previous checkpointing models. We employ Bernoulli trials to derive an expression for the availability of such a checkpointing system, and then use this expression to calculate the checkpoint interval which maximizes availability. Third, we present another derivation of the availability based on a direct calculation of average segment uptime. For the exponential failure distribution function, these two derivations are equivalent. The latter derivation allows for a simple way to numerically approximate availability for other failure distribution functions. We conclude with examples of applying these results.

1 Introduction

Checkpointing is an important topic in fault-tolerant computing as the basis for *rollback recovery*. In a checkpointing system, a user periodically checkpoints his or her program, saving its state to stable storage. If for some reason the program fails, then when the machine becomes functional, the program may be restarted from the stored checkpoint, thereby reducing the amount of recomputation that must be performed.

When using a checkpointing system, a user is confronted with an important question: How frequently should checkpoints be stored? If the user checkpoints too frequently, then the overhead of checkpointing may slow down the program too much. However, if the user checkpoints too infrequently, then the program may spend too much time re-executing code following a failure. To help the user decide upon an *optimal checkpointing interval*, many researchers have quantified the theoretical performance of checkpointing systems under a variety of assumptions [You74, GD78, Dud83, TB84].

The most relevant research to date has been that of Vaidya [Vai97], who has derived equations for average system performance of non-trivial uniprocessor checkpointing systems. As a performance metric, Vaidya introduces the concept of "overhead ratio," r, defined as follows:

$$r = \frac{\Gamma}{T} - 1$$

where Γ is the average running time of a program in the presence of failures with checkpointing and rollback recovery, and T is the optimal running time without checkpointing in the absence of failures. The goal in choosing parameters to optimize the performance of a program in the presence of failures is therefore to minimize r.

In this paper, we make three small contributions to the field. First, we present *availability* as an alternative metric of system performance. Though closely related to overhead ratio, availability is more commonly used in reliability theory, and to some people, it may have a more intuitive meaning.

Second, we derive equations for uniprocessor checkpointing system availability, employing a slightly different system model than Vaidya. This model is derived from **libckpt**, a very popular public-domain checkpointing library for Unix-based uniprocessors [PBKL95]. Our derivation is based on on Bernoulli trials. The resulting equation for availability differs slightly from Vaidya's, due to the slight change in system model. In practical terms, the difference is unimportant. However, we present it so that a theoretical model exists to match existing software systems.

Third, we present an alternate derivation of availability based on a direct calculation of average segment uptime. The significance of this presentation is that it facilitates approximating the availability of non-exponential failure distribution functions. We conclude by showing some availability studies of several checkpointing scenarios.

2 The System Model

The user is executing a long-running application on a uniprocessor. Every I seconds, a checkpoint is initiated. I is called the *checkpoint interval.* It takes L seconds from the initiation of a checkpoint before the checkpoint may be used for recovery. This is called the *checkpoint latency*, and typically corresponds to the time that it takes for the checkpoint to be written to stable storage. We assume that $L \leq I$, meaning that the program cannot store two checkpoints simultaneously.

The impact of checkpointing on the executation time of the program is embodied by the parameter C, called the *checkpoint overhead.* This is the time added to the execution time of the program as a result of storing a single checkpoint. Many checkpointing systems employ an optimization called *forked* or *copy-on-write* checkpointing [LNP90, EJZ92, PBKL95], where an in-memory copy of the checkpoint is created and written to stable storage

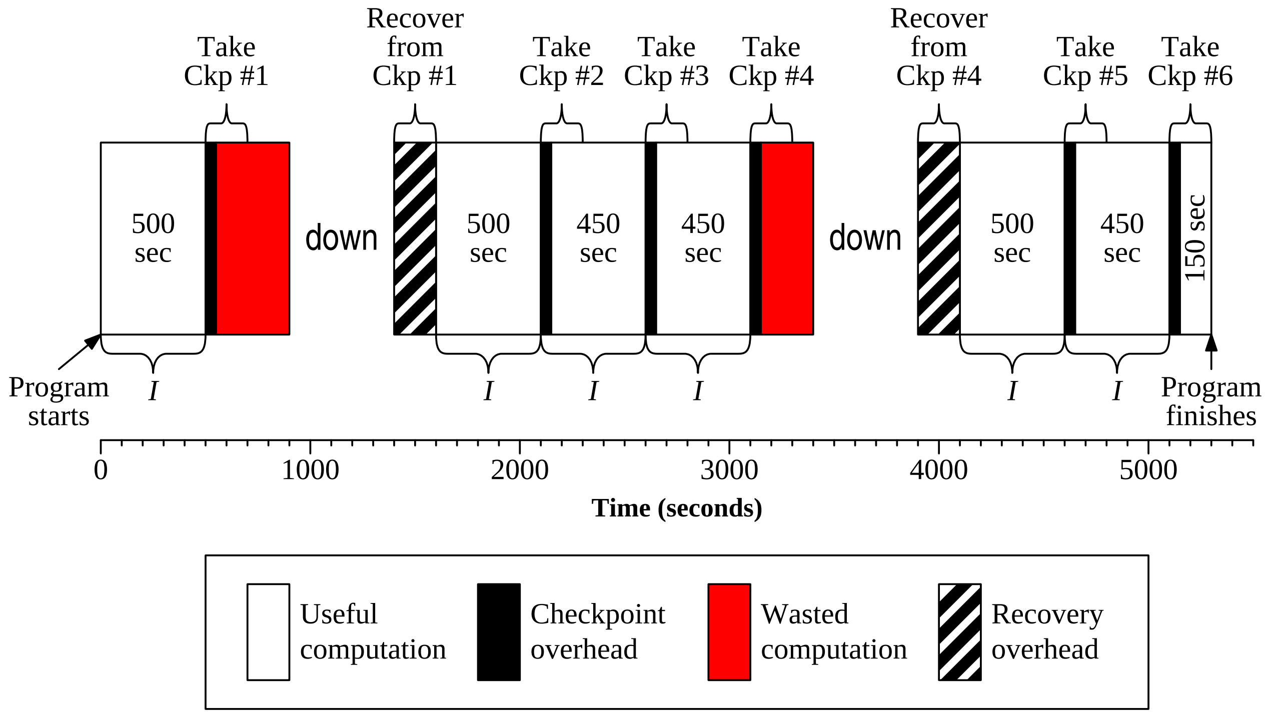


Figure 1: Timeline of an example program with checkpointing.

asynchronously while the program continues execution. To conserve memory, this copy is typically created by the "copy-on-write" technique, where pages that are unmodified by the program during checkpointing are shared by the program and the in-memory checkpoint. The major implication of copy-on-write checkpointing is that C is vastly smaller than L, which in turn drastically improves the performance of the system [EJZ92, PBKL95, Vai97].

Following a failure and subsequent repair of the processor, the program rolls back to the last completed checkpoint. All computation from the beginning of that checkpoint to the failure is lost. The program takes R seconds to restore its state to that of the checkpoint, and then it resumes computation, again checkpointing every I seconds.

Figure 1 shows the time line of an example program that executes with the following parameters:

- The failure-free running time of the program is 3000 seconds.
- Checkpointing is performed with $I = 500$ seconds, $C = 50$ seconds, $L = R = 200$ seconds.
- The machine running the computation fails twice, once at 900 seconds, and once at 3400 seconds. For each failure, the machine remains unusable for 500 seconds.

The execution of the program goes as follows. After 500 seconds, checkpoint #1 is initiated. This takes 200 seconds to complete and takes 50 seconds of computation away from the program. This 50 seconds occurs at the very beginning of the checkpoint. Once checkpoint #1 completes, the 500 seconds of computation preceding it become "useful," meaning that those 500 seconds will never have to be recomputed. At $t = 900$ seconds, the machine fails, meaning that all computation after checkpoint #1 is "wasted," because it will have to be redone upon recovery. At $t = 1400$ seconds, the machine becomes functional, and begins recovery from checkpoint #1. This takes 200 seconds to complete, at which point computation proceeds from checkpoint #1. At $t = 2100$, $t = 2600$ and $t = 3100$ seconds, checkpoints #2, #3 and #4 are initiated, each taking 50 seconds of computation away from the program, and requiring 200 seconds to complete. The completion of checkpoint #2 adds 500 seconds of useful computation to the completion of the program, and the completion of checkpoints #3 and #4 each add 450 more seconds. When checkpoint #4 is complete, the program has completed 1900 seconds (out of 3000) of useful computation.

At $t = 3400$ seconds, the machine again fails, rendering the computation following checkpoint #4 wasted. At $t = 3900$ seconds the machine is repaired, and the computation recovers from checkpoint #4. At $t = 4600$ and $t = 5100$, checkpoints #5 and #6 are initiated. Their completion adds another 950 seconds of useful computation, leaving 150 seconds more to finish the program. At $t = 5300$, that 150 seconds has been added, thus finishing the computation.

2.1 Differences between this model and previous models

Before Vaidya's paper, most research on checkpointing performance assumed that $C = L = R$. The difference between Vaidya's model and the model in this paper is slight. Vaidya denotes the checkpoint interval as T, which is the time between the end of the previous checkpoint's overhead period, and the beginning of the next checkpoint. T is also the time between the beginning of the program and the first checkpoint, and between the end of recovery and the first checkpoint following recovery.

Thus, if we set $T = I - C$, our model and Vaidya's are in agreement, except that in our model the first interval, and the intervals following recovery are all I seconds, whereas in Vaidy's, they are T seconds. We have chosen our model since it conforms to the model implemented by two public domain checkpointing implementations: **libckpt** [PBKL95], and **CLIP** [CPL97]. However, as will be shown, unless failures are quite frequent, the results from our model and Vaidya's do not differ significantly.

2.2 Assumptions

Our model (along with all the others) makes a few assumptions that do not hold in real checkpointing systems. First is that C, L, and R are constants for each program execution. Since L and R depend on the size of each checkpoint, they can only be constants if each checkpoint is the same size. When optimizations such as incremental checkpointing [FB89] or memory exclusion [PBKL95] are employed, it is common for multiple checkpoints of the same program to differ in size. The checkpoint overhead depends on additional factors, such as the memory usage patterns of the program, and again can vary from checkpoint to checkpoint. However, we assume that the programmer will employ average values for C, L, and R.

Another assumption of our model is that all of the overhead induced by a checkpoint occurs when the checkpoint is initiated. As Vaidya has detailed [Vai97], when forked checkpointing is employed, this overhead is in fact distributed throughout the checkpoint's latency. Moreover, when other optimizations such as incremental checkpointing or memory exclusion are employed, some of the checkpoint's overhead arises from tagging regions of memory for inclusion/exclusion *before the checkpoint is initiated.* Therefore, our model (and the others) departs from reality in this respect.

In the analysis that follows, we assume that processor failures occur as a Poisson random variable; hence, the interoccurrence times between failures[1] are independently and identically distributed (*iid*) as an exponential with cumulative distribution function $F(t) = 1 - e^{-\lambda t}$, probability density function $f(t) = \lambda e^{-\lambda t}$, and mean time to failure $MTTF = 1/\lambda$. This model has never been corroborated by observations of real checkpointing systems; however, in [PE98], Plank and Elwasif show that for three sets of observed failure data that do *not* follow a Poisson model, theoretical results based on the Poisson model are applicable for predicting checkpointing performance when I is smaller than, or near its optimal value. Thus, there is utility in predicting checkpointing performance using the Poisson model.

In fact, the exponential is the unique distribution with a constant failure rate [Par62]. Thus, a measure such as *availability* defined below almost surely underestimates a system's long-run, average performance whenever an exponential distribution is used in lieu of a distribution with the same mean value but an increasing failure rate.

3 Average system availability as a metric of checkpointing performance

In standard reliability theory, the *availability* of a system is defined in terms of *uptime* and *downtime* intervals. [BP75]. Relative to system start-up at time 0, the system status is a sequence of time intervals where the start of a new interval is marked by the end of a downtime.

Any interval of system observation may be decomposed into uptime and downtime intervals which occur in alternation. An *uptime interval* is one in which the system produces useful work. Conversely, a *downtime interval* is one in which the system does not produce useful work. In checkpointing systems as modeled in section 2, the uptime intervals are *only those in which useful computation is being produced.* There are other times, for example when the system is recovering from a checkpoint or when the computer is performing wasted computation, that the computer is functional but the checkpointing system is *not* producing useful computation. In terms of availability, these are considered downtimes.

Given a time interval of length T from system start-up, let the sum of the durations of the uptime intervals

[1]In the probability literature, a *renewal process* describes the sequence of independent positive random variables that represent the interoccurrence times of a recurrent event (cf [Par62]).

be U and the sum of the durations of the downtime intervals be D. Since uptimes and downtimes are disjoint,

$$T = U + D.$$

The availability of the system during time T is defined to be

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

As an example, in Figure 1, the availability of the program for the duration of its execution is $3000/5300 = 0.566$. In the first 1000 seconds, the availability is $500/1000 = 0.5$, despite the fact that the computer is functional for the first 900 seconds.

The *long-run, average availability* of a system is defined in terms of uptime and downtime as random variables [BP75] by taking the limit of A_T as T goes to infinity:

$$A = \lim_{T \to \infty} A_T. \tag{2}$$

Since interoccurrence times of failures are *iid*, the uptime and the downtime between consecutive failures are random variables with well-defined mean values, denoted respectively as μ and ν, and allow us to compute the availability as: [Par62]

$$A = \frac{\mu}{\mu + \nu}. \tag{3}$$

MTTF is the mean failure interoccurrence time, and since uptimes and downtime are disjoint, $\mu + \nu = MTTF$ and the availability is computable as:

$$A = \frac{\mu}{MTTF}. \tag{4}$$

In the case of the exponential pdf, we have $MTTF = 1/\lambda$ and thus:

$$A = \lambda\mu. \tag{5}$$

Average system availability A is a convenient metric of checkpointing performance because it has a straightforward meaning. It is the fraction of time that the system is producing computation that counts toward the completion of the program. All other portions of time, be they machine downtime, checkpointing overhead, recovery overhead, or wasted computation, combine to lower the availability. If a computation takes F seconds to complete in the absense of failures and checkpointing, then a F/A is an approximation of the program's expected time to completion in the presence of failures and checkpointing. This approximation becomes more accurate as $F \to \infty$.

When comparing two computing systems S_1 and S_2 that run at different processor speeds R_1 and R_2, the system availability gives the user a very simple way to compute which system will run faster with checkpointing in the presence of failures. The system S_i with the larger value of R_iA_i is the faster system.

Availability is directly related to *overhead ratio* defined in [Vai97] in the following way:

$$r = \frac{\Gamma}{T} - 1$$

and as $T \to \infty$,

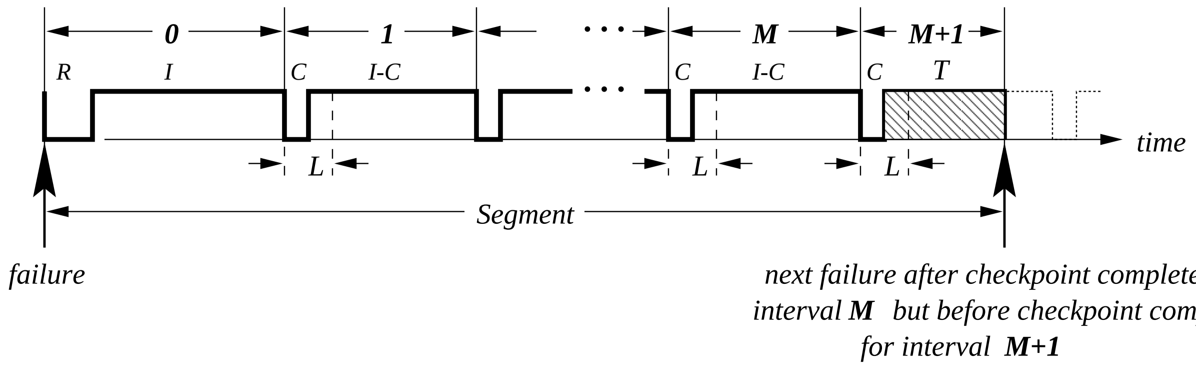
$$\begin{aligned} r &= \frac{\mu + \nu}{\mu} - 1 \\ &= \frac{1}{A} - 1. \end{aligned}$$

Therefore, maximizing A is equivalent to minimizing r.

4 Computing Average System Availability without Markov Chains

4.1 Assumptions and Definitions

For simplicity, we assume zero repair time following a failure (or that R includes a fixed time to repair). In other words, the time between failures is all time during which the computer is available. As detailed in Section 2.2, we assume that processor failures are *iid* as a Poisson random variable with mean value λ. Thus, the interoccurrence times between failures are *iid* as an exponential with *cdf* $F(t) = 1 - e^{-\lambda t}$, *pdf* $f(t) = \lambda e^{-\lambda t}$, and $MTTF = 1/\lambda$.



0
1
M
M+1
R
I
C
I-C
C
I-C
C
T
L
L
L
time
Segment
failure
next failure after checkpoint completes for interval **M** but before checkpoint completes for interval **M+1**

Figure 2: A segment is a sequence of intervals between consecutive failures.

To compute the average system availability of a checkpointing system as detailed in section 2, we define a *segment* s to be a time interval between processor failures. The duration of s is $|s|$. A typical segment is depicted in Figure 2, and composed of checkpoint intervals labelled 0 through $M + 1$. We term a checkpoint interval *complete* or *successful* if the checkpoint that ends the interval completes. In other words, the failure that ends the segment must occur at a time greater than L seconds past when the checkpoint begins. Obviously, if the failure occurs before an interval's checkpoint begins, the interval is unsuccessful.

As pictured, all checkpoint intervals except interval $M + 1$ are successful. In terms of uptime, checkpoint interval 0, if successful, contributes I seconds to the segment. All other successful intervals contribute $I - C$

seconds, and interval $M+1$ contributes zero seconds. Therefore, if a segment fails before checkpoint interval 0 completes, the value of U for the segment is zero. If checkpoint interval 0 completes, then the value of U for the segment is:

$$U = I + M(I - C). \tag{6}$$

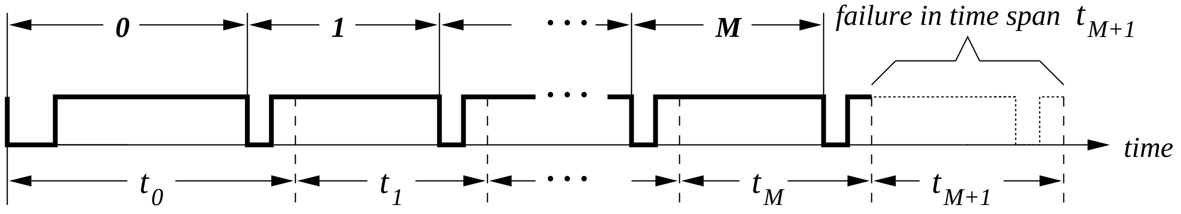


Figure 3: Time spans in a segment.

We define the *time span* t_i of checkpoint interval i to be the interval of time, during which a failure would make that checkpoint interval unsuccessful. Thus, $t_0 = [0, R + I + L)$, and for $i > 0$, $t_i = [R + iI + L, R + (i+1)I + L)$. In terms of interval size, $|t_0| = R + I + L$, and for $i > 0$, $|t_i| = I$.

Finally, we define the sets $\mathcal{F}_i$ and $\bar{\mathcal{F}}_i$ as follows. Both $\mathcal{F}_i$ and $\bar{\mathcal{F}}_i$ are subsets of $\mathcal{S}$, the set of all possible segments. $\mathcal{F}_i$ is the set of all segments that fail in t_i, and $\bar{\mathcal{F}}_i$ is the set of all segments that fail after t_i. In other words:

$$\begin{aligned}\mathcal{F}_i &= \left\{ s \in \mathcal{S} \text{ s.t. } \sum_{k=0}^{i-1} |t_i| \le |s| < \sum_{k=0}^{i} |t_i| \right\} \\ \bar{\mathcal{F}}_i &= \left\{ s \in \mathcal{S} \text{ s.t. } |s| \ge \sum_{k=0}^{i} |t_i| \right\}.\end{aligned}$$

We make three observations about $\mathcal{F}_i$ and $\bar{\mathcal{F}}_i$:

$$\begin{aligned}\mathcal{F}_i \cap \bar{\mathcal{F}}_i &= \emptyset \\ \mathcal{F}_0 \cup \bar{\mathcal{F}}_0 &= \mathcal{S} \\ \mathcal{F}_i \cup \bar{\mathcal{F}}_i &= \bar{\mathcal{F}}_{i-1},\ i > 0\end{aligned}$$

4.2 Determining the availability

Because failures are *iid* and follow the Poisson model, the task of computing availability for an infinitely long computation is equivalent to computing the average availability of an infinite number of segments, where the segment sizes are *iid* in the same manner as the failure interoccurence times. We use Eq. 5 for this determination. Since λ is a given, the problem is to detemine μ, the expected segment uptime.

Segments in $\mathcal{F}_0$ have no uptime. Segments in $\bar{\mathcal{F}}_0$ have uptimes $I + M(I - C)$. Since $\mathcal{F}_0 \cup \bar{\mathcal{F}}_0 = \mathcal{S}$, μ may be determined as:

$$\mu = P(\bar{\mathcal{F}}_0)(I + \mathcal{M}(I - C)), \tag{7}$$

where $P(\bar{\mathcal{F}}_0)$ is the probability that a segment is an element of $\bar{\mathcal{F}}_0$, and $\mathcal{M}$ is the mean value of M for all segments in $\bar{\mathcal{F}}_0$.

$P(\bar{\mathcal{F}}_0)$ is the probability that $|s| > (R+I+L)$:

$$P(\bar{\mathcal{F}}_0) \quad = \quad 1 - F(R+I+L) = e^{-\lambda(R+I+L)} \tag{8}$$

To determine $\mathcal{M}$, we define $P(\bar{\mathcal{F}}_i)$ to be the probability that a segment $s \in \bar{\mathcal{F}}_{i-1}$ is also in $\bar{\mathcal{F}}_i$, and $P(\mathcal{F}_i)$ to be the probability that a segment $s \in \bar{\mathcal{F}}_{i-1}$ is also in $\mathcal{F}_i$. Obviously, $P(\bar{\mathcal{F}}_i) + P(\mathcal{F}_i) = 1$. $P(\mathcal{F}_i)$ is the conditional probability that a segment s with $|s| \geq R+iI+L$ also has $|s| < R+(i+1)I+L$. For the exponential distribution of $|s|$:

$$\begin{aligned} P(\mathcal{F}_i) = F((R+iI+L) - (R+(i+1)I+L) = F(I) \quad &= \quad 1 - e^{-\lambda(I)} &(9)\\ P(\bar{\mathcal{F}}_0) = 1 - P(\mathcal{F}_i) \quad &= \quad e^{-\lambda(I)} &(10)\end{aligned}$$

Let $\mathcal{E}_i$ be a Bernoulli trial with possible outcomes $\{\mathcal{F}_i, \bar{\mathcal{F}}_i\}$. We define an experiment $\mathcal{E}$ to be a sequence of Bernoulli trials $\mathcal{E}_1, \ldots \mathcal{E}_M, \mathcal{E}_{M+1}$ such that for $i \leq M$, $\mathcal{E}_i$ has outcome $\bar{\mathcal{F}}_i$, and $\mathcal{E}_{M+1}$ has outcome $\mathcal{F}_{M+1}$. Then $\mathcal{M}$ is the average value of M for all such experiments $\mathcal{E}$. Using the probabilities in Eqs. 9 and 10, the determination of $\mathcal{M}$ is a standard result in probability theory [Fel68]:

$$\mathcal{M} \quad = \quad \frac{e^{-\lambda I}}{1 - e^{-\lambda I}}. \tag{11}$$

Therefore:

$$\begin{aligned} \mu \quad &= \quad e^{-\lambda(R+I+L)} \left(I + \frac{e^{-\lambda I}}{1 - e^{-\lambda I}} (I - C) \right) \\ &= \quad e^{-\lambda(R+I+L)} \left(\frac{I - Ce^{-\lambda I}}{1 - e^{-\lambda I}} \right) \\ &= \quad \left((I - Ce^{-\lambda I}) \frac{e^{-\lambda I}}{1 - e^{-\lambda I}} \right) e^{-\lambda(R+L)} &(12)\\ A \quad &= \quad \left((I - Ce^{-\lambda I}) \frac{e^{-\lambda I}}{1 - e^{-\lambda I}} \right) \lambda e^{-\lambda(R+L)} &(13)\end{aligned}$$

Therefore, we maximize A by selecting the value of I that maximizes the term:

$$(I - Ce^{-\lambda I}) \frac{e^{-\lambda I}}{1 - e^{-\lambda I}}.$$

Note that as in [Vai97], the optimal value of I is independent of L and R.

5 Direct Calculation of μ

One may also calculate μ directly as an infinite sum of the probability of getting to and failing in interval i multiplied by the uptime associated with failing in that interval:

$$\begin{aligned}
\mu &= \sum_{i=0}^{\infty} P(\text{getting to and failing in interval } i)(I+(I-C)i) \\
&= \sum_{i=0}^{\infty} \left(F(L+(i+2)I+R) - F(L+(i+1)I+R)\right)(I+(I-C)i) \qquad (14)
\end{aligned}$$

For the exponential distribution with mean interoccurence time $1/\lambda$, and $F(t) = 1 - e^{-\lambda t}$, μ has a closed form solution:

$$\begin{aligned}
\mu &= \sum_{i=0}^{\infty} \left(\left(1-e^{-\lambda(L+R+2I+iI)}\right) - \left(1-e^{-\lambda(L+R+I+iI)}\right)\right)(I+(I-C)i) \\
&= \sum_{i=0}^{\infty} \left(-e^{-\lambda(L+R+I)}e^{-\lambda I}e^{-\lambda iI} + e^{-\lambda(L+R+I)}e^{-\lambda iI}\right)(I+(I-C)i) \\
&= e^{-\lambda(L+R+I)}\left(1-e^{-\lambda I}\right)\sum_{i=0}^{\infty}\left(e^{-\lambda iI}\right)(I+(I-C)i) \\
&= e^{-\lambda(L+R+I)}\left(1-e^{-\lambda I}\right)\left(\sum_{i=0}^{\infty} Ie^{-\lambda Ii} + \sum_{i=0}^{\infty}(I-C)ie^{-\lambda Ii}\right) \\
&= e^{-\lambda(L+R+I)}\left(1-e^{-\lambda I}\right)\left(I\sum_{i=0}^{\infty}\left(e^{-\lambda I}\right)^i + (I-C)\sum_{i=0}^{\infty} i\left(e^{-\lambda I}\right)^i\right) \\
&= e^{-\lambda(L+R+I)}\left(1-e^{-\lambda I}\right)\left(\frac{I}{1-e^{-\lambda I}} + \frac{(I-C)e^{-\lambda I}}{(1-e^{-\lambda I})^2}\right) \\
&= e^{-\lambda(L+R+I)}\left(I + \frac{(I-C)e^{-\lambda I}}{1-e^{-\lambda I}}\right) \\
&= e^{-\lambda(L+R+I)}\left(\frac{I - Ie^{-\lambda I} + Ie^{-\lambda I} - Ce^{-\lambda I}}{1-e^{-\lambda I}}\right) \\
&= e^{-\lambda(L+R)}\left(I - Ce^{-\lambda I}\right)\left(\frac{e^{-\lambda I}}{1-e^{-\lambda I}}\right).
\end{aligned}$$

Note that this is the same as Eq. 12 above.

5.1 Other distributions

With Eq. 14, one may calculate μ numerically to an arbitrary precision given any distribution function. For example, in [ML91], the failure interoccurence time for spare CPU availability was observed to best fit a hyper-exponential distribution [Tri82], which has the general cumulative distribution function

$$F(t) = \sum_{i=1}^{k} \alpha_i(1-e^{-\lambda_i t}).$$

Given values for k, $\alpha_1, \ldots, \alpha_k$, and $\lambda_1, \ldots \lambda_k$, one may numerically approximate μ and thus A by calculating Eq. 14 for an appropriately large number of intervals.

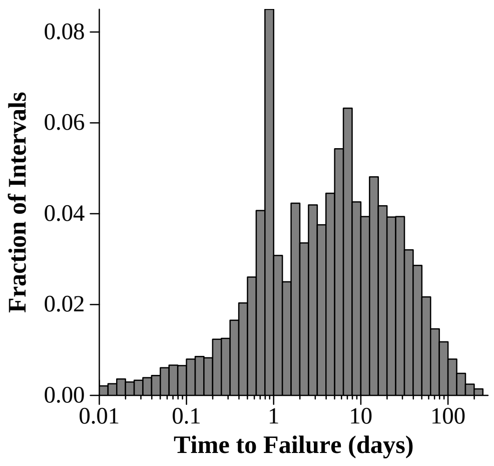


Figure 4: TTF intervals for the **LONG** data set.

6 An Example Case Study: The LONG Failure Data

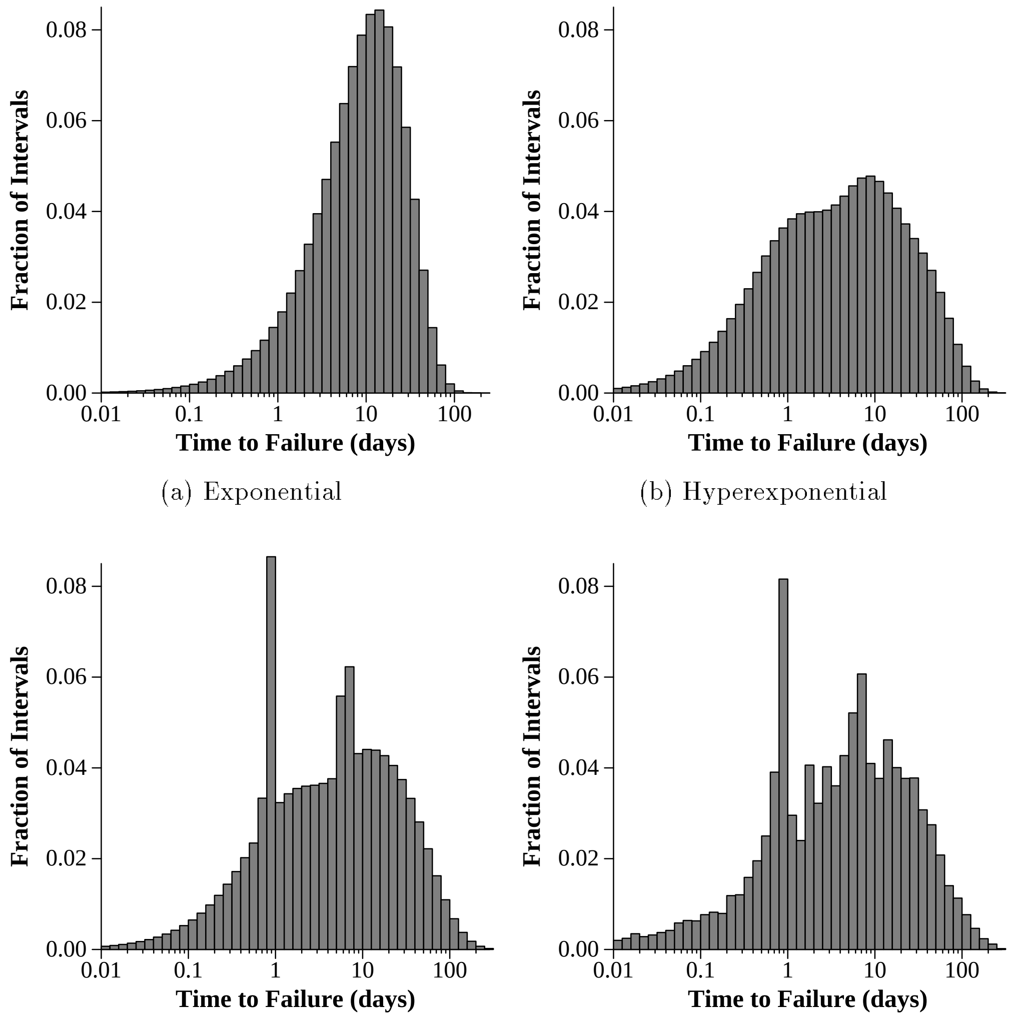
In [PE98], Plank and Elwasif obtained over six months of longitudinal failure data for three separate collections of workstations. The probability that any of these collections failed according to a Poisson model is vanishingly small [LMG95, PE98]. To assess the relationship between equations derived for the Poisson model and these data sets, Plank and Elwasif wrote a simulator, which takes as input a data set, C, L, R, I, and F, the failure-free running time of a program, and determines E_I, the mean running time of that program on the machines in the data set when checkpointing with the given parameters.

One of the surprising results of [PE98] is that even though the TTF intervals in the data sets are extremely unlikely to follow a Poisson model, some of the theoretical results based on the Poisson model still apply. In particular, the value of I that minimizes Γ (i.e. that maximizes A) using Vaidya's analysis typically does a good job of approximating the value of I that minimizes E_I in the simulator.

As an example of using the equations derived in this paper, we consider the **LONG** data set from [PE98]. This is failure data from July, 1994 to May, 1995, for 993 workstations distributed around the world [LMG95]. In the data set, there are 10,958 TTF intervals and 9,965 TTR intervals. The MTTF is 13.306 days, and the MTTR is 1.497 days. A histogram of the TTF intervals is plotted in Figure 4.

A typical way to apply the standard analysis to this data set for the purposes of selecting a checkpointing interval and estimating running time is to approximate the system using an exponential distribution with same MTTF. Thus, $\lambda = 1/13.306$ days. A histogram of the TTF intervals for this distribution function is plotted in Figure 5(a).

The analysis in Section 5 allows us to calculate availability numerically given any distribution function. We therefore selected three other plausible distribution functions. First, as mentioned in Section 5.1, computer systems usage often falls into distinct phases that may be modelled with a hyperexponential distribution. In



(a) Exponential (b) Hyperexponential (c) Mixed regular/hyperexponential (d) F_{LONG}

Figure 5: TTF intervals for various distribution functions.

Figure 5(b) we plot a three-level hyperexponential distribution with $\alpha_1 = 0.370$, $\lambda_1 = 1/5.89$ days, $\alpha_2 = 0.362$, $\lambda_2 = 1/27.64$ days, $\alpha_3 = 0.268$, $\lambda_3 = 1/0.844$ days. These values were obtained by attempting to minimize the least squares difference between the TTF histogram produced by the hyperexponential and the histogram in Figure 5(a). The minimization was performed by a simple Monte Carlo program running for a few days.

The next distribution function is weighted mixture of uniform distributions and exponential distributions:

$$F(t) = \sum_{i=1}^{k_1} \alpha_i \left(\mathcal{U}(a_i, b_i, t)\right) + \sum_{i=k_1+1}^{k} \alpha_i \left(1 - e^{-\lambda_i t}\right),$$

where

$$\mathcal{U}(a,b,t) = \begin{cases} 0 & \text{if } t < a \\ \frac{t-a}{b-a} & \text{if } a \le t < b \\ 1 & \text{if } b \le t \end{cases}$$

Note that $\mathcal{U}(a,a,t)$ is well-defined: $\mathcal{U}(a,a,t) = 0$ if $t < a$, and $\mathcal{U}(a,a,t) = 1$ if $t \ge a$.

We explore this mixed distribution function because the data in Figure 4 has distinct spikes near one day and one week. This corresponds to machines that go through phases where they are rebooted daily or weekly for maintenance, or perhaps power conservation. In Figure 5(c), we plot a mixed distribution function with parameters displayed in Table 1. This was obtained by selecting four regular intervals corresponding to the highest spikes in Figure 4 and then mixing them with a four-level hyperexponential distribution obtained by a Monte Carlo program.

i	α_i	λ_i (1/days)	a_i (days)	b_i (days)
1	0.006632		0.640	0.780
2	0.056758		0.800	0.999
3	0.016478		5.012	6.308
4	0.020878		6.400	7.800
5	0.231590	1/1.07		
6	0.204843	1/5.64		
7	0.312144	1/18.34		
8	0.150676	1/39.49		

Table 1: Parameters for the mixed distribution.

Finally, the last distribution function is one built directly from the data. Specifically, let $S_1, S_2, \ldots, S_N$ be the TTF intervals in data set X. Then:

$$F_X(t) = \frac{1}{N}\sum_{i=0}^{N} \mathcal{U}(S_i, S_i, t)$$

Thus, Figure 5(d), displaying F_{LONG}, matches Figure 4 exactly.

6.1 Calculating the Optimal Checkpoint Interval

In [PE98], the authors use their simulator to plot the optimal checkpoint interval of a program running on the LONG data set. They consider a program with a failure-free running time of 30 days, and then find the checkpoint interval that minimizes expected running time while varying checkpoint overhead (C) from ten seconds to one hour. In their tests, they assume that $C = L = R$. Their results are plotted in Figure 6, along with approximations of the optimal checkpoint interval as determined by Young [You74] and Vaidya [Vai97].

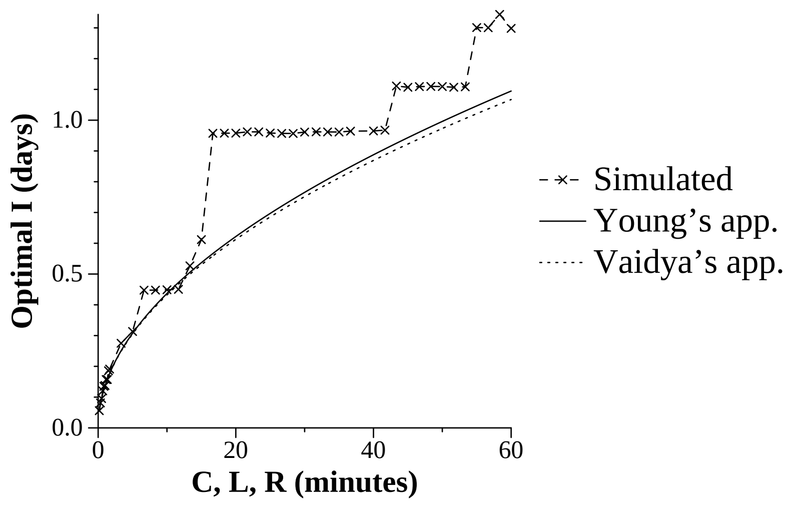


Figure 6: Simulated and theoretical determinations of the optimal checkpoint interval from [PE98].

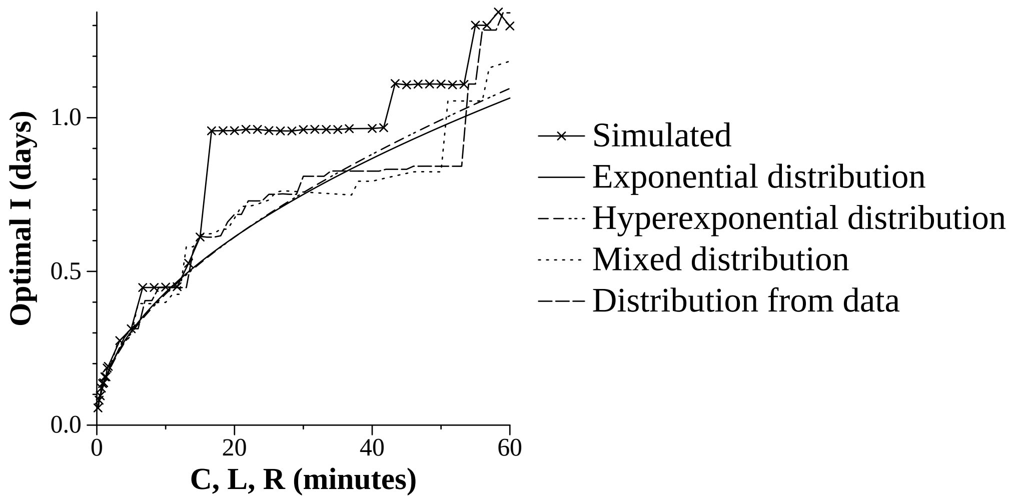


Figure 7: Simulated and theoretical determinations of of the optimal checkpoint interval using Eq. 14.

We use Eq. 14 to calculate the checkpoint interval numerically that maximizes availability for each of the distributions in Figure 5. The results are plotted in Figure 7, along with the values determined by the simulator in [PE98].

The points of interest in Figure 7 are as follows. First, the curve for the exponential and hyperexponential distributions differ very little. Moreover, the optimal values of I as determined by Eq. 14 and the exponential distribution is nearly identical to Vaidya's. The other two distribution functions give curves for optimal I that are much less smooth, and resemble the simulator's values more closely.

Although the distribution function F_{LONG} matches the distribution of the LONG data exactly, the simulator gives different values for optimal I than Eq. 14. There are two reasons for this. First, the exact ordering of TTF intevals in the data is one of many that can create F_{LONG}. Moreover, that ordering affects the results of the simulator. Therefore, it is reasonable to expect the simulated and theoretical values to differ. Second, the simulator's values for optimal I in Figures 6 and 7 are those that minimize the expected running time of the program with failures and downtime due to repair. If the repair time is removed from the simulator, then simulator's values for optimal I change. It is the subject of future work to calculate all of these values and compare them to the values generated by Eq. 14.

7 Conclusion

We have presented two derivations of the availability of uniprocessor checkpointing systems. These are useful for predicting and optimizing the performance of checkpointing when the distribution of failures is known. We have applied the results of these derivations to some real-life data and compared them to simulations of checkpointing systems on that data. It is a subject of future work to further evaluate the difference between simulated and theoretical results, and the implications of this difference in the performance of checkpointing systems. It is also a topic of future work to use these analyses as the bases for deriving the availability of multiprocessor checkpointing systems.

8 Acknowledgement

This material is based upon work supported by the National Science Foundation under grant CCR-9703390.

References

[BP75] R.E. Barlow and F. Proschan. *Statistical Theory of Reliability and Life Testing.* Holt, Reinhart, and Winston, Inc., NY, 1975. Republished by TO BEGIN WITH, Silver Spring, MD, 1981.

[CPL97] 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.

[Dud83] A. Duda. The effects of checkpointing on program execution time. *Information Processing Letters*, 16:221–229, 1983.

[EJZ92] 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.

[FB89] S. I. Feldman and C. B. Brown. Igor: A system for program debugging via reversible execution. *ACM SIGPLAN Notices, Workshop on Parallel and Distributed Debugging*, 24(1):112–123, January 1989.

[Fel68] W. Feller. *An Introduction to Probability Theory and Its Applications (Third Edition)*. John Wiley & Sons, Inc., NY, 1968.

[GD78] E. Gelenbe and D. Derochette. Performance of rollback recovery systems under intermittant failures. *Communications of the ACM*, 21(6):493–499, June 1978.

[LMG95] 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.

[LNP90] K. Li, J. F. Naughton, and J. S. Plank. Real-time, concurrent checkpoint for parallel programs. In *Second ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 79–88, March 1990.

[ML91] M. W. Mutka and M. Livny. The available capacity of a privately owned workstation environment. *Perfomance Evaluation*, August 1991.

[Par62] E. Parzen. *Stochastic Processes*. Holden-Day, San Francisco, CA, 1962.

[PBKL95] J. S. Plank, M. Beck, G. Kingsley, and K. Li. **Libckpt**: Transparent checkpointing under unix. In *Usenix Winter 1995 Technical Conference*, pages 213–223, January 1995.

[PE98] 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.

[TB84] S. Toueg and Ö. Babaoglu. On the optimum checkpoint selection problem. *SIAM Journal on Computing*, 13:630–649, August 1984.

[Tri82] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

[Vai97] N. H. Vaidya. Impact of checkpoint latency on overhead ratio of a checkpointing scheme. *IEEE Transactions on Computers*, 46(8):942–947, August 1997.

[You74] J. S. Young. A first order approximation to the optimum checkpoint interval. *Communications of the ACM*, 17(9):530–531, September 1974.