# Minimizing the Network Overhead of Checkpointing in Cycle-harvesting Cluster Environments*

Daniel Nurmi, John Brevik, Rich Wolski
University of California Santa Barbara
Deptartment of Computer Science
Santa Barbara, CA
{nurmi,jbrevik,rich}@cs.ucsb.edu

## Abstract

*Cycle-harvesting systems such as Condor have been developed to make desktop machines in a local area (which are often similar to clusters in hardware configuration) available as a compute platform. To provide a dual-use capability, opportunistic jobs harvesting cycles from the desktop must be checkpointed before the desktop resources are reclaimed by their owners and the job is evacuated.*

*In this paper, we investigate a new system for computing efficient checkpoint schedules in cycle-harvesting environments. Our system records the historical availability from each resource and fits a statistical model to the observations. Because checkpointing must often traverse the network (i.e. the desktop hosts do not provide sufficient persistent storage for checkpoints), we combine this model with predictions of network performance to the storage site to compute a checkpoint schedule. When an application is initiated on a particular resource, the system uses the computed distribution to parameterize a Markov state-transition model for the application's execution, evaluates the expected time and network overhead as a function of the checkpoint interval, and numerically optimizes with respect to time.*

*We report on the performance of and implementation of this system using the Condor cycle-harvesting environment at the Universiity of Wisconsin. We also evaluate the efficiencies we achieve for a variety of network overheads using trace-based simulation. Finally, we validate our simulations against the observed performance with Condor. Our results indicate that while the choice of model distribution has a relatively small but positive effect on time efficiency, it has a substantial impact on network utilization.*

*This work was supported by grants from the National Science Foundation, numbered NGS-0305390 and CCR-0331654.

# 1 Introduction

As a computing platform, a cluster of commodity workstations interconnected via local-area networking technology offers an extremely high price-performance ratio among the available alternatives. However, for many organizations a dedicated cluster represents a non-trivial investment, both in terms of hardware and maintenance expense. For this reason, "cycle-harvesting" systems such as Condor have been developed to make desktop machines in a local area (which are often similar to a cluster in hardware configuration) available as a compute platform opportunistically. To provide this dual-use capability, opportunistic jobs harvesting cycles from the desktop must be checkpointed before the desktop resources are reclaimed by their owners and the job is evacuated.

These "virtual clusters," implemented by resource-harvesting systems such as Condor [23], SETI@Home [21], Folding@Home [29], UUCS [8], and Entropia [6, 11], offer vast computing potential by leveraging the unused capacity of relatively volatile desktop and "personal" computing resources. From the perspective of the programmer or user wishing to tap this potential, the proffered resources appear significantly less stable than actual clusters. Each resource typically has a primary owner or user who may reclaim or reboot the machine without warning: It is the unused capacity that is available for harvest. Even when these reclamations are controlled (*e.g.*, the resource-harvesting system notices user activity and evacuates any guest load), the effect on the application is the same as if a resource fails: The resource is no longer available for processing, and any application state stored on that resource is in danger of being lost.

Checkpointing is an obvious and widely studied technique for ameliorating the effects of resource volatility in high-performance parallel computing and distributed system settings [2, 4, 13, 24, 25, 26, 27]. However, checkpointing introduces both a system performance overhead (in the

form of additional network and storage load) and an execution performance overhead (the delay required to generate a checkpoint) that can be particularly significant in desktop settings. Often, application checkpoints cannot be stored locally on the resource, either because of security concerns or because resource owners simply do not wish to give up disk storage to guest applications. In this case, the checkpoint state must be stored remotely over the network. While the use of spare cycles on unused workstations only impacts those workstations, over-utilization of a shared network resource will negatively impact the performance of all workstations, whether they are actively contributing to the cycle-harvesting system or not. In addition, while the typical dedicated compute cluster commonly includes a dedicated high-performance network which can handle the load of a parallel application checkpoint, resource-harvesting clusters are commonly connected via a relatively low-speed shared network, and thus more intelligent network usage strategies are required to get high performance during parallel checkpointing.

In this paper, we investigate the effects of optimizing checkpoint overhead on both application execution performance and network load in resource-harvesting settings. Our work automatically derives a checkpoint schedule for an application, based on the amount of state that must be saved in each checkpoint and the historically observed failure and reclamation behavior of each resource it uses. When an application is assigned to a resource by the resource-harvesting system, our system automatically computes a checkpoint schedule.

We use a Markov model to develop a checkpoint schedule "on the fly" for applications running in the Condor [23] environment. We investigate how the choice of distribution that our system automatically fits (which is either Weibull, hyperexponential, or exponential) to historical availability data affects the efficiency of the schedule. We describe the methodology and its implementation, detail its performance for Condor, probe a range of performance response using trace-based sim-

ulation, and validate the simulation against observation. Our results indicate that the choice of parametric distribution affects application execution performance to a small degree but has a dramatic effect on induced network load.

## 2 Related Work

In this paper, we are building upon a great deal of work from two separate but related fields. The first field of interest is that of modeling machine failure/availability distributions. Work such as [7, 10, 22, 30, 31] typically assume an exponential distribution to model machine lifetime data for use in their applications. Usually the exponential is chosen due to the relative simplicity of the distribution as opposed to an actual belief that the exponential represents the data accurately. Other works, most notably works by Long *et. al.* [14] and Plank *et. al.* [17, 18] show that indeed the exponential is a poor fit to their data; sometimes, however, as in [17], the poor fit does not significantly impact their application of the model. In [9, 28], researchers have suggested the use of a Weibull distribution to model machine availability durations but provide no quantitative measure of goodness-of-fit. Others, including [12, 16], show that hyperexponential distributions accurately model certain characteristics of their machine failure data.

The second field of interest for this paper is that of solving the problem of optimal checkpoint interval selection [4]. A great deal of literature has been written on this topic, more than we can comment on here, but we attempt to give the reader some primary and more recent reading. Fundamental work was done on finding optimal checkpoint intervals on transaction processing systems [2]. The work continued, shifting focus to high-performance computing environments and distributed systems [25, 27]. Work in this area is typically predicated on one of two simplifying assumptions. Most authors have assumed that the distribution of availability times is exponen-

tial, because the PDF and CDF formulas are simple enough to allow closed-form solutions to the equations involved in finding optimal checkpoint intervals. Other authors, such as Tantawi and Ruschitzka [24] and Ling *et. al.* [13], consider the problem for general availability distributions, but they make the common assumption that failures do not occur during a checkpoint or recovery in order to produce expressions that are analytically intractable. While assuming no failures during checkpoint or recovery is justified when individual checkpoint and recovery times are insignificant compared to time taken performing computation, we feel the assumption is too restrictive to be applied generally. Clusters provided by cycle-harvesting environments, the focus of this work, provide an example of a system in which a job may be required to make large checkpoints over the network during relatively small availability durations.

In this work, we build upon the checkpoint interval model described by Vaidya [26], without however making the assumption that availability is modeled by an exponential distribution. Since Vaidya's model makes no inherent assumptions regarding failures during recovery or checkpointing, our approach is free of both of the simplifying assumptions discussed above.

# 3 Methodology

In this section, we give a brief description of the statistical methods we use to characterize resource availability. These include the strategies used for fitting statistical distributions to data and the Markov model we use to derive the formula for checkpoint overhead to be minimized.

## 3.1 Fitting a Distribution to Availability Data

In this study, we consider three families of distributions: exponential, Weibull, and hyperexponential. The exponential distribution has been used extensively to model resource availability

because it is computationally simple to use and because of its "memoryless" property, as discussed below, which allows one to specify a single checkpoint interval throughout the execution of a job.

In contrast, the two distribution families that consistently fit the data we have gathered most accurately are the Weibull and the hyperexponential. The *Weibull distribution* is often used to model the lifetimes of objects, including physical system components [20, 1]. Hyperexponentials have been used to model machine availability previously [16], but it is numerically difficult to find estimators which have statistically desirable properties for their parameters.

### 3.2 Probability Function Definitions

We will denote probability density functions using lower-case $f$ and distribution functions using upper-case $F$. For an exponential distribution, the probability density function $f_E$ and distribution function $F_E$ are given respectively as

$$f_E(x) = \lambda e^{-\lambda x} \tag{1}$$

$$F_E(x) = 1 - e^{-\lambda x} \tag{2}$$

where $\lambda$ is a positive real number.

The density and distribution functions $f_W$ and $F_W$ respectively for a Weibull distribution are given by

$$f_W(x) = \alpha\beta^{-\alpha}x^{\alpha-1}e^{-(x/\beta)^\alpha} \tag{3}$$

$$F_W(x) = 1 - e^{-(x/\beta)^\alpha} \tag{4}$$

The parameter $\alpha > 0$ is called the *shape* parameter, and $\beta > 0$ is called the *scale* parameter. [1] When $\alpha = 1$, the Weibull reduces to an exponential distribution.

[1] The general Weibull density function has a third parameter for *location*, which we can ignore since our data has minimum values close to 0.

Hyperexponentials are distributions formed as the weighted sum of exponentials, each having a different parameter. The density function is given by

$$f_H(x) = \sum_{i=1}^{k} [p_i \cdot f_{E_i}(x)],\ x \geq 0 \tag{5}$$

where

$$f_{E_i}(x) = \lambda_i e^{-\lambda_i x} \tag{6}$$

defines the density function for an exponential having parameter $\lambda_i$. In the definition of $f_H(x)$, all $\lambda_i \neq \lambda_j$ for $i \neq j$, and $\sum_{i=1}^{k} p_i = 1$. The distribution function is defined as

$$F_H(x) = 1 - \sum_{i=1}^{k} p_i \cdot e^{-\lambda_i x} \tag{7}$$

Note that for a hyperexponential distribution, one must first specify the number of phases $k$; the distribution is then determined by an additional $2k - 1$ parameters, namely the $\lambda_i$ and all but one of the $p_i$.

### 3.3 The Distribution of Future Lifetimes

Suppose that resource availability lifetimes are represented as a random variable $X$ with probability distribution $F$, and let $t$ be a nonnegative real number. It is natural to consider the distribution of future lifetimes beyond $t$, which we will denote by $F_t(x)$, based on the conditional distribution function of $F$ given that $X \geq t$. Specifically,

$$F_t(x) = F_{X \geq t}(t + x) = \frac{F(t + x) - F(t)}{1 - F(t)},\ \ t \geq 0 \tag{8}$$

computes the probability that a resource will fail within the next $x$ seconds given that it has been available for $t$ seconds. Thus when an application is assigned to a resource, and at any point in time thereafter, we can compute the probability it will be terminated within the next $x$ seconds,

assuming the model distribution to be accurate and given the amount of time the resource has already been available.

In the case of an exponential distribution, the distribution of future lifetimes $(F_E)_t$ reduces to the original distribution for all values of $t$. This is referred to as the *memoryless* property of the exponential.

The future lifetime distribution for a Weibull reduces to

$$(F_W)_t(x) = 1 - e^{[(t/\beta)^\alpha - (x/\beta)^\alpha]}. \tag{9}$$

The future lifetime distribution for the hyperexponential distribution defined above is

$$(F_H)_t(x) = 1 - \left( \frac{\sum_{i=1}^{k} p_i \cdot e^{-\lambda_i (t+x)}}{\sum_{i=1}^{k} p_i \cdot e^{-\lambda_i (x)}} \right) \tag{10}$$

### 3.4 Parameter Estimation

Each of the above-mentioned statistical distributions involves some number of unspecified parameters which must be estimated in order to "fit" a particular distribution to the observed data. The generally accepted approach to the general problem of parameter estimation is based on the principle of *maximum likelihood*. The maximum likelihood estimator (MLE) is calculated for any data set, based on the assumptions that each of the sample data points $x_i$ is drawn from a random variable $X_i$ and that the $X_i$ are independent and identically distributed (i.i.d.). We use Matlab [15] to estimate MLE parameters for exponential and Weibull distributions throughout. Finding MLE parameters for hyperexponential distributions is somewhat more difficult. Therefore, we use the EMPht software package [5] in place of the Matlab MLE routine for all estimated hyperexponential parameters in this paper. We have implemented a software system that takes a set of measurements as inputs and computes Weibull, exponential, and hyperexponential parameters automatically.

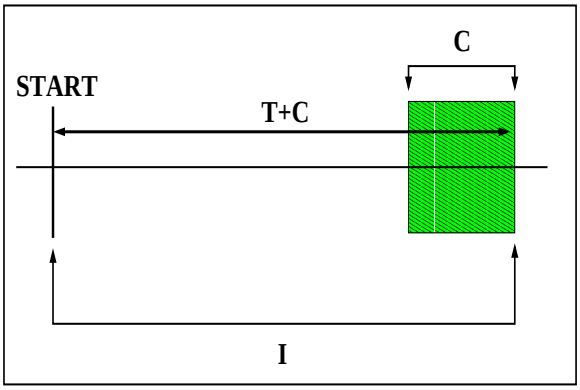


**Figure 1. Diagram of a single checkpoint interval. We wish to find the value of $T$ which maximizes time spent performing useful computation on volatile resources.**

### 3.5 Optimal Checkpoint Intervals

The optimal interval between checkpoints in an application execution balances the cost of checkpointing (lost execution time while each checkpoint is generated) with the cost of recovering from a failure by restoring execution from the last checkpoint. While many solutions to this problem have been proposed, we have chosen to use the description due to Vaidya [26] for this work because it takes into account that failures may occur during both checkpoint and recovery phases. In a resource-harvesting context, we believe that failure (due to reclamation) during checkpointing and/or recovery is significantly more likely than in other settings.

The interval between checkpoints is composed of a computation phase whose duration is $T$ seconds (to which we will sometimes refer as the *work time interval*) and a checkpoint phase using $C$ seconds. If the application is restarting after a failure, it will begin with a recovery phase of duration $R$ seconds. Figure 3.5 shows the decomposition of a single checkpoint interval. Note that we are making the explicit assumption that recovery, computation, and checkpointing occur sequentially without overlap.

With this phased model of application execution, one can compute the expected time spent during the work associated with a single checkpoint interval using a three-state Markov model (shown in Figure 2). The transition probabilities between the states depend on both the work time interval selected and the statistical model chosen for the future lifetime distribution. These probabilities $P_{ij}$ and the cost functions $K_{ij}$ associated with moving from state $i$ to state $j$ are given by the following equations, in which $\Lambda$ represents the set of parameters intrinsic to the distributions used to model resource availability.

$$P_{01}(\Lambda, C, T) = 1 - F(\Lambda, C + T)$$

$$K_{01}(C, T) = C + T$$

$$P_{02}(\Lambda, C, T) = F(\Lambda, C + T)$$

$$K_{02}(\Lambda, C, T) = \int_0^{C+T} \frac{t \cdot f(\Lambda, t)}{F(\Lambda, C + T)} dt$$

$$P_{21}(\Lambda, R, T, L) = 1 - F(\Lambda, L + R + T)$$

$$K_{21}(R, T, L) = L + R + T$$

$$P_{22}(\Lambda, R, T, L) = F(\Lambda, L + R + T)$$

$$K_{22}(\Lambda, R, T, L) = \int_0^{L+R+T} \frac{t \cdot f(\Lambda, t)}{F(\Lambda, L + R + T)} dt$$

Vaidya's work defines these functions explicitly in terms of the exponential distribution and uses them to calculate a single periodic (because the exponential is memoryless) checkpoint interval for the application's execution duration.

Our work generalizes this approach to use other distributions (in our case, Weibull and hyper-exponential, but in fact one can use any family of distributions in this context, as long as one

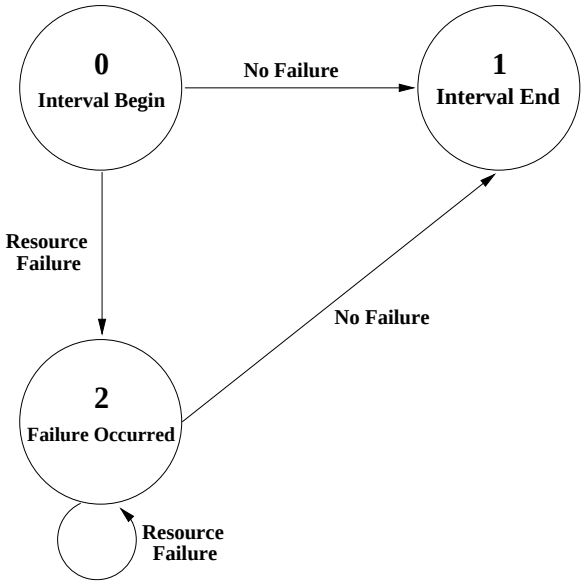


**Figure 2. Three-state Markov model describing a single checkpoint interval in a long-running job.**

has a method for estimating parameters and evaluating the above expressions numerically) and to compute a checkpoint schedule as a sequence of intervals for the application.

Denote by $\Gamma$ the expected value of the amount of time to move from state $0$ to state $1$ in the Markov model. $\Gamma$ can be calculated from the above formulas as

$$\Gamma = P_{01} \cdot K_{01} + P_{02} \cdot (K_{02} + K_{22} \cdot \frac{P_{22}}{P_{21}} + K_{20}) \tag{11}$$

Note that $\frac{\Gamma}{T}$ measures the factor by which we can expect the amount of time spending useful work to be multiplied within a work time interval. Therefore the problem of finding an optimal work time interval can be expressed as the problem of minimizing $\frac{\Gamma}{T}$ with respect to $T$. We use the Golden Section Search method as implemented in Numerical Recipes [19] for this optimization problem. Define $T_{opt}$ to be the optimal work time interval.

Note that the probability and cost calculations made above must be made considering *future* life-

time distributions. Therefore, when calculating $P_{01}, K_{01}, P_{02}$, and $K_{02}$ in the cases of the Weibull and hyperexponential distributions, we must take into account the amount of time that the *specific resource* that the application is using has already been available, since, as noted above, the future lifetime distribution changes as time passes. On the other hand, since a failure has just occurred if we are in state 2, the other $P_{ij}$ and $K_{ij}$ formulas are calculated using the ordinary unconditional versions of the distributions.

From the same considerations, note that if we use a non-memoryless distribution as our model for resource availability, we obtain an aperiodic schedule of $T_{opt}$ values rather than a single value. This schedule takes the form of a sequence of $T_{opt}$ values computed from the beginning of the application's execution time. We denote $T_{opt(i)}$ to be the $i$th such value. $T_{opt(0)}$ is the first interval and it is computed for the time that the application is initiated using the amount of time, denoted $T_{elapsed}$, that has elapsed since the resource running the application has failed. Each successive value of $T_{opt(i)}$ can then be computed based on the amount of time the resource will have been available at the beginning of each work time interval. Note that the schedule remains valid for as long as the resource is available without interruption. After a failure occurs, of course, we need to calculate a new schedule of $T_{opt}$ values.

The schedule we derive in this way is "optimal" in the same sense that Vaidya's model is optimal: Given the information we have at the beginning of execution, and assuming the accuracy of our model, this schedule minimizes $\frac{\Gamma}{T}$ and thus is the best we can do; in fact, the schedule is optimal at the beginning of each checkpoint interval, again assuming our model, in which checkpoint and recovery durations are known constants.

We have written a small, portable routine which implements the evaluation and optimization of $\frac{\Gamma}{T}$ to find $T_{opt}$, taking as input the distribution model chosen, the distribution parameters, the value

of $T_{elapsed}$ (ignored in the case of exponential distributions), and values for $C$ and $R$.

## 4 The Condor System

Condor [3, 23] is a resource-harvesting system designed to support high-throughput computing. It runs as a privileged process on desktop workstations and accepts "batch" job submissions from Condor users. When Condor detects that a machine has become idle (i.e. is not being used by its owner) it initiates a submitted job on the workstation and captures its I/O. Should the owner reclaim the workstation (by moving the mouse, typing at the keyboard, or having a local job start and change the load average), the Condor job is either automatically checkpointed and evacuated (in the *Standard Universe*) or terminated for later restart (in the *Vanilla Universe*).

In this study, we take advantage of the Vanilla (*i.e.,* terminate-on-eviction) execution environment to build a Condor occupancy monitor. A set of monitor processes is submitted to Condor for execution. When Condor assigns a process to a processor, the process wakes periodically and reports the number of seconds that have elapsed since it began executing. When that process is terminated (due to an eviction) the last recorded elapsed time value measures the duration of the occupancy the sensor enjoyed on the processor it was using. For each machine Condor uses to run one of our sensors our system records a sequence of availability durations and time stamps (in UTC units) indicating when those durations occurred.

In this study, Condor used a pool of over 1000 different Linux workstations to run the monitor processes over an $18$-month long measurement period which is ongoing, of which we obtained data for approximately 640 machines. Thus the model fitting component of our overall system computes distributions from up-to-date availability measurements that cover the period from April 2003 until October 2004.

# 5 Experimental Evaluation

We evaluate the method we have outlined in two ways, both of which use the Condor resource-harvesting system as a target execution platform. First, we use discrete event simulation, based on the execution traces gathered from Condor, to compare the effectiveness of exponential, Weibull, and hyperexponential models. Second, we examine their respective effectiveness using the "live" Condor system and a test application. We repeatedly launch the application in Condor, and when it is given access to a host, we compute checkpoint intervals for that host based on availability data we have recorded over the previous 18 months. Because the conditions change from execution run to execution run, we compare the *in vivo* results in terms of their average time and network bandwidth efficiency. Finally, for completeness, we verify the simulations using post-mortem trace data we record during the actual Condor runs.

## 5.1 Simulation Method and Results

For the machines which the Condor scheduler chose to execute our monitoring process on a sufficient number of times over 18 months, we divide each trace in to a "training set" containing the first $25$ values occurring chronologically and an "experimental set" containing the remaining values. We then use each training set to calculate MLE parameters for an exponential model and a Weibull model and EM parameters for both $2$-phase and $3$-phase hyperexponential models. Thus, we model the training set for each trace by four different distributions.

In order to capture "steady-state" efficiency, we simulate a job that begins before the first measurement in each training set and continues to run after the last measurement in the experimental set. Note that for each work time interval, the expected efficiency is just the reciprocal of the quantity $\Gamma$, defined in the "Methodology" section, evaluated at $T_{opt}$.

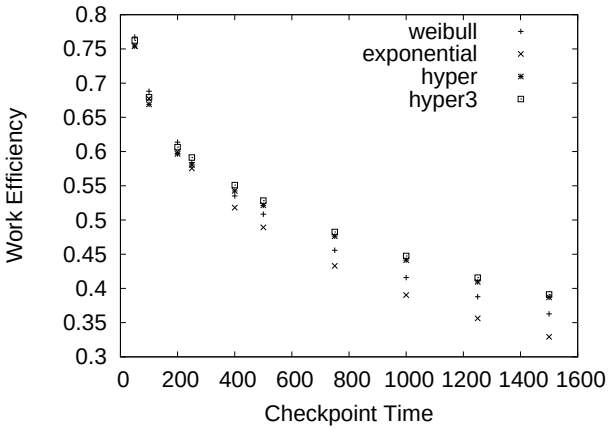


**Figure 3. Simulation experiment: average percent machine utilization using exponential, Weibull, and hyperexponential distributions to compute checkpoint schedules.**

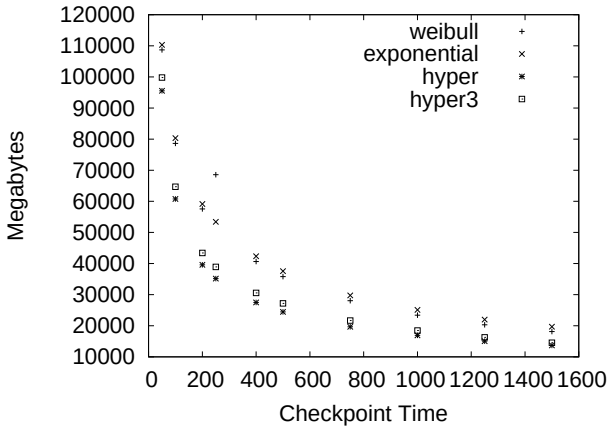


**Figure 4. Bandwidth consumed: average network load (in megabytes) when exponential, Weibull, and hyperexponential distributions to compute checkpoint schedules when each checkpoint is $500$ megabytes in size.**

In order to compare the effectiveness of each model, we consider both application efficiency and the amount of network load generated when each distribution is used to compute a checkpoint schedule. We assume checkpoint cost $C$ is equal to the recovery cost $R$ since this assignment reflects our experience with executing long-running jobs in the vanilla Condor universe.

Figure 3 shows the results of our simulation using fits of exponential, Weibull, and hyperexponential models to machine availability. On the $x$-axis we indicate the time, in seconds, necessary to effect one checkpoint or recovery, and on the $y$-axis we show the fraction of time the application spends doing useful work. Because the overhead varies by machine, each data point represents the average overhead ratio across all machines in the simulation. The figure shows that all four distributions yield approximately the same average efficiency across machines. In Table 1 we show each average from the figure along with its 95% confidence interval.

We indicate statistically significant differences within each row between the results for two dis-

| CTime | Exp. | Weib. | 2-phase Hyperexp. | 3-phase Hyperexp. |
|---|---|---|---|---|
| 50 | $0.754 \pm 0.013$ | $0.767 \pm 0.012$ (e,2,3) | $0.754 \pm 0.014$ | $0.762 \pm 0.013$ |
| 100 | $0.677 \pm 0.017$ (2) | $0.688 \pm 0.016$ (e,2,3) | $0.669 \pm 0.017$ | $0.679 \pm 0.017$ (2) |
| 200 | $0.600 \pm 0.020$ | $0.614 \pm 0.019$ (e,2,3) | $0.597 \pm 0.020$ | $0.606 \pm 0.020$ (2) |
| 250 | $0.576 \pm 0.020$ | $0.584 \pm 0.021$ | $0.581 \pm 0.020$ (e) | $0.591 \pm 0.020$ (e,2) |
| 400 | $0.518 \pm 0.020$ | $0.535 \pm 0.020$ (e) | $0.543 \pm 0.020$ (e) | $0.551 \pm 0.020$ (e,w,2) |
| 500 | $0.489 \pm 0.020$ | $0.508 \pm 0.021$ (e) | $0.521 \pm 0.020$ (e,w) | $0.528 \pm 0.020$ (e,w,2) |
| 750 | $0.433 \pm 0.020$ | $0.456 \pm 0.021$ (e) | $0.476 \pm 0.021$ (e,w) | $0.483 \pm 0.021$ (e,w) |
| 1000 | $0.390 \pm 0.020$ | $0.416 \pm 0.021$ (e) | $0.441 \pm 0.021$ (e,w) | $0.447 \pm 0.021$ (e,w) |
| 1250 | $0.356 \pm 0.020$ | $0.388 \pm 0.020$ (e) | $0.409 \pm 0.021$ (e,w) | $0.416 \pm 0.021$ (e,w,2) |
| 1500 | $0.329 \pm 0.019$ | $0.363 \pm 0.020$ (e) | $0.387 \pm 0.021$ (e,w) | $0.391 \pm 0.021$ (e,w) |

**Table 1.** $95\%$ **confidence intervals for mean efficiency for various checkpoint sizes of the four tested distributions.**

tributions by placing within each cell symbols standing for any distributions whose efficiencies were statistically significantly smaller for that checkpoint duration. (In this scheme, "e" stands for exponential, "w" for Weibull, "2" for 2-phase hyperexponential, and "3" for 3-phase hyperexponential.) For example, the (e,w) in the $500$-second row of the 2-phase hyperexponential column indicates that the efficiency for the 2-phase hyperexponential with a checkpoint duration of $500$ seconds is statistically significantly larger than those for exponential and Weibull distributions with the same checkpoint duration; on the other hand, the absence of such symbols in the $250$-second Weibull cell indicates that its value is not statistically significantly larger than those for any of the other distributions. We measure statistical significance using two-sided paired $t$-tests between each pair of distributions at each checkpoint duration, at a significance level of $.05$.

Although the differences are small within some rows, the table shows that for checkpoint durations shorter than $250$ seconds, the Weibull-based checkpoint schedule outperforms the others. For longer checkpoint intervals, the 3-phase hyperexponential generally does best. These results tend to support those reported in [17], in which the authors assert that an exponential model of machine

| Distribution | C=50 All | C=50 First 25 | C=500 All | C=500 First 25 |
|---|---|---|---|---|
| Exponential | .896 | .896 | .695 | .695 |
| Weibull | .891 | .891 | .685 | .691 |
| 2-Phase Hyper | .862 | .817 | .690 | .671 |
| 3-Phase Hyper | .895 | .897 | .670 | .695 |

**Table 2. Application efficiency when machine availability is defined by a Weibull distribution with shape = $0.43$ and scale = $3409$. We show simulation results with $C = 50$ and $C = 500$, using all $5000$ data points to model fit and only the first $25$ values to model fit.**

availability can be used to develop a checkpoint schedule that is close to optimal.

To quantify the difference as precisely as possible, we generate a synthetic machine availability trace containing $5000$ values in which each availability duration is drawn randomly from a heavy-tailed Weibull distribution with known parameters. To determine the parameters, we compute the MLE Weibull parameter values for a machine trace chosen at random. For the chosen machine, the MLE value for the shape parameter $\alpha$ is $0.43$, and that for the scale parameter $\beta$ is $3409$.

Using our synthetic trace, we repeat the simulation experiment using $C = 50$ and $C = 500$ to reflect empirical observations. Table 2 details the results. The Weibull model used to compute checkpoint intervals is precisely the same model that was used to generate the artificial trace. For the exponential cases, we use MLE-determined model and for the hyperexponentials we use the EM-determined models. Thus, the Weibull results are optimal and the others are approximate. Clearly, using either an exponential or hyperexponential to model the heavy-tailed Weibull data causes only a slight loss of efficiency. Moreover, using only the first $25$ values does not degrade the accuracy with which each approximates the Weibull-generated trace, as shown in Table 2.

While the different statistical models of machine availability yield approximately the same application efficiency, they do not result in the same amount of generated network traffic. In Figure 4

| CTime | Exp. | Weib. | 2-phase Hyperexp. | 3-phase Hyperexp. |
|---|---|---|---|---|
| 50 | $110296 \pm 10317$ (2,3) | $108687 \pm 11448$ (2,3) | $95535 \pm 8952$ | $99788 \pm 10495$ |
| 100 | $80323 \pm 7400$ (2,3) | $78638 \pm 8163$ (2,3) | $60777 \pm 5740$ | $64692 \pm 7306$ |
| 200 | $59153 \pm 5317$ (2,3) | $57557 \pm 5820$ (2,3) | $39603 \pm 3641$ | $43415 \pm 5122$ (2) |
| 250 | $53404 \pm 4775$ (2,3) | $68561 \pm 17864$ (2,3) | $35171 \pm 3225$ | $38926 \pm 4601$ (2) |
| 400 | $42350 \pm 3802$ (2,3) | $40638 \pm 4125$ (2,3) | $27487 \pm 2494$ | $30553 \pm 3645$ (2) |
| 500 | $37546 \pm 3407$ (2,3) | $35809 \pm 3678$ (2,3) | $24474 \pm 2248$ | $27193 \pm 3291$ (2) |
| 750 | $29746 \pm 2794$ (2,3) | $28041 \pm 3002$ (2,3) | $19664 \pm 1868$ | $21671 \pm 2673$ (2) |
| 1000 | $25099 \pm 2427$ (2,3) | $23398 \pm 2590$ (2,3) | $16897 \pm 1652$ | $18458 \pm 2314$ (2) |
| 1250 | $21970 \pm 2172$ (w,2,3) | $20310 \pm 2308$ (2,3) | $15031 \pm 1502$ | $16262 \pm 2065$ |
| 1500 | $19693 \pm 1983$ (w,2,3) | $18137 \pm 2112$ (2,3) | $13671 \pm 1390$ | $14549 \pm 1860$ |

**Table 3.** $95\%$ **confidence intervals for mean bandwdith for various checkpoint sizes of the four tested distributions.**

we show the number of megabytes transferred if each checkpoint were $500$ megabytes in length as a function of checkpoint duration.

As in Figure 3, along the $x$-axis we show the time required to checkpoint or recover, but on the $y$-axis we show the average number of megabytes that traversed the network. Note that the duration $C$ or $R$ associated with a $500$-megabyte transfer depends on the speed of the network linking the resource with the checkpoint storage location. As described in the next subsection, most of the available machines in the Condor cluster have at least $512$ megabytes of memory, motivating our choice of $500$ megabytes as a representative size. Table 3 shows the average values, their respective $95\%$ confidence intervals, and notation for statistically significant differences within a single row identical to that in Table 1. Note that in this table, significantly larger values are undesirable, as they correspond to more consumed bandwidth.

From the table, we see that the exponential-based checkpoint schedule significantly (and substantially) underperforms all of the other approaches. The most bandwidth-parsimonious approach

is that of the 2-phase hyperexponential, which used at least $30\%$ less bandwidth than the exponential for checkpoint overheads $\geq 200$ seconds.

From these results, we conclude that while the application efficiency is good for all availability models, there is a noticeable difference in the network load generated by the different models.

The reason for this difference is that checkpoint overhead comprises both the delay associated with a checkpoint or recovery and the amount of lost work that must be recomputed from the last successful checkpoint when a failure occurs. The heavy-tailed models tend to produce longer intervals between checkpoints, which results in fewer checkpoints generated but more lost work on average at the time of each failure. On the other hand, the exponential model favors shorter intervals, more checkpoints, and less lost work. It is curious that these factors balance almost precisely to produce the same application efficiency in the range of checkpoint costs we have investigated.

### 5.2 Empirical Method and Results

While the simulation permits a quantitative comparison of exponential, Weibull, and hyperexponential models using the same machine traces, it includes several simplifications that could affect the results in practice. First of all, the checkpoint and recovery costs ($C$ and $R$ respectively) are held constant in each simulation. Variation of network performance, particularly in the wide area, makes these costs variable when the system is actually used. Also, Condor imposes some additional overhead at job start-up and termination that is difficult or impossible to determine externally using our measurement methodology. Finally, if the models we use are sensitive to inaccuracies in the parameters supplied to them, the simulation results could be misleading.

To gauge the impact of these issues, we have developed an instrumented test process that im-

plements the recovery-execution-checkpoint cycle that we have simulated. In the experiment, we repeatedly submit copies of the test process to Condor. When Condor assigns a process to a machine, the process opens a network connection to a checkpoint manager. The checkpoint manager initiates a $500$-megabyte transfer to the process in order to emulate an initial recovery of the available memory, and the test process times the transfer [2]. If the test process is terminated during the initial transfer, the checkpoint manager detects the failed connection and records the amount of time as recovery overhead. We choose $500$ megabytes for two reasons: Our target application requires this size checkpoint; and the Condor machines we used had all had at least $512$ megabytes of memory.

As part of the initial transfer, the checkpoint manager also sends the test process a message indicating which model to use to determine a checkpoint schedule and the parameters for that model. Using the initial transfer time as a measurement of $R$ and $C$, the test process then computes one checkpoint interval $T_{opt}$ using the specified model and sends the quantities to the checkpoint manager for logging. It then begins emulating a computation by spinning in a tight loop, which it interrupts every 10 seconds so that it can send the checkpoint manager a heartbeat message. The heartbeat message contains the cumulative time since the process began running, which the checkpoint manager records as execution time. If the job is terminated, the trace of heartbeats simply ends. At the end of the interval, if the process has not been terminated, it transfers $500$ megabytes back to the checkpoint manager to emulate a checkpoint, which it also times. This new time is used as a current measurement of $C$, and $R$, and it computes, based on these values and the amount of time it has been running, a new checkpoint interval $T_{opt}$, and sends this data

[2] Strictly speaking, it records the time from when it sends a request for recovery to the checkpoint manager until the transfer completes, but the latency of the initial request is insignificant compared with the time for the data transfer.

| Distribution | Avg. | Total Time | Megabytes Used | Megabytes/Hour | Sample Size |
|---|---|---|---|---|---|
| Exponential | .680 | 749695 | 338420 | 3842 | 81 |
| Weibull | .689 | 768808 | 363356 | 2734 | 85 |
| 2-phase Hyper. | .726 | 789304 | 150166 | 1313 | 84 |
| 3-phase Hyper. | .676 | 718094 | 329034 | 2374 | 89 |

**Table 4. Average application efficiency and bandwidth consumed using four distributions and the Condor cluster with the checkpoint manager located at the University of Wisconsin.**

| Distribution | Avg. | Total Time | Megabytes Used | Megabytes/Hour | Sample Size |
|---|---|---|---|---|---|
| Exponential | .629 | 491048 | 183339 | 1344 | 40 |
| Weibull | .590 | 491900 | 167195 | 1223 | 48 |
| 2-phase Hyper. | .659 | 491454 | 96264 | 705 | 56 |
| 3-phase Hyper. | .604 | 428626 | 110920 | 931 | 59 |

**Table 5. Average application efficiency and bandwidth consumed using four distributions and the Condor cluster with the checkpoint manager located at our home institution.**

to the manager for logging before it begins emulating computation again. If the transfer back to the checkpoint manager is interrupted, the manager records the elapsed transfer time as checkpoint overhead. The manager keeps a log file for each test process from which the overhead ratio can be calculated *post facto*.

Tables 4 and 5 show the results of the Condor experiment in terms of the average application efficiency we observed across all machines in two different situations. To generate the results in Table 4, we locate the checkpoint manager on a machine at the University of Wisconsin so that all checkpoint traffic would traverse only the campus network. During the experiment, the average checkpoint time is 110 seconds. Thus the efficiency values in column 1 may be compared to row 2 of Table 1.

Column 2 of Table 4 indicates the total execution time for the test application and column 3 shows the number intervals we computed for each method. Table 5 shows the same data, but

for a configuration in which the checkpoint manager is located at our home institution, which is separated from the University of Wisconsin by the Internet. The average checkpoint duration in this case is $475$ seconds, making these results most comparable to row $6$ of Table 1. As the previous simulations indicate, as the application runs for longer and longer periods, the values will converge to the same average efficiency.

Table 4 also shows the average network loads observed for the various statistical models when the checkpoint manager is located at the University of Wisconsin; Table 5 shows the same numbers with the checkpoint manager at our home institution. The third column of each table may be compared to the simulation results shown in rows $6$ and $2$, respectively, of Table 3, because the average checkpoint times are similar to the parameters set in these rows. The third column indicates the total network load, and the forth column reports the average number of megabytes per hour transferred in each case. These results confirm the phenomena that we observed in our simulation data, namely that the differences among the various distributions are relatively small for our application efficiency metric but quite considerable for network usage.

Note that for an individual job, the network savings are likely to be important to the site network administrators (since non-Condor users will see a less congested network). For a parallel job, however, where multiple jobs may be checkpointing simultaneously, the network load savings are likely to improve application efficiency since network collisions will lengthen the amount of time necessary for a checkpoint. We are considering a model of parallel workload that captures the interaction between colliding checkpoints and checkpoint length as part of our future work. In this study, however, the heavy-tailed models parallelize the overhead by incurring it as lost execution work and not sequential network load.

### 5.3 Validating the Simulation

In terms of *validation*, we noticed some discrepancies which contribute to the differences between the empirical results and the results predicted by the simulation (Tables 4 and 1 respectively). First the experimental period only spanned 2 days while the training set spanned 18 months, which tends to *right censor* the data. Second, the Markov model uses constant values of $C$ and $R$ while in reality these values are variable. Close analysis leads us to believe that these factors are not drastically effecting the simulations, but do explain small discrepancies between simulation and empirical results.

## 6 Conclusion

In our work, we examine the effectiveness of four different probability distributions – exponential, Weibull, 2-phase hyperexponential, and 3-phase hyperexponential – as the basis for determining checkpoint schedules. We use availability traces taken from the Condor resource-harvesting system at the University of Wisconsin and simulate both application efficiency and generated network load. We also conduct experiments with the "live" Condor system in which we observe both efficiency and load for a test application that can use different models to compute its checkpoint schedule. Finally, we verify and validate our simulations against the empirical data we have gathered.

Our results indicate that application efficiency is relatively insensitive to the choice of probability distribution (among those we investigate) used to model resource availability. While the differences in average efficiency *are* for the most part statistically significant, they are small in absolute terms for the instances we examine. However, the average network loads generated by schedules derived

from the different distributions are substantially different. In particular, the checkpoint schedule generated from the 2-phase hyperexponential results in considerably less bandwidth consumption than when either the exponential or Weibull are used, and slightly less than that of the 3-phase hyperexponential. Moreover, as the duration of checkpoint and recovery increases, differences in network load become more pronounced. Providing a model which greatly reduces network bandwidth is particularly important for cycle-harvesting systems since the network is a shared resource which, unlike unused workstation cycles, cannot be fairly allocated. Additionally, when loosely coupled resources are combined to form a cluster on which parallel applications can execute, careful usage of the network is crucial to attaining high performance. Thus we conclude that from the perspective of an application user or designer, the choice of availability distribution has little effect on perceived efficiency, but from the perspective of the resource and network administration, heavy-tailed hyperexponential distributions yield considerably better results.

## References


[1] C. E. Beldica, H. H. Hilton, and R. L. Hinrichsen. Viscoelastic beam damping and piezoelectric control of deformations, probabalistic failures and survival times.

[2] K. M. Chandy and C. V. Ramamoorthy. Rollback and recovery strategies for computer programs. *IEEE Trans. on Computers*, 2:546–556, June 1972.

[3] Condor home page – `http://www.cs.wisc.edu/condor/`.

[4] M. Elnozahy, L. Alvisi, Y. M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message passing systems. Technical Report CMU-CS-96-181, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, Oct. 1996.

[5] Empht home page. Available on the World-Wide-Web. `http://www.maths.lth.se/matstat/staff/asmus/pspapers.html`.

[6] The Entropia Home Page. `http://www.entropia.com`.

[7] A. L. Goel. Software reliability models: Assumptions, limitations, and applicability. In *IEEE Trans. Software Engineering, vol SE-11, pp 1411-1423*, Dec 1985.

[8] A. Gupta, B. Lin, and P. Dinda. Measuring and understanding user comfort with resource borrowing. In *HPDC-13*, 2004.

[9] T. Heath, P. M. Martin, and T. D. Nguyen. The shape of failure.

[10] R. K. Iyer and D. J. Rossetti. Effect of system workload on operating system reliabilty: A study on IBM 3081. In *IEEE Trans. Software Engineering, vol SE-11, pp 1438-1448*, Dec 1985.

[11] D. Kondo, M. Taufer, C. Brooks, H. Casanova, and A. Chien. Characterizing and Evaluating Desktop Grids: An Empirical Study. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'04), Santa Fe, NM*, April 2004.

[12] I. Lee, D. Tang, R. K. Iyer, and M. C. Hsueh. Measurement-based evaluation of operating system fault tolerance. In *IEEE Trans. on Reliability, Volume 42, Issue 2, pp 238-249*, June 1993.

[13] Y. Ling, J. Mi, and X. Lin. A variational calculus approach to optimal checkpoint placement. *IEEE Trans. on Computers*, 50:699 – 708, July 2001.

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

[15] Matlab by Mathworks. `http://www.matlab.com`.

[16] M. Mutka and M. Livny. Profiling workstations' available capacity for remote execution. In *Proceedings of Performance '87: Computer Performance Modelling, Measurement, and Evaluation, 12th IFIP WG 7.3 International Symposium*, December 1987.

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

[18] J. Plank and M. Thomason. Processor allocation and checkpoint interval selection in cluster computing systems. *Journal of Parallel and Distributed Computing*, 61(11):1570–1590, November 2001.

[19] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1992.

[20] M. H. Seo, M. L. Realff, M. C. Boyce, P. Schwartz, and S. Backer. Mechanical properties of fabrics woven from yarns produced by different spinning technologies: Yarn failure in fabric.

[21] SETI@home. `http://setiathome.ssl.berkeley.edu`, March 2001.

[22] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and K. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *In Proc. SIGCOMM (2001)*, 2001.

[23] T. Tannenbaum and M. Litzkow. The condor distributed processing system. *Dr. Dobbs Journal*, February 1995.

[24] A. Tantawi and M. Ruschitzka. Performance analysis of checkpointing strategies. *ACM Trans. Computer Systems*, 2(2):123–144, May 1984.

[25] N. Vaidya. On checkpoint latency. In *Proceedings of Pacific Rim Symposium on Fault-tolerant Systems*, December 1995.

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

[27] K. F. Wong and M. A. Franklin. Distributed computing systems and checkpointing. In *HPDC*, pages 224–233, 1993.

[28] J. Xu, Z. Kalbarczyk, and R. K. Iyer. Networked Windows NT System field failure data analysis.

[29] B. Zagrovic, C. Snow, M. Shirts, and V. Pande. Simulation of folding of a small alpha-helical protein in atomistic detail using world-wide distributed computing. *Journal of Molecular Biology*, 323:927–937, 2002.

[30] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. A resilient global-scale overlay for service deployment. *(to appear) IEEE Journal on Selected Areas in Communications*.

[31] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, U.C. Berkeley Computer Science Department, April 2001.