Reliability -aware optimal checkpoint /restart model in high performance computing

Yudan Liu

RELIABILITY-AWARE OPTIMAL CHECKPOINT/RESTART MODEL IN HIGH PERFORMANCE COMPUTING

by

Yudan Liu, M.S.

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

COLLEGE OF ENGINEERING AND SCIENCE
LOUISIANA TECH UNIVERSITY

May 2007

LOUISIANA TECH UNIVERSITY

THE GRADUATE SCHOOL

May 09, 2007
Date

We hereby recommend that the dissertation prepared under our supervision

by YUDAN LIU

entitled RELIABILITY-AWARE OPTIMAL CHECKPOINT/RESTART MODEL IN HIGH PERFORMANCE COMPUTING

be accepted in partial fulfillment of the requirements for the Degree of

Ph.D. in Computational Analysis and Modeling

[illegible]
Supervisor of Dissertation Research

Richard [illegible]
Head of Department

Computational Analysis and Modeling
Department

Recommendation concurred in:

[illegible]

[illegible]

Richard [illegible]

[illegible]

Advisory Committee

Approved:
[illegible]
Director of Graduate Studies

[illegible]
Dean of the College

Approved:
[illegible]
Dean of the Graduate School

GS Form 13
(5/03)

ABSTRACT

Computational power demand for large challenging problems has increasingly driven the physical size of High Performance Computing (HPC) systems. As the system gets larger, it requires more and more components (processor, memory, disk, switch, power supply and so on). Thus, challenges arise in handling reliability of such large-scale systems. In order to minimize the performance loss due to unexpected failures, fault tolerant mechanisms are vital to sustain computational power in such environment. Checkpoint/restart is a common fault tolerant technique which has been widely applied in the single computer system. However, checkpointing in a large-scale HPC environment is much more challenging due to complexity, coordination, and timing issues. In this dissertation, we present a reliability-aware method for an optimal checkpoint/restart strategy. Our scheme aims to address the fault tolerance challenge, especially in a large-scale HPC system, by providing optimal checkpoint placement techniques derived from the actual system reliability. Unlike existing checkpoint models, which can only handle Poisson failure and a constant checkpoint interval, our model can perform a varying checkpoint interval and deal with different failure distributions. In addition, the approach considers optimality for both checkpoint overhead and rollback time. Our validation results suggest a significant improvement over existing techniques.

APPROVAL FOR SCHOLARLY DISSEMINATION

The author grants to the Prescott Memorial Library of Louisiana Tech University the right to reproduce, by appropriate methods, upon request, any or all portions of this Dissertation. It is understood that "proper request" consists of the agreement, on the part of the requesting party, that said reproduction is for his personal use and that subsequent reproduction will not occur without written approval of the author of this Dissertation. Further, any portions of the Dissertation used in books, papers, and other works must be appropriately referenced to this Dissertation.

Finally, the author of this Dissertation reserves the right to publish freely, in the literature, at any time, any or all portions of this Dissertation.

Author Yudan Liu

Date 05/11/2007

GS Form 14
(5/03)

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I want to thank many people for their help and encouragement during the period of my research work. First and foremost, I am grateful to my advisor, Dr. Chokchai (Box) Leangsuksun, for his direction, wise counsel, and endless support through the course of this research; he helped me and taught me more than can be acknowledged in these few sentences. Thanks are also extended to Dr. Raja Nassar for his patient academic guidance and suggestions.

Many thanks to my committee, Dr. Weizhong Dai and Dr. Zeno Dixon Greenwood, and CAM director Dr. Richard Greechie, for their support, guidance, and helpful suggestions. I also would like to thank my teammate, Nichamon Naksinehaboon. She is a smart girl, and helped me a lot in validating the model.

I am in debt to Andy Yoo of Lawrence Livermore National Laboratory, who provided valuable information for the research work to be sound.

I would like to thank all the faculty and staff of the Collage of Engineering and Science for being supportive during the various stages of my study and research at Louisiana Tech University.

And finally, I want to thank my family, my wife Vivian and my lovely daughter Fiana, for they have never lost their faith in me.

CHAPTER 1

INTRODUCTION

Computational power demand for large challenging problems has increasingly driven the physical size of HPC systems. As witnessed in Top500.org, the volatile top500 list currently has the LLNL/IBM Blue Gene/L system with 131072 processors as the world fastest system. The larger the system gets, the more the number of components (processor, memory, disk, switch, power supply and so on). Thus, challenges arise in reliability of such large-scale systems. For example, we analyzed the failure data of the LLNL ASC White system (512 nodes and 8192 processors) and found that the average value of mean time between failures (MTBF) for individual node is over 7000 hours, but MTBF of the system (512 nodes) is only around 20 hours due to the combination of hardware and software failures of each component.

Since MPI is one of the most popular parallel programming paradigms, we target our reliability study on the system running the MPI applications. Normally, a MPI application is decomposed and executed among a group of compute nodes, where individual subtasks communicate by message passing. Because of a static view of an MPI environment [45], a single node failure would cause the whole application to fail and require an application restart. In order to minimize the performance loss due to frequent failures, fault-tolerance (FT) mechanisms have been intensely studied by the HPC research community. The main objective of this dissertation is to address reliability issues, particularly in the area of checkpoint FT mechanisms. Our main goal is to obtain an optimal checkpoint/restart model in HPC environments.

1.1 Reliability Challenges in HPC

A large scale HPC cluster system may consist of hundreds or thousands of compute nodes communicating via interconnection fabric. The main HPC objective is normally targeted to achieve the best possible completion time and performance while executing a parallel application on as many nodes as possible within the system. There are two types of nodes: head nodes (servers) and compute nodes (clients). A head node takes requests and dispatches tasks to the compute nodes, where the actual work is processed. Figure 1.1 illustrates a typical architecture of the large-scale cluster.

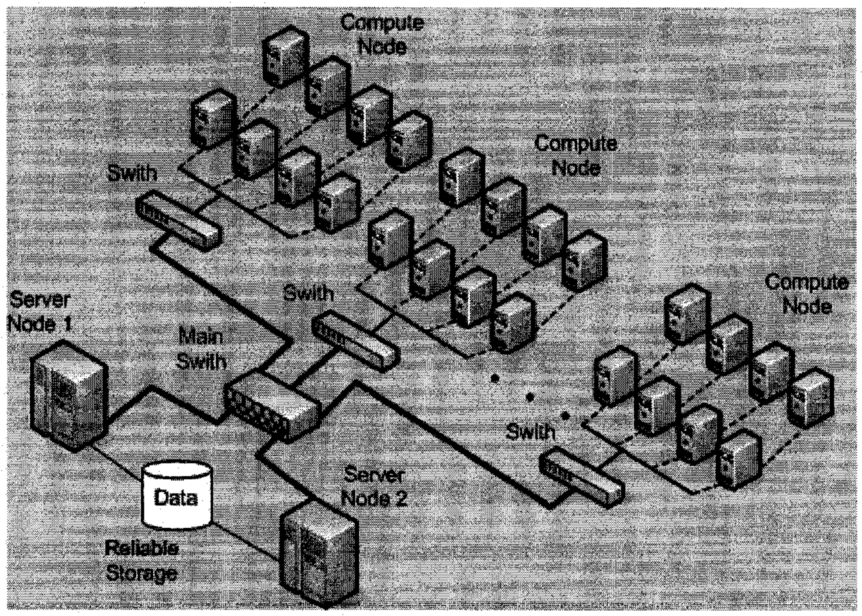


Figure 1.1 Typical architecture of large-scale cluster system.

Besides the size of hardware components in the HPC cluster, the systems depend on software components, such as an operating system (OS) on each node, a parallel computing environment, a job scheduling program, a monitoring and management subsystem, a fault-tolerance mechanism, and so on. To facilitate parallel computing, a parallel application runs simultaneously either on a portion or on all of the nodes in the system. Unfortunately, if one node which the application is running on fails, it will render an application outage. This issue is currently a major challenge of MPI applications on the HPC cluster system.

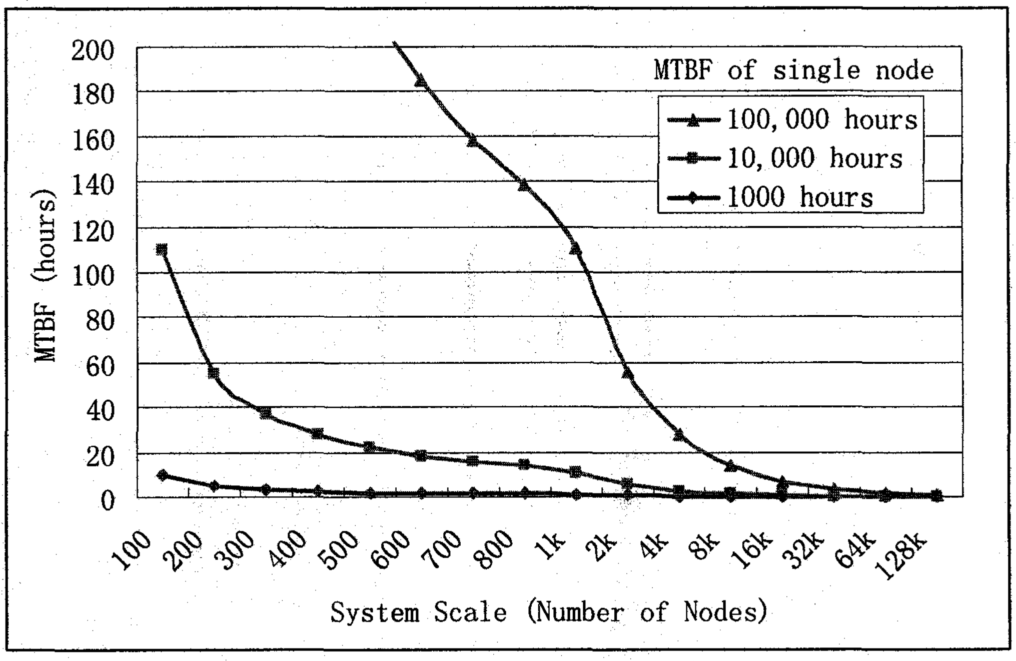


Figure 1.2 System MTBF and scalability.

Although individual components may be very reliable, the total system mean time between failures (MTBF) can be reduced substantially when the number of components is very large. The aggregate system MTBF is given by $(\sum_{k=1}^{N} 1/\mu_k)^{-1}$, where μ_k is the

MTBF of component k. As an example, Figure 1.2 shows the expected system reliability for systems of different sizes when constructed with three different component reliability values (i.e., individual MTBF of 1000, 10,000 and 100,000 hours). In this research, we also analyzed the failure data of the LLNL ASC White system (512 nodes and 8192 processors) and revealed that the average value of MTBF for individual node is over 7000 hours, but MTBF of the system (512 nodes) is less than 20 hours. Therefore, reliability-awareness and fault-tolerance mechanism is a critical component for the success of large-scale system deployments.

1.2 Checkpoint/Restart Model in HPC

The checkpoint/restart mechanism is a common fault tolerant technique to deal with reliability issues in an application runtime environment. However, to perform checkpointing on a HPC system is a non-trivial task when compared to an individual computer. When a checkpointing parallel application is running on a large number of compute nodes, storing application context files and recovering from outages becomes a daunting challenge due to issues such as overheads, stable storage, consistency etc. Therefore, the checkpoint/restart scheme needs to balance between failure caused lost time and checkpoint overhead, in order to find optimal checkpoint placements.

1.3 Organization of the Dissertation

This dissertation is organized as follows: in Chapter Two, we introduce the problem domain and background. It includes the basic concepts of reliability theory, failure data analysis techniques, and fault-tolerance schemes. In Chapter Three, we present the details of failure data analysis and the reliability prediction of a large-scale cluster system. The reliability-aware optimal checkpoint/restart model specification is illustrated in Chapter Four. Chapter Five depicts the analysis and evaluation of the checkpoint/restart model. The incremental checkpointing model and validation are discussed in Chapter Six. The conclusion and future work is presented in Chapter Seven.

CHAPTER 2

TERMINOLOGIES AND BACKGROUND

In this chapter, some background knowledge concerning system reliability and fault-tolerance techniques are described. To begin, some definitions of terminologies relevant to the measure of system failure and reliability are presented. Then we introduce the reliability prediction techniques and related works in the area of events log file analysis in large-scale distributed platform. Various checkpoint/restart methods and their relevant tools are discussed in detail.

2.1 Definitions in Reliability Theory

2.1.1 Definition of Reliability

The reliability $R(t)$ of a system is the probability that the system will perform its intended functions satisfactorily for a length of time under specified operation conditions. Based on the definition, reliability is measured as a probability. Probability theory has been used to analyze the reliability components, as well as the reliability of system consisting of these components. From a mathematical point of view, the reliability $R(t)$ of a system S can be expressed as:

$$R(t) = Pr(S \textit{ is fully functioning in } [0,\ t]).$$

Let X be the random variable representing the lifetime of a system, let f be the probability density function (PDF), and let F be the cumulative density function (CDF.) of the variable X. Then the system reliability at time t can be depicted as

$$R(t) = \Pr(X > t) = 1 - F(t) = 1 - \int_0^t f(x)dx \tag{2.1}$$

because

$$\int_0^\infty f(x)dx = 1.$$

Hence, the reliability of a system can be expressed in

$$R(t) = \int_0^\infty f(x)dx - \int_0^t f(x)dx = \int_t^\infty f(x)dx. \tag{2.2}$$

Normally, it is assumed that the system is working properly at the instant $t = 0$. Yet, it is possible to allow that the system is initially defective with a probability p, i.e. $F(0) = p$. For such a case, the reliability of the system is

$$R(t) = 1 - p - \int_0^t f(x)dx. \tag{2.3}$$

2.1.2 Definition of MTTF

The expected value or mean of the lifetime T is also called the mean time to failure (MTTF) or the expected life of the device. It can be evaluated with the following standard equation:

$$MTTF = E(t) = \int_0^\infty tf(t)dt. \tag{2.4}$$

A computationally more efficient formula for evaluation of MTTF is

$$MTTF = \int_0^\infty R(t)dt. \tag{2.5}$$

2.1.3 Definition of MTBF

Some systems may go though several failures before they are scrapped. Such a system is said to be repairable. For repairable systems, the MTTF represents the mean time to the first failure. After the system is repaired and put into operation again, the average time to the next failure is indicated by mean time between failures (MTBF). The MTBF represents the average operating time from the point that a failed system is recovered to operation to the point of time that it fails again. It does not include the amount of time needed to repair the failed system.

2.1.4 Definition of MTTR

The average amount of time needed to repair a failed system is called mean time to repair (MTTR).

2.1.5 Definition of Availability

For a repairable system, availability is often termed as a measure of its performance. The availability is defined as the probability that the device is available whenever needed. It can be expressed as

$$A = \frac{MTBF}{MTBF + MTTR}. \tag{2.6}$$

From the definitions of reliability and availability given above, the difference between reliability and availability is that reliability requires that at no time within the interval [*0, t*] may the system fail, whereas availability permits the possibility that the system may have failed and subsequently may have been repaired one or more times before time *t*.

2.1.6 Definition of Failure Rate Function

The failure rate function, or the hazard function, denoted by $\lambda(t)$, is defined as the probability that a system will fail in the next time unit, given that it has been working properly up to time t:

$$\lambda(t) = \lim_{\Delta t \to 0} \Pr(T \le t + \Delta t \mid T > t) = \frac{f(t)}{R(t)}. \tag{2.7}$$

2.2 Commonly Used Lifetime Distributions

The lifetime of a system is a random variable of interest in reliability analysis. It is continuous and can only be a nonnegative value. As a consequence, reliability analysis deals with continuous distributions. In this dissertation, only the continuous distributions that have been widely used in reliability analysis are concerned.

2.2.1 Exponential Distribution

A random variable T follows the exponential distribution if and only if its PDF is given by

$$f(t) = \lambda e^{-\lambda t}, \quad t \ge 0, \ \lambda > 0. \tag{2.8}$$

The reliability function is

$$R(t) = e^{-\lambda t}, \quad t \ge 0, \quad \lambda > 0. \tag{2.9}$$

The cumulative distribution function (CDF) is

$$F(t) = 1 - e^{-\lambda t}, \quad t \ge 0, \ \lambda > 0. \tag{2.10}$$

The failure rate function is

$$\lambda(t) = \lambda, \quad t \ge 0, \ \lambda > 0. \tag{2.11}$$

2.2.2 Weibull Distribution

A random variable T follows the Weibull distribution if and only if its PDF is given by

$$f(t) = \frac{\beta t^{\beta-1}}{\eta^{\beta}} e^{-(t/\eta)^{\beta}}, \quad t \geq 0, \beta > 0, \eta > 0, \tag{2.12}$$

where β is the shape parameter, and η is the scale parameter.

The reliability function is

$$R(t) = e^{-(t/\eta)^{\beta}}, \quad t \geq 0, \ \beta > 0, \eta > 0. \tag{2.13}$$

The cumulative distribution function (CDF) is

$$F(t) = 1 - e^{-(t/\eta)^{\beta}}, \quad t \geq 0, \beta > 0, \eta > 0. \tag{2.14}$$

The failure rate function is

$$\lambda(t) = \frac{\beta t^{\beta-1}}{\eta^{\beta}}, \quad t \geq 0, \beta > 0, \eta > 0. \tag{2.15}$$

2.2.3 Gamma Distribution

A random variable follows the gamma distribution if and only if its PDF is given by

$$f(t) = \frac{\lambda^{\beta}}{\Gamma(\beta)} t^{\beta-1} e^{-\lambda t}, \quad t \geq 0, \beta > 0, \lambda > 0, \tag{2.16}$$

where β is the shape parameter, and λ is the scale parameter.

The reliability function is

$$R(t) = \frac{\lambda^{\beta}}{\Gamma(\beta)} \int_t^{\infty} \tau^{\beta-1} e^{-\lambda\tau} \, d\tau, \quad t \geq 0, \beta > 0, \lambda > 0. \tag{2.17}$$

The cumulative distribution function (CDF) is

$$F(t)=\frac{\lambda^{\beta}}{\Gamma(\beta)}\int_0^t \tau^{\beta-1}e^{-\lambda\tau}\,d\tau,\qquad t\geq 0,\beta>0,\lambda>0. \tag{2.18}$$

The failure rate function is

$$\lambda(t)=\frac{t^{\beta-1}e^{-\lambda t}}{\int_t^{\infty}\tau^{\beta-1}e^{-\lambda\tau}\,d\tau},\qquad t\geq 0,\beta>0,\lambda>0. \tag{2.19}$$

2.2.4 Lognormal Distribution

A random variable T follows the lognormal distribution if and only if its PDF is given by

$$f(t)=\frac{1}{\sigma t\sqrt{2\pi}}\exp\left[-\frac{1}{2\sigma^2}(\ln t-\mu)^2\right],\qquad t\geq 0, \tag{2.20}$$

where $\sigma>0$ is the shape parameter, and $\mu>0$ is the scale parameter.

The reliability function is

$$R(t)=1-\Phi\left(\frac{\ln t-\mu}{\sigma}\right),\qquad t>0. \tag{2.21}$$

The cumulative distribution function (CDF) is

$$F(t)=\Phi\left(\frac{\ln t-\mu}{\sigma}\right),\qquad t>0. \tag{2.22}$$

The failure rate function is

$$\lambda(t)=\frac{f(t)}{1-\Phi[(\ln t-\mu)/\sigma]},\qquad t>0,\beta>0,\lambda>0, \tag{2.23}$$

where $\Phi(x)$ is the CDF of a standard normal distribution.

2.3 Failure Data Analysis of HPC

One challenge of a large-scale cluster system is that it is vulnerable to unexpected failures. Therefore, the failure analysis of a large-scale system is the foundational work that provides insights how to deal with the performance loss due to the failures. In this dissertation, we take a first step in literature surveys in the related works.

2.3.1 Architecture and Failure Data Collection

Typically, a monitoring and management subsystem in a large-scale cluster can help gathering the failure information. Most failure events are recorded in the events log file, and each failure record includes the failure start time, end time, the host ID, failure type, and so on. The details of failure data collection are described in Chapter Three.

2.3.2 Analytical Techniques of Failure Data

After the data collection, we perform failure analysis by considering the empirical cumulative distribution function (ECDF) and how well the failure data follows one of four probability distributions commonly used in reliability theory: the exponential distribution; the Weibull distribution; the gamma distribution; and the lognormal distribution. We parameterize the distributions through maximum likelihood estimation and evaluate the goodness of fit both by visual inspection and Chi-Squared test and Kolmogorov-Smirnov test.

For the reliability study of a group of nodes, we consider the most common failure behavior in Message Passing Interface (MPI) applications since MPI is the de facto parallel programming paradigm. Thus, the failure follows a serial reliability model. That is to say when an application is running on a group of nodes, a single node failure may cause the overall application to fail. The technical details of failure data analysis are described in Chapter Three

2.3.3 Related Works

There have been numerous research efforts focusing on the failure data analysis on a large-scale cluster system [15][16][17][18][2][9][10][13][14]. Y. Liang et al. [19] recently presented their failure data analysis on the IBM Blue/Gene supercomputer. In [19], the characteristics of event log files, the collection of failure data, and different failure prediction models are introduced. B. Schroeder et al [20] performed failure data analysis from the point of view in order to improve system dependability and availability. In [20], the authors fitted the time between failures (TBF) into four commonly used distributions and found the Weibull distribution to be a good fit, in which their finding agrees with our results. Heath et al. [2] collected failure data from three different cluster systems, ranging from 18 workstations to 89 workstations. They assumed that the times between failures are independent and identically distributed, and fitted the failure data using a Weibull distribution with a shape parameter less than 1. In addition, they employed the reliability property from this Weibull distribution to motivate a new resource management strategy. Two other studies [5][13] achieved the same conclusion that the Weibull distribution is the best fit for TBF.

2.4 Fault-Tolerance and Checkpoint/Restart

In this section, we introduce backgrounds of checkpoint and restart mechanisms. The definition of fault-tolerance is the ability of a system to respond gracefully to an unexpected hardware or software failure. In a mission critical computer system, for the purpose of fault-tolerance, the system mirrors all operations -- that is, every operation is performed on two or more duplicate systems. Thus, if one fails, the other can take over.

2.4.1 Fault-Tolerance Techniques

One of fault tolerance techniques is checkpoint and rollback restart (checkpoint/restart). To ensure application survivability, it is important to preserve the application's states in order to save completed computation in the event of a system failure. In general, checkpoint-based rollback restart protocols periodically save the current computational state of each process involved in a computation. When checkpoint creation is signaled, the system records a representation of the memory state of the process, so that the process state can be reconstructed at an intermediate state in the computation if the process should fail. There are several subcategories in checkpoint restart protocols: uncoordinated checkpoint, coordinated checkpoint, communication-induced checkpoint, full checkpoint, and incremental checkpoint. Details of these techniques can found in [65].

2.4.2 Checkpoint/Restart Experiments

For understanding the behaviors of checkpoint/restart, we built a small checkpoint/restart enabled cluster as a test bench. In our experiment, we employ BLCR-enabled LAM/MPI which is the coordinated checkpoint to implement a transparent fault-tolerant MPI mechanism on an HA-OSCAR cluster [90].

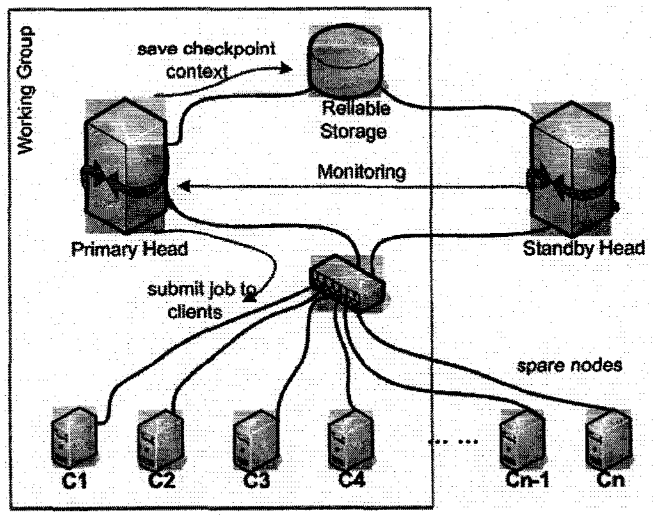


Figure 2.1 The framework of checkpoint/restart.

Figure 2.1 illustrates the normal working stage of the checkpoint/restart scheme on our test bench. This case has no outage in the system. After a user submits an MPI job, the primary head node schedules and dispatches the parallel application to compute nodes. We set up a checkpointing process that periodically checkpoints the application state on the cluster nodes and saves context files to a reliable storage. The standby head node keeps monitoring the health of the primary head and compute nodes.

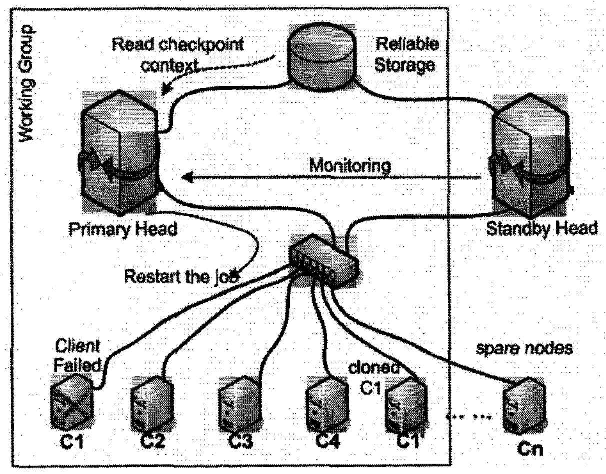


Figure 2.2 The checkpoint/restart behaviors of compute nodes failure.

In a case in which one of the compute nodes fails, as shown in Figure 2.2, the MPI job hangs. Normally, when the node failure causes the application to cease, all MPI processes must be terminated, and the user loses the computation between the failure and the last checkpoint. However, the restart will not succeed in the LAM/MPI environment until the failed node is repaired and brought back to the runtime environment. We address this issue with our self-healing core, resulting in a rapid repair time by grabbing and cloning a spare compute node to impersonate the failed node. Once the head node receives the failure information from the cluster management program, it will take the failure node out of the service and release its IP and other necessary configuration. Then, it applies a recovery mechanism to clone an idle node to be the same as the failed one. Consequently, the primary head will clean up and reinitialize the environment by executing lamclean and lamboot before restarting the application from the checkpointing context files on reliable storage.

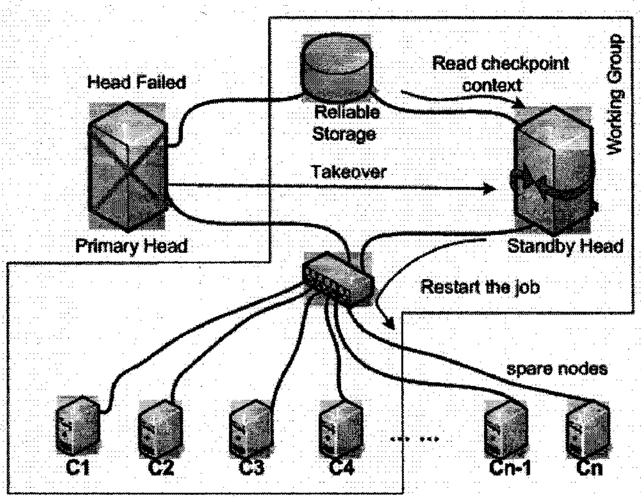


Figure 2.3 The checkpoint/restart behaviors of server nodes failure.

In the second scenario, we consider the primary head failure, shown in Figure 2.3; the standby head node takes over the function and configuration of the primary head. After the failover completes, the standby head node reboots the LAM/MPI with the same node group, and then restarts the checkpointing context files from the shared reliable storage. The job will be successfully restarted from the apparently identical working group (the standby that clones the failed primary and compute nodes).

The high level design of our checkpoint/restart implementation is divided into two parts: the Intelligent Checkpoint Engine (ICE) in Figure 2.4 and the HA-pulse module in Figure 2.5. More details can be found in [90]. However, the optimal checkpoint algorithm, which will be discussed in Four and Chapter Five, is the core of the ICE. Since multiple parameters are involved in the optimal checkpoint model, the modular design and component architecture make it easy for the future extension as well as for promoting reusability. Once, the ICE receives system failure information from cluster management and monitoring, it updates the failure information database and modifies the reliability prediction by the failure prediction module. The details of our failure data analysis and reliability prediction can be found in Chapter Three. In some cases, when a failure is imminent (e.g. CPU temperature rising rate is above a critical threshold), this high priority event can override the optimal checkpoint interval in order to save the application state before the failure occurs. This research on imminent failure prediction is not included in this dissertation; future work on imminent failure prediction with time series technique on IPMI scenario is needed. All these modules are connected to the checkpoint algorithm by ICE interface. The ICE sends a request to HA-pulse for performing the checkpoint. HA-pulse is a daemon that performs the checkpoint/restart. HA-pulse composes of three parts: PulseMaker, PulseMon, and PulseCloner.

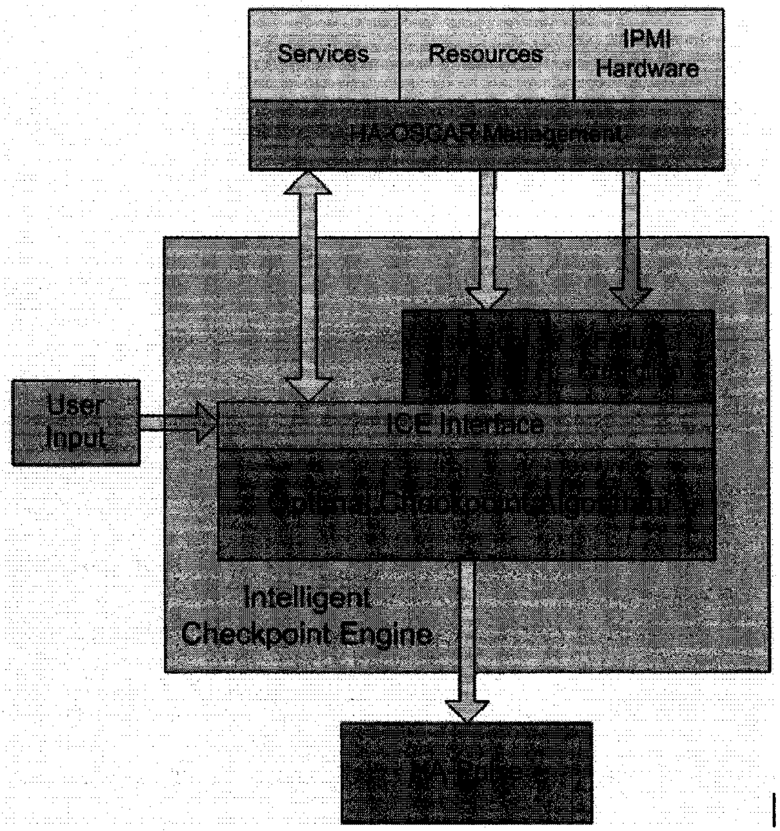


Figure 2.4 The architecture of intelligent checkpoint engine (ICE).

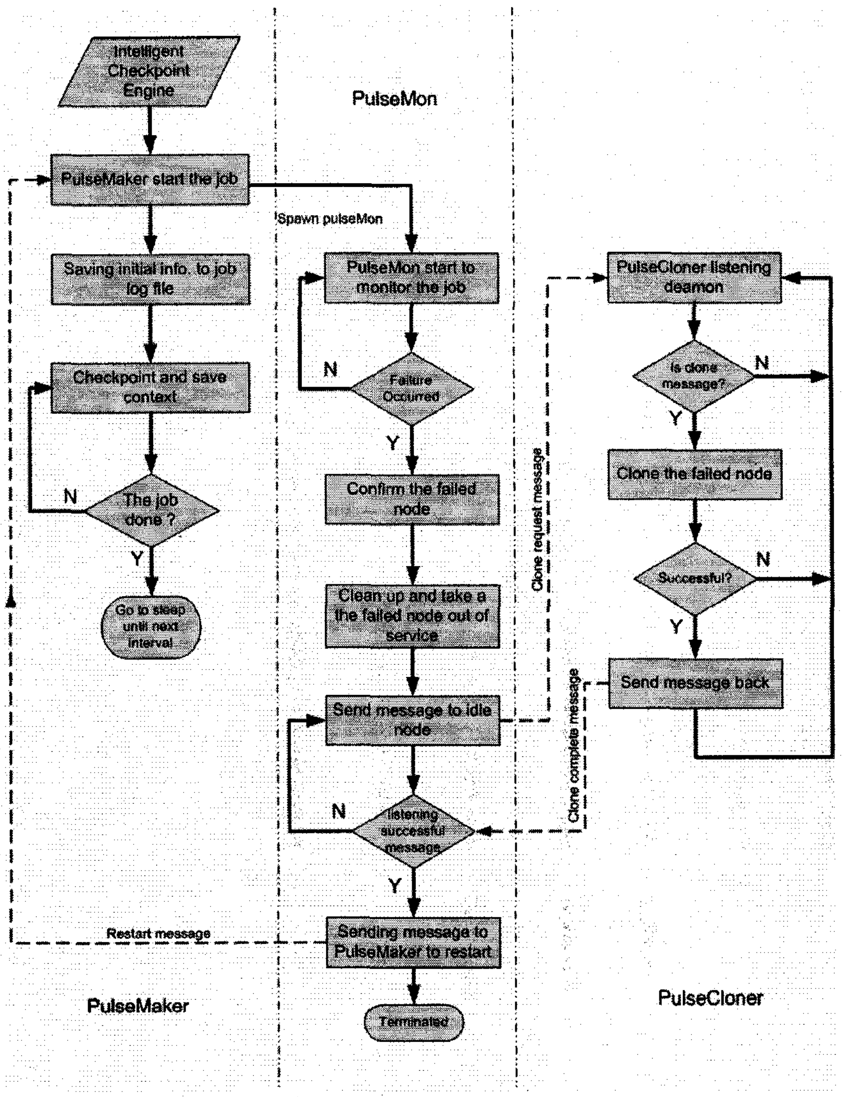

Intelligent Checkpoint Engine
PulseMon
PulseMaker start the job
Spawn pulseMon
Saving initial info. to job log file
PulseMon start to monitor the job
PulseCloner listening deamon
Checkpoint and save context
N
Failure Occurred
Is clone message?
N
Y
Y
Clone the failed node
N
The job done ?
Confirm the failed node
Clone request message
Y
Successful?
N
Go to sleep until next interval
Clean up and take a the failed node out of service
Y
Send message back
Send message to idle node
Clone complete message
N
listening successful message
Y
Restart message
Sending message to PulseMaker to restart
PulseMaker
PulseCloner
Terminated

Figure 2.5 The flow chart of HA-pulse.

CHAPTER 3

FAILURE DATA ANALYSIS AND RELIABLITY PREDICTION

3.1 Overview

Research in the area of system reliability by capturing the potential tendency of the failure in the real system has gained concerns, especially for high performance computing. For example, understanding the failure characteristics can provide insights to better resource allocation and enhance the system availability [7]. In a reliability-aware runtime system, failure analysis is necessary for setting up the checkpoint interval in order to minimize overhead and rollback time.

Nevertheless, the failure analysis is rather complex in a large-scale cluster system. The difficulties of failure study lie in various aspects. First, the root causes could be subtle and numerous from various sources such as hardware component outages, software errors, network failures, or human mistakes within a multitude of nodes. Second, the failure information is a series of statistic data; it is non-trivial to build accurate models for studying the features. Last, the system event log files are normally proprietary and not easily obtained due to providers' negativity that may be perceived when their data are published in the public forum. It is difficult for the researcher to access such source data. Despite existing difficulties, a few studies have focused on failure analysis in the large-scale cluster system.

The failure study is a cornerstone for Reliability, Availability, and Serviceability (RAS) analysis and management. RAS is a critical issue, especially for the large-scale cluster system. The traditional reliability study of cluster system usually makes an unrealistic assumption that the failure is the exponential distribution on each node. Such a simple solution conceals the complexity and multiplicity of the failure in a large-scale cluster system and is detrimental for the RAS management and control.

In this chapter, we discuss methodologies for failure study by analyzing an actual system events log file. The raw data came from the events log files of several major HPC systems from the Lawrence Livermore National Laboratory. The files contain significant system events, from years past, collected from four ASC machines, namely White, Frost, Ice, and Snow.

The organization of this chapter is as follows: in Section 3.2 , we discuss the failure data collection of a large-scale cluster system. It includes the system environment introduction, events log file format, and failure data preprocessing. In Section 3.3 , the method to combine failure data of each node to a group of k nodes is introduced. In this case, the study follows the reliability of the serial model. Section 3.4 describes the failure data processing techniques, and it includes the distribution fitting and goodness-fit-test. At last the sample of failure data analysis results and the conclusion are presented in Section 3.5.

3.2 Failure Data Collection

The failure analysis and reliability prediction for large-scale cluster system has been considered a challenging problem. One of the main reasons is the lack of suitable data from large-scale distributed systems. Fortunately in our research work, we have obtained event logs containing all the failure information in the period from 7/21/2000 to 10/01/2004, of four ASC (Accelerated Strategic Computing) machines in LLNL (Lawrence Livermore National Laboratory). We perform the analysis on these datasets. For the purpose of brevity, only the analysis results of White are presented.

3.2.1 Failure Date Preprocessing

The events log file we have studied ran from July 2000 to October 2004. The raw data contain all the events that occur among different system components. Each record describes an important event using attributes as described in Table 3.1 below:

Table 3.1 Record attributes in the events log file.

Id	Unique number assigned to this event
Pri	Priority
Type	Event type: Hardware; Software; Local problem; Cause not yet categorized
Subtype	Event subtype
Wk-ending	Week ending -- results are binned into weeks ending Friday at 24:00 hours
TDT (hr)	Total down time (in hours)
LC1 (hr)	LC response time (in hours) - time from start of event until LC receives first notification of a problem
LC2 (hr)	LC resolution time (in hours) - time after notification for LC to determine how to resolve event
LC3 (hr)	LC verification time (in hours) - time required for LC to verify problem fix
VR1 (hr)	Vendor response time (in hours) - time from first notification for vendor to start work
VR2 (hr)	Vendor resolution time (in hours) - time for vendor to fix problem once work begins
Pid	Parent id - id of parent event for continuing or cascading events
Vendor id	Vendor id - identifying number assigned to track event by vendor
Status	Event status
Sect	Sector or machine effected (e.g., blue s k y)
Host list	List of host name suffixes
Event-start-date	Date of event start (MM/DD/YY)
Event-start-time	Time of event start (HH:MM:SS)

Table 3.1 Continued.

LC-notified-date	Date of first LC notification (MM/DD/YY)
LC-notified-time	Time of first LC notification (HH:MM:SS)
Event-end-date	Date of event end (MM/DD/YY)
Event-end-time	Time of event end (HH:MM:SS)
Vendor-notified-date	Date of first vendor notification (MM/DD/YY)
Vendor-notified-time	Time of first vendor notification (HH:MM:SS)
Vendor-working-date	Date of vendor start of work (MM/DD/YY)
Vendor-working-time	Time of vendor start of work (HH:MM:SS)
Vendor-done-date	Date of vendor end of work (MM/DD/YY)
Vendor-done-time	Time of vendor end of work (HH:MM:SS)
Follow-up-done-date	Date of LC completion of follow-up work (MM/DD/YY)
Follow-up-done-time	Time of LC completion of follow-up work (HH:MM:SS)
LC-dt (hr)	LC dead time (in hours)- time for which LC fails to work on event resolution due to choice or lack of capacity
Wight	Weight (in cpus) - effective weighting of event if different than the number of CPUs idled
Imp	User impact (YES or NO) - does this event impact user availability
Owner	Owner of the event - person assigned responsibility for following this event
Short description	Short description of the event
Assignee	Person last assigned responsibility for actions on this event
Creator	Person who created original event record

Table 3.1 shows that there is much redundant information included in the raw data of the events log file, and some important information which relate to the failure analysis are not directly presented, such as the time between failure (TBF). We recognize that such events log information must be carefully filtered and preprocessed before being used in decision making, since they contain a large amount of redundant information. We present our preprocessing work, which involves three steps:

The first step is extracting the information we require from the raw data. For the purpose of our failure and reliability study, only the information such as the event ID, failure type, failure start time, failure end time, failure host and down time are concerned. We extract these useful attributes from the raw data and build a smaller dataset.

Secondly, we perform a temporal filtering step to remove duplicate or error reports. Some obvious error report and suspected failure events are removed from the raw data. For example, in some failure cases, the failure ending time are previous to the starting time or the down time is zero.

Thirdly, a categorizing step is performed. Among the failure events, some failures are involved by a single node, and some by multiple nodes, and the target of fundamental failure analysis is the single node reliability. For the convenience of further analysis to access the data, we separate the multiple nodes involved failure from each other.

Therefore, after data preprocessing, the advantages of the new dataset is obvious: 1) Unwanted information has been removed; the new data size is only 18% of the original raw data. 2) Failures can be indexed by node. 3) Failures can be indexed by time. 4) Failures can be indexed by type. 5) Time between failures (TBF) of each node is given. 6) Downtime of each failure is calculated. After preprocessing, it is easy to collect the failure information by node, type, and time period. Table 3.2 is a data example of preprocessed failure information.

Table 3.2 The sample of preprocessed failure data.

Id	Type	TDT (hr)	Host list	Start-date	Start-time	End-date	End-time
1899	HW	24	225	7/19/2000	16:26:00	7/20/2000	16:21:30
1901	HW	20	113	7/20/2000	14:31:00	7/21/2000	10:26:32
1907	SW	3	129	7/21/2000	9:35:00	7/21/2000	12:35:00
1911	HW	3	210	7/24/2000	11:37:06	7/24/2000	14:27:35
1913	HW	2	287	7/24/2000	13:23:25	7/24/2000	15:00:01
1914	HW	2	265	7/24/2000	13:28:56	7/24/2000	15:20:01
1915	HW	1	026	7/24/2000	14:07:21	7/24/2000	15:45:01
1917	HW	33	275	7/24/2000	23:52:00	7/26/2000	8:00:00
1920	SW	142	099	7/25/2000	13:30:31	7/31/2000	11:35:08

3.3 Failure Data Combination of *k* Nodes

In this section, we discuss the reliability of a group of k nodes. In the large-scale cluster, normally, the system job scheduler submits user jobs to a group of nodes, and in such a situation, the group of nodes is treated as a whole system, and each node in the group is a component of the system. Each node failure may cause the whole system to fail: it is the typical series reliability model. For convenient study, we ignore the failure dependency and suppose the failure of each node is independently distributed, so the reliability of a group of k nodes can be achieved by the following failure data combination method. In fact, this failure independency is a trend of HPC hardware architecture design and deployment for which nodes or components are independent and can be repaired in a hot swap manner.

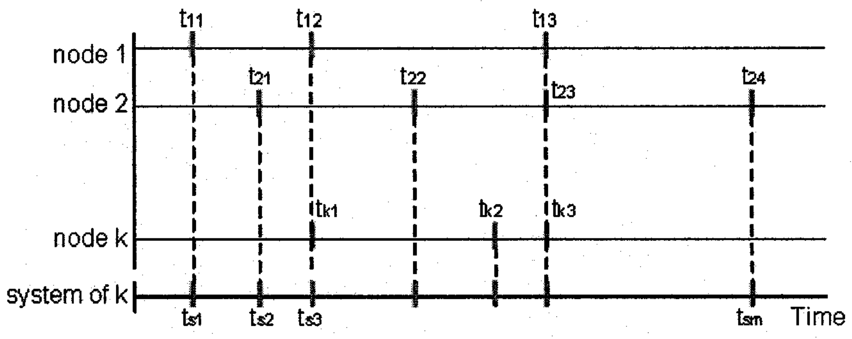


Figure 3.1 Failure data combination of a group of k nodes.

Figure 3.1 shows the method to combine the failure data from individual nodes to the failure dataset of a group. F_{1n}, F_{2n}, F_{3n} ......, F_{kn} are failure data sets of node 1, 2, 3......k , and they are presented in Equation (3.1), where t is the failure time (for example, t_{31} means happen time of the 3rd failure of node 1).

$$\begin{aligned} F_{n1} &= \{t_{11}, t_{21}, t_{31}, ...\} \\ F_{n2} &= \{t_{12}, t_{22}, t_{32}, ...\} \\ &\cdots\cdots \\ F_{nk} &= \{t_{1k}, t_{2k}, t_{3k}, ...\} \end{aligned} \tag{3.1}$$

The combination algorithm of failure data set is in Equation (3.2), and the failure data set of a system of k node is F_s, which is the union operation of each node failure set. After the failure data set F_s is acquired, the time between failures (*TBF*) can be calculated by Equation (3.3).

$$F_s = F_{n1} \cup F_{n2} ...... \cup F_{nk} = \{t_{s1}, t_{s2}, t_{s3}, ...t_{sm}\} \tag{3.2}$$

$$TBF_s = \{t_{s(i+1)} - t_{si}\} \ , i = 1,2,3......m \tag{3.3}$$

3.4 Statistical Analysis

After TBF datasets of individual nodes are acquired, we consider statistical techniques to analyze the TBF datasets. We calculate empirical cumulative distribution function (ECDF) and how well it fits to four probability distributions commonly used in reliability theory: the exponential, the Weibull, the gamma, and the lognormal distribution. We use a maximum likelihood estimation to parameterize the distributions and evaluate the goodness of fit by visual inspection and Chi-square test or Kolmogorov-Smirnov test.

3.4.1 Distribution Fitting of TBF

Distribution fitting is the procedure of selecting the statistical distribution which best fits to a data set generated by some random process. In reliability analysis field, there are four commonly utilized distributions: exponential, Weibull, gamma and lognormal. The details of their mathematic definition have been introduced in Section 2.2. We compared the empirical cumulative distribution function (ECDF) of the TBF dataset with these four normally used distributions. The fittings are performed with TBF datasets which come from a different node group, and within different time periods. The graphs of fitting results are listed in Section 3.5 .

3.4.2 Goodness-Fit-Test

It is difficult to determine the best distribution fitting result from observation or visual inspection. We perform the goodness-fit-test, Chi-Squared test and Kolmogorov-Smirnov test to measure the fitting result in quantitative comparisons. The details of goodness-fit-test theories listed in Appendix C.

3.5 Analysis Results

In this section, some distribution fitting results are presented in Figure 3.2 to Figure 3.13. It is based on the TBF dataset of ASC White system from January 2001 to October 2004. We fitted the failure dataset in a different time period. Because the White system was unstable and got more failure events involved in year 2001 and 2002, we separated the failure data in half year periods to make the fitting. In years 2003 and 2004, we fitted the data in the whole year. The distribution fitting graphs over different period are shown in Figure 3.2 to Figure 3.13.

The parameters of each fitted distribution over different data set are listed in Table 3.3, Table 3.5, Table 3.7, Table 3.9, Table 3.11, and Table 3.13. In the goodness-fit-test, we did the Chi-square test and Kolmogorov-Smirnov Test, and the result shows that Kolmogorov-Smirnov Test has manifest difference in P-value to identify the best fitting distribution; therefore, we only list results of Kolmogorov-Smirnov Test of different data set in Table 3.4, Table 3.6, Table 3.8, Table 3.10, Table 3.12, and Table 3.14.

P-values in Kolmogorov-Smirnov Test denote that Weibull Distribution is the best fit in most cases on the given failure data set. Thus, we will focus our discussion on the checkpoint/restart model centered on the Weilbull distribution in Chapter Four.

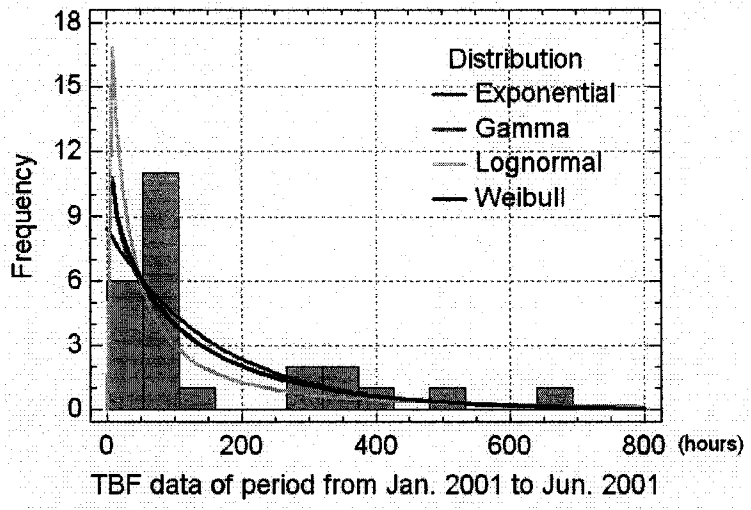


Figure 3.2 Histogram of TBF data of White in period Jan. 2001-Jun. 2001.

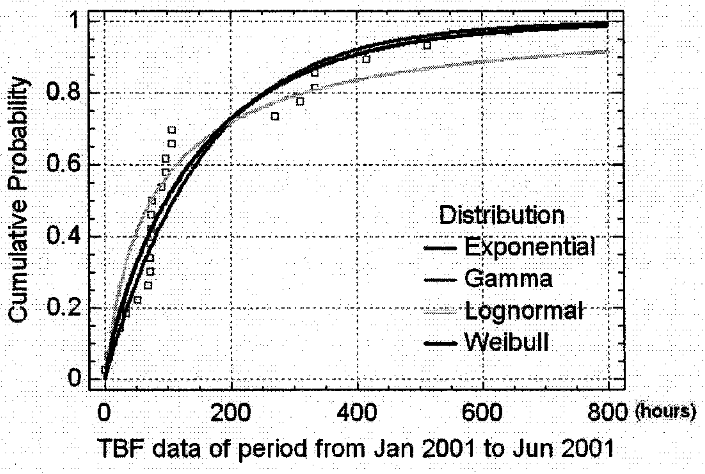


Figure 3.3 CDF Comparison of TBF data of White in period Jan. 2001-Jun. 2001.

Table 3.3 Fitted Distributions of data in period Jan. 2001-Jun. 2001.

Exponential	Gamma	Lognormal	Weibull
mean = 158.25	shape = 0.782987	mean = 329.945	shape = 0.870012
	scale = 0.00494779	standard deviation = 1433.69	scale = 148.213
		Log scale: mean = 4.30404	
		Log scale: std. dev. = 1.7291	

Table 3.4 K-S test of the fitting of data in period Jan. 2001-Jun. 2001.

	Exponential	Gamma	Lognormal	Weibull
DPLUS	0.228156	0.193604	0.135276	0.190498
DMINUS	0.115434	0.164208	0.245532	0.16395
DN	0.228156	0.193604	0.245532	0.190498
P-Value	0.148168	0.307531	0.0981672	0.326552

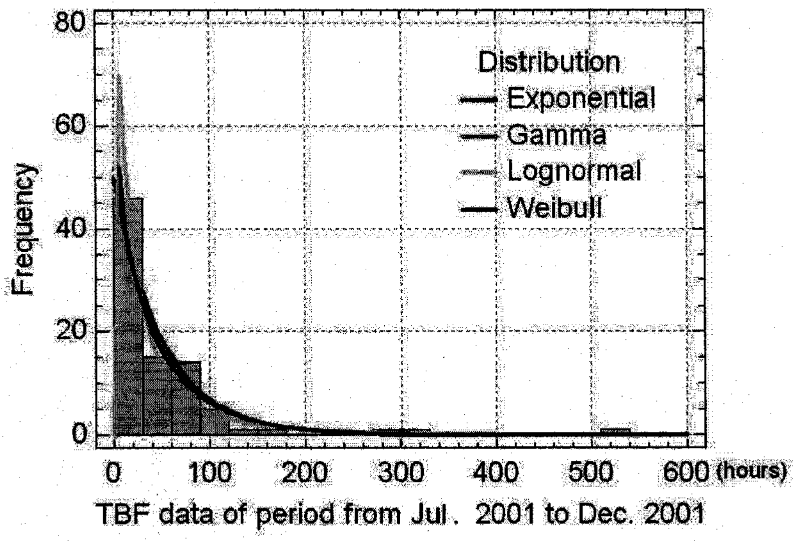


Figure 3.4 Histogram of TBF data of White in period Jul. 2001-Dec. 2001.

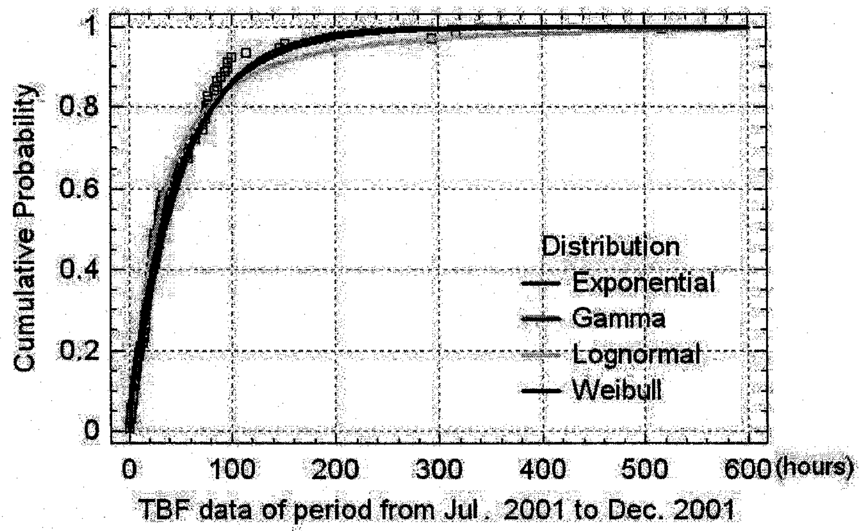


Figure 3.5 CDF Comparison of TBF data of White in period Jul. 2001-Dec. 2001.

Table 3.5 Fitted distributions of data in period Jul. 2001-Dec. 2001.

Exponential	Gamma	Lognormal	Weibull
mean = 51.1349	shape = 0.859612	mean = 61.2761	shape = 0.875449
	scale = 0.0168107	standard deviation = 131.993	scale = 47.4309
		Log scale: mean = 3.25045	
		Log scale: std. dev. = 1.31525	

Table 3.6 K-S test of the fitting of data in period Jul. 2001-Dec. 2001.

	Exponential	Gamma	Lognormal	Weibull
DPLUS	0.117513	0.0937095	0.0806108	0.0754125
DMINUS	0.0320601	0.0647599	0.117052	0.0827226
DN	0.117513	0.0937095	0.117052	0.0827226
P-Value	0.191281	0.452011	0.194836	0.605893

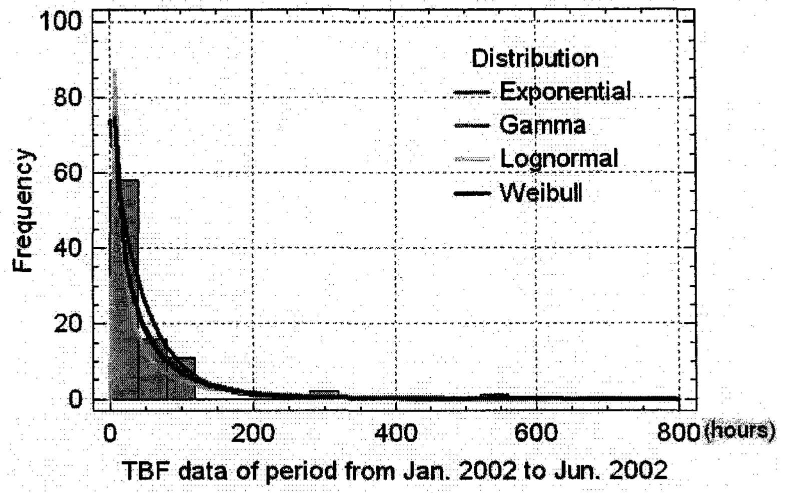


Figure 3.6 Histogram of TBF data of White in period Jan. 2002-Jun. 2002.

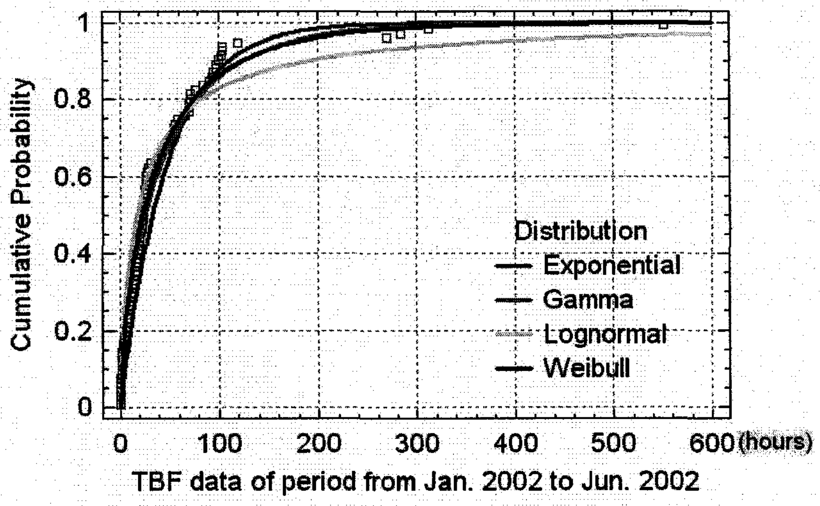


Figure 3.7 CDF Comparison of TBF data of White in period Jan. 2002-Jun. 2002.

Table 3.7 Fitted distributions of data in period Jan. 2002-Jun. 2002.

Exponential	*Gamma*	*Lognormal*	*Weibull*
mean = 48.1436	shape = 0.600256	mean = 100.322	shape = 0.71623
	scale = 0.012468	standard deviation = 576.495	scale = 38.6631
		Log scale: mean = 2.8449	
		Log scale: std. dev. = 1.87803	

Table 3.8 K-S test of the fitting of data in period Jan. 2002-Jun. 2002.

	Exponential	*Gamma*	*Lognormal*	*Weibull*
DPLUS	0.18572	0.100186	0.112908	0.0771213
DMINUS	0.0412279	0.0927998	0.172895	0.102403
DN	0.18572	0.100186	0.172895	0.102403
P-Value	0.00431175	0.335838	0.00977668	0.309882

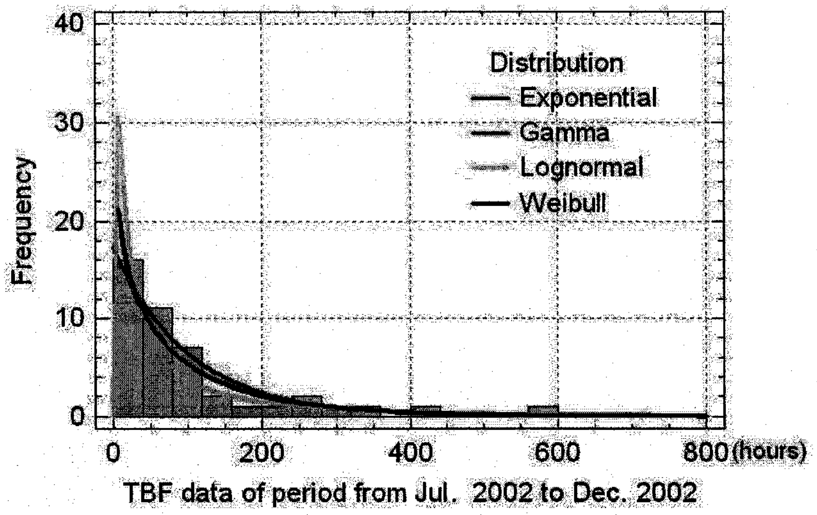


Figure 3.8 Histogram of TBF data of White in period Jul. 2002-Dec. 2002.

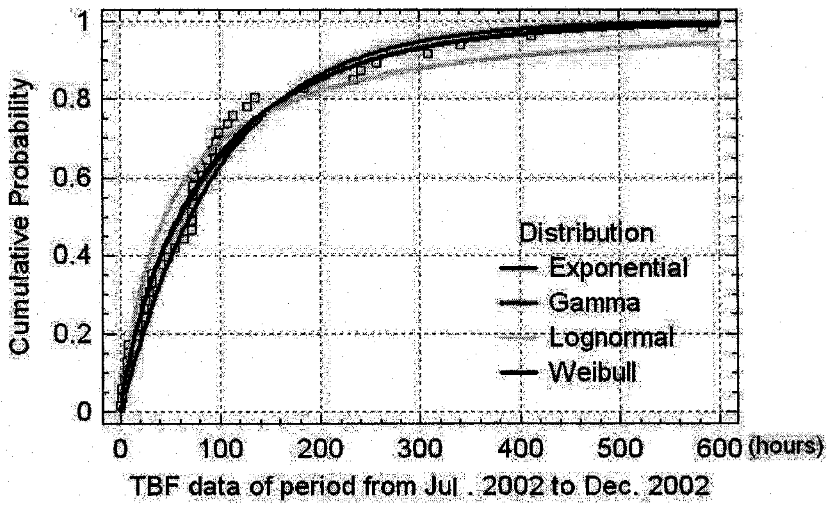


Figure 3.9 CDF Comparison of TBF data of White in period Jul. 2002-Dec. 2002.

Table 3.9 Fitted distributions of data in period Jul. 2002-Dec. 2002.

Exponential	Gamma	Lognormal	Weibull
mean = 101.241	shape = 0.740914	mean = 168.113	shape = 0.825671
	scale = 0.00731831	standard deviation = 604.122	scale = 91.5017
		Log scale: mean = 3.8082	
		Log scale: std. dev. = 1.62261	

Table 3.10 K-S test of the fitting of data in period Jul. 2002-Dec. 2002.

	Exponential	Gamma	Lognormal	Weibull
DPLUS	0.105249	0.0857462	0.0666097	0.0761836
DMINUS	0.059705	0.0920687	0.157378	0.103101
DN	0.105249	0.0920687	0.157378	0.103101
P-Value	0.714328	0.849785	0.226341	0.737854

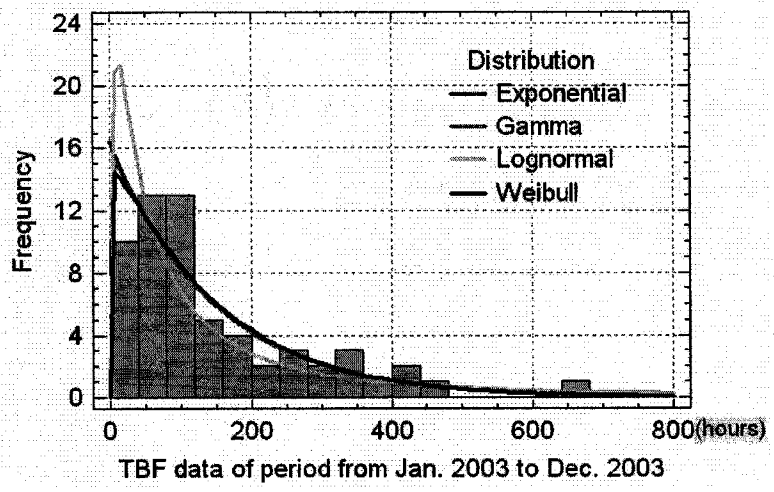


Figure 3.10 Histogram of TBF data of White in period Jan. 2003-Dec. 2003.

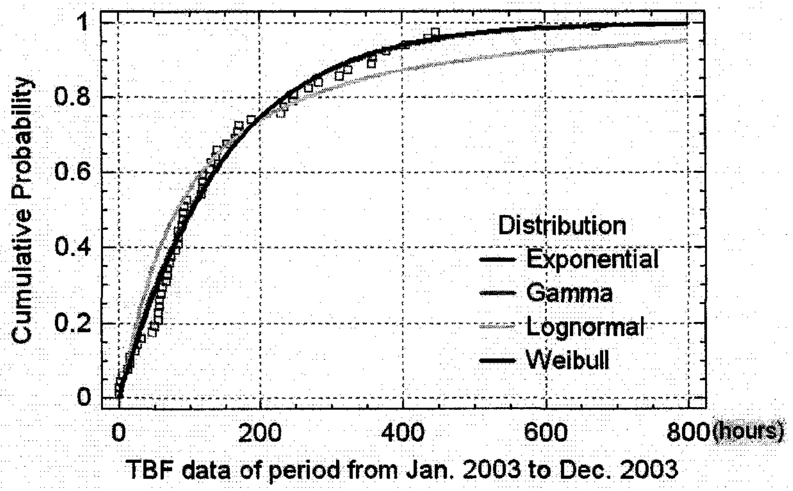


Figure 3.11 CDF Comparison of TBF data of White in period Jan. 2001-Jun. 2001.

Table 3.11 Fitted distributions of data in period Jan. 2003-Dec. 2003.

Exponential	Gamma	Lognormal	Weibull
mean = 145.922	shape = 1.01495	mean = 216.717	shape = 0.83365
	scale = 0.00695545	standard deviation = 524.853	scale = 147.787
		Log scale: mean = 4.41535	
		Log scale: std. dev. = 1.38798	

Table 3.12 K-S test of the fitting of data in period Jan. 2003-Dec. 2003.

	Exponential	Gamma	Lognormal	Weibull
DPLUS	0.0539832	0.0560491	0.0958043	0.0632786
DMINUS	0.118476	0.115451	0.189113	0.106782
DN	0.118476	0.115451	0.189113	0.106782
P-Value	0.3723	0.405675	0.0273654	0.513283

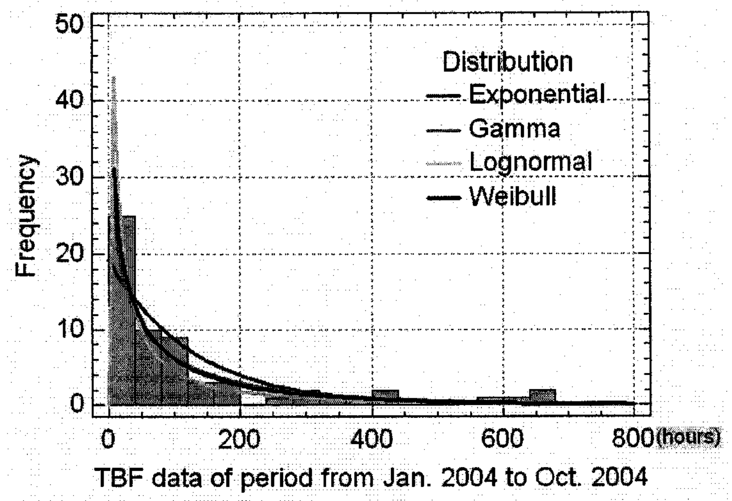


Figure 3.12 Histogram of TBF data of White in period Jan. 2004-Oct. 2004.

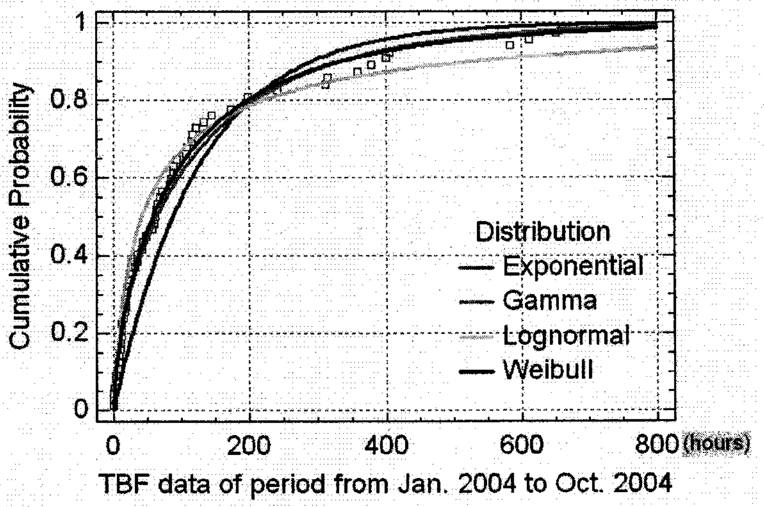


Figure 3.13 CDF Comparison of TBF data of White in period Jan. 2004-Oct. 2004.

Table 3.13 Fitted distributions of data in period Jan. 2004-Oct. 2004.

Exponential	Gamma	Lognormal	Weibull
mean = 126.681	shape = 0.566654	mean = 293.391	shape = 0.687184
	scale = 0.00447308	standard deviation = 2019.51	scale = 99.2228
		Log scale: mean = 3.74196	
		Log scale: std. dev. = 1.96954	

Table 3.14 K-S test of the fitting of data in period Jan. 2004-Oct. 2004.

	Exponential	Gamma	Lognormal	Weibull
DPLUS	0.17424	0.072916	0.0811963	0.0516248
DMINUS	0.0792948	0.0561746	0.101738	0.0532796
DN	0.17424	0.072916	0.101738	0.0532796
P-Value	0.0492558	0.901922	0.552961	0.99515

CHAPTER 4

AN OPTIMAL CHECKPOINT/RESTART MODEL

4.1 Introduction

Our analysis of the failure data of LLNL ASC White system (512 nodes and 8196 processors) shows the average value of mean time between failures (MTBF) for any individual node is over 7000 hours, but MTBF of the system is only around 20 hours due to the combination of hardware and software failure of each components. Because of the characteristics of the parallel applications running in a large-scale system, the reliability of the whole system or reliability of k of n nodes is a bigger concern than of any individual component. The MPI is the popular parallel environment for scientific computing. Normally, the MPI jobs are decomposed and spread over a bunch of nodes, and each part of the job cooperates to work by passing the message. Because the job data is partitioned over the nodes, single node failure or network connection blockage would cause the whole job to be lost. In order to minimize the performance loss due to the frequently happened failures, the optimal checkpoint/restart scheme is widely used in computing systems. These works, either the cost function models or Markov availability models, they theoretically assume that the system failure is Poisson process (with fixed failure rate λ). But in the practices of a large-scale cluster system, the characteristics of system failure are more complicated than for single component systems.

In this chapter, we develop an optimal checkpoint/restart model related to the system reliability function. The reliability functions are obtained by analyzing the historical failure data from the system events log files. The methods of failure data analysis have been discussed in Chapter Three. We then apply a theory of stochastic renewal reward process for an optimal solution of our checkpoint/restart model. The detailed descriptions for this chapter are organized as follows: Section 4.2 presents related works of optimal checkpoint/restart model. In Section 4.3 , we describe the checkpoint/restart behaviors in distributed environments. The introduction of applying renewal reward process theory to checkpoint/restart model is in Section 4.4 . In Section 4.5 , the mathematical solution of reliability-aware optimal checkpoint/restart model is presented.

4.2 Related Works

The checkpoint/restart method is a typical fault tolerant technique in computer systems. Works by Chandy [63][64] and Treaster [65] can serve as good review documents for the checkpoint/restart technique. Young [66] presented an optimal checkpoint and rollback recovery model, and obtained a constant optimal checkpoint interval by which the total waste time was minimized. Based on Young's work, Daly [68][69] improved the solution to an optimal checkpoint placement from a first order to a higher order approximation. Both Young's and Daly's studies established a principle on a cost function through which the whole execution period was considered, and the optimal checkpoint interval was derived to minimize the output of the cost function. An alternate technique concerning the optimal checkpoint interval is to consider the system availability. In [71][72][73], authors presented a Markov availability model, and obtained an optimal checkpoint placement that maximizes system availability. Essentially, with a particular system failure rate, minimizing the cost function is equivalent to maximizing system availability. However, the limitation of existing optimal placement solutions (either the cost function or availability model) is the fact that system failure is assumed to follow a Poisson process with a fixed failure rate. This implies that the time between failures follows an exponential distribution. A more recent study [44] on the reliability-aware approach in a BlueGene/L proposed a so-called cooperative checkpointing, which is a hybrid checkpointing scheme based on Young's [66] periodic checkpoint model. The cooperative checkpointing scheme may reduce the checkpointing cost, especially in a large-scale parallel system. However, it risks of increasing the rollback cost. Our technique addresses both overhead and rollback time.

4.3 Checkpoint/Restart in Distributed System

A typical parallel application is divided into subtasks running on multiple nodes. In order to deal with the reliability issue, the application states must be periodically saved by a well designed checkpoint protocol. There are three common checkpoint protocols: uncoordinated checkpointing, coordinated checkpointing, and communication-induced checkpointing. In our cost model construction, we experimented and measured actual checkpoint overhead based on the Berkeley Lab's Linux Checkpoint/Restart (BLCR) implementation. BLCR is the Linux kernel-based coordinate Checkpoint/Restart technique, and is integrated with the LAM/MPI to provide checkpoint and restart for a parallel application.

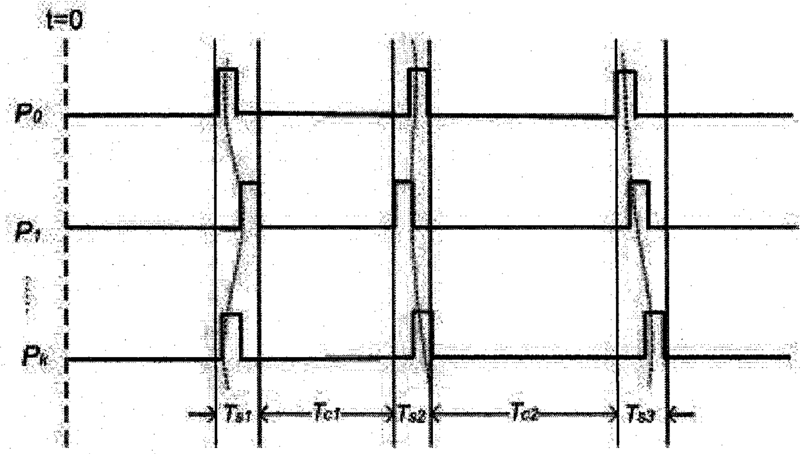


Figure 4.1 Coordinate checkpointing for parallel application.

Figure 4.1 shows a general idea of the coordinate checkpoint mechanism in the case of a parallel application. Consider a parallel program running with k processes, P_0, $P_1 \ldots \ldots P_k$, and coordinate checkpoint placements, $T_{c1}, T_{c2}, \ldots \ldots T_{cm}$, that represent the checkpoint intervals between two checkpoints. Let $T_{s1}, T_{s2}, \ldots \ldots T_{sm}$ be the checkpoint overhead of each checkpoint. Since the coordinate checkpoint protocol behaves in a synchronized fashion, we assume that there is no time difference for each individual process checkpoint, and treat it as a single checkpoint overhead, T_{si}. Thus, in this study, we focus on how to determine a checkpoint interval or placement that minimizes the job execution time or total waste time.

4.4 Checkpoint/Restart and Stochastic Renewal Process

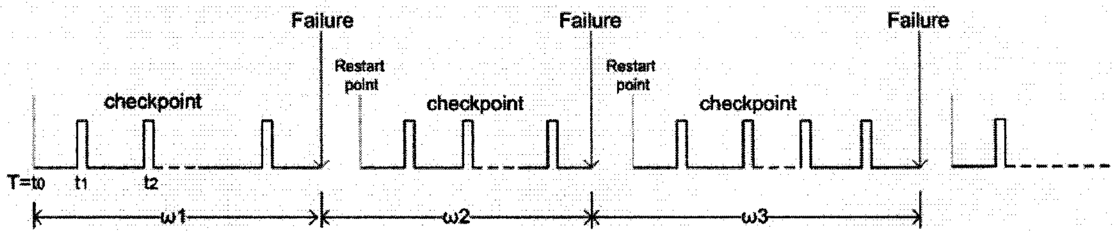


Figure 4.2 Checkpoint/restart is a stochastic renewal reward process.

We consider an application failure model that allows more than one failures during a lifetime of a given application. The checkpoint/restart model can be shown as in Figure 4.2 which represents by a typical stochastic renewal reward process. Ideally, the whole checkpoint procedure is divided by random failures, where $\omega_1, \omega_2, \omega_3 ...$ are random time intervals between failures, and $t_1, t_2, t_3 ...$ are the checkpoint placement time. In the stochastic renewal process, ω denotes a renewal interval or repeat cycle. For each cycle ω, the behavior of the process is similar. The overall checkpoint/restart procedure can be described as C_t in Equation (4.1)

$$C_t = \max\{n : \sum_{i=1}^{n} \omega_i \le t\}. \tag{4.1}$$

We suppose that execution time lost due to the checkpoints and failure on each cycle is $W_1, W_2, W_3 ...$ (reward), then the total lost time on process C_t is:

$$Y_t = \sum_{i=1}^{C_t} W_i . \tag{4.2}$$

Y_t is called a renewal reward process. The important theorem of renewal reward process is described in Equation (4.3).

$$\lim_{t \to \infty} \frac{E(\sum_1^n W_i)}{t} = \frac{E(W_1)}{E(\omega_1)} \tag{4.3}$$

Equation (4.3) represents the total average reward equal to the average reward in the first cycle. In the checkpoint/restart model, this fact means that the problem of how to minimize the overall time lost is equal to how to minimize the lost time in cycle ω_1.The details of the renewal reward process theory are presented in Appendix A

4.5 Description of Checkpoint/Restart Model

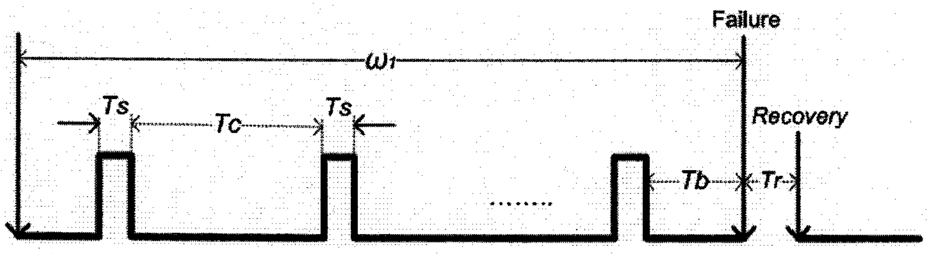


Figure 4.3 Behaviors of checkpoint/restart model.

Figure 4.3 shows the behaviors of checkpoint restart scheme on an application interrupted by failures. The meaning of each symbol that we use in the model is shown in Table 4.1.

Table 4.1 Parameters of checkpoint/restart model.

Parameters	Meaning
Tc	Checkpoint Interval
Ts	Checkpoint Overhead
Tr	Recovery Time
T_b	Time to rollback to the last checkpoint
n(t)	Checkpoint frequency function
f(t)	Probability density function of TBF
ω_i	The cycle between failure *i* and failure *(i+1)*

Consider a checkpoint restart scheme in the first cycle ω_1 (interval between two failures). Figure 4.3 illustrates the model with parameters such as checkpoint interval (*Tc*), checkpoint overhead (*Ts*), restart (*Tr*) and rollback time (*Tb*). Through the rest of the paper, our failure model is based on the following assumptions:

1. A running application may be interrupted by a series of random failures *Si*, (i = 1,2,3...), where the time between failures has a certain PDF, f(t).
2. The system failure can be detected by a monitoring mechanism.
3. Checkpoint intervals need not be fixed, but can vary.
4. Each checkpoint overhead *Ts* is a constant. In practice, we can take this constant to be the average value of multiple checkpoint overheads.
5. The system can be recovered from the last checkpoint, and the rollback cost *Tb* is a period between the last checkpoint and the present failure.
6. The repair time *Tr* is a constant.

Remark: Assumption 2 is satisfied, since a well-managed system [45] can be engineered with an efficient mechanism to immediately detect the failure. In assumption 3, the checkpoint interval may be constant or variable depending on the PDF of system failure. Thus, the checkpoint interval can be described by a checkpoint frequency function, *n(t)*. The mathematical derivation of *n(t)* is given in the next subsection. Finally, assumption 6, T_r, is satisfied if there is a mechanism in place to replace the failed node with a spared node. In fact, HA-OSCAR and its new fault tolerance extension [45] can provide a transparent recovery time *Tr* less than a minute.

In Figure 4.3, ω_i is the ith cycle between ith failure and (i+1)th failures. From the discussion in Section 4.4 , we recognize that an optimal checkpoint/restart model must follow a specific failure distribution. Therefore, we look for the best checkpoint placement sequence that minimizes the total waste time. In addition, according to the renewal reward process and from Equation (4.3), one can minimize the total waste time by minimizing lost execution time in the first cycle, ω_1.

Let the sequence of discrete checkpoint placement moments are $0 = t_0 < t_1 < ... < t_n$. If the *n(t)* is the checkpoint frequency function, then

$$\int_{t_i}^{t_{i+1}} n(\tau)d\tau = 1, \qquad i = 0, 1, 2 ... m. \tag{4.4}$$

For any given time interval from the beginning of each cycle, denoted by *L,* the number of checkpoint placements is given by

$$N(L) = \int_0^L n(\tau)d\tau \ . \tag{4.5}$$

In Figure 4.3, let W_1 be the time wasted due to the checkpointing in cycle ω_1,

$$W_1 = T_s \int_0^{\omega_1} n(\tau)d\tau + T_b + T_r. \tag{4.6}$$

From assumption 6, T_r is a constant, and from assumption 5, we suppose that the system can be successfully recovered from the last checkpoint. The relationship between T_b and checkpoint interval is illustrated in Figure 4.4.

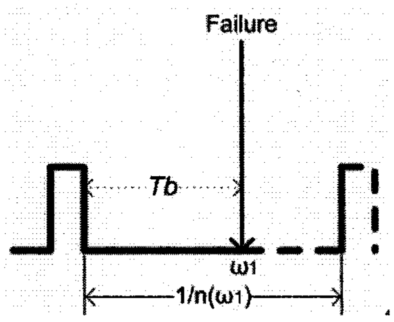


Figure 4.4 The relationship of Tb and checkpoint interval.

Since ω_1 is the value between these checkpoint placements, by Mean Value theorem, we can estimate the frequency of this interval by $n(\omega_1)$. Therefore, T_b can be expressed by Equation (4.7), where k is a rollback coefficient variable between (0,1). The detailed discussion of k value is explained in Section 5.1 .

$$T_b \approx k/n(\omega_1),\ (0 < k < 1)\,. \tag{4.7}$$

Therefore, (4.6) can be written as

$$W_1 = T_s \int_0^{\omega_1} n(\tau)d\tau + \frac{k}{n(\omega_1)} + T_r. \tag{4.8}$$

The discussion in Section 4.4 suggests that the checkpoint/restart is a renewal reward process (4.2). The theorem of renewal reward process in Equation (4.3) shows that the mean of overall time lost in the checkpoint/restart process is $E(W_1)/E(\omega_1)$. Therefore, a process to minimize the total time lost in the checkpoint process is to minimize $E(W_1)$. Let *f(t)* be the probability density function of the system failure, then the probability for the system failure within $[t, t+\Delta t]$ is $f(t)\cdot\Delta t$. The expected time lost during a cycle in the checkpoint process $E(W_1)$ is

$$E(W_1) = \int_0^{\infty}\left[\int_0^{t} T_s \cdot n(\tau)d\tau + \frac{k}{n(t)} + T_r\right]\cdot f(t)dt\,. \tag{4.9}$$

Now we aim for the optimal checkpoint frequency $n^*(t)$ in order to minimize the expected time lost, as defined by the Equation (4.9).

<u>Solution</u>: Let $x(t) = \int_0^t n(\tau)d\tau$, and then Equation (4.9) changes to

$$E(W_1) = \int_0^{\infty}[T_s \cdot x(t) + \frac{k}{x'(t)} + T_r]\cdot f(t)dt\,. \tag{4.10}$$

Let

$$\Phi(x, x', t) = [T_s \cdot x(t) + \frac{k}{x'(t)} + T_r]\cdot f(t)\,. \tag{4.11}$$

Based on the theorem of calculus of variations, if the integral in Equation (4.10) has a minimum value, $\Phi(x, x', t)$ must satisfy Euler's equation in (4.12) [83],

$$\frac{\partial \Phi}{\partial x} - \frac{d}{dt} \cdot \frac{\partial \Phi}{\partial x'} = 0 . \tag{4.12}$$

In Equation (4.11), we take the partial derivative of Φ and obtain

$$\begin{cases} \dfrac{\partial \Phi}{\partial x} = T_s \cdot f(t) \\ \\ \dfrac{\partial \Phi}{\partial x'} = \dfrac{-k}{(x'(t))^2} \cdot f(t) \end{cases} . \tag{4.13}$$

We substitute (4.13) into (4.12)

$$T_s \cdot f(t) + \frac{d}{dt} \cdot \frac{k}{(x'(t))^2} \cdot f(t) = 0 . \tag{4.14}$$

Taking Integral on both sides of (4.14), we get

$$T_s \cdot F(t) + \frac{k}{(x'(t))^2} \cdot f(t) = C . \tag{4.15}$$

Where C is a constant, and because if $x'(t) = n(t)$, then

$$T_s \cdot F(t) + \frac{k}{(n(t))^2} \cdot f(t) = C . \tag{4.16}$$

Since $\lim_{t \to \infty} F(t) = 1$, and $\lim_{t \to \infty} f(t) = 0$, so we have $C = T_s$, Solving for n(t) , we obtain the following optimal frequency function:

$$n^*(t) = \sqrt{\frac{k \cdot f(t)}{T_s(1 - F(t))}} = \sqrt{\frac{k}{T_s}} \cdot \sqrt{\frac{f(t)}{R(t)}} . \tag{4.17}$$

1) Exponential Distribution

For the exponential distribution, $f(t) = \lambda e^{-\lambda t}$, and $R(t) = e^{-\lambda t}$, $t \ge 0$, $\lambda > 0$, we get

$$n^*(t) = \sqrt{\frac{k}{T_s}} \cdot \sqrt{\lambda}\,. \tag{4.18}$$

So the optimal checkpoint interval for exponential failure distribution is

$$\frac{1}{n^*(t)} = \sqrt{\frac{T_s}{k\lambda}}\,. \tag{4.19}$$

Form (4.19) shows that the optimal checkpoint interval for exponential is fixed value.

2) Weibull Distribution

For Weibull distribution, where $f(t) = \dfrac{\beta t^{\beta-1}}{\eta^\beta} e^{-(t/\eta)^\beta}$, $t \ge 0$ and

$R(t) = e^{-(t/\eta)^\beta}$, $t \ge 0$, then the Equation (4.17) becomes

$$n^*(t) = \sqrt{\frac{k}{T_s}} \cdot \sqrt{\frac{\beta t^{\beta-1}}{\eta^\beta}} = \sqrt{\frac{k\beta}{T_s\eta^\beta}} \cdot t^{\frac{\beta-1}{2}}\,. \tag{4.20}$$

Form (4.20) satisfies (4.4), i.e.

$$\int_{t_{i-1}}^{t_i} \sqrt{\frac{k\beta}{T_s\eta^\beta}} \cdot t^{\frac{\beta-1}{2}} dt = 1,\ i = 1,2,3.... \tag{4.21}$$

$$\left(\frac{2}{\beta+1}\sqrt{\frac{k\beta}{T_s\eta^\beta}}\right) \cdot (t_i^{\frac{\beta+1}{2}} - t_{i-1}^{\frac{\beta+1}{2}}) = 1\,. \tag{4.22}$$

$$t_i = \left(\frac{\beta+1}{2}\sqrt{\frac{T_s\eta^\beta}{k\beta}} + t_{i-1}^{\frac{\beta+1}{2}}\right)^{\frac{2}{\beta+1}}. \tag{4.23}$$

Because $t_0 = 0$, we obtain the sequence of optimal checkpoint placements (timestamps)

$$t_1 = \left(\frac{\beta+1}{2} \sqrt{\frac{T_s \eta^\beta}{k\beta}} \right)^{\frac{2}{\beta+1}}, t_2 = \left(\frac{\beta+1}{2} \sqrt{\frac{T_s \eta^\beta}{k\beta}} + t_1^{\frac{\beta+1}{2}} \right)^{\frac{2}{\beta+1}}, \ldots\ldots$$
$$t_i = \left(i \frac{\beta+1}{2} \sqrt{\frac{T_s \eta^\beta}{k\beta}} \right)^{\frac{2}{\beta+1}} \tag{4.24}$$

where t_i is the i^{th} checkpoint placement. In the next chapter, we use t^i from Equation (4.24) in our model evaluation.

In next chapter, we perform model analysis and comparison to current state-of-art techniques.

CHAPTER 5

MODEL ANALYSIS AND VALIDATION

In Chapter Four, we presented an optimal checkpoint/restart model aiming to minimize the total applications lost time due to checkpoint overhead and rollback time when failure occurs. The solution of the optimal checkpoint frequency (Equation 4.17) is a function of system failure rate, checkpoint overhead, and the rollback coefficient *k*. The checkpoint overhead is the time cost for a checkpointing program to take the snapshot of the application. In the distributed environment, the checkpoint overhead is affected by many factors: the performance of checkpointing program, the node size or process size that the application is spread on, the memory usage of the application in the checkpointing moment, the system load of the platform, and so on.

In the model evaluation, we assume the checkpoint overhead T_s is a constant for the particular execution environment. The main focus of this dissertation is the relationship between the checkpoint interval or frequency and system failure. First, we present the method to evaluate the rollback coefficient *k* in the model. Secondly, we compare the total cost of our solution vs. other checkpoint models.

5.1 Evaluation of Rollback Coefficient *k*

In Figure 4.4 in Chapter Four, we consider T_b, a rollback time of the application after a given failure. T_b is the time interval between the failure and the last checkpoint. It is a random value which depends on the time when the failure occurs from the last checkpoint placement.

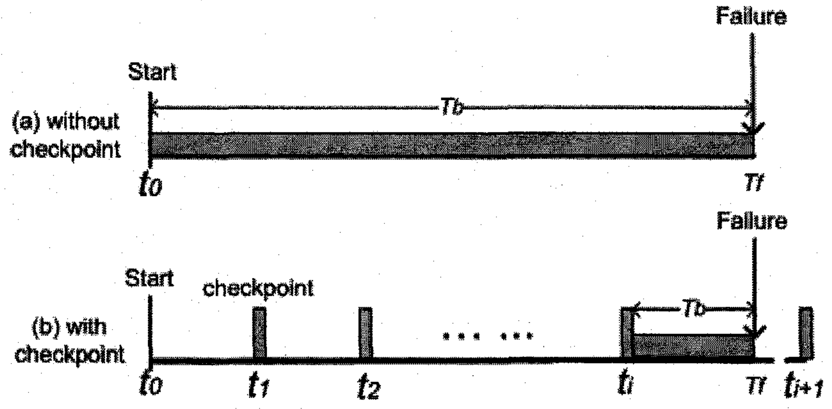


Figure 5.1 T_b in different case. (a) without checkpoint, (b) with checkpoint.

If a failure occurs at time T_f, during an application execution without checkpoint (Figure 5.1 (a)), and then the rollback time T_b will be 100% of the time interval $T_f - t_0$. With checkpoints (Figure 5.1 (b)), it is obvious that T_b is a random value dependent on the time the failure occurs. Therefore, if we know the distribution of the time between failures, then T_b can be estimated.

First, let's define the rollback time coefficient k in Equation (5.1).

$$k = \frac{T_b}{t_{i+1} - t_i} = \frac{T_f - t_i}{t_{i+1} - t_i}, \quad k \in (0,1) \tag{5.1}$$

where $k \in [0,1]$, is the proportion of T_b in the interval $t_{i+1} - t_i$, which depends on the failure time and the checkpoint time sequence.

For a given system TBF dataset followed Weibull distribution, so the checkpoint timestamps $t_1, t_2 \ldots\ldots, t_i$ can be find out by Equation (4.24),

$$t_i = \left(i\frac{\beta+1}{2}\sqrt{\frac{T_s\eta^\beta}{k\beta}} \right)^{\frac{2}{\beta+1}}, \qquad 0<k<1$$

where T_s, the checkpoint overhead, is a constant that can be figure out from the checkpoint program in the actual system. β and η are parameters of fitted Weibull distribution on dataset *I*. In the first step, we know that *k* is a value between 0 and 1, for example *k*=0.1, for convenience, we denote the assumed k as $\hat{k}$, and the expected k to be presented as $\bar{k}$.

Let $\hat{k}=a, 0<a<1$, and let the corresponding checkpoint time sequence, $\{t_1, t_2 \ldots\ldots, t_i\}$, calculated from Equation.(4.24), be as shown in Figure 5.1 (b). We denote the probability that a failure occurs in the time interval (t_0, t_1) as

$$P_0 = P[0<T_f<t_1]$$

Therefore, the probability that failure will happen in (t_1, t_2) is

$$P_1 = P[t_1<T_f<t_2]$$

In general, the probability that failure will happen in (t_i, t_{i+1}) is

$$P_i = P[t_i<T_f<t_{i+1}]. \tag{5.2}$$

Since it is more realistic to assume that the time to failure follows a Weibull distribution with $F(t)=1-e^{-(t/\eta)^\beta}, t\geq 0$,

$$\begin{aligned} P_i = P[t_i<T_f\leq t_{i+1}] &= P[T_f\leq t_{i+1}]-P[T_f<t_i] \\ &= 1-e^{-t_{i+1}^\beta/\eta^\beta}-(1-e^{-t_i^\beta/\eta^\beta}) \, . \\ &= e^{-t_i^\beta/\eta^\beta}-e^{-t_{i+1}^\beta/\eta^\beta} \end{aligned} \tag{5.3}$$

The excess life probability function of a Weibull distribution shown in Equation (5.4), is based on the conditional probability, $P(t+s\,|\,t)$, that the system survives until time $t+s$ (s equal or larger than 0), given that it survived to time t.

$$P[t+s\,|\,t]=\frac{P[t+s]}{P[t]}=\frac{1-F(t+s)}{1-F(t)} \tag{5.4}$$
$$=\frac{e^{-(\frac{t+s}{\eta})^{\beta}}}{e^{-(\frac{t}{\eta})^{\beta}}}=e^{-(\frac{t+s}{\eta})^{\beta}+(\frac{t}{\eta})^{\beta}}=e^{\frac{1}{\eta^{\beta}}[t^{\beta}-(t+s)^{\beta}]}$$

Taking the derivative of Equation (5.4) with respect to s, one obtains the PDF of the excess life distribution in Equation (5.5)

$$f[t+s\,|\,t]=\frac{d}{ds}(e^{\frac{1}{\eta^{\beta}}[t^{\beta}-(t+s)^{\beta}]})=\frac{1}{\eta^{\beta}}\beta(t+s)^{\beta-1}\cdot e^{\frac{t^{\beta}-(t+s)^{\beta}}{\eta^{\beta}}} \tag{5.5}$$
$$=\frac{\beta\cdot(t+s)^{\beta-1}}{\eta^{\beta}}\cdot e^{\frac{t^{\beta}-(t+s)^{\beta}}{\eta^{\beta}}}$$
$$=e^{\frac{t^{\beta}}{\eta^{\beta}}}\cdot\frac{\beta\cdot(t+s)^{\beta-1}}{\eta^{\beta}}\cdot e^{\frac{-(t+s)^{\beta}}{\eta^{\beta}}}$$

We denote the expected excess life at time t_i as $E(s_i)$. If a failure occurs within the time interval (t_i,t_{i+1}), then on average the rollback time $T_b=E(s_i)$, where

$$E(s_i)=\frac{\int_0^{t_{i+1}-t_i}s\cdot f[t_i+s\,|\,t_i]ds}{\int_0^{t_{i+1}-t_i}f[t_i+s\,|\,t_i]ds} \tag{5.6}$$
$$=\frac{e^{\frac{t^{\beta}}{\eta^{\beta}}}\int_0^{t_{i+1}-t_i}s\cdot\frac{\beta\cdot(t_i+s)^{\beta-1}}{\eta^{\beta}}\cdot e^{\frac{-(t_i+s)^{\beta}}{\eta^{\beta}}}ds}{e^{\frac{t^{\beta}}{\eta^{\beta}}}\int_0^{t_{i+1}-t_i}\frac{\beta\cdot(t_i+s)^{\beta-1}}{\eta^{\beta}}\cdot e^{\frac{-(t_i+s)^{\beta}}{\eta^{\beta}}}ds}$$
$$=\frac{e^{\frac{t^{\beta}}{\eta^{\beta}}}\int_0^{t_{i+1}-t_i}s\cdot\frac{\beta\cdot(t_i+s)^{\beta-1}}{\eta^{\beta}}\cdot e^{\frac{-(t_i+s)^{\beta}}{\eta^{\beta}}}ds}{1-e^{\frac{t_i^{\beta}}{\eta^{\beta}}-\frac{t_{i+1}^{\beta}}{\eta^{\beta}}}}$$

From the definition of k in Equation (5.1)

$$k_i = \frac{E(s_i)}{t_{i+1} - t_i}. \tag{5.7}$$

Therefore, the expected k is

$$\bar{k} = \frac{\sum_{i=1}^{n} P_i k_i}{\sum_{i=1}^{n} P_i}. \tag{5.8}$$

where n is the number of checkpoint intervals.

In order to determine β and η in Equation (5.6), we fitted the Weibull distribution to the actual failure data, in our case, from the LNLL ASC White system during the period July, 2004 to September, 2004. The fitted Weibull distribution shown in Figure 5.2 has the shape parameter β= 0.6732 and the scale parameter η=15.56. We assume the checkpoint overhead to be 0.1667 hours. Using these β and η values, we estimated $\bar{k}$ by Equations (5.3), (5.7), and (5.8). For each $\hat{k}$ =a (0<a<1), one has a sequence of checkpoints $(t_1, t_2, t_3, \ldots\ldots, t_i)$ and a corresponding $\bar{k}$ value from Equation (5.8). Figure 5.2 shows a plot of $\hat{k}$ (assumed k) vs. expected k. Our objective is to find the k value in Equation (5.1) such that it is equal to the point where $\hat{k}$ is equal to $\bar{k}$. From our data set shown in Figure 5.3, the k value is approximately 0.4614.

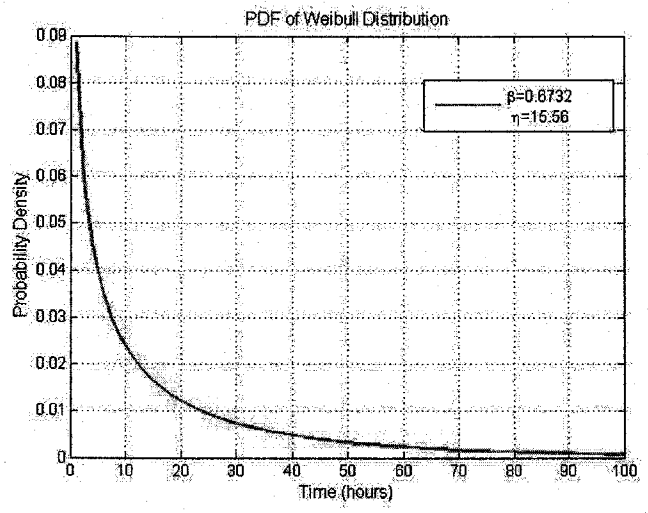


Figure 5.2 Fitted Weibull distribution to the failure data set of the ASC White system.

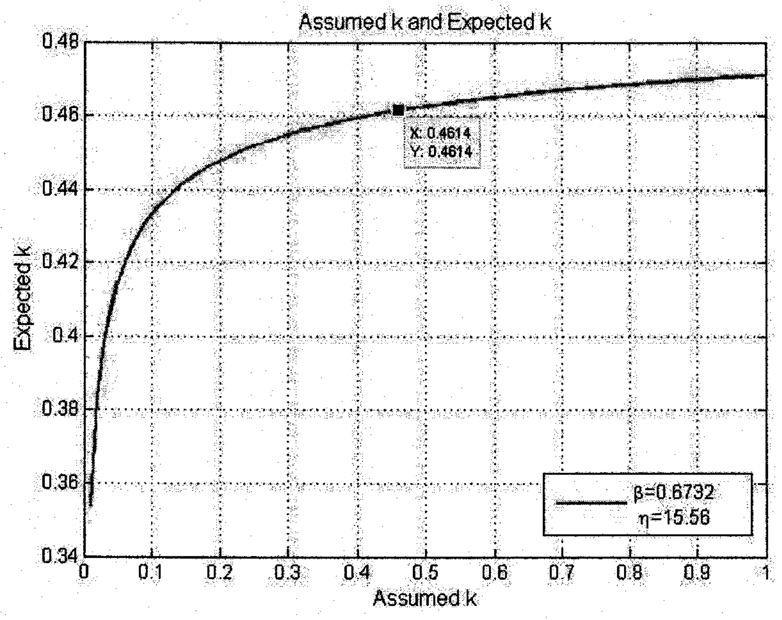


Figure 5.3 Expected k value of a failure data set from the ASC White system.

5.2 Model Validation Results

For an evaluation purpose, we compared our checkpoint/restart model with three existing schemes from a recent large-scale system checkpoint study [44], namely periodic, exponential backoff, and risk-based. The detail of these three models can be found in [44]. One of our main goals is to identify which technique will result in the least lost time by varying the checkpoint overhead. The comparison was performed based on the dataset from the LNLL ASC White system in the period July, 2004 to September, 2004. The time between failures in this data set has a Weibull distribution with β=0.673189 and η=15.5612.

Table 5.1 The k values for model comparisons.

Ts (hrs)	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
$\bar{k}$	0.4682	0.4587	0.4519	0.4564	0.4417	0.4375	0.4338	0.4304	0.4273	0.4244

The $\bar{k}$ values we used in our model are calculated by the method in Section 5.2. The values of different checkpoint overhead T_s are listed in Table 5.1.

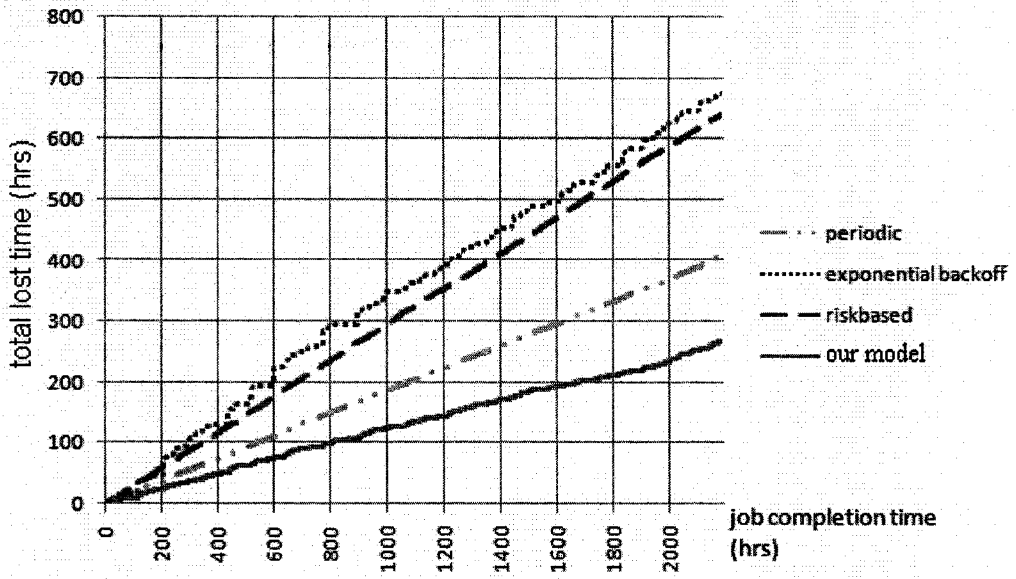


Figure 5.4 Total waste time against job completion time from 1 to 2,200 hours.

In our study, we defined the job completion time that includes overall execution time, checkpoint overhead, recovery and rollback times. We compared waste time improvement results from our model against there existing methods [44]. We then ran a simulation with the LLNL failure information by varying job completion times, when the checkpoint overhead was 0.1667 hour (i.e. around 10 minutes), and k was 0.4614. Moreover, the cooperative checkpoint request interval [44] in the compared methods was at every half an hour. In the experiments, we varied the job completion time from 1 hour to 2200 hours (i.e. around 3 months). Results in Figure 5.4 clearly show that our model produces the least waste time or lost time. Our technique provides significant improvement over the existing techniques, especially when the application total completion time goes larger. This is due to the fact that the longer job completion time, the higher failure probability in the system. In addition, our improvements come from optimality interplay among reliability-awareness and a balance between checkpoint overhead and rollback time factors.

CHAPTER 6

INCREMENTAL CHECKPOINT/RESTART MODEL

6.1 Introduction

Generally, checkpoint/restart on a large-scale distributed system is much more challenging than checkpoint/restart on a single system. This difference is caused by the multiplicity nature and coordination of state saving and recovery of applications are quite complex. Furthermore, the larger system is the more impact by potential failures. For example, the analysis of ASC White error log shows that the MTBF of a single node may be several thousands hours, but for the whole system with 512 nodes, the MTBF can be reduced to only 20 hours. This phenomenon suggests that applications running on a large-scale system need more checkpoint placements to reduce lost computational time. Also, for checkpointing on a large-scale system, huge memory contexts must potentially be transferred through the network and saved on reliable storage, so that the checkpoint overhead becomes a critical issue which directly impacts the application execution time and disk storage requirement. Therefore, in recent years, research in reducing checkpoint overhead has gained significant attentions in the high performance computing community [47][48][49] [50][52] [53] [44].

One of the recent checkpoint overhead reduction techniques is cooperative checkpointing [44]. The main concept of the cooperative checkpointing schedules the basic checkpoint placements following the traditional fixed interval checkpoint model (Young's model). However, in order to reduce the checkpoint cost, the technique skips some scheduled checkpoints according to the estimation of system failure risk. The performance of cooperative checkpointing depends on the accuracy of risk estimation. Nevertheless, an accurate failure prediction or risk estimation is a challenging problem [54][55].

On the other hand, incremental checkpoint schemes [47][48][49][52][53] focus on reducing the checkpoint overhead by saving only necessary application states or only modified states. In contrast to a traditional checkpoint (i.e. full checkpoint) which copies all the information (all memory pages related to the running application) to a stable storage, the incremental checkpoint takes a full checkpoint the first time, and then typically exploits an OS's page protection mechanism to observe, at checkpoint time, which pages have changed since the last checkpoint, and it only saves those pages. One of the disadvantages of the incremental checkpoint is that it requires the system first to restore the states as captured in the previous full checkpoint, and then to apply all the incremental checkpoints in order, before the recovery can be completed. Thus, the restart mechanism of an incremental checkpoint is more complex than the full checkpoint.

Some efforts [47][48][52][53] have focused on incremental checkpointing algorithm, e.g. how to efficiently implement incremental checkpointing in system level or user level. However, not much attention has been given to an optimal incremental checkpoint model. Based on our previous reliability-aware full checkpoint/restart model in Chapter Four and Five, we extend our study to include incremental checkpoint placements in a large-scale distributed system. Because the checkpoint overhead and restart mechanism of incremental checkpointing are different from full checkpointing, we describe the incremental checkpoint model, derive our solution, and compare it to the full checkpoint counterpart in the next sections.

6.2 Incremental Checkpoint/Restart Model

6.2.1 Behaviors of Incremental Checkpointing Model

The behaviors of incremental checkpoint/restart model are illustrated in Figure 6.1, and this model is similar to the full checkpoint/restart model as shown in Chapter Four Figure 4.3. The difference between these two models is that the incremental checkpoint model has two types of checkpoints (full checkpoint and incremental checkpoints), whereas the full checkpoint model only has a full checkpoint. The meaning of each parameter in the incremental checkpoint/restart model is listed in Table 6.1.

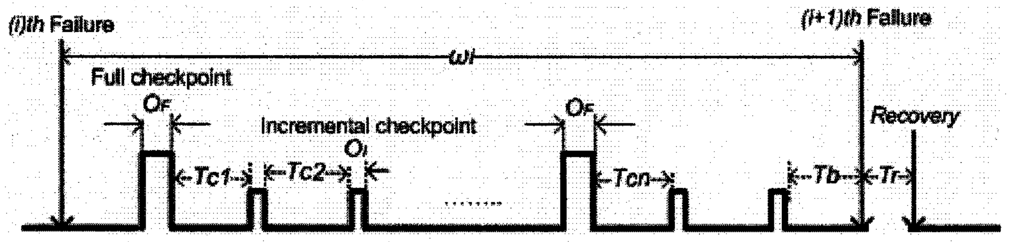


Figure 6.1 Incremental checkpoint/restart model.

Table 6.1 Parameters Meaning in Incremental Checkpoint/Restart Model.

Parameters	Meaning
$T_{c1},\ldots T_{cn}$	Checkpoint Intervals (they may be different).
O_F	Full Checkpoint Overhead.
O_I	Incremental Checkpoint Overhead.
T_b	Time lost between failure and last checkpoint.
T_r	Recovery time from an incremental checkpoint.
T_{rf}	Recovery time from a full checkpoint

σ	Additional recovery cost per incremental checkpoint.
m	Number of incremental checkpoint between two full checkpoints.
μ	Incremental checkpoint overhead ratio $\mu = O_I / O_F$.

In our incremental checkpoint model, the first checkpoint is a full checkpoint, which saves the entire data section and the stack of the application. The full checkpoint is followed by a sequence of incremental checkpoints, which only saves the address spaces that have changed since the previous checkpoint. The recovery cost is decided by the number of incremental checkpoints. After m incremental checkpoints, a full checkpoint may be taken again if a breakeven point indicates that the full checkpoint cost may be cheaper than the recovery cost if a failure occurs. The main idea is to balance a cost saving function with full and incremental checkpoint overheads and the complexity of the recovery that is introduced by the incremental model.

6.2.2 Assumptions

For the clear and convenient discussion, we list the following assumptions in our model:

1) About failures

An application can be interrupted by a series of unexpected failures Y, and Y could be a Poisson process following a frequency function $n(t)$.

2) Checkpoint interval.

The checkpoint interval times $T_{c1}, T_{c2}, \dots\dots, T_{cn}$ may be different.

3) Checkpoint overhead.

Let the full checkpoint overhead be O_F.

The incremental checkpoint overhead O_I, is a constant proportion to the full checkpoint overhead, $O_I = \mu \cdot O_F$, where μ is the incremental checkpoint overhead ratio, $0 < \mu < 1$.

4) The recovery time.

Let T_r be the recovery time from a incremental checkpoint, and T_{rf} be the recovery time from a full checkpoint. They are including failure detection time, component repair or replace time, and restart time.

Each incremental checkpoint puts additional cost on the restart phase. If the application is recovered from a series of m incremental checkpoints between the failure and the previous full checkpoint, then the additional restart time is $\sum_{i=1}^{m} \delta_i$, where δ_i is the recovery cost of i^{th} incremental checkpoint. We assumed that the cost of each incremental checkpoint is the same, i.e. $\delta_1 = \delta_2 = \cdots\cdots = \delta_m = \delta$, then the total additional restart cost generated by incremental checkpoints is $m\delta$, i.e. $T_r = T_{rf} + m \cdot \delta$.

5) The conditions for full checkpoints.

The first checkpoint in an application is a full checkpoint.

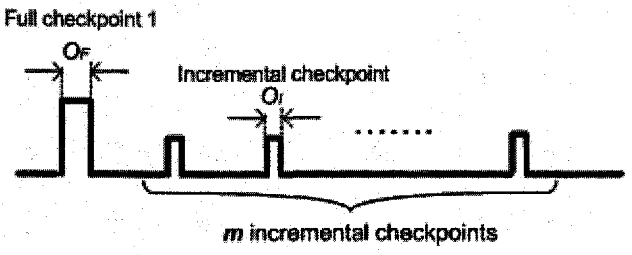
After an application is recovered from failure, the first checkpoint is a full checkpoint.

After m consecutive incremental checkpoints, a full checkpoint may be performed if the overall cost reaches a breakeven point for a choice between incremental vs. full checkpoint. We will determine the value of m in the next section.

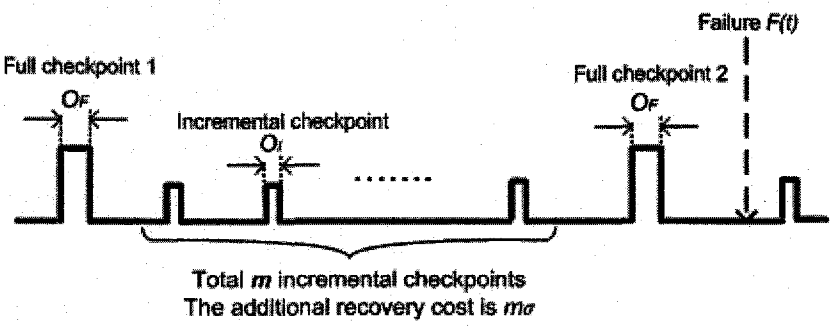
6.2.3 Sequential Incremental Checkpoint Number *m*

We denote the number of incremental checkpoints in the sequence as *m* between two full checkpoints. The value of *m* is decided by the breakeven cost consideration between a choice of the next checkpoint type, either incremental or full after m continuous incremental checkpoint. As discussed earlier, the incremental checkpoint aims to reduce the checkpoint overhead; on the other hand, the recovery cost will increase as the number of subsequent incremental checkpoints (*m*) increase, when the failure occurs. This increase is due to the application state reconstruction phase that requires information from each and every incremental checkpoint from the previous full checkpoint.

1)



2a)



2b)

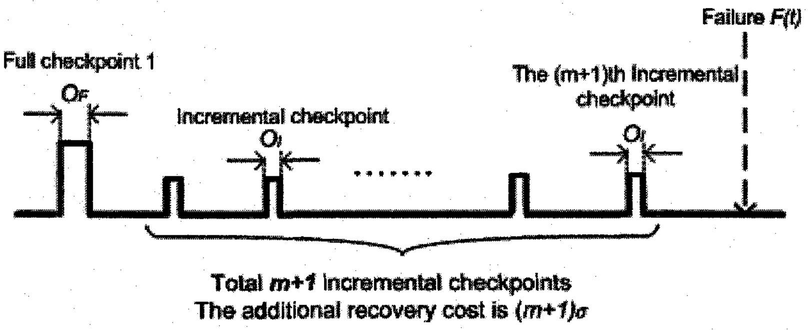

Failure *F(t)*
Full checkpoint 1
O_F
Incremental checkpoint
O_I
....
The (m+1)th Incremental checkpoint
O_I
Total *m+1* incremental checkpoints
The additional recovery cost is *(m+1)σ*

Figure 6.2 Sequential incremental checkpoint sernerio

From the model description of the incremental checkpoint in Section 6.2.1 , we assume that the first checkpoint is a full checkpoint. Thereafter, it is followed by a sequence of incremental checkpoints. Let us assume that the number of sequential incremental checkpoint placements is *m*. The key finding of our incremental checkpoint model is how to derive *m,* so that the overall cost, including recovery and rollback time, remains minimal when a failure occurs.

Therefore, our purpose of this checkpointing model study is to find an incremental checkpoint placement solution which will minimize the total lost time. We follow this rule to find *m* by comparing the lost time in two possible cases. In the first case, as shown in Figure 6.2 (2a), *m* continuous incremental checkpoints are followed by a full checkpoint. Alternatively, as shown as in Figure 6.2 (2b), after placing m continuous incremental checkpoints, we continue to take the *(m+1)*th incremental checkpoint. In each case, we consider the probability of failure event. The details are discussed in the following section.

<u>Case (a), when the next checkpoint *(m+1)*th is a full checkpoint.</u>

After placing *m* continuous incremental checkpoint, a full checkpoint is performed next as shown in Figure 6.2. Suppose the probability that a failure will occur after the second full checkpoint and before the next incremental checkpoint is P_I. Hence the probability that failure will not happen in that period is $1-P_I$.

If no failure occurs during this period, the overall cost C_{a1} is

$$C_{a1} = O_F + mO_I + O_F .$$

Alternatively, if the failure occurs, the cost C_{a2} is

$$C_{a2} = O_F + mO_I + O_F + T_{rf}.$$

Therefore the total cost is

$$\begin{aligned} C_a &= (1-P_I)C_{a1} + P_I C_{a2} \\ &= (1-P_I)(2O_F + mO_I) + P_I(2O_F + mO_I + T_{rf}). \end{aligned} \tag{6.1}$$

Case (b), when the next checkpoint *(m+1)*th is another incremental checkpoint.

After reaching m consecutive incremental checkpoints, another incremental checkpoint is performed. We consider that the probability of the failure events is approximately the same as in case (a).

When no failure occurs, the cost C_{b1} is

$$C_{b1} = O_F + (m+1)O_I \ .$$

If the failure will happen, the cost C_{b2} is

$$C_{b2} = O_F + (m+1)O_I + T_{rf} + (m+1)\delta.$$

Therefore, the total cost in case (b) is

$$\begin{aligned} C_b &= (1-P_I)C_{b1} + P_I C_{b2} \\ &= (1-P_I)[O_F + (m+1)O_I] + P_I[O_F + (m+1)O_I + T_{rf} + (m+1)\delta]. \end{aligned} \tag{6.2}$$

In order to minimize the time lost in the model, the solution of m is satisfied by the following condition

If $C_b \geq C_a$, it means the cost of case (b) is larger than the cost of case (a); thus, we will choose case (a) and perform a full checkpoint after m sequential incremental checkpoints.

Therefore, we obtain

$$(1-P_I)[O_F+(m+1)O_I]+P_I[O_F+(m+1)O_I+T_{rf}+(m+1)\delta] \geq(1-P_I)(2O_F+mO_I)+P_I(2O_F+mO_I+T_{rf}). \tag{6.3}$$

By simplifying the above inequality, we obtain the following:

$$m \geq \frac{O_F-O_I}{P_I\cdot\delta}-1. \tag{6.4}$$

Inequality (6.4) tells us that if $m \geq \frac{O_F-O_I}{P_I\cdot\delta}-1$, the cost in case (b) will be greater than the cost in case (a). Thus, we take m as in (6.5):

$$m=\left\lfloor \frac{O_F-O_I}{P_I\cdot\delta}-1 \right\rfloor, \tag{6.5}$$

where $\lfloor x \rfloor$ is the floor function, also called the greatest integer function. Because $O_I=\mu O_F$ (Assumption 3) in Section 6.2.2, we substitute O_I in Equation (6.5) and obtain

$$m=\left\lfloor \frac{(1-\mu)O_F}{P_I\cdot\delta}-1 \right\rfloor. \tag{6.6}$$

6.2.4 Mathematical Solution of the Model

In Chapter Four, we consider that the checkpointing procedure is a renewal process. Therefore, whenever the failure occurs, the new cycle starts. We follow the same renewal reward theory as in Chapter Four to derive the optimal incremental checkpoint/restart model.

Let the sequence of discrete checkpoint placements be $0=t_0<t_1<\ldots<t_n$, and $n(t)$ be the checkpoint frequency function. Then

$$\int_{t_i}^{t_{i+1}} n(\tau)d\tau=1, \qquad i=0,1,2\ldots\,. \tag{6.7}$$

In Figure 6.1, the total number of checkpoints in cycle ω_i is

$$N(\omega_1) = \int_0^{\omega_1} n(\tau)d\tau = n_F + n_I \ , \tag{6.8}$$

where n_F is the number of full checkpoints in cycle ω_1, and n_I is the number of incremental checkpoints in the first cycle ω_1, and $n_I = m \cdot n_F$.

If W_1 is the time lost due to the checkpointing in cycle ω_1, then

$$\begin{aligned} W_1 &= \frac{1}{m+1} \cdot O_F \int_0^{\omega_1} n(\tau)d\tau + \frac{m}{m+1} \cdot \mu O_F \int_0^{\omega_1} n(\tau)d\tau + T_b + T_r \\ &= \frac{1+\mu m}{m+1} \cdot O_F \int_0^{\omega_1} n(\tau)d\tau + T_b + T_r. \end{aligned} \tag{6.9}$$

We suppose that the system can be successfully recovered from the last checkpoint, and the rollback cost T_b is the same as what we have discussed in Chapter Four and Five, i.e. $T_b \approx k/n(\omega_1)$, $(0 < k < 1)$, where $n(\omega_1)$ is the checkpoint frequency at time ω_1, and k can be evaluated by the same method as in Section 5.2. In assumption 4, the recovery time is $T_r = T_{rf} + m \cdot \delta$, where T_{rf} is the recovery time from a full checkpoint. Therefore, we substitute T_r in Equation (6.9) and obtain:

$$W_1 = \frac{1+\mu m}{m+1} \cdot O_F \int_0^{\omega_1} n(\tau)d\tau + \frac{k}{n(\omega_1)} + T_{rf} + m\delta \, . \tag{6.10}$$

By following the same stochastic renewal reward process theory as described in Section 4.4 , the overall checkpoint/restart process can be described as $A(t) = \sum_1^{N(t)} W_i$.

From the basic limit theorem of renewal reward processes, we obtain

$$\lim_{t\to\infty} \frac{E(\sum_1^{N(t)} W_i)}{t} = \frac{E(W_1)}{E(\omega)} \, .$$

The left hand side of the above equation represents the total average reward (in this case, the waste or lost time), and it is a function of the average reward in the first cycle, $E(W_1)$. In the checkpoint/restart model, this theorem denotes that minimizing the overall time lost is equivalent to minimizing lost time in cycle ω_1. Let $f(t)$ be the probability density function of the system failure, so that the probability for the system failure within $[t, t+\Delta t]$ is $f(t)\cdot\Delta t$. The expected time lost during a cycle in the checkpoint process $E(W_1)$ is

$$E(W_1) = \int_0^{\infty}\left[\int_0^{t}\frac{1+\mu m}{m+1}\cdot O_F\cdot n(\tau)d\tau + \frac{k}{n(t)} + T_{rf} + m\delta\right]\cdot f(t)dt. \tag{6.11}$$

We are now looking for the solution of overall checkpoint frequency $n^*(t)$ to minimize Equation (6.11).

Let $x(t) = \int_0^t n(\tau)d\tau$. From Equation (6.11) we obtain:

$$E(W_1) = \int_0^{\infty}[\frac{1+\mu m}{m+1}\cdot O_F\cdot x(t) + \frac{k}{x'(t)} + T_{rf} + m\delta]\cdot f(t)dt. \tag{6.12}$$

Let the function in the integral in Equation (6.12) be $\Phi(x, x', t)$. Then

$$\Phi(x, x', t) = [\frac{1+\mu m}{m+1}\cdot O_F\cdot x(t) + \frac{k}{x'(t)} + T_{rf} + m\delta]\cdot f(t). \tag{6.13}$$

Based on the theorem of calculus of variations, if the integral in Equation (6.12) has a minimum value, $\Phi(x, x', t)$ in Equation (6.12) must satisfy Euler's equation in (6.14) [83],

$$\frac{\partial\Phi}{\partial x} - \frac{d}{dt}\cdot\frac{\partial\Phi}{\partial x'} = 0. \tag{6.14}$$

In Equation (6.13), we take the partial derivative of Φ

$$\begin{cases} \dfrac{\partial \Phi}{\partial x} = \dfrac{1+\mu m}{m+1} \cdot O_F \cdot f(t) \\ \\ \dfrac{\partial \Phi}{\partial x'} = \dfrac{-k}{(x'(t))^2} \cdot f(t). \end{cases} \tag{6.15}$$

By substituting (6.15) into (6.14), we obtain

$$\frac{1+\mu m}{m+1} \cdot O_F \cdot f(t) + \frac{d}{dt} \cdot \frac{k}{(x'(t))^2} \cdot f(t) = 0. \tag{6.16}$$

By integrating both sides of (6.16), we obtain

$\dfrac{1+\mu m}{m+1} \cdot O_F \cdot F(t) + \dfrac{k}{(x'(t))^2} \cdot f(t) = C$, where C is a constant.

Because $x'(t) = n(t)$, we have

$$\frac{1+\mu m}{m+1} \cdot O_F \cdot F(t) + \frac{k}{(n(t))^2} \cdot f(t) = C. \tag{6.17}$$

We know that $\lim\limits_{t \to \infty} F(t) = 1$, $\lim\limits_{t \to \infty} f(t) = 0$, and $C = \dfrac{1+\mu m}{m+1} \cdot O_F$; therefore, we obtain the solution for the incremental checkpoint frequency in Equation (6.18)

$$n^*(t) = \sqrt{\frac{k \cdot f(t)}{\dfrac{1+\mu m}{m+1} \cdot O_F (1 - F(t))}} = \sqrt{\frac{(m+1)k}{(1+\mu m)O_F}} \cdot \sqrt{\frac{f(t)}{R(t)}}. \tag{6.18}$$

6.3 Model Analysis and Validation

In Section 6.2 , we obtained the general solution, Equation (6.18), for our incremental checkpointing model. Equation (6.18) is a checkpoint frequency function which is derived from a probability distribution function of system time between failures (TBF). In previous chapters, the failure distribution could be obtained from the system failure analysis.

For the purpose of the incremental checkpoint/restart study and evaluation, we validate our model results only when the system failure follows the exponential distribution. This assumption will help simplify our validation and clearly demonstrate the deference between the full checkpoint solution vs. the incremental counterpart. In order to manifest the problems of incremental checkpointing model itself, we reduce the complexity brought by other distributions, and only perform the model analysis with the exponential distribution. However, we plan to use these results as a guidance to further our study with other distributions in future works.

6.3.1 Exponential Distribution

For the time between failures (TBF) that follows the exponential distribution, we substitute $f(t) = \lambda e^{-\lambda t}$, and $R(t) = e^{-\lambda t}$, $t \geq 0$, $\lambda > 0$ in Equation (6.18), and then obtain

$$n^*(t) = \sqrt{\frac{(m+1)k}{(1+\mu m)O_F}} \cdot \sqrt{\lambda}\,. \tag{6.19}$$

The optimal checkpoint interval is a fixed value

$$I = \frac{1}{n^*(t)} = \sqrt{\frac{(1+\mu m)O_F}{k(m+1)}} \cdot \sqrt{\frac{1}{\lambda}}\,. \tag{6.20}$$

In Equation (6.20), the full checkpoint overhead O_F and incremental checkpoint overhead coefficient μ are given as constants. The exponential failure rate fitted from the failure data set is λ. The number of incremental checkpoints m can be obtained by Equation (6.6) in Section 6.2.3 .

$$m = \left\lfloor \frac{(1-\mu)O_F}{P_I \cdot \delta} - 1 \right\rfloor.$$

In the above formula, we can obtain δ, the recovery cost per incremental checkpoint, from an experiment. Therefore, the failure probability P_I can be analytically derived as follows.

We suppose that the checkpoint placements are at $t_0, t_1, t_2, \ldots\ldots t_n, t_{n+1}$, where $t_0 = 0$, when the application starts, and t_n is a time stamp of the n^{th} checkpoint. In the exponential distribution case, the checkpoint interval does not change over time as described in Equation (6.20),

$$t_0 = 0, t_1 = I, t_2 = 2I, \ldots\ldots, t_n = nI.$$

From Equation (6.6), the probability P_I of failure during (t_n, t_{n+1}) interval is

$$\begin{aligned} P_I &= P[T < t_{n+1} \mid T > t_n] \\ &= \frac{P[t_n < T < t_{n+1}]}{P[T > t_n]} \\ &= \frac{F(t_{n+1}) - F(t_n)}{1 - F(t_n)}. \end{aligned} \tag{6.21}$$

For the exponential distribution, the CDF is $F(t) = 1 - e^{-\lambda t}$,

$$\begin{aligned} P_I &= \frac{(1 - e^{-\lambda t_{n+1}}) - (1 - e^{-\lambda t_n})}{1 - (1 - e^{-\lambda t_n})} \\ &= \frac{e^{-\lambda t_n}(1 - e^{-\lambda t_1})}{e^{-\lambda t_n}} \\ &= 1 - e^{-\lambda t_1}. \end{aligned} \tag{6.22}$$

From Equations (6.19) to (6.22), we can find m based on the following algorithm. Figure 6.3 illustrates the flow chart of the algorithm.

Algorithm description:

Step 1. To initialize O_F, μ, δ, and λ. Let $m = 1$.

Step 2. To calculate checkpoint interval I by Equation (6.20).

Step 3. To calculate P_I by Equation (6.22).

Step 4. To compare m with $\frac{(1-\mu)O_F}{P_I \cdot \delta} - 1$

If $m < \frac{(1-\mu)O_F}{P_I \cdot \delta} - 1$, then $m = m+1$, and go to Step 2

Else go to Step 5.

Step 5. To output $m = m-1$.

Step 6. End.

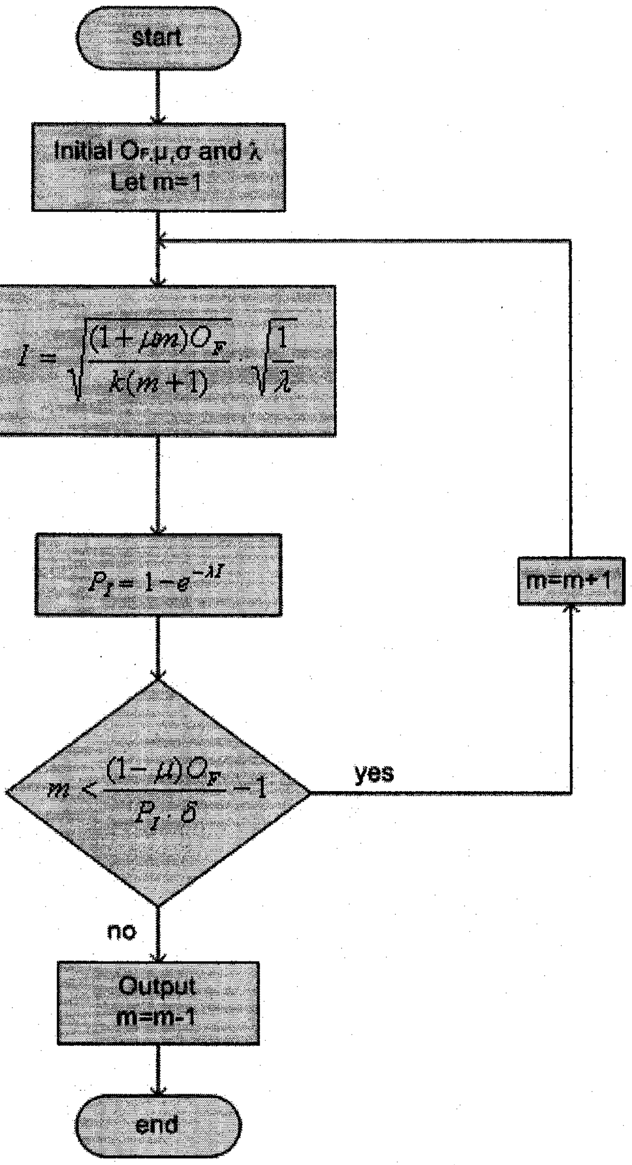

start
Initial O_F, μ, σ and λ
Let m=1
$I = \sqrt{\frac{(1+\mu m)O_F}{k(m+1)}} \cdot \sqrt{\frac{1}{\lambda}}$
$P_I = 1-e^{-\lambda I}$
$m < \frac{(1-\mu)O_F}{P_I \cdot \delta} - 1$
yes
m=m+1
no
Output
m=m-1
end

Figure 6.3 Algorithm to find incremental checkpoint number m

6.3.2 Validation Results

In this section, the incremental checkpoint model is studied and compared with the full checkpoint technique. For the comparison, we assume the full checkpoint overhead O_F=0.1667 hour, the incremental checkpoint overhead radio μ=0.1, and the rollback coefficient k is given by same method as in Section 5.1 . In exponential distribution case, k equal to 0.5. Our evaluation also employs the failure information from which TBF data set is from ASC White system, and its corresponding exponential distribution with the failure rate, λ, is 0.051876. We compared the total lost time between the full checkpoint model and incremental checkpoint model. We also varied the job completion time from 1 hour to 2200 hours, and we have taken $\delta = 30$ second. The result in Figure 6.4 shows that under the same failure scenario, the incremental model performs better than the full checkpoint model. We observe that the best case occurs when the total lost time less than the half of the full checkpoint is.

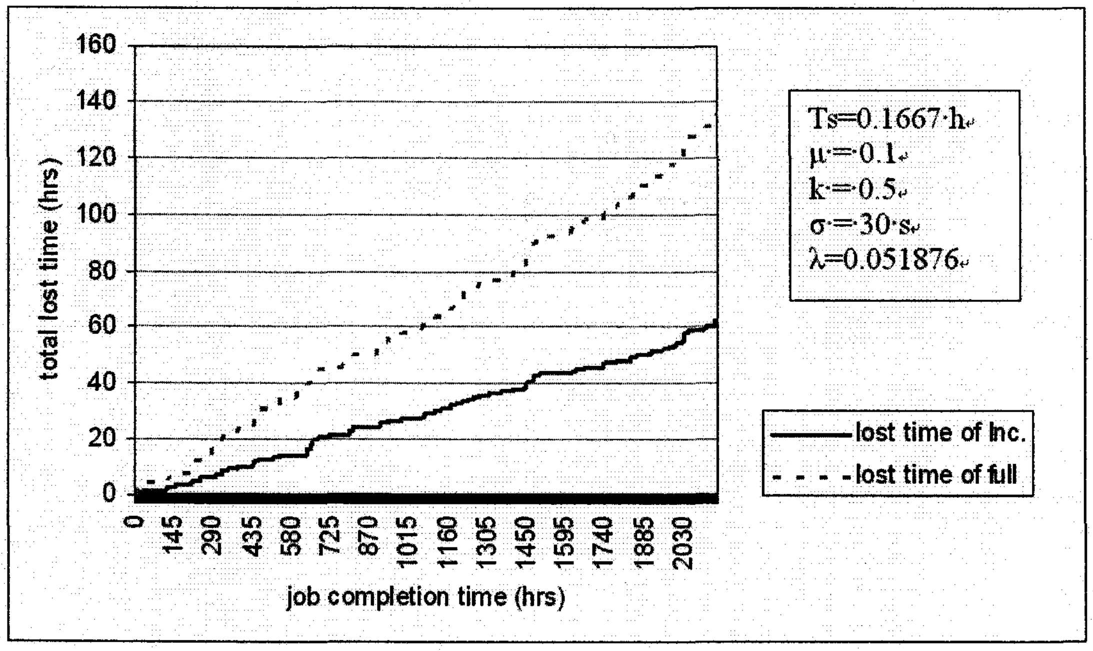


Figure 6.4 Lost time comparison between full checkpoint and incremental checkpoint

6.4 Summary

In this chapter, we have discussed the incremental checkpoint/restart model in the large-scale distributed system. It is an extension of the full checkpoint/restart model that is discussed in Chapter Four and Five. The incremental checkpointing aims to further improve overhead reduction, especially for a large-scale distributed system where the full checkpoint overhead may be significant. We have built the model for the incremental checkpoint/restart solution, and have derived the consecutive incremental checkpoint number *m* between two typical full checkpoints that yields the breakeven cost. The m value provides guidance where we can perform additional checkpoints without paying much penalty and gain better benefits when the failure occurs. We have also presented the comparison between the incremental checkpoint model and full checkpoint model. Our results suggest that the total time lost in the incremental checkpoint is significantly smaller than the time lost in the full checkpoint model.

CHAPTER 7

CONCLUSION AND FUTURE WORK

In this dissertation, we first introduce the problem domain and background. The reliability challenges in large-scale cluster, basic concepts of reliability theory, failure data analysis techniques and fault-tolerance schemes are described. In Chapter Three, we detailed failure data analysis techniques through the process of analyzing actual failure data from the Lawrence Livermore ASC White system. Three key techniques are described: reliability theory of k of n system, distribution fitting of TBF, and goodness-fit test. In Chapter Four, the optimal checkpoint/restart model, we applied the stochastic renewal reward process theory to analyze the behaviors of the checkpoint/restart procedure, and figured out the optimal checkpoint placement solution to meet the requirement of minimum total cost. In model evaluation part, Chapter Five, we present our model solution in exponential failure distribution and Weibull failure distribution, and provide the method to evaluate the rollback cost ratio *k*, since it is the significant part in the model. We also compare our model with three checkpoint schemes, namely periodic, exponential backoff, and risk-based, described in [44]. The comparison is based on a TBF dataset which comes from ASC White system. The distribution fitting and goodness-fit test results show that the dataset follows Weibull distribution. The comparison outcome demonstrated that our model has the smallest waste time amongst the current state-of-art methods.

One major contribution of this dissertation is that we present a reliability-aware optimal checkpoint/restart model in a large-scale distributed cluster environment. In the model, the optimal checkpoint placements make use of the system reliability which is derived from actual system failure data. Unlike most previous studies of optimal checkpoint/restart models, they assume that the system failure follows exponential distribution (fixed failure rate λ). Thus, the advantage of our model is that the checkpoint strategy depends on actual system failure analysis. The analysis is performed on the dataset which comes from system events log file. The analysis result shows that the failure may follow different distributions: exponential, Weilbull, or gamma distribution. Therefore, our model can provide more accurate checkpoint placements and reduce the cost in the actual application.

Another contribution is that we present an improved checkpoint solution with an incremental checkpoint/restart model in Chapter Six. Although, there are some efforts in the area of developing of incremental checkpoint mechanism, but until now, there is no effective incremental checkpoint system for the large-scale HPC environment. Therefore, our incremental checkpoint model study can be considered a pioneer work in this area. However, it is an early foundation that aims to provide a guidance to further performance improvement by balancing interplays among a mixed between full and incremental checkpoints, and corresponding recovery methods as well as rollback time. The key issue of our incremental checkpoint model lies into how to find the number of incremental checkpoints between two neighbored full checkpoints that provides a better cost model.

Because of the computation complexity of other distributions like gamma or lognormal, in this dissertation, we only present our model in two failure distributions which are popular in the reliability field: exponential and Weilbull. In future study, we will extend our findings in the optimal checkpoint solutions in gamma or lognormal failure distributions. Another future work should be how to further improve and obtain the optimal incremental checkpoint model.

APPENDIX A

STOCHASTICS RENEWAL PROCESSES

A class of stochastic processes is *renewal processes.* It is used to model independent identically distributed occurrences with arbitrary distribution.

1) Renewal Processes

Let $\{X_1, X_2, X_3, \ldots X_n\}$ be a sequence of nonnegative independent random variables with a common distribution F, and to avoid trivialities suppose that $F(0) = P\{X_n = 0\} < 1$. We shall interpret X_n as the time between the *n*th and *(n+1)*th event . Let

$$\mu = E[X_n] = \int_0^\infty x dF(x).$$

Denote the mean time between successive events and note that from the assumptions that $X_n \geq 0$ and $F(0) < 1$, it follows that $0 < \mu \leq \infty$. Letting

$$S_0 = 0 \quad ,$$

$$S_n = \sum_{i=1}^{n} X_i \quad n \geq 1,$$

it follows that S_n is the time of the *n*th event. As the number of events by time *t* will equal the largest value of *n* for which the *n*th event occurs before or at time *t*, we have that $N(t)$, the number of events by time *t*, is given by

$$N(t) = \max\{n : S_n \leq t\}.$$

The counting process $\{N(t), t \geq 0\}$ is called a renewal process

Since the interarrival times are independent and identically distributed, it follows that each renewal process probabilistically starts over.

2) Renewal Reward Processes.

Consider a renewal process $\{N(t), t \geq 0\}$ having interarrival times $X_n, n \geq 1$ with distribution *F*, and suppose that when each time a renewal occurs, we receive a reward. We denote by R_n the reward earned at the time of the *n*th renewal. We shall assume that the $R_n, n \geq 1$, are independent and identically distributed. However, we do allow for the possibility that R_n may depend on X_n, the length of the nth renewal interval, and so we assume that the pair $(X_n, R_n), n \geq 1$, are independent and identically distributed. If we let

$$R(t) = \sum_{n=1}^{N(t)} R_n .$$

then $R(t)$ represents the total reward earned by time *t*.

Theorem: let

$$E[R] = E[R_n], \qquad E[X] = E[X_n].$$

If $E[R] < \infty$ *and* $E[X] < \infty$, then

(i) $$\lim_{t \to \infty} \frac{R(t)}{t} = \frac{E[R]}{E[X]}$$

(ii) $$\lim_{t \to \infty} \frac{E[R(t)]}{t} = \frac{E[R]}{E[X]}$$

If we define that a cycle is completed every time a renewal occurs, then the theorem states that the expected long-run average return is just the expected return earned during a cycle, divided by the expected time of a cycle.

APPENDIX B

THE MATLAB CODE

Algorithm 1: The Matlab Code for Distribution Fitting

```
function TBFfit(M)
%TBFFIT   Create plot of datasets and fits
%  TBFFIT(M)
%  Designed by Yudan Liu

% Remove missing values
t_ = ~isnan(M);
if ~isempty(M), t_ = t_ & ~isnan(M); end
M = M(t_);
if ~isempty(M), M = M(t_); end

% Set up figure to receive datasets and fits
f_ = clf;
figure(f_);
set(f_,'Units','Pixels','Position',[33 147 680 468.45]);
legh_ = []; legt_ = {};   % handles and text for legend
ax_ = newplot;
set(ax_,'Box','on');
hold on;


% --- Plot data originally in dataset "M data"
M = M(:);
[Y_,X_,yL_,yU_] = ecdf(M,'Function','cdf','alpha',0.05...
          ,'freq',M...
          ); % compute empirical function
display(Y_);

h_ = stairs(X_,Y_);
set(h_,'Color',[0.333333 0 0.666667],'LineStyle','-', 'LineWidth',1);
[XX1_,YY1_] = stairs(X_,yL_);
[XX2_,YY2_] = stairs(X_,yU_);
hb_ = plot([XX1_(:); NaN; XX2_(:)], [YY1_(:); NaN; YY2_(:)],...
  'Color',[0.333333 0 0.666667],'LineStyle',':', 'LineWidth',1);
xlabel('Time Between Failure (TBF)');
ylabel('Cumulative Distribution Function (CDF)')
legh_(end+1) = h_;
legt_{end+1} = 'TBF data';
legh_(end+1) = hb_;
legt_{end+1} = '95% confidence bounds';

% Nudge axis limits beyond data limits
xlim_ = get(ax_,'XLim');
if all(isfinite(xlim_))
```

```
  xlim_ = xlim_ + [-1 1] * 0.01 * diff(xlim_);
  set(ax_,'XLim',xlim_)
end

x_ = linspace(xlim_(1),xlim_(2),11);

% --- Create fit "Exponential"


p_ = expfit(M, 0.05, [], M);
y_ = expcdf(x_,p_(1)); % compute cdf
nlogLe = explike(p_, M);
display(nlogLe);
display(y_(:));

h_ = plot(x_,y_,'Color',[1 0 0],...
      'LineStyle','-', 'LineWidth',2,...
      'Marker','none', 'MarkerSize',6);
legh_(end+1) = h_;
legt_{end+1} = 'Exponential';

% --- Create fit "Weibull"


p_ = wblfit(M, 0.05, [], M);;
y_ = wblcdf(x_,p_(1), p_(2)); % compute cdf
nlogLw = evlike(p_,M);
display(nlogLw);
display(y_(:));
h_ = plot(x_,y_,'Color',[0 0 1],...
      'LineStyle','-', 'LineWidth',2,...
      'Marker','none', 'MarkerSize',6);
legh_(end+1) = h_;
legt_{end+1} = 'Weibull';


% --- Create fit "Lognormal"


p_ = lognfit(M, 0.05, [], M);;
y_ = logncdf(x_,p_(1), p_(2)); % compute cdf
nlogLl = evlike(p_,M);
display(nlogLl);
display(y_(:));
h_ = plot(x_,y_,'Color',[0.666667 0.333333 0],...
      'LineStyle','-', 'LineWidth',2,...
```

```
        'Marker','none', 'MarkerSize',6);
legh_(end+1) = h_;
legt_{end+1} = 'Lognormal';

% --- Create fit "Gamma"


p_ = gamfit(M, 0.05, [], M);;
y_ = gamcdf(x_,p_(1), p_(2)); % compute cdf
nlogLg = evlike(p_,M);
display(nlogLg);
display(y_(:));
h_ = plot(x_,y_,'Color',[0.333333 0.333333 0.333333],...
        'LineStyle','-', 'LineWidth',2,...
        'Marker','none', 'MarkerSize',6);
legh_(end+1) = h_;
legt_{end+1} = 'Gamma';

 hold off;
 h_ = legend(ax_,legh_,legt_,'Location','NorthWest');
 set(h_,'Interpreter','none');

% --- Create "Ecdf"
[f,x] = ecdf(M);
display(f);
```

Algorithm 2: The Matlab Code for excess life time calculation

```
function evaluek2()

b1=1.5;
a1=50;

b2=1.5;
a2=100;

b3=1.5;
a3=200;

Ts=0.167;

k_hat1=0.6;
k_hat2=0.5;
k_hat3=0.9;

CT1(1)=0;
```

```
Es1(1)=0;
P1(1)=1;
K1(1)=0;
A1(1)=0;

CT2(1)=0;
Es2(1)=0;
P2(1)=1;
K2(1)=0;
A2(1)=0;

CT3(1)=0;
Es3(1)=0;
P3(1)=1;
K3(1)=0;
A3(1)=0;

sum1=0;
sum2=0;
m=1;

% calculate group1

  while (P1(m)>=0.00000001)

    CT1(m+1)= ((m*(b1+1)/2)*((Ts*a1^b1)/(k_hat1*b1))^0.5)^(2/(b1+1));
    t1=CT1(m);
    t2=CT1(m+1);
    F1 = @(s)s.*b1.*(t1+s).^(b1-1)./a1.^b1.*exp(-(t1+s).^b1./a1.^b1);
    Q1 = exp(t1^b1/a1^b1)*quad8(F1,0,(t2-t1));
    A1(m+1)=t2-t1;
    Q2 = 1-exp((t1^b1)/(a1^b1)-(t2^b1)/(a1^b1));
    Es1(m+1)=Q1/Q2;

    P1(m+1)=exp(-(t1^b1)/(a1^b1))-exp(-(t2^b1)/(a1^b1));
    K1(m+1)=Es1(m+1)/(t2-t1);
    sum1=sum1+P1(m+1)*K1(m+1);
    sum2=sum2+P1(m+1);
    k_bar1=sum1/sum2;
    m=m+1;
  end
m=1;
  % calculate group2
  while (P2(m)>=0.00000001)

    CT2(m+1)= ((m*(b2+1)/2)*((Ts*a2^b2)/(k_hat2*b2))^0.5)^(2/(b2+1));
    t1=CT2(m);
    t2=CT2(m+1);
    F1 = @(s)s.*b2.*(t1+s).^(b2-1)./a2.^b2.*exp(-(t1+s).^b2./a2.^b2);
    Q1 = exp(t1^b2/a2^b2)*quad8(F1,0,(t2-t1));
    A2(m+1)=t2-t1;
    Q2 = 1-exp((t1^b2)/(a2^b2)-(t2^b2)/(a2^b2));
    Es2(m+1)=Q1/Q2;

    P2(m+1)=exp(-(t1^b2)/(a2^b2))-exp(-(t2^b2)/(a2^b2));
    K2(m+1)=Es2(m+1)/(t2-t1);
    sum1=sum1+P2(m+1)*K2(m+1);
    sum2=sum2+P2(m+1);
    k_bar2=sum1/sum2;
    m=m+1;
  end
m=1;
  % calculate group3
  while (P3(m)>=0.00000001)

    CT3(m+1)= ((m*(b3+1)/2)*((Ts*a3^b3)/(k_hat3*b3))^0.5)^(2/(b3+1));
    t1=CT3(m);
    t2=CT3(m+1);
    F1 = @(s)s.*b3.*(t1+s).^(b3-1)./a3.^b3.*exp(-(t1+s).^b3./a3.^b3);
    Q1 = exp(t1^b3/a3^b3)*quad8(F1,0,(t2-t1));
    A3(m+1)=t2-t1;
    Q2 = 1-exp((t1^b3)/(a3^b3)-(t2^b3)/(a3^b3));
    Es3(m+1)=Q1/Q2;

    P3(m+1)=exp(-(t1^b3)/(a3^b3))-exp(-(t2^b3)/(a3^b3));
    K3(m+1)=Es3(m+1)/(t2-t1);
    sum1=sum1+P3(m+1)*K3(m+1);
    sum2=sum2+P3(m+1);
    k_bar3=sum1/sum2;
    m=m+1;
  end

  format long

y1=[CT1' A1' Es1' P1' K1'];
y2=[CT2' A2' Es2' P2' K2'];
y3=[CT3' A3' Es3' P3' K3'];

fid = fopen('out.txt','w');
fprintf(fid,' Chkpt time      Chkpt Interval     E(Si)         P(i)          K(i)\n');
fprintf(fid,'%12.6f,  %12.6f,  %12.6f,   %12.6f,  %12.6f\n',y1');
fprintf(fid,'\n\nThe expected k: %12.6f\n',k_bar1);
```

```
fclose(fid);
% display(y);
display(k_bar1);



%plot weibull pfd
%x=0:a/20:2*a;
x=0:20:600;
w1=wblpdf(x,a1,b1);
w2=wblpdf(x,a2,b2);
w3=wblpdf(x,a3,b3);
subplot(2,2,1);

plot(x,w1,'r',x,w2,'-.b',x,w3,'--g','LineWidth',2);

title('PDF of Weibull Distribution');
xlabel('Time (hours)');
ylabel('Probability Density');
legend('\beta=1.5 \eta=50','\beta=1.5 \eta=100','\beta=1.5 \eta=200');
grid on;

% %plot weibull cfd
c1=wblcdf(x,a1,b1);
c2=wblcdf(x,a2,b2);
c3=wblcdf(x,a3,b3);
subplot(2,2,2);

plot(x,c1,'r',x,c2,'-.b',x,c3,'--g','LineWidth',2);

title('CDF of Weibull Distribution');
xlabel('Time (hours)');
ylabel('Probability');
legend('\beta=1.5 \eta=50','\beta=1.5 \eta=100','\beta=1.5 \eta=200',4);
grid on;

% %plot Es(i)
subplot(2,2,3);
plot(CT1,Es1,'-.r',CT2,Es2,'--b',CT3,Es3,'g','LineWidth',2);

xlim([max(max(CT1(2),CT2(2)),CT3(2)) CT3(m)]);
title('E(Si)');
xlabel('Time (hours)');
ylabel('Average excess life time');
legend('\beta=0.5 \eta=150','\beta=1.5 \eta=150','\beta=3.0 \eta=150');
grid on;
```

```
% %plot K(i)
subplot(2,2,4);
plot(CT1,K1,'-.r',CT2,K2,'--b',CT3,K3,'g','LineWidth',2);
xlim([max(max(CT1(2),CT2(2)),CT3(2)) CT3(m)]);
title('The expected k-value');
xlabel('Time (hours)');
ylabel('k-value');
legend('\beta=0.5 \eta=150','\beta=1.5 \eta=150','\beta=3.0 \eta=150');
        grid on;
```

APPENDIX C

GOODNESS-FIT-TESTS

1) Chi-Squared Test

Chi-Squared test is defined for the hypothesis:

H0: The data follow a specified distribution.

Ha: The data do not follow the specified distribution.

For the chi-square test computation, the data are divided into k bins and the test statistic is defined as

$$\chi^2 = \sum_{i=1}^{k} (O_i - E_i)^2 / E_i$$

where O_i is the observed frequency for bin *i* and E_i is the expected frequency for bin i. The expected frequency is calculated by

$$E_i = N(F(Y_u) - F(Y_l))$$

where F is the cumulative distribution function for the distribution being tested, Y_u is the upper limit for class *i*, Y_l is the lower limit for class *i*, and *N* is the sample size.

This test is sensitive to the choice of bins. There is no optimal choice for the bin width. Most reasonable choices should produce similar, but not identical, results. This test is not valid for small samples, and if some of the counts are less than five, then some bins in the tails may need to be combined. The test statistic follows, approximately, a chi-square distribution with (*k-c*) degrees of freedom where *k* is the number of non-empty cells and *c* equal to the number of estimated parameters (including scale parameters and shape parameters) for the distribution plus one.

Therefore, the hypothesis that the data are from a population with a specified distribution is rejected if $\chi^2 > \chi^2_{(\alpha,k-c)}$, where $\chi^2_{(\alpha,k-c)}$ is the chi-square percent point function with k - c degrees of freedom and a significance level of α.

2) Kolmogorov-Smirnov (K-S) Test

The Kolmogorov-Smirnov (K-S) test is based on the empirical distribution function (ECDF). Given N ordered data points $Y_1, Y_2, \ldots, Y_N$, the ECDF is defined as

$$E_N = n(i) / N$$

where $n(i)$ is the number of points less than Y_i ,and the Y_i are ordered from smallest to largest value. This is a step function that increases by $1/N$ at the value of each ordered data point.

The Kolmogorov-Smirnov test is defined by:

H0: The data follow a specified distribution

Ha: The data do not follow the specified distribution

The Kolmogorov-Smirnov test statistic is defined as

$$D = \max_{1 \le i \le N}(F(Y_i) - \tfrac{i-1}{N}, \tfrac{i}{N} - F(Y_i))$$

where F is the theoretical cumulative distribution of the distribution being tested which must be a continuous distribution. The hypothesis regarding the distributional form is rejected if the test statistic, D, is greater than the critical value obtained from a table. There are several variations of these tables in the literature that use somewhat different scaling for the K-S test statistic and critical regions. These alternative formulations should be equivalent, but it is necessary to ensure that the test statistic is calculated in a way that is consistent with how the critical values are tabulated.

The Kolmogorov-Smirnov test is another null hypothesis test used to determine whether two probability distributions differ, or whether an underlying probability distribution differs from a hypothesized distribution. It computes the maximum distance between the empirical cumulative distribution of failure dataset and the CDF of the fitted distribution.

In the goodness-fit-test results, P-value of each fitting distribution is obtained to identify the best fit. P-value is the probability of obtaining a finding at least as "impressive" as that obtained, assuming the null hypothesis is true, so that the finding was the result of chance alone. The fact that p-values are based on this assumption is crucial to their correct interpretation. If the p-values are less than 0.05, then that value would indicate that the failure datasets do not come from the selected distribution with 95% confidence.

REFERENCES

[1] J. Gray. "Why do computers stop and what can be done about it". In Proc. of the 5th Symposium on Reliability in Distributed Software and Database Systems, 1986.

[2] T. Heath, R. P. Martin, and T. D. Nguyen. "Improving cluster availability using workstation validation". In Proc. of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, 2002.

[3] R. K. Iyer, D. J. Rossetti, and M. C. Hsueh. "Measurement and modeling of computer reliability as affected by system activity". ACM Transactions Computer System, 4(3), 1986.

[4] M. Kalyanakrishnam, Z. Kalbarczyk, and R. Iyer. "Failure data analysis of a LAN of windows NT based computers". In Proc. of the 18th IEEE Symposium on Reliable Distributed Systems, 1999.13

[5] T.-T. Y. Lin and D. P. Siewiorek. "Error log analysis: statistical modeling and heuristic trend analysis". IEEE Transactions on Reliability, 39, 1990.

[6] J. Meyer and L. Wei. "Analysis of workload influence on dependability". In Proc. International Symposium on Fault-tolerant computing, 1988.

[7] B. Murphy and T. Gent. "Measuring system and software reliability using an automated data collection process". Quality and Reliability Engineering International, 11(5), 1995.

[8] D. L. Oppenheimer, A. Ganapathi, and D. A. Patterson. "Why do internet services fail, and what can be done about it" In USENIX Symposium on Internet Technologies and Systems, 2003.

[9] R. K. Sahoo, R. K., A. Sivasubramaniam, M. S. Squillante, and Y. Zhang. "Failure data analysis of a large-scale heterogeneous server environment". In Proc. of the 2004 international Conference on Dependable Systems and Networks (DSN'04), 2004.

[10] D. Tang, R. K. Iyer, and S. S. Subramani. "Failure analysis and modeling of a VAX cluster system". In Proc. International Symposium on Fault-tolerant computing, 1990.

[11] N. H. Vaidya. "A case for two-level distributed recovery schemes". In Proc. of the 1995 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems, 1995.

[12] K. F. Wong and M. Franklin. "Checkpointing in distributed computing systems". J. Parallel Distributed Computing, 35(1), 1996.

[13] J. Xu, Z. Kalbarczyk, and R. K. Iyer. "Networked Windows NT system field failure data analysis". In Proc. of the 1999 Pacific Rim International Symposium on Dependable Computing, 1999.

[14] Y. Zhang, M. S. Squillante, A. Sivasubramaniam, and R. K. Sahoo. "Performance implications of failures in large-scale cluster scheduling". In Proc. 10th Workshop on Job Scheduling Strategies for Parallel Processing, 2004.

[15] K. Sahoo, R. K., A. Sivasubramaniam, M. S. Squillante, and Y. Zhang. "Failure data analysis of a large-scale heterogeneous server environment". In Proc. of DSN'04, 2004.

[16] R. K. Sahoo, A. J. Oliner , I. Rish, M. Gupta, J.E. Moreira, S. Ma. " Critical event prediction for proactive management in large-scale computer clusters " Conference on Knowledge Discovery in Data Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining Pages: 426 - 435 ,Year of Publication: 2003 ISBN:1-58113-737-0

[17] Gary M. Weiss, Haym Hirsh. "Learning to predict rare events in categorical time-series data". Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98)

[18] T.-T. Y. Lin and D. P. Siewiorek. "Error log analysis: statistical modeling and heuristic trend analysis". IEEE Transactions on Reliability, 39, 1990.

[19] Y. Liang, Y. Zhang, M. Jette, A. Sivasubramaniam, and R. K. Sahoo. "Blue gene/l failure analysis and prediction models". In Proc. of the International Conference on Dependable Systems and Networks (DSN), 2006.

[20] Bianca Schroeder, Garth Gibson "A large-scale study of failures in high-performance-computing systems". May 2006. 3rd Symposium on Networked System Design and Implementation (NSDI 2006)

[21] A. J. Oliner "Performance implications of periodic checkpointing on large-scale cluster systems", Parallel and Distributed Processing Symposium, 2005. Proc. 19th IEEE International (2005), pp. 299b-299b.

[22] Armando Fox, Emre Kiciman, David Patterson, Michael Jordan, and Randy Katz. "Combining statistical monitoring and predictable recovery for self-management" 2004 Workshop on Self-Managed Systems (WOSS'04).

[23] Dohi, T.; Kaio, N.; Trivedi, K.S. "Availability models with age-dependent checkpointing". Reliable Distributed Systems, 2002. In Proc. Of 21st IEEE Symposium on 13-16 Oct. 2002 Page(s):130-139.

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

[25] Sancho, J.C.; Petrini, F.; Davis, K.; Gioiosa, R.; Song Jiang. "Current practice and a direction forward in checkpoint/restart implementations for fault tolerance". Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International 04-08 April 2005 Page(s):300b - 300b

[26] Coit, D.W. "System reliability prediction prioritization strategy". Reliability and Maintainability Symposium, 2000. Proc. Annual 24-27 Jan. 2000 Page(s):175 - 180

[27] Collas, G. "Dynamic reliability prediction: how to adjust modeling and reliability growth". Reliability and Maintainability Symposium, 1991. Proceedings, Annual 29-31 Jan. 1991 Page(s):301 - 306

[28] Duda, "The effects of checkpointing on program execution time". Information Processing Letters, vol. 16, no. 5, pp. 221-229, June 1983.

[29] V.F. Nicola, "Checkpointing and the modeling of program execution time". Software Fault Tolerance, M.R. Lyu, ed., pp. 167-188, John Wiley & Sons, 1995.

[30] K. G. Shin, T.-H. Lin, Y.-H. Lee, "Optimal checkpointing of real-time tasks". IEEE Transactions on Computers, v.36 n.11, p.1328-1341, Nov. 1987

[31] Elmootazbellah N. Elnozahy, James S. Plank. "Checkpointing for peta-scale systems: a look into the future of practical rollback-recovery". IEEE Transactions on Dependable and Secure Computing, vol.01,no.2, pp. 97-108,April-June,2004.

[32] James S. Plank and Michael G. Thomason, "The average availability of parallel checkpointing systems and its importance in selecting runtime parameters". 29th International Symposium on Fault-Tolerant Computing, Madison, WI, June, 1999, pp. 250-259.

[33] Michael Treaster, "A survey of fault-tolerance and fault-recovery techniques in parallel systems". ACM Computing Research Repository (CoRR), Technical Report cs.DC/0501002, January 2005.

[34] Elnozahy, E. N., Alvisi, L., Wang, Y., and Johnson, D. B. 2002. "A survey of rollback-recovery protocols in message-passing systems". ACM Computing Survival. 34, 3 (Sep. 2002), 375-408.

[35] Plank, J. S., Beck, M., Kingsley, G., and Li, K. 1995b. Libckpt: "Transparent checkpointing under UNIX". In Proceedings of the USENIX Winter 1995 Technical Conference, 213--223.

[36] Plank, J. S., Xu, J., and Netzer, R. H. 1995a. "Compressed differences: an algorithm for fast incremental checkpointing". Technical Report CS-95-302, University of Tennessee at Knoxville.

[37] Gioiosa, R.; Sancho, J.C.; Jiang, S.; Petrini, F., "Transparent, incremental checkpointing at kernel level: a foundation for fault tolerance for parallel computers". Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference, vol., no.pp. 9- 9, 12-18 Nov. 2005

[38] Sancho, J.C.; Petrini, F.; Johnson, G.; Frachtenberg, E., "On the feasibility of incremental checkpointing for scientific computing," Parallel and Distributed Processing Symposium, 2004. Proc. 18th International, vol., no.pp. 58-, 26-30 April 2004

[39] Junyoung Heo , Yookun Cho , Gwangil Jeon , Haklin Kimm, "The overhead model of word-level and page-level incremental checkpointing". Proc. of the 2006 ACM symposium on Applied computing, April 23-27, 2006, Dijon, France

[40] Saurabh Agarwal, Rahul Garg, Meeta S. Gupta, Jose Moreira. "Adaptive incremental checkpointing on massively parallel systems". In Proc. of 18th Annual ACM International Conference of Supercomputing (ICS'04), June 26 – July 1, 2004, p. 277-286.

[41] Palaniswamy, A. C. and Wilsey, P. A. 1993. "An analytical comparison of periodic checkpointing and incremental state saving". In Proc. of the Seventh Workshop on Parallel and Distributed Simulation (San Diego, California, United States, May 16 - 19, 1993). R. Bagrodia and D. Jefferson, Eds. PADS '93. ACM Press, New York, NY, 127-134.

[42] Vaidya, N. H. 1997. "Impact of checkpoint latency on overhead ratio of a checkpointing scheme". IEEE Transactions on Computer 46, 8 (Aug. 1997), 942-947.

[43] Sheldon M. Ross. "Stochastic Processes" Wiley; 2nd edition (January 1995), ISBN-10: 0471120626

[44] A. J. Oliner, L. Rudolph, R. K. Sahoo, "Cooperative checkpointing: a robust approach to large-scale systems reliability". In Proc. of the 20th Annual International Conference on Supercomputing (ICS), Cairns, Australia, June 2006.

[45] Anand Tikotekar, Chokchai Leangsuksun Stephen L. Scott. "On the survivability of standard MPI applications". In Proc. of 7th LCI International Conference on Linux Clusters: The HPC Revolution 2006

[46] S. Sankaran, J. M. Squyres, B. Barrett, A. Lumsdaine, J. Duell, P. Hargrove, and E. Roman. "The LAM/MPI checkpoint/restart framework: system-initiated checkpoint". The 2003 Los Alamos Computer Science Institute Symposium, Santa Fe, NM. October 2003.

[47] J. S. Plank, J. Xu, and R. H. Netzer, "Compressed differences: an algorithm for fast incremental checkpointing". Technical Report CS-95-302, University of Tennessee at Knoxville, Aug. 1995.

[48] K. Li, J. F. Naughton, and J. S. Plank, "Low-latency, concurrent checkpointing for parallel programs". IEEE Transactions on Parallel and Distributed Systems, vol. 5, Aug. 1994.

[49] N. H. Vaidya, "Impact of checkpoint latency on overhead ratio of a checkpointing scheme". IEEE Transactions on Computers, vol. 46, Aug. 1997.

[50] J. S. Plank, M. Beck, G. Kingsley, and K. Li, "Libckpt: transparent checkpointing under UNIX". In Usenix Winter Technical Conference, pp. 213–223, Jan. 1995.

[51] Saurabh Agarwal, Rahul Garg, Meeta S.Gupta and Jose E.Moreira "Adaptive incremental checkpointing for massively parallel systems", Proc. of the 18th annual international conference on Supercomputing, Malo, France SESSION: Middleware for high performance computing Pages: 277 – 286,2004

[52] J. C. Sancho, F. Petrini, G. Johnson, J. Fernandez, and E. Frachtenberg, "On the feasibility of incremental checkpointing for scientific computing", in International Parallel and Distributed Processing Symposium, (Santa Fe, NM, USA), April 2004.

[53] Roberto Gioiosa, Jose Carlos Sancho,Song Jiang and Fabrizio Petrini "Transparent, incremental checkpointing at kernel level: a foundation for fault tolerance for parallel computers". Conference on High Performance Networking and Computing Proc. of the 2005 ACM/IEEE conference on Supercomputing Page: 9.

[54] K. Sahoo, R. K., A. Sivasubramaniam, M. S. Squillante, and Y. Zhang. "Failure data analysis of a large-scale heterogeneous server environment". In Proc. of DSN'04, 2004.

[55] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J.E. Moreira, S. Ma. "Critical event prediction for proactive management in large-scale computer clusters " International conference on Knowledge discovery and data mining Pages: 426 - 435,Year of Publication:2003 ISBN:1-58113-737-0

[56] K. S. Trivedi and M. Malhotra, "Reliability and performability techniques and tools: a survey". Proc. of Seventh ITG/GI Conference, MMB, Aachen University of Technology, Sept. 27 - 48, 1993.

[57] J. Huang and M. J. Zuo, "Generalized multi-state k-out-of-n:g systems". IEEE Transaction on Reliability, vol. 49, no. 1, March, 2000, pp. 105-111.

[58] C. Hirel, R. A. Sahner, X. Zang, and K. S. Trivedi, "Reliability and performability modeling using SHARPE 2000," Proceedings of the Eleventh International Conference on Computer Performance Evaluation: Modeling Techniques and Tools, March 27 - 31, 2000.

[59] C. Hirel, B. Tuffin, and K. S. Trivedi, "SPNP: stochastic Petri. nets. Version 6.0". Proceedings of the Eleventh International Conference on Computer Performance Evaluation: Modeling Techniques and Tools, 2000.

[60] A. Cumani, "ESP - A package for the evaluation of stochastic Petri nets with phase-type distributed transition times". In Proc. of International Workshop Timed Petri Nets, Torino, Italy, 1985, pp. 144-151.

[61] Kishor S. Trivedi, "Probability and statistics with reliability, queuing, and computer science applications". John Wiley & Sons, Inc., New York, 2002.

[62] R.A. Sahner, K.S. Trivedi, and A. Puliafito, "Performance and reliability analysis of computer systems: An Example-Based Approach Using the SHARPE Software Package". Kluwer Academic Publishers, New York, 1996.

[63] Chandy, K. M. "A survey of analytic models of roll-back and recovery strategies Computer", 8, 5 (May 1975), 40-47

[64] Chandy, K M, Browne, J C, Dissly, C W, and Uhrig, W. R. "Analytic models for rollback and recovery strategms m data base systems" IEEE Transactions Software Engineering. SE-1, (March 1975), 100-110

[65] M. Treaster. "A survey of fault-tolerance and fault-recovery techniques in parallel systems". Technical Report Computer Sciences. DC/ 0501002, ACM Computing Research Repository (CoRR), January 2005.

[66] Young, J W "A first-order approximating to the optimum checkpoint interval" Communication ACM 17, 9 (Sept 1974), 530-531

[67] V.F. Nicola, "Checkpointing and the modeling of program execution time, software fault tolerance", M.R. Lyu, ed., pp. 167-188, John Wiley & Sons, 1995.

[68] Daly, J.: "A model for predicting the optimum checkpoint interval for restart dumps". ICCS 2003, LNCS 2660, Proceedings 4 (2003) 3–12

[69] Daly, J.T.: "A higher order estimate of the optimum checkpoint interval for restart dumps". Future Generation Computer Systems (Elsevier, Amsterdam, 2004).

[70] A.K.Basu, "An introduction to stochastic process", (Dec. 2002).

[71] F. K. Wong, M. A. Franklin, "Distributed computing systems and checkpointing". HPDC 1993: 224-233

[72] J. S. Plank, M. G. Thomason, "The average availability of parallel checkpointing systems and its importance in selecting runtime parameters". IEEE Proc. Int'l Symposium on Fault-Tolerant Computing, 1999.

[73] R. Geist, R. Reynolds and J. Westall, "Selection of a checkpoint interval in a critical-task environment". IEEE Transactions on Reliability, 37, (4), 395--400 (1988)

[74] E. Hendriks, "BProc: the Beowulf distributed process space". In Proc. of the 16th Annual ACM International Conference on Supercomputing, New York City, June 22–26, 2002.

[75] H. Zhong and J. Nieh. CRAK: "Linux checkpoint/restart as a kernel module". Technical Report CUCS-014-01, Department of Computer Science, Columbia University, New York, November2001.

[76] D. J. Becker, T. Sterling, D. Savarese, J. E. Dorband, U. A. Ranawak, and C. V. Packer, "Beowulf: a parallel workstation for scientific computation". Proc. of International Conference on Parallel Processing, 1995.

[77] T. Naughton, S. L. Scott, Y. Fang, P. Pfeiffer, B. Ligneris, and C. Leangsuksun, "The OSCAR toolkit: current and future developments". Dell Power Solutions, Nov., 2001.

[78] C. Leangsuksun, L. Shen, T. Liu, H. Song, and S. L. Scott, "Availability prediction and modeling of high availability OSCAR cluster," IEEE International Conference on Cluster Computing, Hong Kong, December 1-4, 2003.

[79] S. Osman, D. Subhraveti, G. Su, and J. Nieh. "The design and implementation of Zap: a system for migrating computing environments". In Proc. of the Fifth Symposium on Operating Systems Design and Implementation, Boston, MA, December 9–11, 2002.

[80] J. Duell, P. Hargrove, and E. Roman. "The design and implementation of Berkeley Lab's Linux Checkpoint/Restart". Available from http://old-www.nersc.gov/research/FTG/checkpoint/blcr.pdf.

[81] D. Jin and S. G. Ziavras, "A super-programming technique for large sparse matrix multiplication on PC clusters". IEICE Transactions on Information and Systems, Special Issue on Hardware/Software Support for High Performance Scientific and Engineering Computing, vol. E87-D, no. 7, July 2004.

[82] B. Venners, "Inside The Java Virtual Machine", McGraw-Hill, 1997.

[83] O. Bolza, "Lectures on the calculus of variations", Third Edition, Chelsea Publishing Company, New York, 1973.

[84] J. L. Devore, Probability and statistics, Fifth Edition., Brook/Core, New York, 2000.

[85] M. H. DeGroot and M. J. Schervish, "Probability and statistics", Third Edition, Addition-Wesley, New York, 2002.

[86] R. V. Hogg and A. T. Craig, "Introduction to mathematical statistics", the Fourth edition, Macmillan Publishing Co., Inc., New York, 1978.

[87] T. Y. Lin and D. P. Siewiorek, "Error log analysis: statistical modeling and heuristic trend analysis". IEEE Transactions on Reliability, vol. 39, no. 4, Oct. 1990, pp. 419-432.

[88] P. Moran, P. Gaffney, J. Melody, M. Condon and M. Hayden, "System availability monitoring". IEEE Transactions on Reliability, vol. 39, no. 4, Oct., 1990, pp. 480-485

[89] T. Mattson, B. Sanders, and B. Massingill, "Patterns for parallel programming". Addison-Wesley, 2005.

[90] Y. Liu, C. Leangsuksun, H. Song, and S. L. Scott, "Reliability-aware checkpoint /restart scheme: a performability trade-off". IEEE International Conference on Cluster Computing, 2005.

[91] H. Song, C. Leangsuksun, R. Nassar, and Y. Liu "Availability specification and evaluation of HA-OSCAR servers – an object-oriented approach". The Third International Conference on Computing, Communications and Control Technologies, Austin, Texas, July, 2005.

[92] C. Leangsuksun, C. Kottapalli, H. Song, and Y. Liu, "Reliability-aware intelligent checkpointing of MPI programs in Beowulf clusters", High Availability and Performance Computing Workshop, Santa Fe, New Mexico, 2004.

[93] C. Leangsuksun, L. Shen, T. Liu, H. Song and S. L. Scott, "Availability prediction and modeling of high availability OSCAR cluster". IEEE International Conference on Cluster Computing, Hong Kong, December, 2003.

[94] C. Leangsuksun, L. Shen, T. Liu, H. Song and S. L. Scott, "Dependability prediction of high availability OSCAR cluster server". Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, Nevada, June, 2003.