Model-Based Checkpoint Scheduling for Volatile Resource Environments*

UCSB Computer Science Technical Report Number 2004-25

Daniel Nurmi
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106

Rich Wolski
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106

John Brevik
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106

ABSTRACT

In this paper, we describe a system for application checkpoint scheduling in volatile resource environments. Our approach combines historical measurements of resource availability with an estimate of checkpoint/recovery delay to generate checkpoint intervals that minimize overhead.

When executing in a desktop computing or resource harvesting context, long-running applications must checkpoint, since resources can be reclaimed by their owners without warning. Our system records the historical availability from each resource and fits a statistical model to the observations using either Maximum Likelihood Estimation (MLE) or Expectation Maximization (EM). 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 overhead as a function of the checkpoint interval, and numerically optimizes this quantity.

Using Condor as a target platform, we investigate the effectiveness of this technique fitting exponential, Weibull, 2-phase hyperexponential and 3-phase hyperexponential distributions to observed availability data. To verify our method and compare the distributions each against the same conditions, we use observations taken from the Condor pool at the University of Wisconsin and trace-based simulation. We examine the practical value of our approach by observing an implementation of our system when applied to a test application that is then run on the "live" Condor system. Finally, we conclude with a verification of the simulated results against the experimental observations. Our results indicate that application efficiency is relatively insensitive to the choice of distribution (among the ones we investigate) but that induced network load is not.

General Terms

Statistical Modeling of Performance Data and Impact on Optimal Checkpoint Interval Selection

Keywords

distributed systems modeling, resource availability, statistical analysis, optimal checkpoint interval selection, aperiodic checkpointing

1. INTRODUCTION

Distributed computing systems as the source of reliable computational and storage capabilities continue to mature. However, most of these systems rely on careful administration of the resources in the system to minimize the rate of failure as a way of providing reliability to applications. For example, most successful "grid computing" [3, 10] systems use machines that are housed and maintained by professional administrative organizations as opposed to individual users [11, 24, 25, 35]. In this context, the failure rate of the resources (*e.g.* machines, networks, storage devices, *etc.*) is relatively low compared to the lifetime of the typical application.

In contrast, resource-harvesting systems such as Condor [33] SETI@Home [31], Folding@Home [40], UUCS [13], and Entropia [16, 9] offer vast computing potential by leveraging the unused capacity of more volatile desktop and "personal" computing resources. From the perspective of the programmer or user wishing to tap this potential, the proffered resources appear less stable for two reasons. First, each resource in these settings typically has a primary owner or user who maintains ultimate physical control over when and how the resource behaves. Users may reboot the machines under their control, disconnect them from the network, *etc.*, without warning or sanction. Second, resource owners must be able to reclaim their resources at will from the resource-harvesting system, since it is typically only 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 ef-

*This work was supported by National Science Foundation Grants numbered NGS-0305390 and CCR-0331654.

fect 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 [4, 7, 18, 34, 36, 37, 38]. However, checkpointing introduces an execution performance overhead on the application that can be particularly significant in desktop settings. Often, application checkpoints cannot be stored locally on the resource, either because security concerns prevent the use of local persistent storage 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 same network that permits the resource-harvesting system ingress.

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 when the application should checkpoint.

We assume that either system availability logs or active health-and-status monitoring systems such as Nagios [23] or Ganglia [20] are in place to provide a long-term record of resource availability for each resource. From an automatically derived statistical distribution (in its conditional form if it is not a "memoryless" distribution) we then compute the checkpoint schedule for the application that minimizes the expected execution time using a Markov model similar to the one proposed in [37]. This computation requires estimates of the delay imposed on an application when a checkpoint is generated and when the application recovers from a checkpoint. Additionally, when the distribution that is fit is not memoryless, the schedule calculation requires the elapsed time from the last resource failure until the start of the application as a parameter (which we assume is derivable from the availability logs).

We investigate the application of this methodology to checkpoint scheduling for the Condor [33] system using exponential, Weibull, and hyperexponential distributions, and compare the results using both simulation, and observation of test applications run under Condor. We detail the relevant specific characteristics of Condor, the techniques we use for automatic model fitting, the checkpoint scheduling algorithm and the results it generates. in terms of application execution efficiency.

While the problem of checkpoint optimization has been widely studied, and the general approach we pursue is not new, our results nonetheless make several significant contributions to this important research domain:

- We develop a new method for computing an optimal checkpoint schedule based on the original work of Vaidya [37] and later improved upon by Plank and Elwasif [27]. In particular, our method combines heavy-tailed statistical models of machine availability with the Markov model they propose to compute a checkpoint schedule that includes the possibility of failure during execution, during checkpoint, and during recovery.
- We investigate the efficacy of this new theoretically optimal model using trace-based simulation and compare it to the previously reported optimality results. Our results indicate that application performance is insensitive to the choice of distribution but that the additional network load introduced by checkpointing is not.
- We have developed a working implementation for computing checkpoint schedules that is compatible with the Condor [33] resource-harvesting system. Using this system, we report on empirical results gathered with a test application in the "live" resource-harvesting setting the Condor team maintains at the University of Wisconsin.
- We check the validity of the simulation against measurements we gather during each live experiment, so that we are able to quantify the agreement among the theory, the simulation, and practice.

We know of no other work that provides a theoretically optimal checkpoint schedule (including failure possibility during checkpoint and recovery), verifies the results in both simulation and in practice, and quantifies the differences among all three.

In addition, the utility of this investigation comprises both academic and practical engineering considerations. The Condor deployment at the University of Wisconsin is a nationally supported computing resource that we intend to use as a distributed supercomputer for two large-scale scientific calculations [1]. The implementation we have developed and the results we have gathered with it are specifically for the purpose of supporting these two application efforts.

2. RELATED WORK

There is a great deal of work that we are building upon in this paper in two separate but related fields. The first field of interest is that of modeling machine failure/availability distributions. Work such as [12, 15, 32, 41, 42] 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 [19] and Plank [27, 28] show that indeed the exponential is a poor model to employ, but sometimes, as in [27], the poor fit does not significantly impact their application of the model. In [14, 39] researchers have suggested the use of a Weibull distribution to model machine availability duarations, but do not show how well the Weibull fits the data using more than visual analysis of the fit. Others, including [17, 22], obtain good results using a hyperexponential distribution to approximate machine failure data.

The second field of interest for this paper is that solving the problem of optimal checkpoint interval selection [7]. 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

[1]We omit further details of the applications themselves for the purposes of blind submission.

on transaction processing systems [4]. The work continued, shifting focus to high performance computing environments and distributed systems [36, 38]. Work in this area is typically predicated on one of two simplifying assumptions. Most authors have assumed that the distribution of availability times is exponential, because the PDF and CDF formulas are simple enough to allow closed-form solutions to the equations involved in finding optimal checkpoint intervals. As mentioned above, it is generally accepted that this assumption is not realistic; however, the question remains whether making it has any significant impact on performance. Other authors such as Tantawi [34] and Ling [18] consider the problem for general availability distributions. Ling makes the common assumption that failures do not occur during a checkpoint or recovery in order to avoid intractable complexity in their equations while Tantawi circumvents the assumption by proposing a suboptimal expression for checkpoint interval selection. While assuming no failures during checkpoint or recovery is justified when individual checkpoint and recovery times are insignificant when compared to time taken performing computation, we feel the assumption is too restrictive to be applied generally. 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 [37], 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. Long, Muir and Golding [19] and Plank and Elwasif [27] both note inaccuracies in fitting exponentials to empirical observations of resource availability; nonetheless, its simplicity of use has led to widespread use of exponentials in the study of checkpoint scheduling.

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 [30, 2]. Hyperexponentials have been used to model machine availability previously [22], 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 lowercase f and distribution functions using upper-case F; we will subscript these letters to distinguish different families of distributions. 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 λ 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. [2] When $\alpha = 1$, the Weibull reduces to an exponential distribution.

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 λ_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 λ_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,

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

This function is useful because it 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

[2]The general Weibull density function has a third parameter for *location*, which we can eliminate from the density simply by subtracting the minimum lifetime from all measurements. In this paper, we will work with the two-parameter formulation.

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; in other words, if availability lifetimes follow an exponential distribution, the amount of time a resource has already been available has no impact on how long it is likely to remain available. For this reason, the exponential distributions are called *memoryless*; in fact, they are the only memoryless (continuous) distributions.

The future lifetime distribution for a Weibull reduces to

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

This function clearly depends on t as well as x when $\alpha \neq 1$. When $0 < \alpha < 1$, the probability that a resource will survive another time unit *increases* as t increases. For $\alpha > 1$, this probability *decreases*, and when $\alpha = 1$ the distribution reduces to an exponential and is therefore memoryless. Thus a Weibull distribution is capable of modeling different aging effects, depending on its shape parameter.

The future lifetime distribution for the hyperexponential distribution defined above is

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

A hyperexponential is only capable of modeling increasing expected lifetime. One can show this by demonstrating that the *failure rate function* $\frac{f_H(x)}{1-F_H(x)}$ is a decreasing function of time for any hyperexponential; this is a straightforward but tedious calculus exercise. Intuitively, throughout the lifetime of a hyperexponentially distributed object, the condition that it has survived as long as it has makes it increasingly probable that its lifetime is governed by one of the longer phases of the hyperexponential, and so its expected future lifetime will increase.

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. Given a set of sample data points $\{x_1 ... x_n\}$, there are many common techniques for estimating the parameters, including both visual inspection (*e.g.* using a graph) and analytic methods. 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 an that the X_i are independent and identically distributed (i.i.d.). The method defines the *likelihood function* L, depending on the parameters of the distribution, as the product of the density function evaluated at the sample points. For example, in the case of the Weibull distribution, L is a function of α and β given by

$$L(\alpha, \beta) = \prod_i f(x_i) = \prod_i \alpha \beta^{-\alpha} {x_i}^{\alpha - 1} e^{-(x/\beta)^\alpha}.$$

Intuitively (and precisely in the case of a discrete distribution), maximizing L is equivalent to maximizing the joint probability that each random variable X_i will take on the sample value x_i. Large values of the density function correspond to data that is "more likely" to occur, so larger values of L correspond to values of the parameters for which the data was "more likely" to have been produced. Thus, the MLE for the parameters is simply the choice of parameters (if it exists) which maximizes L. We find that the MLE value for λ in an exponential distribution is simply the reciprocal of the sample mean $\bar{x}$. MLE values for the Weibull parameters can be found using standard optimization routines on $\log L$.

Finding MLE parameters for hyperexponential distributions is somewhat more difficult. It is necessary to specify the number k of phases in the distribution before it even makes sense to employ MLE methods; we are then left with an optimization problem in $2k - 1$ variables. Even for small values of k, this problem is often too complex for commonly available computers to solve, especially for larger data sets.

One therefore needs to use another method in practice to estimate hyperexponential parameters. We use the EMpht software package [8] in place of the MLE approach for all estimated hyperexponentials in this paper. EMpht implements the estimation maximization (EM) algorithm described in [1]. While this method often yields a good fit with the data (as is evidenced by our results) it is not guaranteed to converge to the MLE solution. As for the problem of specifying k, we have found that, in practice, a 2- or 3-phase hyperexponential is sufficient to model accurately the data sets of interest. In fact, for many data sets the 3-phase hyperexponential produced by the EMpht software is for all practical purposes identical to a 2-phase (because two of the λ_i values become equal).

We have implemented a software system that takes a set of measurements as inputs and computes both the MLE Weibull and exponential as well as the EM-based hyperexponential automatically. Perhaps unsurprisingly, the quality of the numerical methods that we use is critical to the success of the method. In particular, the MLE computations may involve hundreds or thousands of terms (the data sets can be quite large) requiring robust and efficient techniques. At present, the implementation uses a combination of the Octave [26] numerical package, Mathematica [21] (for solver quality), and the aforementioned EMpht. The resulting system, however, takes data and automatically determines the necessary parameters for each model.

3.5 Optimal Checkpoint Intervals

The optimal interval between checkpoints in an application execution balances the cost (in terms of lost execution time while each checkpoint is generated) with the cost of recovering from a failure by restoring execution from the last checkpoint. If the application checkpoints too often, with respect to the duration of resource availability, a great deal of time is wasted taking checkpoints instead of doing useful computation. On the other hand, if the application checkpoints too infrequently, then the amount of useful work lost while the program "backs up" to the last checkpoint becomes unacceptable. While many solutions to this problem have been proposed, we have chosen to use the description due to Vaidya [37] for this work because it takes into account the possibility that a failure could occur while a checkpoint is generated, and during checkpoint recovery. In a resource-harvesting context, we believe that failure (due to reclama-

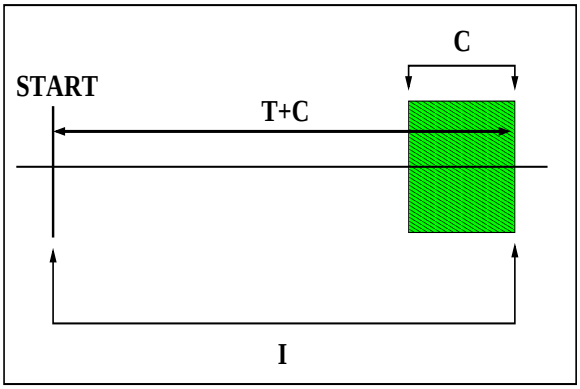

Figure 1: Diagram of a single checkpoint interval. We wish to find the value of T which maximizes time spent performing useful computation on volatile resources.

tion) during checkpointing and/or recovery is 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). This model begins in state 0 and ends at state 1; state 2 is used to model failures. Thus, from either state 0 or state 2, transition to state 1 represents completion of the interval without a failure; if there is a failure, the transition is to state 2, and the model continues until it reaches state 1. 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 and the cost functions associated with visiting each state are given by the following equations, in which Λ represents the set of parameters intrinsic to the distributions used to model resource availability.

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

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

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

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

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

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

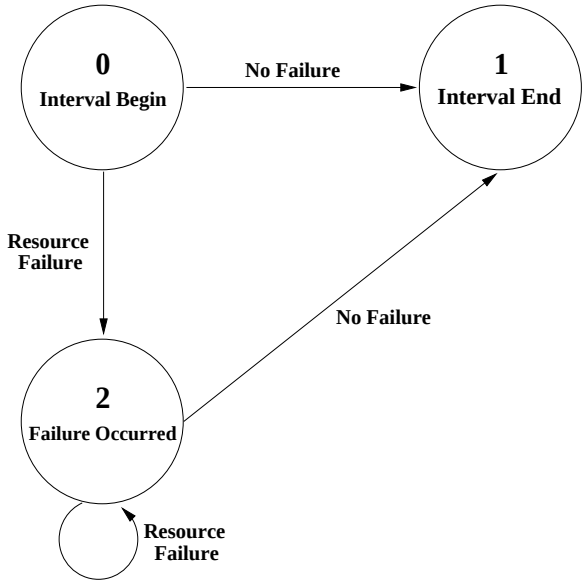

Figure 2: Three-state Markov model describing a single checkpoint interval in a long-running job.

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

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

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

Our work generalizes this approach to use other distributions (in our case, Weibull and hyperexponential, but in fact one can use any family of distributions in this context, as long as one 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 Γ the expected value of the amount of time to move from state 0 to state 1 in the Markov model. Γ can be calculated from the above formulas as

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

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 [29] 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* lifetime 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 re-*

source 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 ith 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 [5, 33] is a resource-harvesting system designed to support high-throughput computing. Under the Condor model, the owner of each machine allows Condor to launch an externally submitted job (*i.e.* one not generated by the owner) when the machine becomes idle. Each owner is expected to specify under what conditions on, *e.g.*, load average, memory occupancy, keyboard activity, *etc.*, his or her machine can be considered idle. When Condor detects that a machine has become idle, it takes an unexecuted external job from a queue it maintains and assigns it to the idle machine for execution. If the machine's owner begins using the machine again, Condor detects the local activity and evacuates the external job. The result is that resource owners maintain unfettered access to their own resources, and Condor uses them only when they would otherwise be idle.

When a process is evicted from a machine because the machine's owner is reclaiming it (*e.g.*, begins typing at the console keyboard), Condor offers two options: Either the evicted Condor process is checkpointed and saved for a later restart, or it is killed. Condor implements checkpointing through a series of libraries that intercept system calls to ensure that a job can be properly restarted. Using these libraries, however, places certain restrictions on the system calls that the job can issue (forking or threaded processes are disallowed, for example). "Vanilla" jobs, however, are unrestricted but will be terminated (and not checkpointed) when the resource is reclaimed. Condor's extensive documentation [6] details these features to a greater extent.

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. We associate availability with Internet address and port number; therefore, if a monitor process is subsequently restarted on a particular machine (because Condor determined the machine to be idle), the new measurements will be associated with the machine running the process, thus allowing us to accumulate data for an individual machine. For each machine Condor uses, then, our system records a sequence of availability durations and time stamps (in UTC units) indicating when those durations occurred.

Note that Condor does not make guarantees about which individual machines it will use, or when it will use them. As a result, the number of measurements in each trace, and the durations between measurements, are highly variable. In our 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 (the time of this writing).

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 described in the previous section. Using simulation, we compare the effectiveness of exponential, Weibull, and hyperexponential models against the same set of execution traces. 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 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

From a database of machine availability observations gathered from the Condor pool, we identify traces with at least 50 observations in them. There are approximately 1000 workstations in the Condor pool to which we have access. The Condor scheduler decides (based on load, job priority, *etc.*) which machines will be allocated and when. After 18 months of observation, some machines were only chosen a few times. We discard these machines from our analysis, as we do not believe their measurement histories are sufficient to effect a good model fit.

For the remaining machines, 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.

The simulation is designed to capture "steady-state" efficiency, as opposed to the interplay between the length of a particular simulated job and the computed checkpoint intervals. Rather than choosing an arbitrary length for a simulated job, as in [27], or sampling from a distribution of job lengths, we choose instead to 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. That is, we examine the efficiency of execution while the job is running, under the assumption that its lifetime is significantly longer than the simulated time period. To be precise, define the *efficiency* of a job execution to be the amount of useful computation time divided by the total time spent computing, recovering, and checkpointing. Note that for each work time interval, the expected efficiency is just the reciprocal of the quantity Γ, defined in the "Methodology" section, evaluated at T_{opt}.

Our goal is to compare the effectiveness of each model as a function of the checkpoint overhead cost. To do so, we consider both application efficiency and the amount of network load generated when each distribution is used to compute a checkpoint schedule. We only consider the case where the 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. Typically, we submit a set of jobs to the Condor pool which are then scheduled individually on various workstations at the University of Wisconsin according to an internal (and hidden) scheduling algorithm used by Condor. When Condor eventually runs a process on a workstation, it insulates the workstation from any persistent state the process generates. Thus, each process (in the vanilla universe) must manage its own checkpointing through a network connection to a system outside of Condor's control. On this set of external machines, each application runs a set of processes that "feed" checkpoints to newly initiated processes inside the Condor pool and collect checkpoints as those processes continue to execute.

After Condor initiates a process, the process first contacts one of the external checkpoint managers and requests a checkpoint from which it will continue to execute. We model this initial start-up as a recovery operation at the beginning of an availability period in our simulation. As the process continues to execute, it generates checkpoints at various intervals and sends them over the network to a checkpoint manager. The applications we have written to use Condor in this way attempt to consume all of the available local memory in an attempt to limit the number of separate processes they require. As a result, each recovery and checkpoint (except for the first recovery) is the same size, and that size is approximately the size of the available memory on each machine. At some point in the execution, Condor terminates the process without warning and, later, re-initiates it on some workstation (possibly the same one, but most likely not) in the Condor pool. Thus the overall application consists of individual processes that are constantly scheduled and terminated by Condor using checkpoints and recoveries of equal length.

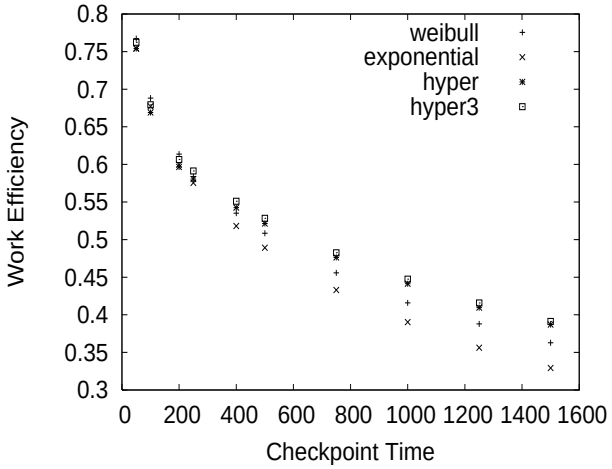

Figure 3: Simulation experiment: average percent machine utilization using exponential, Weibull, and hyperexponential distributions to compute checkpoint schedules.

To simulate this state of affairs, we consider each machine availability trace individually. For each availability trace, and each of the four models we have fit to its training set, we iterate setting $C = R$ to values ranging from 50 to 1500 seconds. During the duration of time covered by the experimental set in each machine trace, a simulated job first spends R seconds recovering. If, during the initial recovery period, the machine trace indicates that the resource has failed, the amount of elapsed time is recorded as recovery overhead, and the simulation skips to the next availability duration for the machine. Otherwise, the simulation computes T_{opt} based on the observation that R seconds have elapsed and assigns T_{opt} seconds of work to the job. If the experimental set indicates that the machine does not fail during this period, the job spends C seconds taking a checkpoint, computes the next value of T_{opt}, and continues. If, at any point, the machine trace indicates that the resource has failed, the period of simulated time back until the last checkpoint completed is recorded as lost work.

When the time period in the experimental set is exhausted, the simulator outputs the fraction of the resource occupancy time that the simulated application spent computing useful results (as opposed to checkpointing, recovering, or recomputing lost work). This entire process is repeated for each combination of model and availability trace.

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 inter-

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

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

val. We indicate statistically significant differences within each row between the results for two distributions 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 support those reported in [27], in which the authors assert that an exponential model of machine availability can be used to develop a checkpoint schedule that is close to optimal. However, since other work has hypothesized the effectiveness of heavier-tailed models for machine availability [17, 22] and the potential for their use to determine checkpointing intervals [18, 34], we wish to quantify the difference precisely.

To do so, 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 α is 0.43, and that for the scale parameter β is 3409.

Using our synthetic trace, we repeat the simulation experiment for $C = 50$ and $C = 500$ as these two cases reflect the conditions we observe in our empirical study (described in the next subsection). Table 2 details the results. For the Weibull cases, we use the known parameters. That is, the Weibull model used to compute checkpoints intervals is precisely the same model that was used to generate the artificial trace. For the exponential cases, we use MLE-determined model and the 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.

C	Weib.	Exp.	2-p Hypexp.	3-p Hypexp.
50	0.891	0.896	0.862	0.895
500	0.685	0.695	0.690	0.670

Table 2: Application efficiency when machine availability is defined by a Weibull distribution with shape = 0.43 and scale = 3409. Each exponential and hyperexponential fit uses all 5000 values.

C	Weib.	Exp.	2-p Hypexp.	3-p Hypexp.
50	0.891	0.896	0.817	0.897
500	0.691	0.695	0.671	0.695

Table 3: Application efficiency when machine availability is defined by a Weibull distribution with shape = 0.43 and scale = 3409. Each exponential and hyperexponential fit only the first 25 values.

Table 2 compares efficiency when all 5000 measurements are used in the respective MLE and EM fits. That is, each fit is made as accurate as possible for the data set through the use of all the data points. In Table 3 we compare efficiency when only the first 25 measurements are used to compute the exponential and hyperexponential models. 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 to fit the MLE exponential and EM hyperexponentials does not degrade the accuracy with which each approximates the Weibull-generated trace.

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 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

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

Table 4: 95% confidence intervals for mean bandwdith for various checkpoint sizes of the four tested distributions.

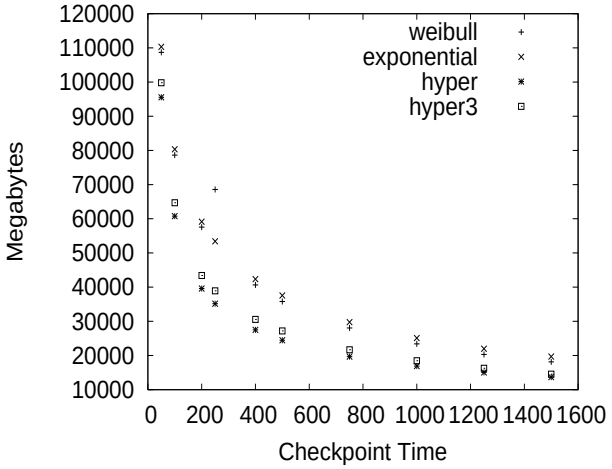

Figure 4: Bandwidth consumed: average network load (in megabytes) when exponential, Weibull, and hyperexponential distributions to compute checkpoint schedules when each checkpoint is 500 megabytes in size.

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 pool have at least 512 megabytes of memory, motivating our choice of 500 megabytes as a representative size. Table 4 shows the average values and their respective 95% confidence intervals, and notation for statistically significant differences within a single row identical to that in Table 1 in 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, although the 3-phase does almost as well and, as we saw before, tends to outperform the 2-phase in application efficiency.

From these results, we conclude that while the application efficiency is insensitive to the choice of availability model, 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 implements the recovery-execution-checkpoint cycle that we have simulated. Because the target applications for this system are memory-intensive, our test process generates and recovers from checkpoints that are 500 megabytes in size. We made this choice because most of the machines in the Linux Condor pool have at least 512 megabytes of memory, and because one of the applications that we intend to deploy with the checkpoint system routinely fills all of the memories in the machines it uses.

However, we report results for a test application, as opposed to the two real applications that will use this system initially, for two reasons. First, the engineering effort required to instrument fully the working applications is significant. Each is a complex, carefully engineered distributed program that already includes fault-tolerance features. Before integrating these codes with our system, we first wanted to determine the extent of the expected performance im-

Distribution	Avg.	Total Time	Sample Size
Weibull	.689	768808	85
Exponential	.680	749695	81
2-phase Hyper.	.726	789304	84
3-phase Hyper.	.676	718094	89

Table 5: Average application efficiency observed using four distributions and the Condor pool with the checkpoint manager located at the University of Wisconsin.

Distribution	Avg.	Total Time	Sample Size
Weibull	.590	491900	48
Exponential	.629	491048	40
2-phase Hyper.	.659	491454	56
3-phase Hyper.	.604	428626	59

Table 6: Average application efficiency observed using four distributions and the Condor pool with the checkpoint manager located at our home institution.

provement. Second, we were concerned that instrumenting the codes might introduce programming errors which affect the correctness of the internal algorithms. As a result, we opt for a test application that performs only the checkpointing functions (the computation phase is simply a tight loop) to generate the following results.

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 [3]. 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.

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 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 5 and 6 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 5, 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 5 indicates the total execution time for the test application and column 3 shows the number intervals we computed for each method. Table 6 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 7 shows the average network loads observed for the various statistical models when the checkpoint manager is located at the University of Wisconsin; Table 8 shows the same numbers with the checkpoint manager at our home institution. The first column of each table may be compared to the simulation results shown in rows 2 and 6, respectively, of Table 4, because the average checkpoint times are similar to the parameters set in these rows. The second column indicates the total network load, and the third 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

Distribution	Megabytes Used	Megabytes / Hour
Weibull	363356	2734
Exponential	338420	3842
2-phase Hyper.	150166	1313
3-phase Hyper.	329034	2374

Table 7: Average network load (in megabytes) using four distributions and the Condor pool with the checkpoint manager located at the University of Wisconsin.

Distribution	Megabytes Used	Megabytes / Hour
Weibull	167195	1223
Exponential	183339	1344
2-phase Hyper.	96264	705
3-phase Hyper.	110920	931

Table 8: Average network load (in megabytes) using four distributions and the Condor pool with the checkpoint manager located at our home institution.

[3]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.

among the various distributions are relatively small for our efficiency metric but quite considerable for network usage.

5.3 Verifying and Validating the Simulation

To *verify* the precision with which simulation results match empirical results, we record the exact conditions that each empirical experiment experienced while it was running; we then use these observations as parameters to our simulation and compare the simulated results to the empirical results. Note that the checkpoint and recovery durations are difficult to predict due to fluctuating network performance. By using the observed durations, however, the simulation should be able to compute correctly the empirical results.

As Table 9 shows, the simulator yields results that are almost identical to those observed in practice, shown in Table 5. Almost all of the real-life traces, when run back through the simulator, result in output that is identical to the live experiment output. Thus, in terms of verification, we conclude that the precision of the simulator sufficient and that if given accurate inputs it will produce good results.

In terms of *validation*, however, there are some discrepancies between the empirical results and the results predicted by the trace-based simulation (as can be seen by comparing the data in Tables 5 and 6 with Table 1 and Tables 7 and 8 with Table 4). We attribute these discrepancies to three characteristics of this study that we do not currently model.

First, while the period of time over which we observed the Condor pool in order to build the database of availability measurements is 18 months, the experimental period for the application test runs is approximately 2 days in length. Thus the experimental availability data are *right-censored*. This censoring tends to favor shorter measurements for a number of reasons. The most obvious reason is that no interval can be longer than the 2-day period. While such measurements only account for 1% of the observations, their cumulative duration constitutes 29% of the total Condor availability. Additionally, we do not consider any jobs that are executing at the time that we terminate our experiment; the interrupted availability interval in which such a job is executing is more likely to be a long interval than a short one, by virtue of the very fact that longer intervals take up more space on the time axis. Thus the data values removed by censoring are skewed toward longer intervals, and it is therefore natural to expect the empirical data over this short time period to appear "lighter-tailed" than the trace data used for the simulations.

Second, as we have mentioned before, the Markov model we use takes as parameters constant (and equal) values for C and R. In practice, these values are variable, especially so when the checkpoints and recoveries traverse a shared network as they do in our study. Recall that after each successful recovery and each successful checkpoint we record the observed delay for use as a new estimate of R and C. This method of predicting future checkpoint duration is likely to be quite inaccurate. Moreover, since this estimate is likely to lose accuracy with time, sequence of short intervals will be based on more accurate overhead costs that sequences that contain many long intervals. For the heavy-tailed distributions, as the sequence length increases so does the length of the intervals between checkpoints thereby introducing greater potential for estimation error. Finally, we assume in our investigation that failure is due to resource reclamation by Condor or machine failure. However, other components in the testbed, most notably the network, can fail as well and by doing so introduce a source of error. For example, a temporary network outage would result in the loss of messages which report checkpoint times, and thus the simulation would report a large amount of lost work due to failure to checkpoint; in reality, we believe we are simply missing a checkpoint time or two for some of the traces.

Distribution	Avg.	Total Time	Sample Size
Weibull	.589	492053	49
Exponential	.620	491206	41
2-phase Hyper.	.642	491547	59
3-phase Hyper.	.602	428719	60

Table 9: Simulation results using variable checkpoint and recovery costs recorded during Condor experiment with the checkpoint manager located at our home institution.

6. CONCLUSION

To use resource-harvesting systems such as Condor effectively, long-running applications must generate intermediate checkpoints of their internal state. Each time a resource is reclaimed by its owner, any applications using the resource's spare cycles are evicted and, without a checkpoint, must be restarted from scratch. Storing and recovering from checkpoints, however, can introduce substantial execution overhead and network load, particularly since resource-harvesting systems often prohibit the use of persistent storage local to the resource.

Previous research has examined the problem of determining when, in an application execution lifetime, checkpoints should be generated so that overhead is minimized. Much of this work assumes that the availability of the resource used by the application is adequately described by some probability distribution, the parameters of which can be derived from previous observations of the resource.

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 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 only slightly less consumption than does the schedule generated from the 3-phase

hyperexponential. Moreover, as the duration of checkpoint and recovery increases, differences in both efficiency and network load become more pronounced. 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.

7. ACKNOWLEDGMENTS

The authors wish to thank and gratefully acknowledge Professor Miron Livny and the Condor Team at the University of Wisconsin for their support of this work. Condor is a production computing environment with demanding users. Even with the conflicting priorities inherent such an enterprise, the research support provided by the Condor Team was responsive and professional. Without it, this work would not have been possible.

8. REFERENCES

[1] S. Asmussen, O. Nerman, and M. Olsson. Fitting phase-type distributions via the EM algorithm. *Scandinavian Journal of Statistics*, 23:419–441, 1996.

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

[3] F. Berman, G. Fox, and T. Hey. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley and Sons, 2003.

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

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

[6] The Condor Reference Manual. `http://www.cs.wisc.edu/condor/manual`.

[7] 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.

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

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

[10] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1998.

[11] R. Gardner et. al. The grid2003 production grid: Principles and practice. In *HPDC-13*, 2004.

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

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

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

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

[20] M. Massie, B. Chun, and D. Culler. he ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing*, 30(7), 2004.

[21] Mathematica by Wolfram Research. `http://www.wolfram.com`.

[22] 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.

[23] The Nagios Network Monitor. `http://www.nagios.org`.

[24] National Computational Science Alliance. `http://www.ncsa.uiuc.edu`.

[25] National Partnership for Advanced Computational Infrastructure. `http://www.npaci.edu`.

[26] GNU Octave Home Page. `http://www.octave.org`.

[27] 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.

[28] 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.

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

[30] 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.

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

[32] 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.

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

[34] A. Tantawi and M. Ruschitzka. Performance analysis

of checkpointing strategies. *ACM Trans. Computer Systems*, 2(2):123–144, May 1984.

[35] TeraGrid. `http://www.teragrid.org`.

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

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

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

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

[40] 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.

[41] 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*.

[42] 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.