Modeling and Tolerating Heterogeneous Failures in Large Parallel Systems

Eric Heien
INRIA, France
eric.heien@inria.fr

Derrick Kondo
INRIA, France
derrick.kondo@inria.fr

Ana Gainaru
UIUC, NCSA, Urbana, IL, USA
University Politehnica of Bucharest
againaru@ncsa.illinois.edu

Dan LaPine
UIUC, NCSA, Urbana, IL, USA
dlapine@ncsa.uiuc.edu

Bill Kramer
NCSA, UIUC, Urbana, IL, USA
wkramer@ncsa.illinois.edu

Franck Cappello
INRIA, France
UIUC, Urbana, IL, USA
fci@lri.fr

ABSTRACT

As supercomputers and clusters increase in size and complexity, system failures are inevitable. Different hardware components (such as memory, disk, or network) of such systems can have different failure rates. Prior works assume failures equally affect an application, whereas our goal is to provide failure models for applications that reflect their specific component usage. This is challenging because component failure dynamics are heterogeneous in space and time.

To this end, we study 5 years of system logs from a production high-performance computing system and model hardware failures involving processors, memory, storage and network components. We model each component and construct integrated failure models given the component usage of common supercomputing applications. We show that these application-centric models provide more accurate reliability estimates compared to general models, which improves the efficacy of fault-tolerant algorithms. In particular, we demonstrate how applications can tune their checkpointing strategies to the tailored model.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Reliability, availability, and serviceability; C.4 [**Performance of Systems**]: Fault tolerance

1. INTRODUCTION

With the rapid increase in the size and complexity of large-scale parallel computing systems, failures are the norm and no longer the exception. New emerging PetaFLOP systems with hundreds of thousands of nodes are expected to have 1 component failure every 30 minutes [1]. The failure rate of these systems is expected to increase as they incorporate higher numbers of commodity components, and the reliability of such components is not expected to improve [2]. Large-scale applications using large numbers of processors and memory in parallel are relatively more sensitive to individual component failures.

The components (such as memory, CPU, disk, and the network) of such systems arguably have different failure dynamics in terms of time and space. Some components may fail randomly and frequently while others may fail in a correlated fashion though rarely. Most modern supercomputers concurrently use multiple heterogeneous types of networks, storage and processors (such as GPU's, general purpose processors, and FPGA's). This potentially makes the failure dynamics even more diverse. Moreover, applications may use such components to different degrees. Some applications may by design fit entirely in memory and avoid disk accesses. Other applications may be CPU-bound and have little network use.

With those factors in mind, our goal is to design application-centric failure models that depend on both the component usage of the application and the failure dynamics of each component. This is challenging as the components exhibit a diverse mixture of failure patterns, and obtaining failure traces of different components is difficult. Nevertheless, such models, which can be used for generative, analytical, or predictive purposes, are critical for the design of efficient and effective fault-tolerant algorithms. For example, an application which frequently uses local storage is more likely to be affected by local hard disk failure than an application which does not use local storage. If the frequently faulty factor is the hard disk, an application which uses this component should checkpoint more often than one which does not.

To the best of our knowledge, this is the first work to explore a range of heterogeneous failures in large-scale systems and investigate methods for addressing them in a holistic manner. Previous models (reviewed in Section 5) treat failures identically, often viewing system nodes as a single black-box or focusing on one specific component (such as disks or memory) exclusively. In particular, they do not distinguish, compare, nor integrate the different failure rates of components nor consider the component usage of applications.

Table 1: Characteristics of Mercury System.

Resource	Phase I	Phase II
# of Nodes	256	635
Processors	2x Itanium II @ 1.3 GHz	2x Itanium II @ 1.5 GHz
Memory	4 or 12 GB DDR1600 ECC RAM	4GB DDR2100 ECC RAM
Network Storage	AFS, NFS (1TB), GPFS (90TB)	
Local Storage	1x18GB, 1x73 GB UltraSCSI drives	2x73 GB UltraSCSI drives
Network	Gigabit Ethernet, Myrinet, Management Network (Ethernet)	

We summarize our approach as follows:

1. We analyze five years of event logs from a production high-performance computing system with hundreds of nodes, and multiple storage and networking systems.
2. We develop a failure model for this system that includes multiple heterogeneous components and captures failure correlation between components.
3. We propose techniques for checkpointing and fault-tolerance on the system, based on the model incorporating component failure heterogeneity.

Section 2 gives an overview of the failure collection methodology we used. Then we proceed to analyze the system and build a heterogeneous failure model in Section 3. Using this model, we propose a failure aware checkpointing scheme in Section 4 and analyze its effectiveness. We review related work in Section 5, describing how our contribution fits in. We conclude with a summary of our work and possible future research directions in Section 6.

2. SYSTEM EVENT COLLECTION AND PROCESSING

In this paper we base our model of component failure on event logs taken from the Mercury cluster at the National Center for Supercomputing Applications (NCSA). We begin by examining the technical details of this cluster, the event recording methodology and how event log messages are correlated with component failures.

2.1 Cluster Specification

The NCSA Mercury cluster was a production high-performance computing system used for scientific applications as part of TeraGrid over a 5-year period with roughly 98% uptime over its lifetime. During its operation, it ran millions of parallel computing jobs for hundreds of researchers in fields ranging from molecular and fluid dynamics simulation to DNA and gene expression analysis.

Detailed technical information regarding the cluster is shown in Table 1. The cluster started with 256 compute nodes, half having large amounts of memory (12GB). Later, an additional 635 compute nodes were added with faster processors. The cluster was operational from January 2004 until March 2010 when it was decommissioned. Over time, system components were replaced due to failure and nodes repurposed to/from computing or storage purposes. We tracked between 936 to 1050 nodes actively reporting log messages over the lifetime of the cluster - this includes compute, login and storage nodes.

Each compute node consisted of two Itanium processors running at 1.3 or 1.5 GHz with 4 or 12 GB ECC protected memory. Login and storage nodes had roughly similar specifications. Storage was a combination of a network file system and local hard disks serving as mount and scratch devices, as well as a wide-area file system using AFS. The AFS system was generally not used directly by applications, so we omit it from our analysis. High speed I/O was handled by a GPFS file system connected by fiber channel.

The Mercury system contained three separate networks - a Gigabit Ethernet network for computation, a high speed Myrinet network for latency sensitive parallel applications, and a management network for node maintenance and software updates. By default there was no checkpointing or fail-over mechanism for applications running on the cluster. Each application was responsible for managing its own fault tolerance.

2.2 Event Logs

To understand failures in the Mercury cluster we analyze system error logs maintained by the cluster administrators. Logs from each node were collected centrally with each log message being sent as a single packet to avoid truncation. Some events generated multiple messages which could be interleaved in the logs with messages from other machines. These logs contain the time of the message, node on which it occurred and possibly details regarding the application which generated it.

Figure 1 shows the dates for which event logs are available, with blue colored sections indicating log availability and blank sections indicating a lengthy log lack. There are a handful of small (less than a few days) gaps in the logs, as well as longer inter uptions on the order of several months such as from March to July 2007 and February to May 2009. These gaps are the result of system downtime, inactive data recording or accidental deletion of message logs. We are interested in understanding the distribution of long-term failure behavior of components. Therefore, in this paper we extract event data from logging periods uninterrupted by excessively lengthy gaps. We refer to these periods as epochs.

For this work we select five epochs which cover most of the operational span of the cluster and have relatively small periods of missing data (on the order of 1 or 2 weeks at the most). Table 2 shows the five epochs for event log data used in this paper. The chosen epochs range in length from 203 to 329 days (29-47 weeks) with an average length of 257 days (36 weeks). Two of the epochs have no missing days, while epoch 3 has the most with 15 days missing. Overall, our selected epochs cover 1285 days (183 weeks) with only 25 days missing.

Table 2: Event Log Data.

Epoch	Time Span	Days Recorded	Days Missing
1	Jul. 2004 to Jun. 2005	329	6
2	Jun. 2005 to Jan. 2006	203	0
3	Feb. 2006 to Oct. 2006	227	15
4	Jul. 2007 to May 2008	301	4
5	Jun. 2008 to Feb. 2009	225	0
Total		1285	25

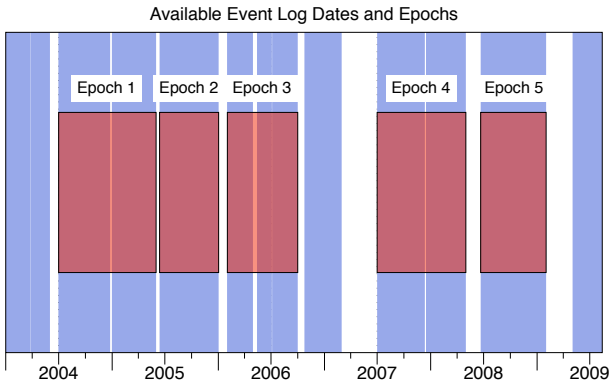

Figure 1: Dates of event log availability.

2.3 Event Processing

To summarize the common event log messages, we performed an initial pass over the logs to identify frequently occurring messages with similar syntactic patterns. In particular, we applied the Hierarchical Event Log Organizer [3] to the logs, resulting in a list of message templates. These message templates are essentially regular expressions that describe a set of syntactically-related messages.

The cluster system administrator, one of the paper co-authors, then inspected these message templates, and identified which messages indicated hardware failures and which were caused by user error or misconfiguration. The system administrator had over 6.5 years of experience identifying and resolving failures of this particular cluster, since its first construction.

Table 3: Example error messages.

Code	Message
F1	`scsi error: (1:0:0) status=02h key=4h (hardware error); fru=02h asc/ascq=11h/00h ""`
F2	`rpc: bad tcp reclen 0x47455420 (non-terminal)`
F3	`pbs_mom: sister could not communicate (15059) in xxxxxx, job_start_error from node xxxxx in job_start_error`
F4	`ifup: could not get a valid interface name: -> skipped`
F5	`+ mem error detail: physical address: 0x5fa56180, address mask: 0xfffffffff80, node: 0, card: 0, module: 4, bank: 2, device: 0, row: 6098, column: 3252`
F6	`processor error map: 0x4000 processor state param: xxx processor lid: 0xc0180000`

In order to have a good statistical measure of events, we discarded patterns which occurred less frequently than once a week over the entire system. Many log messages were ambiguous and could indicate either user or hardware error. In this regard, we were conservative, and chose only error messages which clearly indicated hardware errors.

Based on our analysis, we developed a set of 6 message patterns corresponding to errors affecting processor, disk, memory and network resources. Examples of these messages are shown in Table 3 with identifiable elements replaced by x's. Rather than using the full message, in this work we refer to each error by a code F1, F2, etc.

The first error code, **F1**, refers to a hardware reported error in a device on the SCSI bus. These messages were reported by one of three storage devices - compute node local primary or scratch storage devices or RAID configured SAN devices (on storage nodes). The messages tended to be classified into two categories, either SCSI bus resets or unrecoverable read or write failures. Roughly 13% of the local device errors were SCSI resets and among the remaining failures, 82% were unrecoverable read failures, 9% were unrecoverable write failures and the remaining 9% were problems such as mechanical failures or record corruption. The SAN messages were 100% SCSI resets, which should not affect jobs and we therefore ignore. Although unrecoverable failures might not necessarily cause immediate job failure, they would at least result in node shutdown and replacement of the faulty drive within a short time frame.

Event code **F2** is an NFS related error indicating unavailability of the network file system for a machine. These messages were almost exclusively reported by the NFS server machines. According to administrators, these errors indicate temporary unavailability of the NFS system to a request, possibly due to excessive load. In applications using the network file system this could cause file operations to fail and the application to quit.

Code **F3** indicated a failure of a PBS (Portable Batch System) daemon to communicate. Although this can indicate incorrect setup, generally PBS was functioning correctly and thus this message would indicate network unavailability for a node. Code **F4** occurs when a node is restarted but has not yet connected to either the Gigabit or management networks which can indicate an unexpected node restart caused by unexpected hardware failure.

Finally codes **F5** and **F6** indicate errors in memory and processor cache, respectively. These errors are expected on occasion due to manufacturing imperfections, cosmic rays or overheating. They are often correctable thanks to the ECC capabilities of the memory. However, when large numbers of them occur in a short time span it likely represents permanent failure of a component.

2.4 Event Filtering

To correctly match messages to a failure, we processed each message type in a manner reflecting its nature. With his in-depth experience, the cluster system administrator determined different thresholds that allowed us to isolate unique failures from redundant messages.

Table 4: Failure interevent statistics (in days).

		E1	E2	E3	E4	E5
All	Mean	0.390	0.543	0.373	0.278	0.287
	Median	0.210	0.236	0.157	0.089	0.103
F1	Mean	1.45	2.72	3.27	12.7	14.9
	Median	0.986	1.01	2.23	3.46	9.08
F2	Mean	31.2	7.90	6.90	12.2	12.4
	Median	34.6	7.90	0.778	14.0	14.0
F3	Mean	N/A	1.10	0.828	0.425	0.407
	Median	N/A	0.078	0.316	0.093	0.114
F4	Mean	1.45	1.45	1.50	1.21	1.80
	Median	0.120	0.068	0.121	0.073	0.090
F5	Mean	1.18	5.11	9.88	7.95	3.39
	Median	0.841	2.89	5.42	4.41	1.51
F6	Mean	1.52	2.65	2.68	4.27	4.09
	Median	0.908	1.74	1.82	3.25	2.70

For example, some component failures may cause logging of large numbers of messages. In particular, memory and processor cache failures (F5 and F6) can result in a single faulty component generating hundreds or thousands of messages in less than a day. Conversely, transient correctable errors in these components can generate isolated single messages but not affect an application.

Along these lines, codes F5 and F6 appear frequently due to repeated access to the corrupted component when a memory module or processor failed. Replacement of a faulty memory module by the administrators could take up to a day or more. Therefore, consecutive F5 or F6 messages on the same host within a 24 hour span are treated as a single failure.

However, to avoid transient correctable errors being treated as fatal failures, we only view a host as having a failure if there are more than 100 F5 or F6 messages. Similarly, code F1 can be generated repeatedly by a faulty drive, so we treat multiple F1 messages from a machine in a 24-hour time span as a single failure.

Conversely, codes F2, F3 and F4 should not be generated repeatedly for a long time by a single failure, and a single instance of any of these codes indicates a potentially fatal failure to an application using the resource. Therefore, we treat any number of these messages on a single machine within a 1-hour time span as a single failure. We chose these thresholds with the advice of the cluster system administrator, who had extensive experience with the logs.

3. FAILURE ANALYSIS AND MODELING

Given the six error codes in the previous section, we next develop a model of heterogeneous failures in the system from the event log data. We assume that failure events are instantaneous, and focus on modeling the time between two consecutive failure events.

3.1 Heterogeneous Failure Statistics

We derive statistical distributions of failure rates over the whole system.

Mean and median time to failure. Table 4 shows the mean and median rate of each failure over the entire cluster across all epochs. Among all types of failures, there was an average of between 1.8 and 3.6 failures per day. Assuming an equal probability of failure over all nodes, this is a per-node mean rate of between 248 to 484 days to failure. This table also shows there is a wide range in interevent times for different types of failures. For example, in epoch 3 failure type F1 has a mean interevent time of 77 hours, while failure type F3 has a mean of 20 hours. Interevent times can also vary significantly between different epochs for the same failure. For example, mean interevent time between F1 failures increases from roughly 35 hours in epoch 1 to 78 hours in epoch 3.

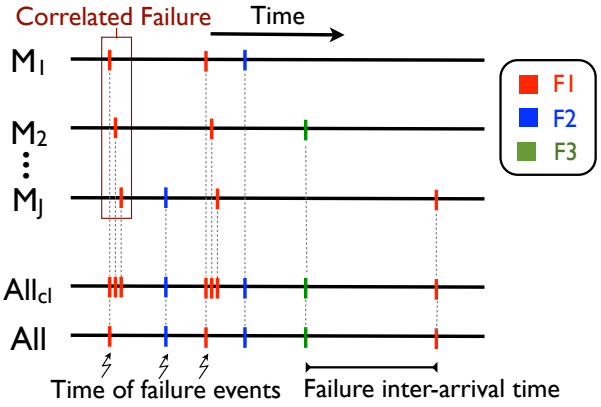

Figure 3: Example of failure events. $M_1, M_2, \cdots M_J$ denote the J nodes in use. ALL_{CL} refers to the set of failure event times over the entire cluster. ALL refers to the set of unique failure event times over the entire cluster.

Cumulative distribution of failure interarrivals. Figure 2 shows cumulative distributions (CDF) of the time between failures for the whole cluster over different epochs plotted on a log scale. The solid lines indicate interevent times for the cluster as a whole. These interevent times correspond to the time line labeled $\texttt{ALL}_{CL}$ in Figure 3.

Some correlated failures (for example, network failures corresponding to F2 and F4) can occur nearly simultaneously on multiple machines. In the logs, a correlated failure appears as sequence of time-clustered events of the same failure type across machines. In particular, we assume that a correlated failure occurs when the time separating two consecutive failure events is less than 1 minute. We justify this threshold as overheads for fault tolerance often exceed this threshold by a factor of 10 [4]. Also, the CDF for failures on the whole cluster showed a spike near this threshold (not visible in Figure 2), indicating a correlated failure.

After interpreting nearly simultaneous failures as a single failure, we plot also in Figure 2 the resulting cumulative distribution shown as the dotted line. These interevent times correspond to the time line labeled ALL in Figure 3. Finally, the red line indicates the line of best fit. The derivation of this line of best fit is explained below.

Correlation across nodes. We find that some of the failures are correlated across different machines, shown in the graphs as a large cumulative distribution at the start of the plot. For example, depending on the epoch, between 30-70%

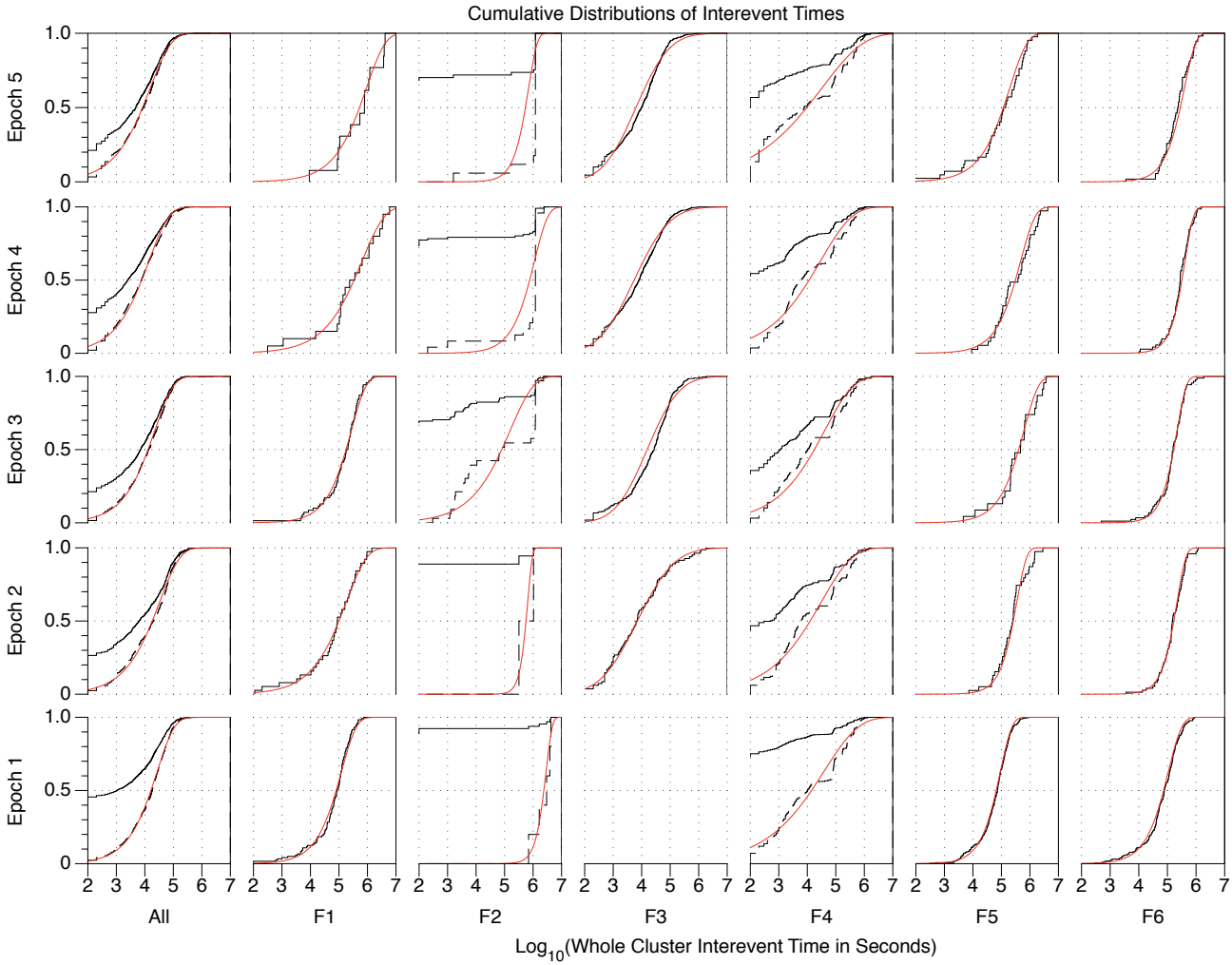

Figure 2: Failure interevent cumulative distributions for different epochs. (Solid line: cluster as a whole, dotted line: discounts simultaneous failures, red line: best fit.)

of the F4 failures on different machines occur within 30 seconds of each other. This could be caused by multiple nodes being restarted simultaneously after a shared power failure, for example. On the other hand, other failures such as F1, F5, and F6 show almost no correlation between machines, with over half the failures occurring more than a day apart.

Correlation across components. Although some failure types are correlated between machines, we must also investigate whether different failure types have correlation, such as memory and processor failures being correlated due to common causes like overheating or motherboard failure. To determine whether different failures are time-correlated between machines, we divided each epoch into hour-long periods where the value of each period is 1 if a specified failure type occurred and 0 otherwise. We chose a length of 1 hour as most failure events are separate by several hours. We calculate the cross-correlation over all epochs between all combinations of failure types, with a cross-correlation of 1 indicating exact correlation (at some time delta) between two failure patterns within an epoch.

The average cross-correlations between different failures over all epochs ranged from 0.04 to 0.11, none of which indicate strong positive or negative correlation. Therefore, in our model we feel it is reasonable to assume there is no correlation between different types of failures. However, it is worth noting that this does not mean there are no correlations between failures in large-scale systems, as some correlated aspects (e.g. long term failure rates of the same type of component) have been found in previous studies [5]. Rather, over extended periods we find there is little correlation between different types of failures. We also looked into auto-correlation among failure arrival times for a single component, but not see any noticeable periodic pattern.

3.2 Per-Component Failure Model

Time to failure. Next we perform statistical analysis to create a model of resource failure for the Mercury cluster. Since we found no correlation between different component failures, we choose to model the failure types as independent probability distributions. We also treat the model of failure interevent times differently from the model of the number

of nodes affected. Previous work [6] indicates a Weibull distribution tends to fit failure interevent times well. For completeness we also test log-normal, log-gamma and exponential distributions.

To fit the parameters of each candidate distribution, we use the standard method of maximum likelihood estimation (MLE) [7]. To evaluate the fit of these distributions with our data, we use the Kolmogorov-Smirnov test which generates a p-value between 0 and 1. Low p-values below a standard threshold of 0.05 indicate that the data did not come from the specified distribution.

With the combined failure data, we find log-normal and Weibull distributions best fit with p-values of 0.41 and 0.49 respectively. For F1 we find that log-normal and Weibull distributions fit best with average p-values around 0.52 to 0.61. F2 is difficult to model due to the irregular sharp spike shape in the distribution around 11 days, but it fits reasonably to an exponential, log-normal or Weibull distribution (p-values 0.33, 0.22 and 0.12 respectively).

F3 is fit best by a log-normal distribution with p-values around 0.38 (ignoring the first epoch). F4 is not particularly well fit by a distribution because of the crease in the distribution around 17 hours, but it fits reasonably well to log-normal or Weibull distributions (p-values around 0.15). F5 and F6 are both well fit by log-normal distributions (p-values 0.82 and 0.68), Weibull distributions (p-values 0.68 and 0.58) and exponential distributions (p-values around 0.55). Based on these results, we use Weibull distributions for each of the failure interevent times except F3. The parameters for these distributions are shown in Table 5.

In addition to having heterogeneous scale and shape parameters, the hazard rates are different as well for the component types. Intuitively, if the hazard rate is decreasing, the longer the component has been without failure, the higher the probability that it will not fail in the future. A shape parameter with value less than 1 indicates a decreasing hazard rate. For F1 and F4, the shape parameter is clearly below one. So the hazard rate is clearly but surprisingly decreasing for disk and network failures. For F5 and F6, the shape parameter fluctuates above or slightly below 1. This means that the hazard rate is relatively constant or slight increasing for memory or processor failures. Overall, the F1, F2, and F4 failures are dominant, and the hazard rate is decreasing.

Number of nodes in failure. Finally, we examine the number of nodes affected by a failure. As seen in Figure 2, F1, F3, F5 and F6 rarely occur at close intervals on separate nodes. Therefore we consider each of these as affecting only a single node. However, F2 and F4 can occur simultaneously on multiple nodes, as well as the combination of all failures. A plot of the CDF for the number of nodes affected by a failure over all epochs is shown in Figure 4 with the black line indicating the actual data and the red line showing the fitted distribution. For example, with the combined failure model, 91% of failures affect just one node, 3% affect two nodes, and 6% affect more than two nodes. We model the number of nodes affected by the combined failure as a Weibull distribution, by an F2 failure as a log normal distribution, and by an F4 failure as an exponential distribution.

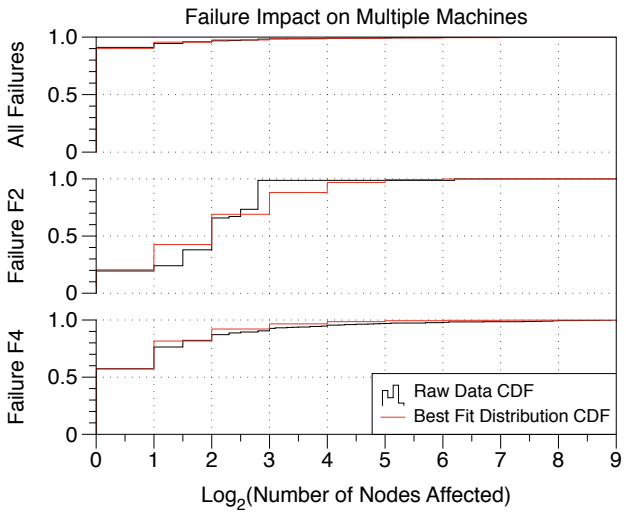

Figure 4: Distributions of nodes affected by failures.

The relevant parameters are shown in Table 5.

Although this model is for the Mercury system, we believe the underlying principles should be applicable to other large-scale parallel systems. In particular this model demonstrates that failure interevent time tends to be well modeled by a Weibull distribution, and that some failures can affect multiple nodes, the modeling of which may require a heavy-tailed distribution.

3.3 Integrated Failure Model

Next we develop a complete failure model integrating the different resource failure distributions and affected node distributions determined in Section 3.1. In general, the distribution depends on two factors, namely the specific components and number of nodes used by the application. Our approach is to use Monte Carlo simulations to determine the failure distribution for different applications.

Application resource usage. We consider two profiles of application resource usage, representative of real workloads. We assume that applications using a subset of components are only affected by failures of components in that subset.

The first configuration [F5,F6] corresponds to an application that uses mainly the processor and memory with little disk or network usage. Many bag-of-task applications, such as those commonly used in Grids [8] and Desktop Grids [9], match this profile. Examples of such real-world applications include Rosetta@home [10] (a project investigating protein folding) and EINSTEIN@home [11] (a project searching for gravitational wave sources).

The second configuration [F1, F5, F6] corresponds to data-intensive applications that use disks frequently in addition to memory and the processor. An example of such a data-intensive application is the CM1 application used for atmospheric research at NCSA.

The third configuration is the combination of all failure types, which we refer to as `combined`. Massively parallel

Table 5: Failure model parameters. (λ: scale, k: shape)

Interevent Time (days)						
	Distribution	Epoch 1	Epoch 2	Epoch 3	Epoch 4	Epoch 5
All	Weibull	$\lambda = 0.3166$ $k = 0.7097$	$\lambda = 0.3868$ $k = 0.6023$	$\lambda = 0.2613$ $k = 0.6629$	$\lambda = 0.1624$ $k = 0.6161$	$\lambda = 0.1792$ $k = 0.5841$
F1	Weibull	$\lambda = 1.436$ $k = 0.8300$	$\lambda = 2.110$ $k = 0.6052$	$\lambda = 3.167$ $k = 0.8418$	$\lambda = 7.579$ $k = 0.5560$	$\lambda = 10.684$ $k = 0.6510$
F2	Weibull	$\lambda = 33.16$ $k = 1.994$	$\lambda = 7.515$ $k = 2.631$	$\lambda = 1.831$ $k = 0.5317$	$\lambda = 13.08$ $k = 0.9249$	$\lambda = 8.077$ $k = 1.416$
F3	Log Normal	N/A	$\mu = -2.509$ $\sigma = 2.361$	$\mu = -1.717$ $\sigma = 2.030$	$\mu = -2.767$ $\sigma = 2.249$	$\mu = -2.622$ $\sigma = 2.125$
F4	Weibull	$\lambda = 0.4498$ $k = 0.3593$	$\lambda = 0.3639$ $k = 0.4009$	$\lambda = 0.4776$ $k = 0.4317$	$\lambda = 0.3400$ $k = 0.3931$	$\lambda = 0.3792$ $k = 0.2979$
F5	Weibull	$\lambda = 1.071$ $k = 1.065$	$\lambda = 4.032$ $k = 1.253$	$\lambda = 7.181$ $k = 0.8464$	$\lambda = 5.506$ $k = 0.8510$	$\lambda = 2.274$ $k = 0.7092$
F6	Weibull	$\lambda = 1.260$ $k = 0.9258$	$\lambda = 2.520$ $k = 1.392$	$\lambda = 2.520$ $k = 1.323$	$\lambda = 4.788$ $k = 1.455$	$\lambda = 4.548$ $k = 1.091$
Number of Nodes						
	Distribution	Parameters				
All	Weibull	$\lambda = 0.1387, k = 0.4264$				
F1, F3, F5, F6	Constant	1				
F2	Log Normal	$\mu = 2.273, \sigma = 2.137$				
F4	Exponential	$\lambda = 0.8469$				

Variable	Definition
M_J	Number of nodes used by job J
M_{tot}	Total number of nodes in cluster
$Q_i(N)$	Probability of number of nodes N affected by failure for component i
$P_{node}(M_J)$	Probability of a failure of component i on a node used by job J
$P_J(M_J, t)$	Probability of job J failing at time t

Table 6: Failure Variable Definitions.

applications would use many or all components of the system, including multiple networks and storage devices. One such application is NAMD [12], a portable parallel application for biomolecular simulations.

Holistic failure model. We construct an application-centric failure model as a function of the types of components used as well as the number of nodes used. Assume a parallel job J utilizes M_J nodes out of a total M_{tot} on the cluster while periodically checkpointing its computation state. For simplicity we assume checkpointing occurs simultaneously across all nodes and the act of checkpointing does not affect the failure rates. We also assume a failure on one node means all M_J nodes performing the job must restart from the last checkpoint. The job uses resources subject to failures $F_J \subseteq (F_1, F_2, \ldots)$. Each failure F_i has a corresponding probability density function Q_i of the number of nodes affected by the failure and probability density function P_i of failure probability at different time intervals - the corresponding distributions/parameters for this work are shown in Table 5. Table 6 summarizes these variable definitions.

If a job uses M_J of M_{tot} nodes, the probability P_{node} of a given failure F_i occurring on a node used by a job is:

$$P_{node}(M_J) = \sum_{N=1}^{M_{tot}} Q_i(N) \left[1 - \prod_{R=0}^{N-1} \left(1 - \frac{M_J}{M_{tot} - R} \right) \right] \quad (1)$$

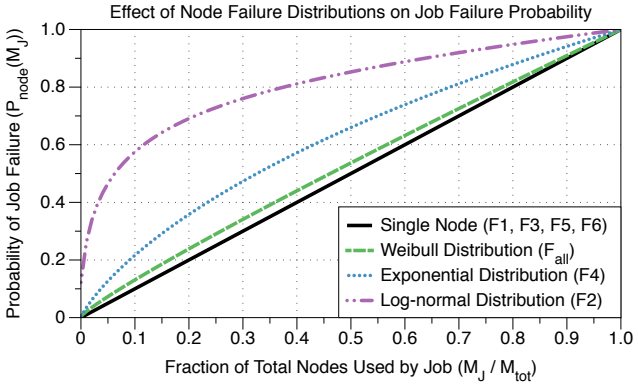

Figure 5: Effect of node failure distributions on job failure probability.

which indicates the weighted sum of the probabilities that the failure affects one or more of the nodes the job is running on. This assumes all nodes are equally likely to experience a given failure.

Figure 5 graphically depicts this function for the four distributions we use in the model. When failures affect a single node, the job failure probability rises linearly as seen in the black solid line. This is because the probability of a single node failure affecting a job rises linearly with the number of nodes used by the job. However, for failures that are likely to affect large numbers of nodes (e.g. network failure F2) the probability of job failure is high even when using few nodes. The F_{all} job failure probability is similar to the single node, but slightly higher because it has a small chance (roughly 10%) of affecting multiple nodes. This demonstrates that the heterogeneity in number of nodes affected by a given failure will in turn affect the probability of job failure.

Since we determined the failures are not correlated with each other, the total failure probability density function of the job

is given by:

$$P_J(M_J, t) = 1 - \prod_{i \in F_J} (1 - P_{node}(M_J) P_i(t)) \qquad (2)$$

Ideally we would solve this analytically to determine the failure distribution for a given application using a specified set of nodes and resources. However, this would involve a mathematical maelstrom and be intractable for most configurations. Instead, we perform Monte Carlo simulations using the node failure law in Equation 1. These simulations create failures according to the distributions in Table 5, then determine if the failure would affect a job of a given size (M_J). By running the simulations to generate thousands of failures we can approximate the expected failure arrival distributions for different job sizes with different resource usage.

The results of these simulations for parameters from epoch 3 are shown in Figure 6 for three job sizes, $M_J = 8, 64, 512$ using the Mercury cluster size of $M_{tot} = 891$. This figure shows the result of the simulation - an approximation of the cumulative distribution function for P_J. As we expect, using fewer resources or fewer nodes results in a longer time between failures. For example, the mean time between failures for jobs subject only to processor or memory failures (F_5 and F_6) is 428 days when using 8 nodes, but jumps to 7.21 days when using 512 nodes. However, if the jobs are subject to disk failures as well, the mean time between failures becomes 135 days for 8 nodes and 2.13 days for 512 nodes. This demonstrates how by using additional types of resources or nodes, applications become subject to a higher rate of failures.

Our model considering all failure types provides an upper bound on the failure rate. Ideally, the distribution for combined failures and F_{all} should be identical. However, our model uses approximations for the distribution of nodes affected by a failure and overestimates the number of nodes affected by F_2 and F_4 failures. This is likely the reason why the combined failure distribution (created by merging failure times from F_1 through F_6) shows more frequent failures than F_{all}, especially for jobs using fewer numbers of nodes.

Based on the simulations, we create an integrated model of failure time distributions for different resource configurations and job node counts. We find that the failure time distributions such as those shown in Figure 6 are best modeled by a Weibull distribution. The fitted parameters of these distributions for Epoch 3 are shown in Table 7.

Interestingly, for a given set of failures types, the scale parameter (λ) changes as the number of job nodes (M_J) increases but the shape parameter (k) stays mostly the same. Also, a doubling in the number of job nodes results in roughly a halving of the λ parameter, though this effect is less pronounced when including failures that affect multiple nodes (e.g. the "Combined" row).

These distributions and their parameters are useful for predictive, generative, and analytical purposes. With respect to prediction, the model can be used to predict the time to failure given the time the system has been continuously available. With respect to generation, the model can be used to easily simulate failures realistically in large parallel systems. In the next section, we show how to apply these models to determine the optimal checkpoint interval for applications.

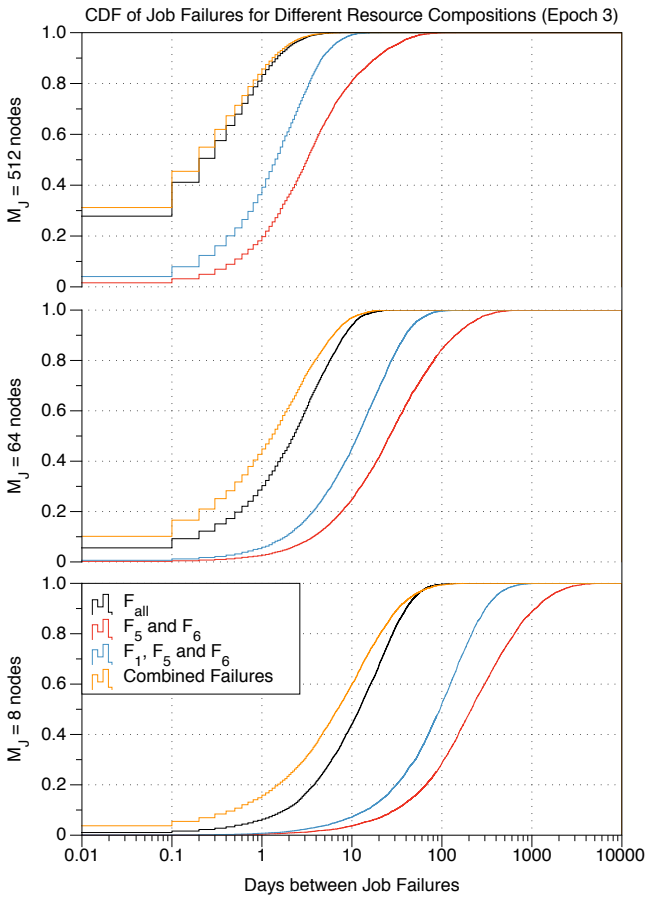

Figure 6: The effect of heterogeneous resource usage on expected failure rates for different job sizes.

4. TOLERATING HETEROGENEOUS FAILURES

We next examine several common supercomputing applications and propose a scheme for fault tolerance in light of the heterogeneous failure model developed in Section 3.3. Given this model, we develop a scheme for determining the optimal checkpointing period in Section 4.1.

The novelty of our checkpoint investigation is the failure model. Prior works on checkpointing assume failures occur randomly and homogeneously in terms of time and space. In contrast, we look a checkpointing strategies that account for correlated failures and mixtures of component failure rates. We also discover correlations between optimal checkpoint period with required checkpoint time and node count.

4.1 Checkpoint-Based Fault Tolerance

In any checkpointing application, there is a tradeoff between the cost (lost time, network/storage use, etc) due to checkpointing and the expected time lost from a potential fail-

Table 7: Monte Carlo-based integrated failure model parameters for epoch 3. (λ: scale, k: shape)

Weibull Distribution of Time Between Failures (days), $M_{tot} = 891$							
	$M_J = 8$	$M_J = 16$	$M_J = 32$	$M_J = 64$	$M_J = 128$	$M_J = 256$	$M_J = 512$
F_{all}	$\lambda = 17.75$ $k = 1.013$	$\lambda = 10.68$ $k = 0.9989$	$\lambda = 6.284$ $k = 0.9915$	$\lambda = 3.379$ $k = 0.9198$	$\lambda = 1.776$ $k = 0.8860$	$\lambda = 0.9600$ $k = 0.8190$	$\lambda = 0.4765$ $k = 0.7406$
F_5, F_6	$\lambda = 363.9$ $k = 0.7167$	$\lambda = 172.8$ $k = 0.7153$	$\lambda = 89.14$ $k = 0.7180$	$\lambda = 44.15$ $k = 0.7013$	$\lambda = 21.40$ $k = 0.7270$	$\lambda = 10.90$ $k = 0.7074$	$\lambda = 5.282$ $k = 0.7020$
F_1, F_5, F_6	$\lambda = 132.7$ $k = 0.9327$	$\lambda = 65.87$ $k = 0.9867$	$\lambda = 33.52$ $k = 0.9963$	$\lambda = 16.52$ $k = 1.006$	$\lambda = 8.349$ $k = 1.049$	$\lambda = 4.225$ $k = 0.9905$	$\lambda = 2.025$ $k = 1.031$
Combined	$\lambda = 10.86$ $k = 0.7562$	$\lambda = 5.963$ $k = 0.7654$	$\lambda = 3.653$ $k = 0.8173$	$\lambda = 2.236$ $k = 0.7959$	$\lambda = 1.429$ $k = 0.7887$	$\lambda = 0.8193$ $k = 0.7513$	$\lambda = 0.4419$ $k = 0.7222$

Variable	Definition
T_C	Computation time
T_S	Time to checkpoint
T_L	Lost computation since last checkpoint
T_{step}	Single computation and checkpoint step $(T_C + T_S)$
T_F	Time from (re)start of computation to next failure
T_W	Total time wasted

Table 8: Checkpoint Variable Definitions.

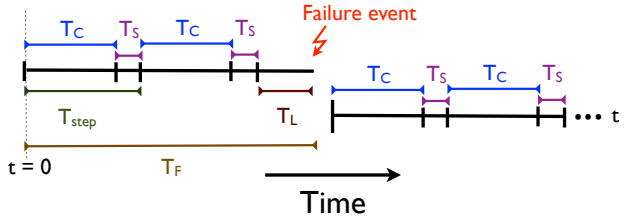

Figure 7: Checkpoint and restart variables [13].

ure/restart. By checkpointing more often, an application loses less computation when a failure occurs, but checkpointing too frequently leads to wasted time and resources.

One of the first analytic checkpointing schemes [13] determined the optimal interval between checkpoints by attempting to minimize the expected wasted/lost time before failure. The author assumed an exponential distribution for failure interarrival times. While we use a similar formulation to derive the optimal checkpoint interval, our failure model is significantly different and more realistic as it considers failure heterogeneity, correlation and use of multiple nodes.

Each node computes for T_C time, then performs a checkpoint requiring T_S time to save - a total time of $T_{step} = T_C + T_S$. These compute/checkpoint steps are repeated until the program finishes. If a failure occurs, the lost computation time is T_L, which will depend on the time since last checkpoint. We assume the time to restart a computation is negligible compared to T_{step} and can therefore be ignored. Table 8 summarizes these variable definitions, and Figure 7 illustrates them.

The time from the start or restart of computation to the next failure is T_F, and the probability density function of job failure at time t is $P_J(M_J, t)$ as shown in Equation 2. P_J is determined both by the number of nodes involved in the job and the failure rate of the resources used by the nodes. If a job uses many nodes in the cluster, it is more likely to be affected by a failure than a job which only uses a few. Similarly, if a job requires multiple resources (local disk, network, etc) it is more likely to be affected by a failure than a job which uses only the processors and memory.

A failure occurring at time T_F will fall between $n(T_C + T_S)$ and $(n+1)(T_C + T_S), n = 0, 1, \ldots$, which will result in nT_S time lost from saving checkpoints and T_L time lost from the failure. Another way of writing this is $T_F = nT_{step} + T_L$. This gives a total time wasted $T_W = nT_S + T_L$.

Assuming a Weibull failure inter-arrival distribution with scale and shape parameters λ and k, the total expected wasted time due to lost computation and checkpointing is:

$$T_W = \sum_{n=0}^{\infty} \int_{nT_{step}}^{(n+1)T_{step}} [t - nT_C] \left[\frac{k}{\lambda} \left(\frac{t}{\lambda} \right)^{k-1} e^{-(\frac{t}{\lambda})^k} \right] dt \quad (3)$$

We want to find the value of T_C (time between checkpoints) which minimizes T_W (expected wasted time). Other work [14] uses a dynamic programming approach to solve a similar problem, however we use standard mathematical packages to determine optimal T_C values for the parameters in Table 7. The calculated optimal T_C values are shown in Table 9 - note the days/hours difference between the tables when comparing them.

We notice several striking aspects of the optimal T_C values including some very useful relations. First, as the number of machines increases the optimal checkpoint interval decreases. This is to be expected, since an increase in failure prone elements implies the mean time to failure will be shorter and thus require more frequent checkpointing. For our model we find the relation $T_C \propto 1/\sqrt{M_J}$, indicating optimal checkpoint time will roughly halve with every four-fold increase in nodes. This is especially accurate for failures which only affect single nodes.

Next, the required time to checkpoint (T_S) also has a significant impact on T_C with greater T_S implying greater T_C. This is because when we optimize T_C in Equation 3 we include the checkpointing time in T_{step}. Intuitively this makes sense as well - more expensive checkpoints should be performed less often. For our model we find the relation $T_C \propto \sqrt{T_S}$. It is worth noting this is the same relation in the approximation used in Young's paper [13]. In fact, with our model for $k = 1$ (where the Weibull distribution sim-

Table 9: Optimal T_C Checkpoint Intervals (hours)

		$M_J = 8$	$M_J = 16$	$M_J = 32$	$M_J = 64$	$M_J = 128$	$M_J = 256$	$M_J = 512$
F_{all}	T_S =1 min	3.746	2.912	2.236	1.669	1.222	0.923	0.681
	T_S =10 min	11.77	9.136	6.995	5.218	3.810	2.881	2.134
	T_S =30 min	20.24	15.68	11.98	8.923	6.495	4.906	3.640
F_5, F_6	T_S =1 min	18.99	13.38	9.430	6.713	4.594	3.327	2.328
	T_S =10 min	60.17	42.42	29.90	21.30	14.57	10.56	7.392
	T_S =30 min	104.55	73.67	51.91	36.97	25.25	18.32	12.81
F_1, F_5, F_6	T_S =1 min	10.44	7.345	5.171	3.619	2.547	1.831	1.253
	T_S =10 min	33.16	23.24	16.28	11.37	7.972	5.718	3.882
	T_S =30 min	57.69	40.27	28.06	19.55	13.66	9.767	6.579
Combined	T_S =1 min	3.216	2.370	1.806	1.427	1.145	0.8866	0.6650
	T_S =10 min	10.18	7.493	5.680	4.488	3.594	2.786	2.089
	T_S =30 min	17.61	12.94	9.766	7.707	6.158	4.769	3.570

plifies to an exponential distribution), Equation 3 results in almost exactly the same value as Young's approximation of $T_C = \sqrt{2T_S T_F}$.

Finally, we find that by using only failures relevant to an application, such as F_5, F_6 or F_1, F_5, F_6 we can checkpoint less frequently. For example, when using 64 nodes the optimal T_C for F_5, F_6 is roughly 4 times less frequent than for F_{all}. These results demonstrate that by tuning the checkpoint interval to the actual node count and resource usage, large-scale applications can improve their run time by minimizing unnecessary checkpointing.

5. RELATED WORK

There is a significant body of work related to this topic covering multiple areas including tools for message log analysis, analysis of failures in large-scale systems, and fault tolerance for parallel computing.

There are several tools available for system event log analysis which take different approaches. One such tool is Nodeinfo [15], which uses an automatic identification algorithm in combination with a binary scoring metric to identify important alert messages. Another tool uses a bioinformatic inspired algorithm to classify messages [16]. In this work, particularly in Section 2.3, we use a hierarchical log analysis tool [3] to summarize the logs. This tool shows higher accuracy than other comparable tools with our data set.

There are numerous studies investigating failures in large-scale systems. These studies range from comprehensive reviews of failures on entire systems to studies of failure for particular components. Studies of large-scale systems include investigations of failure and repair in a variety of high-performance computing systems [6, 17, 18, 19, 20, 21, 22, 4] to computational error rates in desktop computing systems [23]. In general, such studies do not distinguish failures among component types nor consider how an application would be affected if it uses specific components, or they focus exclusively on one particular component type.

Other studies examine in more detail the failure of particular components, taking into account the effects of environment, utilization and aging. These include recoverable and unrecoverable errors in memory [24, 25] and disk [5]. Usually, these studies focus on one component in isolation, and the models cannot always be easily applied to applications that use multiple components.

In regards to fault tolerance, there are numerous proposed techniques. In this work we focus on fault tolerance through checkpointing since this is the most widely applicable software based technique, though perhaps not the most efficient for some applications. There are several studies of transparent checkpointing [26] or large-scale parallel application oriented checkpointing [27]. Our checkpointing study is unique in terms of its failure model, which considers heterogeneous component failures and node counts.

6. CONCLUSION

In this paper we presented application-centric failure models that consider both the components used by the application and the reliability of such components. Specifically our contributions were as follows:

1. Measurement – We identified and analyzed failures contained in five years of event logs from a production high-performance computing system. The event traces and failure identification are certainly useful for other failure studies and algorithm evaluation.
2. Modeling – We determined the distribution of failure interarrivals of specific components. We then formed holistic failure models based on the component-usage of applications. These distributions are useful for generative, predictive, and analytical purposes.
3. Fault-tolerance – We applied this model to derive the optimal time to checkpoint. Our failure model is significantly more realistic than previous models in that it takes into account correlated and random failures in the system. With more realistic models, the efficiency and efficacy of algorithms is improved.

Future work could investigate the optimal number of nodes for a job in order to minimize the expected failure rate; we noticed that using larger numbers of nodes increases the expected rate of job failure. Developing a closed form expression for optimal checkpoint period in a heterogeneous parallel system with different expected failure distributions should also be investigated.

7. ACKNOWLEDGMENTS

We thank Slim Bouguerra for useful discussions. This work was carried out in part under the ANR project Clouds@home (ANR-09-JCJC-0056-01).

8. REFERENCES

[1] William D. Gropp. Personal communication, May 2010.

[2] Franck Cappello, Al Geist, Bill Gropp, Laxmikant Kale, Bill Kramer, and Marc Snir. Toward exascale resilience. *Int. J. High Perform. Comput. Appl.*, 23:374–388, November 2009.

[3] Ana Gainaru, Franck Cappello, Stefan Trausan-Matu, and Bill Kramer. Hierarchical event log organizer. *Technical Report of the INRIA-Illinois Joint Laboratory on PetaScale Computing*, pages 1–24, Sep 2010.

[4] Daniel Ford, Francois Labelle, Florentina Popovici, Murray Stokely, Van-Anh Truong, Luiz Barroso, Carrie Grimes, and Sean Quinlan. Availability in globally distributed storage systems. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation*, 2010.

[5] Bianca Schroeder and Garth Gibson. Understanding disk failure rates: What does an mttf of 1,000,000 hours mean to you? *Transactions on Storage (TOS*, 3(3), Oct 2007.

[6] Bianca Schroeder and Garth A. Gibson. A large-scale study of failures in high-performance computing systems. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 249–258, Washington, DC, USA, 2006. IEEE Computer Society.

[7] G. Casella and R. Berger. *Statistical Inference*. Duxbury, 2002.

[8] A. Iosup, C. Dumitrescu, D. H. J. Epema, H. Li, and L. Wolters. How are real grids used? the analysis of four grid traces and its implications. In *GRID*, pages 262–269, 2006.

[9] Catalog of boinc projects. `http://www.boinc-wiki.info/Catalog_of_BOINC_Powered_Projects`.

[10] D. Baker. ROSETTA@home. `http://boinc.bakerlab.org/rosetta/`.

[11] EINSTEN@home. `http://einstein.phys.uwm.edu`.

[12] James C. Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D. Skeel, Laxmikant V. Kalé, and Klaus Schulten. Scalable molecular dynamics with namd. *Journal of Computational Chemistry*, 26(16):1781–1802, 2005.

[13] John W. Young. A first order approximation to the optimum checkpoint interval. *Commun. ACM*, 17:530–531, September 1974.

[14] M. S. Bouguerra, D. Kondo, and D. Trystram. On the scheduling of checkpoints in desktop grids. In *Proceedings of the 11th IEEE International Symposium on Cluster Computing and Grid (CCGrid)*, 2011.

[15] A Oliner, A Aiken, and J Stearley. Alert detection in system logs. *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on*, pages 959 – 964, 2008.

[16] J. Stearley. Towards informatic analysis of syslogs. In *Proceedings of the 2004 IEEE International Conference on Cluster Computing*, pages 309–318, Washington, DC, USA, 2004. IEEE Computer Society.

[17] Derrick Kondo, Bahman Javadi, Alexandru Iosup, and Dick Epema. The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems. *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 398 – 407, 2010.

[18] R. Bhagwan, S. Savage, and G. Voelker. Understanding Availability. In *Proceedings of IPTPS'03*, 2003.

[19] B. Javadi, D. Kondo, JM. Vincent, and D.P. Anderson. Mining for statistical availability models in large-scale distributed systems: An empirical study of seti@home. In *17th IEEE/ACM International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, September 2009.

[20] John R. Douceur. Is remote host availability governed by a universal law? *SIGMETRICS Performance Evaluation Review*, 31(3):25–29, 2003.

[21] J. Brevik, D. Nurmi, and R. Wolski. Quantifying Machine Availability in Networked and Desktop Grid Systems. Technical Report CS2003-37, Dept. of Computer Science and Engineering, University of California at Santa Barbara, November 2003.

[22] Mehmet Bakkaloglu, Jay J. Wylie, Chenxi Wang, and Gregory R. Ganger. On correlated failures in survivable storage systems. Technical Report CMU-CS-02-129, Carnegie Mellon University, 2002.

[23] D Kondo, F Araujo, P Malecot, P Domingues, LM Silva, G Fedak, and F Cappello. Characterizing result errors in internet desktop grids. *Lecture Notes in Computer Science*, 4641:361, 2007.

[24] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. Dram errors in the wild: a large-scale field study. *SIGMETRICS '09: Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, Jun 2009.

[25] Xin Li, Michael Huang, Kai Shen, and Lingkun Chu. A realistic evaluation of memory hardware errors and software system susceptibility. *USENIXATC'10: Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, Jun 2010.

[26] Jason Ansel, Kapil Arya, and Gene Cooperman. Dmtcp: Transparent checkpointing for cluster computations and the desktop. *Parallel and Distributed Processing Symposium, International*, 0:1–12, 2009.

[27] Camille Coti, Thomas Herault, Pierre Lemarinier, Laurence Pilard, Ala Rezmerita, Eric Rodriguez, and Franck Cappello. Blocking vs. non-blocking coordinated checkpointing for large-scale fault tolerant mpi. In *SC 2006 Conference, Proceedings of the ACM/IEEE*, page 18, 2006.