

Spring 2008

Failure analysis and reliability -aware resource allocation of parallel applications in High Performance Computing systems

Narasimha Raju Gottumukkala
Louisiana Tech University

Follow this and additional works at: <https://digitalcommons.latech.edu/dissertations>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Gottumukkala, Narasimha Raju, "" (2008). *Dissertation*. 510.
<https://digitalcommons.latech.edu/dissertations/510>

This Dissertation is brought to you for free and open access by the Graduate School at Louisiana Tech Digital Commons. It has been accepted for inclusion in Doctoral Dissertations by an authorized administrator of Louisiana Tech Digital Commons. For more information, please contact digitalcommons@latech.edu.

**FAILURE ANALYSIS AND RELIABILITY-AWARE RESOURCE
ALLOCATION OF PARALLEL APPLICATIONS IN HIGH
PERFORMANCE COMPUTING SYSTEMS**

by

Narasimha Raju Gottumukkala, 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

March 2008

UMI Number: 3298937

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI[®]

UMI Microform 3298937

Copyright 2008 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

LOUISIANA TECH UNIVERSITY

THE GRADUATE SCHOOL

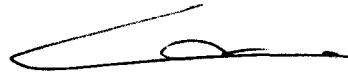
12/11/2007

Date

We hereby recommend that the dissertation prepared under our supervision
by NARASIMHA RAJU GOTTUMUKKALA

entitled FAILURE ANALYSIS AND RELIABILITY- AWARE RESOURCE ALLOCATION
OF PARALLEL APPLICATIONS IN HIGH PERFORMANCE COMPUTING SYSTEMS

be accepted in partial fulfillment of the requirements for the Degree of
Ph.D. in COMPUTATIONAL ANALYSIS AND MODELING



Supervisor of Dissertation Research
Wigberto Du'
Head of Department

Department

Recommendation concurred in:

Raja S. Sankaranarayanan

Patil

J. S. Rao

Wigberto Du'

Advisory Committee

Approved:

Patil Sankaranarayanan
Director of Graduate Studies

Approved:

Wigberto Du'
Dean of the Graduate School

Dean of the College

Stan Nigam

ABSTRACT

The demand for more computational power to solve complex scientific problems has been driving the physical size of High Performance Computing (HPC) systems to hundreds and thousands of nodes. Uninterrupted execution of large scale parallel applications naturally becomes a major challenge because a single node failure interrupts the entire application, and the reliability of a job completion decreases with increasing the number of nodes. Accurate reliability knowledge of a HPC system enables runtime systems such as resource management and applications to minimize performance loss due to random failures while also providing better Quality Of Service (QOS) for computational users.

This dissertation makes three major contributions for reliability evaluation and resource management in HPC systems. First we study the failure properties of HPC systems and observe that Times To Failure (TTF's) of individual compute nodes follow a time-varying failure rate based distribution like Weibull distribution. We then propose a model for the TTF distribution of a system of k independent nodes when individual nodes exhibit time varying failure rates. Based on the reliability of the proposed TTF model, we develop reliability-aware resource allocation algorithms and evaluated them on actual parallel workloads and failure data of a HPC system. Our observations indicate that applying time varying failure rate-based reliability function combined with some heuristics reduce the performance loss due to unexpected failures by as much as

30 to 53 percent. Finally, we also study the effect of reliability with respect to the number of nodes and propose reliability-aware optimal k node allocation algorithm for large scale parallel applications. Our simulation results of comparing the optimal k node algorithm indicate that choosing the number of nodes for large scale parallel applications based on the reliability of compute nodes can reduce the overall completion time and waste time when the k may be smaller than the total number of nodes in the system.

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 N R GOTTUMUKKALA *G. R. N. G. U.*

Date 02/18/2008

TABLE OF CONTENTS

ABSTRACT.....	iii
LIST OF TABLES.....	viii
LIST OF FIGURES.....	ix
ACKNOWLEDGEMENTS.....	xi
CHAPTER 1 INTRODUCTION.....	1
1.1 Failures in HPC Systems.....	1
1.2 Resource Allocation in HPC Systems.....	2
1.3 Organization of Dissertation.....	5
CHAPTER 2 RELATED WORK.....	6
2.1 TTF Properties and System Reliability.....	6
2.2 Reliability-Aware Resource Allocation.....	7
2.3 Optimal K Node Allocation of Parallel Applications.....	8
CHAPTER 3 THE TIME TO FAILURE PROPERTIES OF HPC SYSTEMS.....	10
3.1 Introduction.....	10
3.2 Description of Failure Data.....	11
3.3 TTF Distribution of Individual Nodes.....	13
3.3.1 Distributions and Goodness Of Fit Test.....	13
3.3.2 Comparison of TTF Distributions.....	14
3.4 TTF Distribution of System of K Nodes.....	16
3.5 Correlation of TTF's Between Nodes.....	17
3.6 Autocorrelation of TTF's.....	19
3.7 The Failure Parameters of Various Nodes.....	22
CHAPTER 4 THE DISTRIBUTION OF TIME TO FAILURE PROPERTIES OF K NODES IN HPC SYSTEMS.....	24
4.1 Introduction.....	25
4.2 TTF Distribution of a System of K Nodes.....	27
4.3 Goodness Of Fit Tests.....	34

4.4 Numerical Example	36
4.5 Conclusion	43
CHAPTER 5 RELIABILITY AWARE RESOURCE ALLOCATION IN HPC SYSTEMS.....	45
5.1 Introduction.....	45
5.2 Reliability Model for a Parallel Application.....	46
5.2.1 Reliability of Job Completion Time	46
5.2.2 Reliability Model for a Parallel Application.....	48
5.2.3 Reliability-Aware Resource Allocation Algorithms.....	48
5.2.4 A Study of Waste Time for a Parallel Program	50
5.2.5 Heuristics for Reliability-Aware Resource Allocation	52
5.3 Comparison of Reliability-Aware Resource Allocation Algorithms.....	54
5.3.1 Simulation Study.....	55
5.3.2 Parallel Workloads.....	55
5.3.3 Comparison of Reliability Prediction	56
5.4 Conclusions.....	59
CHAPTER 6 RELIABILITY AWARE OPTIMAL K NODE ALLOCATION OF PARALLEL APPLICATIONS.....	60
6.1 Introduction.....	60
6.2 The Expected Completion Time of a Parallel Program.....	62
6.3 Performance and Scalability of Parallel Programs	65
6.3.1 Amdahl's Law	66
6.3.2 Gustafson's Law	67
6.4 Reliability-Aware Resource Allocation.....	68
6.4.1 Resource Allocation Algorithms.....	69
6.4.1.1 All Nodes (ALL).....	69
6.4.1.2 Round Robin Allocation (RR)	69
6.4.1.3 Reliability-aware Allocation (RA).....	70
6.4.2 Reliability-Aware Optimal_K Node Allocation	70
6.4.3 Numerical Example	73
6.4.4 Simulation Study.....	74
6.4.5 Performance Metrics.....	75
6.4.6 Experimental Results	76
6.6 Conclusion and Future Work.....	79
CHAPTER 7 CONCLUSIONS AND FUTURE WORK.....	81
APPENDIX A PROOFS FOR SYSTEM RELIABILITY MODEL.....	83
APPENDIX B MATLAB PROGRAMS FOR THE SYSTEM TTF MODEL	86
REFERENCES	90

LIST OF TABLES

Table 3.1 The CDF's of various distributions.	14
Table 3.2 Comparison of various distributions based on p-value for all nodes.....	16
Table 3.3 Comparison of failure distributions using the Kolmogorov-Smirnov Goodness.....	17
Table 3.4 Significance of correlation with respect to correlation coefficient values.....	20
Table 3.5 The statistical properties of shape and scale parameters of the TTF's of individual nodes obtained from White.....	23
Table 4.1 The failure times of three nodes	37
Table 6.1 A Numerical Example showing the expected completion time of a parallel program on k nodes.....	74

LIST OF FIGURES

Figure 3.1 The number of failures in 3-month intervals over 4 years 3 month's period on the ASC White	12
Figure 3.2 Comparison of the empirical CDF with the theoretical CDF of gamma Weibull and lognormal distributions for node 277.....	16
Figure 3.3 The cross correlation coefficient among the i^{th} TTF among different nodes.	20
Figure 3.4 Autocorrelation among TTF's of Individual nodes.....	21
Figure 3.5 The Autocorrelation among system TTF's.....	22
Figure 3.6 The scale parameters and MTTF's of various nodes in White.....	23
Figure 4.1 Algorithm to determine the system TTF's from TTF's of individual nodes.	28
Figure 4.2 Schematic diagram of system TTF's obtained from the TTF's of individual nodes.....	29
Figure 4.3 The Chi-Square GOF tests comparing the actual and expected number of failures.	36
Figure 4.4. Job submitted at (x=100, t=0).....	37
Figure 4.5 Job submitted at (x=500, t=300).....	39
Figure 4.6 Job submitted at t=200, x=350 when the system failed at 1114 Hrs.....	40
Figure 4.7 Job submitted at t=50, x=150 when the system failed at 1167 Hrs.....	41
Figure 4.8. The effect of failure rate and MTTF with the increase in number of nodes (k).....	43
Figure 4.9 The effect of system reliability with the increase in the number of nodes	43

Figure 5.1. The Reliability-Aware Scheduling Algorithm	49
Figure 5.2 Description of workloads and comparison of Waste Times for reliability-aware policies for individual jobs requiring different number of processors.	51
Figure 5.3 Description of workloads and comparison of Waste Times for reliability-aware policies for individual jobs requiring different job run lengths.....	52
Figure 5.4 The Longest Job Reliability-Aware Scheduling Algorithm.....	54
Figure 5.5 The comparison of overall waste times for the two workloads namely LANL and SDSC.	56
Figure 5.6 Description of LANL and SDSC workloads based on run lengths and comparison of Total Waste Time and % average Waste Time for each run lengths of jobs.	58
Figure 6.1 Effect of reliability with the increase in number of nodes.	61
Figure 6.2 An un checkpointed parallel application with failures.	63
Figure 6.3 The expected completion times for various values of the shape and scale parameters for a Weibull distribution.	65
Figure 6.4 The scalability effect with respect to job completion time for different performance models (Amdahl's Law and Gustafson's Law)	68
Figure 6.5 The expected completion times of parallel programs considering reliability for Amdahl's Law (a) and Gustafson's law (b).....	68
Figure 6.6 The Optimal k node allocation algorithm.....	72
Figure 6.7 The Optimal k node selection by the algorithm.	73
Figure 6.8 Comparison of MTA and WTA for various resource allocation algorithms.....	77
Figure 6.9 The comparison of ATA and AWT with respect to run-lengths.	78

ACKNOWLEDGEMENTS

Finally, as I stand here at a point closer to finishing my dissertation and look back at the journey from where I have started, it has been the most challenging, enduring, educating and rewarding experience of my career. Several people that have helped me to come this far.

First of all I owe most of my achievements to Dr. Chokchai Box Leangsuksun, who has been the most influential person in my career and to whom I shall ever be grateful. He has guided me from the start in every aspect of my research and career from picking a high-quality dissertation problem, writing and presenting research, providing critical feedback, supporting me, and motivating me always to go a step further. He has also given me the experience of working on various exciting HPC technologies and has also brought in several researchers in the HPC area from ORNL like Dr. Stephen Scott, Christian Engelmann, Dr. Hong Ong and Dr. George Ostrouchov who also have provided valuable inputs on this research.

Dr. Raja Nassar has validated each and every idea and equation in this dissertation, and helped me improve my technical writing to a great extent. He has taken time even during his retirement and shown me far more patience than I deserve. I also sincerely thank Dr. Mihaela Paun for discussing ideas, providing valuable feedback and spending time to correct each and every chapter in my dissertation. Dr. Dileep Sule has taken a very special interest in my research by providing me excellent direction. He has

taken valuable time on several occasions to discuss my dissertation. He has provided some valuable insights on the scheduling aspect. I also thank Dr. Dai for correcting my dissertation, providing valuable feedback and guiding me through the CAM program. Dr. Greechie was also helpful in guiding me through the CAM program.

I also specially thank my very dear friend and colleague Kiran Balagani for always being there to share my research and proofread my papers. I also thank my other dear friend and Nethken Hall late-night partner Shrijit Joshi for helpful discussions. I also thank my excellent colleagues at the Extreme Computing Research Group: Hertong Song, Yudan Liu, Narate Taerat, Nichamon Naksinehaboon, Vishal Rampure, Anand Tikotekar and Kiriti whom I worked in several occasions on various HPC-related projects.

On a personal level, I would like to thank my dearest wife Siri who has shown great love and patience and has given never ending support throughout even in the toughest of times. She has put up with me for working late nights in the lab and has taken great care of me throughout. Finally, I thank my parents Krishnaveni and Janaki Rama Raju, and my Guruji Dr. Umar Alisha who have been a pillar of support throughout.

I would like to dedicate this dissertation to my Sathguru Dr. Umar Alisha and my late grand father Manthena Rama Krishna Raju.

CHAPTER 1

INTRODUCTION

The demand for more computational power to solve complex scientific applications has given rise to High Performance Computing (HPC) systems that comprise hundreds of thousands of nodes. HPC has gained significant prominence in recent years because of its cost-effective way to build systems from Commercial-Off-The-Shelf (COTS) components. Uninterrupted execution of large scale parallel applications naturally becomes a major challenge because a single node failure interrupts the entire application. Increasing the number of nodes for a parallel application increases the failure probability, therefore providing reliability for parallel applications running on large scale computational resources becomes a major challenge.

1.1 Failures in HPC Systems

HPC systems comprise several hardware and software resources required for uninterrupted completion of a parallel application. Unexpected failures and downtimes have severe effects both on the performance of a HPC system and the Quality of Service (QOS) for the computational users. It is unrealistic, at least in the near future, to completely eliminate failures and predict the next failure event and time [1]. However, several monitoring tools and fault tolerance mechanisms can presently deal with failures. Monitoring tools like CluMon [2], Ganglia [3] and Nagios [4] report detailed health

information of various hardware and software components, and provide failure warnings in event of abnormal activities. These monitoring frameworks lack the capability to predict reliability or future failure events. There are also fault-tolerance mechanisms like task duplication and checkpoint/restart. Checkpoint/restart is a mechanism that enables saving the software state at regular intervals so that the program does not have to restart over from the beginning. Current checkpoint schemes may, however, be very inefficient for large scale parallel programs because of the overhead of saving the state of multiple processes running on large number of nodes [5][6]. Therefore the checkpoint algorithm must rely on the reliability information of the resources to optimize checkpoint placement and minimize the performance loss.

The failure events of various compute nodes in HPC systems may be recorded in order to assess system reliability and failure rates. Dynamic reliability analysis of selected components provides up-to-date system reliability. RAS frameworks [7][8] have recently been proposed for online monitoring and modeling of HPC systems to complement resource management and checkpoint frameworks. An accurate reliability model and up-to-date failure information would complement reliability-aware resource management, reliability-aware checkpoint/restart, and scheduled maintenance of computational resources.

1.2 Resource Allocation in HPC Systems

Scheduling or resource allocation involves task assignment to computational resources to satisfy certain job criteria. In a broader sense, scheduling in parallel super computers can be at two levels. The first or top level is in application level or meta-level that deals with allocating a parallel application to a partition of compute nodes or

systems. Second is the lower level or the operating system level that takes a task and allocates various local resources like memory, CPU, disks and I/O devices. There are different characteristics of jobs and different domains for application scheduling in general [9]. In this dissertation we concentrate on scheduling of parallel applications at the application level.

A scheduler basically consists of a job queue, and a decision algorithm that decides where to allocate the job based on the job's requirements and other policies. The objective of the scheduler can be to maximize throughput and utilization, to minimize completion time, to prioritize jobs, or just to execute the jobs. One of the most important aspects for HPC applications is maximizing performance, which means minimizing the completion time. There are several resource parameters that affect the completion time of a parallel program like the CPU speed, I/O bandwidth, memory, system architecture, network bandwidth and latency, etc. Completion time is usually a sum of other times like the job submission time, scheduler response time, waiting time or execution time [10]. In this dissertation we focus on completion time. The job completion time is basically a sum of three components, (1) the waiting time (2) job execution time (3) and waste time due to failures. We further discuss these definitions in Chapter 5 and Chapter 6.

There are several application level schedulers that are widely used for distributed platforms. Each scheduler has various capabilities in terms of monitoring resources, giving priorities and task assignment. PBS/MAUI [11], SGE [12], and SLURM [13] are the most widely used job schedulers in HPC systems. PBS/MAUI and SLURM have two basic scheduling policies namely First In First Out (FIFO) and Backfilling. The FIFO policy allocates jobs based on the arrival order. Backfilling enables moving short jobs

ahead if longer jobs in the queue are not interrupted. In addition some other policies that are aimed at giving fair share of resources to users. For example SGE uses several ticket-based policies to prioritize jobs based on users [12].

Parallel applications may also be further categorized into four types based on a decision factor such as the number of processors required by a given job. Rigid jobs require the number of processors to be specified during job submission, and the job does not change the processor requirement during execution. An evolving job is the one that keeps changing the processor requirements while the job is executing. A moldable job allows the scheduler or resource manager to dictate the number of processors, and a malleable job is adaptable to changes in the processor count during execution. More details of these jobs are given in [9]. In this dissertation we concentrate on rigid jobs and moldable jobs. In the parallel workloads available at [15], the users already give the number of processors that would let us to use the actual workloads for rigid jobs. Secondly, some of the MPI applications are flexible in the sense that the number of nodes can be decided by the scheduler during run time (moldable jobs). We will further discuss reliability-aware scheduling of moldable jobs in Chapter 6.

Currently existing resource managers/schedulers do not consider reliability as an important factor. However, reliability becomes crucial research for very large scale systems that span hundreds and thousands of nodes. Obtaining accurate reliability information provides insights for resource managers to minimize the overall performance loss of the HPC system. In addition reliability knowledge of resources lets resource manager to ensure Quality of Service to users.

1.3 Organization of Dissertation

This dissertation is organized as follows: Chapter 2 discusses the related work on the failure properties, system reliability models, and resource allocation techniques for HPC systems. The description of failure data, a detailed statistical analysis such as the comparison of Goodness Of Fit (GOF) tests of various TTF distributions of individual nodes, and system of k nodes, correlation analysis, and variability of MTTF's and failure parameters are demonstrated in Chapter 3. Chapter 4 presents a new model for the TTF distribution of a system of k nodes in a HPC system when individual nodes have time varying failure rates. Chapter 5 describes our proposed reliability-aware resource allocation techniques for parallel applications. The effect of scalability on reliability and performance, and an optimal reliability-aware k node allocation algorithm for moldable parallel jobs are further discussed in Chapter 6. The conclusions and future work are given in Chapter 7.

CHAPTER 2

RELATED WORK

This chapter discusses the related work on the TTF properties, system TTF model when individual nodes have time varying failure rates, resource allocation techniques in HPC systems and reliability-aware resource allocation algorithms.

2.1 TTF Properties and System Reliability

Failure properties like the TTF distribution, and correlations between failures enable us to develop reliability or failure models that represent the actual behavior of systems. Failure properties of computing systems have also been studied by many researchers[14]. The TTF's in various distributed computing platforms [16], [17], [18], [19] and [20] are observed to follow the Weibull distribution. Various aspects of correlations have been studied in the literature. In [21], the analysis on failures and error logs on heterogeneous servers have shown that there are significant autocorrelation, cross correlation, and long term temporal correlations among nodes. Also, the temporal correlation patterns vary across different nodes. Iyer [20] found a significant correlation of workload on failures due to increase in CPU activity. According to [22], the correlated failures comprise 27% of all the data and the impact of correlated failures is significant. In Chapter 3, we discuss the TTF distribution of a HPC system, the correlation of TTF's

between individual nodes, and autocorrelation of individual nodes and a system of k nodes of a production HPC system.

The system reliability model for a constant failure rate is uncomplicated as compared with time varying failure rate assumptions to model various aspects of system reliability. Dey [23] presents the system reliability and proposes a parameter estimation technique when individual components in a series system are Weibull distributed, and individual components may have different failure parameters. Also, Hassett [24] discussed the availability and reliability of a system with components having time varying failure rates. Individual compute nodes are shown to follow a Weibull distribution instead of an exponential. We therefore propose a time varying failure rate based distribution for a system of k nodes when nodes have different Weibull based shape and scale parameters in Chapter 4.

2.2 Reliability-Aware Resource Allocation

Failure data analysis and reliability-aware scheduling research in general have recently gained much attention in HPC community. Zhang et al. [25] studied failure rates of HPC systems that affect performance metrics like job-slow-down and work-loss ratio based on the spatial and temporal correlation of time to failures. Oliner et al. [26] discussed the effect of HPC reliability metrics on parallel applications with periodic check pointing under an assumption that failures follow Poisson process, proposed event based failure prediction on Blue Gene/L failure logs [27] and presented failure-aware scheduling techniques. Also, Linping [28] discusses the failure aware scheduling policies based on the Longest Uptime of nodes.

Prior work on reliability-aware task allocation for heterogeneous distributed systems models tasks and communications links as a directed graph by Shatz et al. [29]. While the graph based reliability model is a good theoretical model for task allocation in heterogeneous systems [29][30][31], this model relies on a constant failure rate assumption, and the simulation results are not based on the failure data of actual systems. Recently, Ming [32] proposed exponential-based performance prediction and fault aware allocation of parallel applications using failure rates, downtimes, checkpoint overhead and fault aware heuristic algorithms. To our knowledge, no work that developed reliability-aware algorithms based on a time-varying distribution, or techniques that use the failure data of actual system for reliability-aware resource allocation.

2.3 Optimal K Node Allocation of Parallel Applications

Various aspects of scalability were studied in [33][34][18][35][36][37][38]. Amdahl's law [33] suggests that there is a serial part of the program that limits scalability. However, according to Gustafson [38] other parameters in computation that can be overlapped to achieve better parallelism while executing the serial portion of the program. Kumar et al. [35] have given a survey of scalability models for parallel architectures and algorithms for a given parallel architecture and problem size. Nicol [39] discussed the optimal selection of the number of processors for a numerical approximation problem and architecture. The impact of reliability for large scale parallel applications is not new and has also been addressed recently [40][41]. The effect of reliability on the completion time of parallel programs is discussed in [42], and the effect of coordinated check pointing on large scale parallel applications due to failures is

discussed in [6][41]. Also, Plank[43] discusses the importance of considering the number of processors as an important performance attribute for check pointed jobs.

In Chapter 6, we study the scalability limitations of a parallel application with respect to system reliability. We also propose a reliability-aware optimal k node selection algorithm with the objective of minimizing the completion time and the waste time of large scale parallel applications in the presence of failures.

CHAPTER 3

THE TIME TO FAILURE PROPERTIES OF HPC SYSTEMS

3.1 Introduction

Several factors cause failures in a HPC system. Understanding various failure properties enable better assessment of the overall system reliability. This chapter describes some important failure properties such as the various factors that cause failures, the (Time To Failure) TTF distribution of individual nodes and system of k nodes, the correlations between TTF's of various nodes, the autocorrelation of TTF's and the variability of MTTF's among various nodes in a HPC system.

HPC systems contain a significant number of hardware and software components that are required for uninterrupted running of applications. We classify the factors that affect failures broadly into three categories:

(1) *Inherent defects during development/manufacturing*

Both hardware and software are prone to defects or bugs. Many of these defects become visible when the system becomes operational. Hardware components like processors, memory and hard disks may have design problems, or manufacturing problems due to fabrication and processing [44]. Similarly, software is prone to have bugs, which may be due to design or logical errors during coding and to the software that is not tested adequately.

(2) Failures caused due to environments

Both hardware and software failures could occur due to the operating conditions. For example, overuse of hardware, power fluctuations, and noise can cause hardware failures, and certain software upgrades or operating system patches may affect other software already residing in the system. Exhausted resources such as low swap space and CPU overload may also cause some failures.

(3) Failures caused due to misconfiguration by administrators

Failures are also be caused by the administrators who may misconfigure the system after installation or upgrade. Administrators have to be careful because a minor configuration error could lead to an outage of the entire system affecting all the jobs that were running.

3.2 Description of Failure Data

In the reliability analysis, the failure data source is the failure logs obtained from Lawrence Livermore National Laboratory (LLNL) ASC White [45], a 512-node cluster. Each node is a 16-way Symmetric Multi Processor (SMP), and thus the total number of processors is 8196. Each event log includes the type of failure, length of downtime, time of failure, and the impacted node. Since we focus on the TTF's, we extract the failure times and the node information from the failure logs. The ASC white failure information consists of a large dataset including significant failure events over the period of four years, from 2000 to 2004. The failure data have 72.3 percent hardware failures, 13.8 percent software, and 13.8 percent are other type of failures. Single node failures usually occur from a disk/memory failure. Failures from a network switch or a common power failure affects a group of nodes, and failures from a head node or scheduler master

daemon may affect all the nodes. About 75 percent of the failures are from a single node category, 15 percent belong to a group of nodes, and 10 percent failures affect all 512 nodes. The event logs also have both scheduled and unscheduled downtime times. In our TTF analysis, we consider the failures that affect a single node, and a group of nodes. Figure 3.1 shows the number of failure events recorded during the period 7/1/2000 to 10/1/2004 from the ASC White system logs. The total operational time is approximately 4 years 3 months (0.2×10^6 minutes). We observe that in the first two years and three months there were more failures as compared with the last two years.

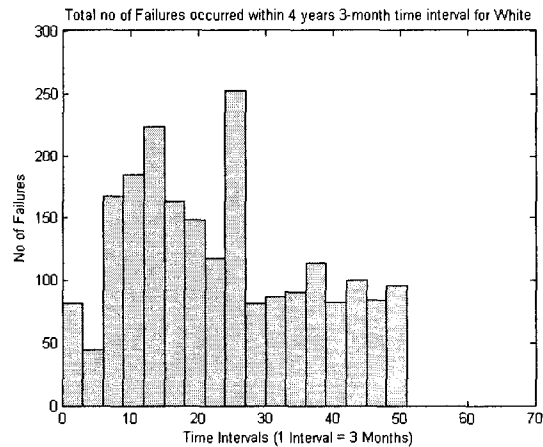


Figure 3.1 The number of failures in 3-month intervals over 4 years 3 month's period on the ASC White

The newly deployed components tend to have more failures in the initial phase; this may be called the infancy stage when the overall failure rate is represented by the bathtub curve based failure rate. After the infancy stage, failures approximately remain steady. We use the most recent failure data available from 4/1/2003 to 10/1/2004 for both TTF analysis and reliability-aware resource allocation study, because data during this period represent the most recent failure behavior.

3.3 TTF Distribution of Individual Nodes

3.3.1 Distributions and Goodness Of Fit Test

Failure distribution functions such as exponential, Weibull, lognormal and gamma have been widely used in reliability analysis [16][46][46]. The Cumulative Density Function (CDF) for a given TTF distribution is given by,

$$F(t) = \int_0^t f(\tau) d\tau \quad (3.1)$$

Where $f(\tau)$ is the probability density function (Pdf). The Cumulative Distribution Functions (CDF) of various TTF distributions are given in Table 3.1.

The Kolmogorov-Smirnov (K-S) Goodness Of Fit (GOF) test [47] compares the theoretical time to failure distributions with the empirical distribution. If $F_0(x)$ is the empirical CDF based on N data points, $F_1(x)$ is the theoretical CDF, the K-S statistic D is defined as the maximum absolute difference between the two CDF's which is given by

$$D = \max_{-\infty < x < \infty} |F_0(x) - F_1(x)|$$

The K-S GOF test gives the maximum distance between the empirical and theoretical distribution [47]. The p-value ($0 \leq p \leq 1$) in K-S test represents the probability that the sample data belongs to certain distributions. A p-value ($p \leq 0.05$) indicates that the distribution does not fit the data. Greater the p-value, greater is the probability that the empirical distribution closely represents theoretical distribution. Thus, we apply K-S test to identify which theoretical distribution best represents the empirical failure distribution.

Table 3.1 The CDF's of various distributions.

Distribution Name	CDF
Exponential	$F(t) = 1 - e^{-\lambda t}$ where λ is the failure rate.
Weibull	$F(t) = 1 - e^{-\left(\frac{t}{\alpha}\right)^\beta}$ where α is the scale parameter, and β is the shape parameter.
Lognormal	$F(t) = \Phi\left\{\frac{\ln \frac{t}{T_{50}}}{\sigma}\right\}$ σ is the shape parameter. T_{50} is the medial life at 50% failure point.
Gamma	$\frac{\Gamma_x(\gamma)}{\Gamma(\gamma)} \quad x \geq 0$ $\Gamma(\gamma) \quad \gamma > 0$ where $\Gamma_x(a) = \int_0^x t^{\alpha-1} e^{-t} dt$ $\Gamma_\infty(a) = \int_0^\infty t^{\alpha-1} e^{-t} dt$ and γ is the shape parameter.

3.3.2 Comparison of TTF Distributions

The TTF distribution lets us understand whether the TTF increases, decreases, or remains constant over time, and an appropriate theoretical TTF distribution can be used for reliability prediction for minimizing the performance loss due to failures. The empirical failure distributions of individual nodes are compared with the theoretical distribution based on the p-value of the Kolmogorov-Smirnov GOF test. As an example, Figure 3.2 shows the empirical CDF's compared with the theoretical CDF's of

exponential, Weibull, gamma and lognormal distributions. The greater the p-value of the K-S test, the better is the GOF. Different nodes could fit better to different distributions so we compared all the 512 nodes, based on the p-values to understand which distribution fits better in most cases. Table 3.2 shows the percentages of how well each distribution fits to the empirical data for all the 512-nodes. In some cases, two or more distributions perform equally well because the Weibull distribution is a general case of the exponential, and the gamma distribution is general enough to give results similar to those of the Weibull or lognormal. Weibull, lognormal, and gamma have time varying failure rates, and in most cases, they fit the data equally well. For the given data, we observed that 89.9 percent of the nodes have TTF's that fit the Weibull better than or as good as the other three distributions. The gamma distribution fits well for 88.8 percent of the nodes, lognormal for 84.3 percent and 60.7 percent of nodes fit to exponential. For the case where the p-value is greater than 0.8, 99 percent of the nodes fit the Weibull distribution. Hence, for the given data set, the Weibull distribution gave the best fit to the TTF's of the different nodes.

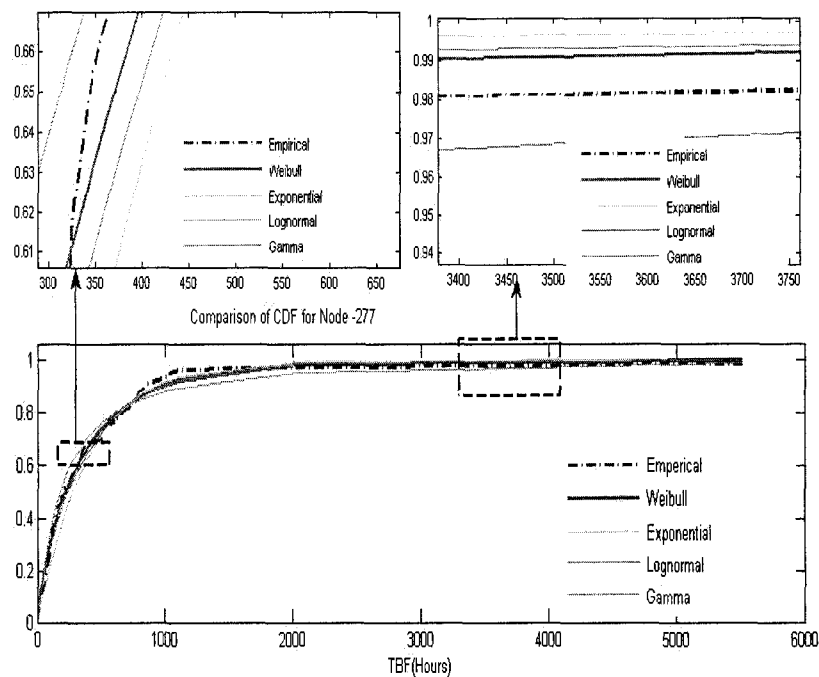


Figure 3.2 Comparison of the empirical CDF with the theoretical CDF of gamma Weibull and lognormal distributions for node 277.

Table 3.2 Comparison of various distributions based on p-value for all nodes.

Distribution name	Number of nodes having highest p-value	Percentage of times
Gamma	79	88.8
lognormal	75	84.3
exponential	54	60.7
Weibull	80	89.9

3.4 TTF Distribution of System of K Nodes

Parallel applications are normally allocated to a set of k nodes for execution. Each node has an individual failure distribution. In our model, we assume that the system fails

when at least one node fails. The TTF's of individual nodes are combined to obtain the TTF distribution of k nodes when the first failure occurs using the algorithm given in Figure 4.1. We compare various distributions, namely exponential, Weibull, Gamma and Lognormal using the K-S test.

Parallel programs are allocated to k processors, and k is usually a power of two. From the given failure data, we show the time to failure distributions for k= 8, 16, 32, 64, 128, 256. The GOF tests for various models are compared for two cases, namely when nodes are selected randomly, and when nodes are selected in order. Table 3.3 shows the comparisons among various distributions for these two cases from the K-S test. In both the cases Weibull is observed to be a better model for reliability of a system of k nodes as compared to exponential and lognormal fit.

Table 3.3 Comparison of Failure distributions using the Kolmogorov-Smirnov Goodness.

Comparison of K-S Test of various number of nodes (nodes selected in order)				Comparison of K-S Test of various number of nodes (nodes selected randomly)			
No of Nodes	p-value	p-value	p-value	No of Nodes	p-value	p-value	p-value
2	0.2628	0.8679	0.5409	2	0.6060	0.4460	0.6573
4	0.2049	0.4310	0.9034	4	0.9940	0.8151	0.9852
8	0.1916	0.9980	0.8571	8	0.2272	0.5758	0.7485
16	0.0818	0.9845	0.3269	16	0.3193	0.7091	0.4671
32	0.0002	0.6300	0.0438	32	0.4829	0.4829	0.2460
64	0.0001	0.7122	0.1538	64	0.0224	0.2484	0.0785
128	0.0001	0.2652	0.0779	128	0.0001	0.1169	0.0061
256	0.0001	0.0599	0.0001	256	0.0001	0.0453	0.0001
350	0.0001	0.0388	0.0001	350	0.0001	0.0159	0.0001

3.5 Correlation of TTF's Between Nodes

Correlation is a way to measure how two variables are related. A correlation of TTF's between two nodes refers to whether the TTF of one node affects the probability of TTF of another node. There are three possible patterns in TTF correlations.

(1) *TTF's are positively correlated*

The i^{th} TTF of one node increases the probability of i^{th} TTF of another node

(2) *TTF's are negatively correlated*

The i^{th} TTF of one node decreases the probability of i^{th} TTF of another node

(3) *There is no correlation between the TTF's of two nodes*

The TTF of one node does not affect the probability of TTF of another node. This implies that the TTF's of two nodes are independent.

Several studies [48][50][51] show that node failures are in fact correlated with failures in multiple nodes occurring nearly simultaneously. Correlated failures have been studied on several distributed systems. Nath et al. [48] discussed the impact of correlated failures on the availability of three distributed systems, namely PlanetLab, Public Web Servers and RON test bed. A study of correlated failures based on conditional probability on PlanetLab[49], a distributed storage platform show that for 75 percent of the nodes there is no correlation; however, 10 percent of the node pairs have correlated failures. Tang et al. [51] have studied the impact of correlated failures on VAX clusters, and proposed dependability models to evaluate correlations. Nath et al. [48] discusses the impact of correlated failures on performability and software reliability and presents a Markov renewal process-based framework to model dependencies between failures.

Distributed systems consist of several hardware and software components with different configurations. Correlation between failures mostly depend on the system environment factors like the operating system configuration, shared resources like network storage and routers, workload, middleware software bugs, and system configurations. Correlated failures lead to additional costs and performance loss, and

these failures have to be minimized during the design phase. For example, Nath et al. [48] propose design principles to minimize correlated failures on distributed systems. In addition, manufacturers attempt to provide fault tolerance for crucial components like disk redundancy, and high available storage solutions to minimize correlation failures.

We use the spearman correlation coefficient on the i^{th} TTF of different combinations of nodes to understand how well the failures are correlated. The Spearman correlation coefficient [52] is given by

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} \quad (3.2)$$

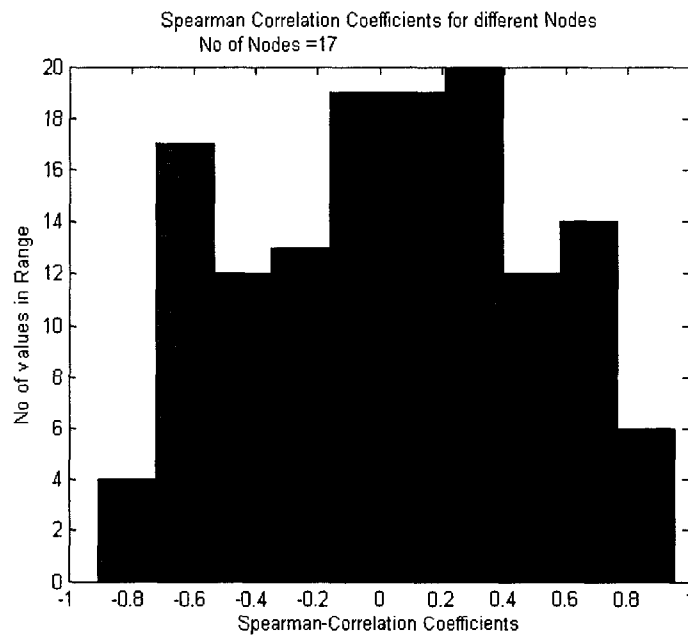
Where d is the difference between each ranks of i^{th} TTF's of different nodes, and n is the number of pairs of values. The significance of correlations with respect to the correlation coefficient values is given in Table 3.4. We performed a correlation test for all the combinations of nodes, and plot a histogram, given in Figure 3.3. We observe that about 60 percent of the TTF's have a weak correlation, 30 percent have a strong correlation and 10 percent have very strong correlation. Future systems have to be designed to be independent of failures, therefore assuming independence assumption is still valid.

3.6 Autocorrelation of TTF's

Autocorrelation between TTF's is another important statistical property that determines whether each failure affects the net consecutive failures. Prior studies use the Autocorrelation Function to determine long-range dependence on the number of failures with age. A study on the failure and error processes on several heterogeneous clusters by Sahoo et al. [21] reveals significant levels of autocorrelation with a periodic behavior suggesting long range dependence of failure and error process.

Table 3.4 Significance of correlation with respect to correlation coefficient values.

Correlation coefficient	Significance of correlation
$\rho \sim 0$	uncorrelated
$-0.4 < \rho < 0.4$	weak correlation
$0.4 \leq \rho \leq 0.8$	high correlation
$-0.8 \leq \rho \leq -0.4$	high correlation
$-0.8 \leq \rho \leq 1$	strong correlation
$-1 \leq \rho \leq -0.8$	strong correlation

Figure 3.3 The cross correlation coefficient among the i^{th} TTF among different nodes.

In addition, a study on the autocorrelation of disk failures by Bianca [53] suggests a long-range dependence among failures. To understand whether there is autocorrelation among TTF's we test the autocorrelation among TTF's of the system using the Durbin Watson statistic [52]. There are two aspects of autocorrelation when we have to consider when we want to develop a model for a system of k nodes

- (1) If TTF's of individual nodes have autocorrelation
- (2) If TTF's of a system of k nodes have autocorrelation when individual nodes may have autocorrelation

Figure 3.4 shows the autocorrelation of TTF's among individual nodes where nodes are ordered based on the number of failures. We observe that the autocorrelation among TTF's for nodes which have lesser number of failures is very insignificant.

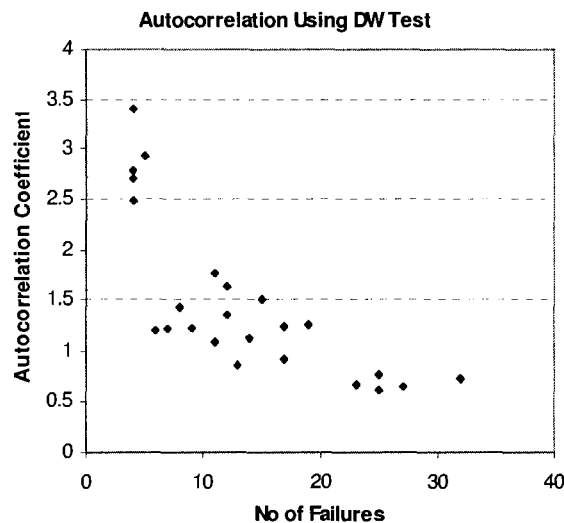


Figure 3.4 Autocorrelation among TTF's of Individual nodes.

However, when the number of failures increases, results show that the autocorrelation becomes significant. In the given data, we observed that the 52 percent of the nodes have significant autocorrelation among TTF's. Because we are interested in

modeling the system TTF behavior we therefore combine the TTF's of nodes into a system in order to understand whether there would be autocorrelation. We combined the nodes in the order of node numbers using the algorithm given in Figure 4.1 in Chapter 4. We observe in Figure 3.5 that the autocorrelation is not very significant for a system of k nodes.

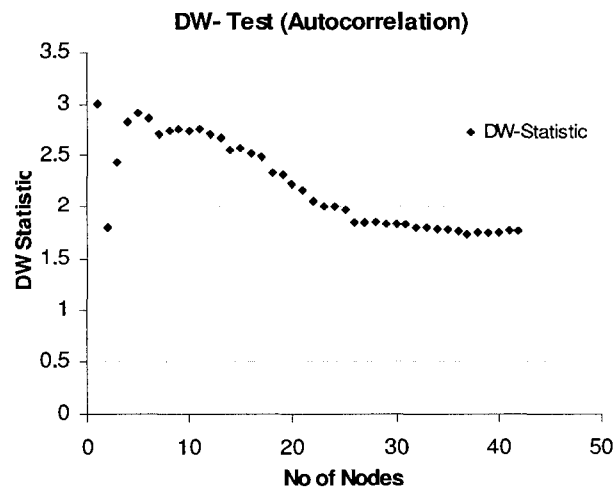


Figure 3.5 The Autocorrelation among system TTF's.

3.7 The Failure Parameters of Various Nodes

Typically, nodes obtained from same manufacturers are assumed to have identical reliabilities. However, as the nodes are put into usage, the failure properties may change. Table 3.5 shows the statistical summary of shape and scale parameters of ASCII White, assuming that TTF's of nodes follow a Weibull distribution. The scale parameters are shown in Figure 3.6 (a), and the MTTF's of individual nodes are shown in Figure 3.6 (b).

Table 3.5 The statistical properties of shape and scale parameters of the TTF's of individual nodes obtained from White.

	Scale Parameter	Shape Parameter
Minimum	0.4659	323.4
Maximum	1.582	5107
Mean	0.8396	2132
Median	0.7923	1884
Standard dev	0.2374	1459

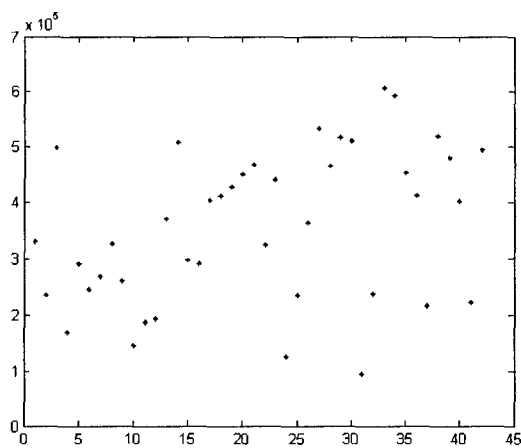


Figure 3.6(a)

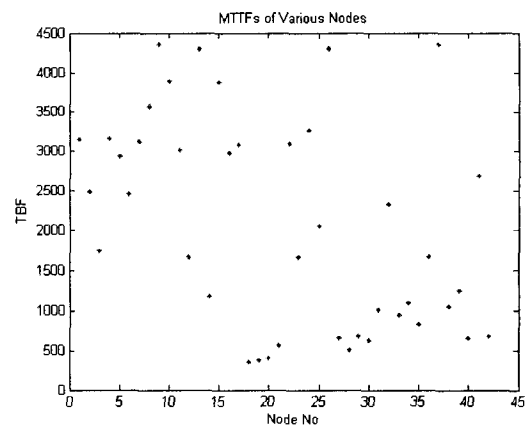


Figure 3.6(b)

Figure 3.6 The scale parameters and MTTF's of various nodes in White.

Figure 3.6(a) shows the scale parameters for different nodes, and Figure 3.6(b) shows the MTTF's for various nodes.

CHAPTER 4

THE DISTRIBUTION OF TIME TO FAILURES

FOR A SYSTEM OF K NODES

Reliability estimation of High Performance Computing (HPC) systems allows resource allocation and fault tolerance frameworks to minimize the performance loss due to unexpected failures. Recent studies have shown that compute nodes in HPC systems follow a time-varying failure rate distribution like the Weibull instead of the exponential distribution. In this chapter, we propose a model for the Time To Failure (TTF) distribution of a system of k independent nodes when individual nodes exhibit time varying failure rates. We also present the system reliability, failure rates, the Mean Time To Failure (MTTF) of the proposed system TTF model and validate the model using the chi-square Goodness Of Fit (GOF) test.

Acronyms

CDF	Cumulative Distribution Function
Pdf	Probability Density Function
TTF	Time To Failure or Time Between Failures
MTTF	Mean Time To Failure
HPC	High Performance Computing
GOF	Goodness-of-fit

K-S Test Kolmogorov Smirnov Test

Notations

t_j j^{th} TTF of a node

$w_i(t_j)$ the Weibull Pdf of node i for the j^{th} failure

$W_i(t_j)$ the Weibull CDF of node i , where $W_i(t_j) = \int_0^{t_j} w_i(\tau) d\tau$

α_i Shape parameter of the i^{th} node

β_i Scale parameter of the i^{th} node

$h_i(t_j)$ the excess Weibull Pdf for node i

$H_i(t_j)$ the excess Weibull CDF, where $H_i(t_j) = \int_0^{t_j} h(\tau) d\tau$

$g(t_j)$ the Pdf of a mixture of excess Weibull's.

$G(t_j)$ the CDF of a mixture of excess Weibull's, where $G(t_j) = \int_0^{t_j} g(\tau) d\tau$

$s_j(x)$ the Pdf of the Time to Failure (TTF) of a system of k nodes after the j^{th} failure

$S_j(x)$ the CDF of the Time to Failure (TTF) system of k nodes after the j^{th} failure,
where $S_j(x) = \int_0^x s_j(\tau) d\tau$

p -value The probability that the sample belongs to a particular distribution (for K-S Test).

4.1 Introduction

Current HPC systems utilize hundreds and thousands of compute nodes simultaneously to solve computationally challenging problems. The parallel tasks of a HPC application simultaneously executes on several nodes, and the failure of a single

compute node may interrupt the entire application. Runtime systems in HPC platforms, like resource managers and checkpoint/restart rely on reliability prediction to minimize the performance loss. For example, reliability-aware checkpoint requires the accurate failure rate of the system to allocate an optimal checkpoint interval. Similarly, resource managers need system reliability information to select resources to provide better quality of service to users.

In HPC systems several individual hardware and software components may affect the failure behavior of the system. Accurate reliability estimation in the presence of multiple failure events is a non-trivial problem. An exponential model is a simple model to analytically obtain the failure rate, reliability, and MTTF of a system of k independent nodes because of its memory-less property [54]. However, several studies on the failures of HPC systems have shown that the failure rate varies over time [53][17][18], A time varying failure rate distribution like the Weibull or gamma typically results in a better GOF. Also, applying a time varying failure rate distribution in check pointing algorithms and reliability-aware resource allocation algorithms [55][56] is observed to minimize performance loss. In this chapter, we develop a TTF distribution model for a system of k nodes when individual nodes have a Weibull distribution. We also give analytical formulae for the system failure rate, MTTF, and reliability for a system of k independent nodes.

In this section, we derive the distribution of TTF for a system of k nodes, for the case when the first node that fails interrupts the entire application, using the first-order statistics approach. Then, we calculate the failure rate and MTTF of the derived system reliability model. We also validate the model TTF distribution using the chi-square GOF

test. Section 4.2 gives the Pdf of the TTF of a system of k nodes, an algorithm to obtain the system TTF from the TTF's of individual nodes, the CDF, failure rate, and expected time to failure of the system. In Section 4.3 we validate the model with the GOF tests, and finally in Section 4.4 we present a numerical example.

Many studies assume that failures of different nodes are independent. However, few studies have shown some dependencies in failures, especially software failures [57]. These dependencies among failures depend on the system configuration and operation environment [17]. In this study we built our model based on the assumption that nodes fail independently. This assumption seems to be true in many cases. The model has merit because cluster computing systems and servers are usually designed to be independent in failures and provide fault tolerance in the event of failures. HPC systems where nodes fail independently are expected to be more reliable.

4.2 TTF Distribution of a System of K Nodes

We make the following assumptions on the failure properties of individual nodes in an HPC system based on our discussion in Chapter 3.

1. Individual nodes are Weibull distributed, but each node may have different shape and scale parameters.
2. The first failure interrupts the entire application, i.e the node's TTF's are in series
3. The TTF's of nodes are statistically independent.
4. After a failure, the node returns to operation at the next time instant.
5. No more than one failure occurs at a single time instant for the system (i.e. only one node fails at a time)

We consider a parallel application on a system of k nodes, where any failure in one of the k nodes interrupts the entire application. In a failure event, the node is renewed back into operation. There may be a downtime associated with the node during recovery operation, but in our study, we omit the downtimes and assume that there are spare nodes, and a node is available immediately after a failure because we are currently interested in modeling the TTF's. The system TTF's are obtained from the TTF's of individual nodes using the algorithm shown in Figure 4.1. The algorithm first calculates the actual failure times for each individual node (n_{ij}) then the system failure times, and finally it calculates the system TTF's as the time between system failures (see Figure 4.2).

Algorithm to determine the system TTF's

1. $i = 1 : k$ // number of nodes
2. $j = 1 : m_i$ // number of failures of i^{th} node
// Let n_{ij} be the actual failure times of i^{th} node
// where the TTF of i^{th} node is $n_{ij} - n_{ij-1}$
3. let $i = 1 : \max(\text{size}(n_{ij}))$
4. $p_1 = 0; t_1 = 0;$
5. $x_1 = p_1 = \min(n_{ij})$ // find the min
6. remove p_1 from n_{ij}
7. while ($n_{ij} \neq \phi$)
8. $p_{i+1} = \min(n_{ij})$ // find the new min
9. remove p_{i+1} from n_{ij}
10. $i = i + 1$
11. $x_i = p_{i+1} - p_{i+2}$ // the system TTF's
12. end while

Figure 4.1 Algorithm to determine the system TTF's from TTF's of individual nodes.

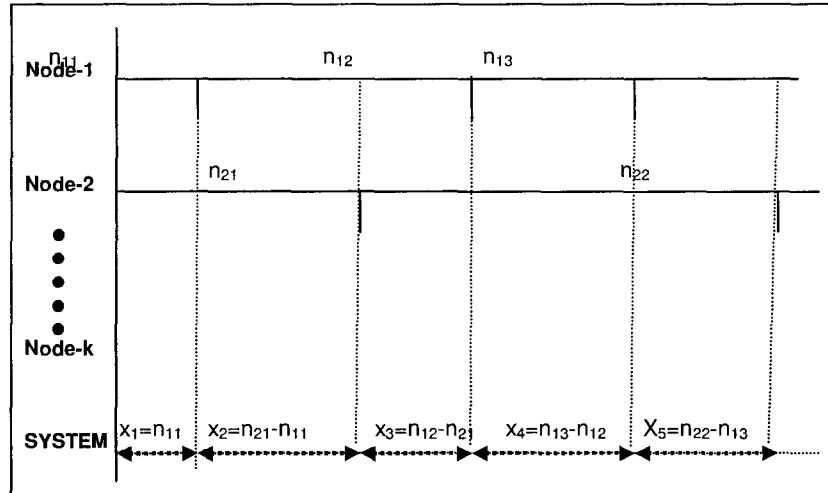


Figure 4.2 Schematic diagram of system TTF's obtained from the TTF's of individual nodes.

Figure 4.2 illustrates a schematic diagram explaining how the TTF's of the system are obtained from the TTF's of individual nodes. Let n_{11} , n_{12} , n_{13} be the failure times of node 1 and n_{21} , n_{22} the failure times of node 2. The system TTF's denoted by x_1 , x_2 , x_3 are obtained from the failure times of the individual nodes using the system TTF algorithm given in Figure 4.1. Now we first obtain the system TTF distribution from the TTF distributions of individual nodes for the example shown in Figure 4.2, and generalize it for the system of k nodes.

The Pdf of a Weibull for an i^{th} node and time between $j-1$ and j^{th} failure (x_j) is given by:

$$w_i(x_j) = \frac{\beta_i}{\alpha_i} \left(\frac{x_j}{\alpha_i} \right)^{\beta_i - 1} e^{-\left(\frac{x_j}{\alpha_i} \right)^{\beta_i}} \quad (4.1)$$

The corresponding CDF is given by

$$W_i(x_j) = 1 - e^{-\left(\frac{x_j}{\alpha_i} \right)^{\beta_i}} .$$

From Figure 4.2, the first TTF of the system x_1 is obtained from Node1 ($x_1=n_{11}$). Since Node1 has a Weibull (from Equation (4.1)), the system TTF x_1 also has a Weibull distribution. If there are k nodes, it is possible that any of the k nodes can have the first failure, therefore the system TTF may belong to any one of the k nodes.

For the case of two nodes, we have

$$s_1(x_1) = w_1(x_1)[1 - W_2(x_1)] + w_2(x_2)[1 - W_1(x_1)]$$

Therefore, for k nodes, the first TTF has the Pdf

$$s_1(x_1) = \sum_{i=1}^k w_i \prod_{\substack{l=1 \\ l \neq i}}^k (1 - W_l) \quad (4.2)$$

After the first failure, the failed node is renewed back into operation, and the second system failure may be due to any of the k nodes failing. The same node (the one that failed first) may fail again, or any of the remaining $(k-1)$ nodes (those that have not had the first failure) can fail. If the same node fails (in our example in Figure 4.2, the 1st node), then the second TTF will have a Weibull, $w_1(x_2)$. For the remaining $(k-1)$ nodes that did not have the first failure, we know that they have survived until the first failure. We define the probability that a node will fail in time 'x', given that the node has survived until time 't' as

$$P(X \leq t + x | t) = H(t + x | t) = 1 - e^{-\frac{t^\beta - (t+x)^\beta}{\alpha^\beta}} \quad (4.3)$$

We denote by $H(\cdot)$ the excess Weibull distribution function, having CDF $H(x+|t)$ and

$$\text{Pdf } h(t+x|t). \text{ where } \frac{\beta}{\alpha^\beta} (t+x)^{\beta-1} e^{-\frac{t^\beta - (t+x)^\beta}{\alpha^\beta}} \quad (4.4)$$

Note that for the shape parameter $\beta = 1$, the Weibull reduces to an exponential distribution and for the excess Weibull $H(\cdot)$, we have $P(X \leq t + x | t) = P(X \leq x)$.

Hence, the second TTF has an excess Weibull distribution if any of the remaining (k-1) nodes fails. Therefore the Pdf of the 2nd TTF 'x₂' can be

(1) Weibull $w_1(x_2)$, if the node that first failed would fail again

or

(2) A mixture of (k-1) excess Weibull's $g(x_2)$.

The Pdf of the second TTF of the system may be obtained as

$$s_2(x_2) = w_1(x_2)[1 - G(x_2)] + g(x_2) [1 - W_1(x_2)], \quad (4.5)$$

where $w_1(x_2)$ is the Pdf, and $W_1(x_2)$ the CDF of the node that has failed previously, and $g(x_2)$ is the Pdf of the mixture of (k-1) nodes that have not failed previously and have an excess Weibull with $G(x_2)$ as the corresponding CDF.

In general, the procedure to obtain the Pdf $g(x_2)$ is presented below.

For 1 node that has not failed previously, we have $g(x_2) = h_1(t_1 + x_2 | t_1)$, where $h_1(t_1 + x_2 | t_1)$ is the distribution of excess life for Node 1 with survival time t_1 and excess life x_2 .

For 2 nodes, we have

$g(x_2) = h_1(t_1 + x_2 | t_1)[1 - H_2(t_1 + x_2 | t_1)] + [1 - H_1(t_1 + x_2 | t_1)]h_2(t_1 + x_2 | t_1)$, where $h_1(t_1 + x_2 | t_1)$ is the Pdf and $H_1(t_1 + x_2 | t_1)$ is the CDF of excess life distribution for Node 1 with survival time t_1 and excess life x_2 , and $h_2(t_1 + x_2 | t_1)$ is the Pdf and $H_2(t_1 + x_2 | t_1)$ is the CDF of excess life distribution for Node 2 with survival time t_1 and excess life x_2 .

Therefore, for (k-1) nodes that have not failed previously, we obtain

$$g(x_j) = \sum_{i=1}^{k-1} [h_i(t_{j-1} + x_j | t_{j-1}) \prod_{\substack{l=1 \\ l \neq j}}^{k-1} [1 - H_l(t_{j-1} + x_j | t_{j-1})]]. \quad (4.6)$$

In Equation (4.6), note that each of the (k-1) nodes may have a different probability density functions. The CDF for the mixture of (k-1) nodes with excess Weibull is given

$$\text{by } G(x_j) = 1 - \prod_{l=1}^{k-1} [1 - H_l(t_{j-1} + x_j | t_{j-1})]. \quad (4.7)$$

The Pdf of the j^{th} TTF for a system when there are k nodes may be written in the general

$$\text{form as } s_j(x_j) = \sum_{i=1}^k f_i \prod_{\substack{l=1 \\ l \neq i}}^k (1 - F_l) \quad (4.8)$$

where $f_i = w_i(x_j)$ if the (j-1)th TTF belongs to the i^{th} node and

$f_i = h_i(t_{j-1} + x_j | t_{j-1})$ if the (j-1)th TTF does not belong to the i^{th} node.

We verify that the function in Equation (4.8) is a Pdf. The proof is provided in the Appendix A, Theorem (1).

The CDF of the system TTF for Equation (4.8) is given by

$$S_j(a) = P(t \leq a) = \int_0^a \sum_{i=1}^k f_i(\tau) \prod_{\substack{l=1 \\ l \neq i}}^k (1 - F_l(\tau)) d\tau. \quad (4.9)$$

The system reliability of the k nodes is given by

$$R_j(a) = 1 - S_j(a) = \prod_{l=1}^k (1 - F_l(a)). \quad (4.10)$$

The proof can be found in Theorem (2) in the Appendix.

The failure rate of the system TTF is given by

$$\lambda_j(x) = \frac{s_j(x)}{1 - \int_0^x s_j(\tau) d\tau}.$$

Therefore,
$$\lambda_j(x) = \frac{\sum_{i=1}^k f_i \prod_{\substack{l=1 \\ l \neq i}}^k (1 - F_l)}{1 - \int_0^x \sum_{i=1}^k f_i(\tau) \prod_{\substack{l=1 \\ l \neq i}}^k (1 - F_l(\tau)) d\tau} . \quad (4.11)$$

The MTTF or the expected time to failure is given by $E(x) = \int_0^{\infty} x[s(x)]dx$,

Substituting $S_j(x_j)$ in $E(x)$, we obtain

$$E(x_j) = \int_0^{\infty} x_j \sum_{i=1}^k f_i \prod_{\substack{l=1 \\ l \neq i}}^k (1 - F_l) dx \quad . \quad (4.12)$$

We present the system TTF model for two special cases,

- (1) The shape parameter $\beta = 1$ for a Weibull distribution, which gives rise to an exponential distribution
- (2) The shape and the scale parameters for all nodes are equal to some values α and β .

Case 1: The shape parameter $\beta = 1$ for a Weibull distribution.

When $\beta = 1$ the CDF of Weibull becomes, $W(x) = 1 - e^{-\left(\frac{x}{\alpha}\right)}$, which is an exponential

distribution. The corresponding Pdf is given by $w(x) = \frac{1}{\alpha} e^{-\left(\frac{x}{\alpha}\right)}$, and the system Pdf

for the i^{th} node is given by $s_j(x_j) = w_i(x_j) = \frac{1}{\alpha_i} e^{-\left(\frac{x}{\alpha_i}\right)}$.

Case 2: The shape parameter α and scale parameter β for all nodes are equal.

That is, $\alpha = \alpha_1 = \alpha_2 = \alpha_3 = \dots = \alpha_k$ and $\beta = \beta_1 = \beta_2 = \beta_3 = \dots = \beta_k$.

The Pdf for the mixture of (k-1) excess Weibull distributed nodes in Equation (4.6) is given by,

$$g(x_j) = (k-1)[1 - H(t_{j-1} + x_j | t_{j-1})]^{k-2} h(t_{j-1} + x_j | t_{j-1}),$$
 and the corresponding CDF

from Equation (4.7) is given by

$$G(x_j) = 1 - [1 - H(t_{j-1} + x_j | t_{j-1})]^{k-1}.$$

Therefore, the Pdf of the system TTF in Equation (4.8) becomes

$$s_j(x_j) = k[f(1-F)^{k-1}], \quad (4.13)$$

where $f = w(x_j)$ if the (j-1)th TTF belongs to the failed node and $f = h(t_{j-1} + x_j | t_{j-1})$

if the (j-1)th TTF does not belong to the node that failed.

For Case 1, f in Equation (4.13) is exponential $\frac{1}{\alpha} e^{-\frac{x}{\alpha}}$ for all nodes.

4.3 Goodness Of Fit Tests

We adopted a simulation based approach to validate the model using the chi-square GOF test. First, we derived the expression to generate random variables for an excess Weibull model for a given survival time 't'. The expression for generating TTF's that follow an excess life distribution is given by,

$$H^{-1}(t + x | t) = (t^\beta - \alpha^\beta \cdot \ln(1 - U))^{1/\beta} - t. \quad (4.14)$$

Where t is the survival time, x is the system TTF or the excess life, α is the scale parameter and β is the shape parameter and U is the uniform random variable. The proof for Equation (4.14) is given in Theorem 3 in Appendix A. We use the Chi-square GOF test to validate the model. The Chi-square test statistic if given as

$$\chi^2 = \sum_{i=1}^m \frac{(O_i - E_i)^2}{E_i},$$

where m is the number of time intervals, O_i the observed number of failures in the i^{th} interval, $E_i = np_i$ the expected number of failures in the i^{th} interval, n the total number at risk (sample size) assuming that the TTF's are not correlated, and p_i the probability of a failure occurring in the i^{th} interval, where $p_i = S_j(a_i) - S_j(a_{i-1})$, and

$$S_j(a_i) = \int_0^{a_i} h_1(t+x|t)[1-H_2(t+x|t)] + h_2(t+x|t)[1-H_1(t+x|t)] dx \quad (4.15)$$

$$\text{The expected number of failures in the } i^{\text{th}} \text{ interval is } n * [S_j(a_i) - S_j(a_{i-1})] \quad (4.16)$$

We use the following algorithm to calculate the system TTF's

Simulation Algorithm:

1. Generate TTF for two nodes with same alpha and beta using Equation (4.14) for a sample size ($N=200$) assuming $\alpha = 1000$ and $\beta = 0.8$.
2. Draw a random observation from node-1 and node-2, the system TTF is the $\min(\text{TTF of node-1, TTF of node-2})$.
3. Repeat the above process 100 times to obtain 100 TTF's of the system, group them into 'm' intervals and perform the chi-square test.

The hypothesis we want to test is formulated below:

Null hypothesis H0: The system TTF has the specified distribution given in Equation (4.15)

Alternate hypothesis H1: The data does not fit the specified distribution

Figure 4.3 shows the comparison of observed number of failures, expected number of failures calculated from Equation (4.16), and chi-square values for various TTF intervals. In calculating the expected value in Figure 4.3 from Equation (4.16) α and β are estimated from the sample of generated TTF's. As such the degrees of freedom for the test statistic is $m-c-1=6-3=3$, where m is the number of intervals and c is the number of estimated parameters. The value of the test statistic is $\chi^2 = 1.365$. The critical value χ_{α}^2 for a level of significance $\alpha = 0.05$ and 3 degrees of freedom is given by $\chi_{0.05,3}^2 = 7.814$. We observe that $1.365 < \chi_{0.05,3}^2$. Therefore, we fail to reject H_0 and conclude that our model is consistent with the data. The GOF test shows 95 percent confidence that the data follows the distribution specified in Equation(4.15).

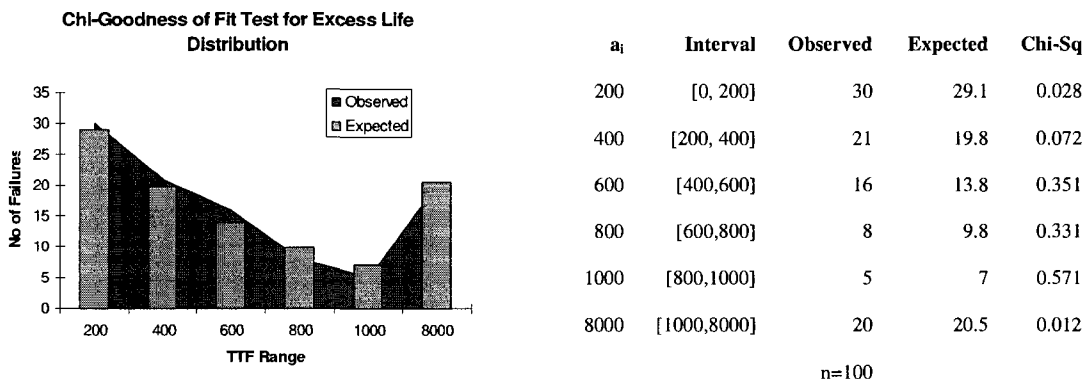


Figure 4.3 The Chi-Square GOF tests comparing the actual and expected number of failures.

4.4 Numerical Example

To illustrate the TTF distribution model, CDF, reliability and failure rate for a system of k nodes we give a numerical example. We chose the scale parameters $\alpha_1 = \alpha_2 = \alpha_3 = 1542$ and shape parameters $\beta_1 = \beta_2 = \beta_3 = 0.8606$ for three nodes. The

failure times (the actual wall clock times that the nodes failed) for the three nodes are shown in Table 4.1. The calculation of CDF, reliability and failure rate are calculated for various cases illustrated in Figure 4.4, Figure 4.5, Figure 4.6, and Figure 4.7. We also discuss the effect of system reliability, failure rate and MTTF with the increasing number of nodes. Figure 4.4 shows the failure times of the three nodes when the system started functioning at time $t=0$. Here, the job is submitted at time $t=0$ where the job running time is 100 Hrs.

Table 4.1 The failure times of three nodes

Node-1	Node-2	Node-3
1667	1114	1503
6472	2347	3875
10273	3358	3933
10587	4264	5610
12643	4662	5917
12722	7592	5925
14945	13018	6203
17201	13719	6262
17202	13779	15143
17957	13800	17812

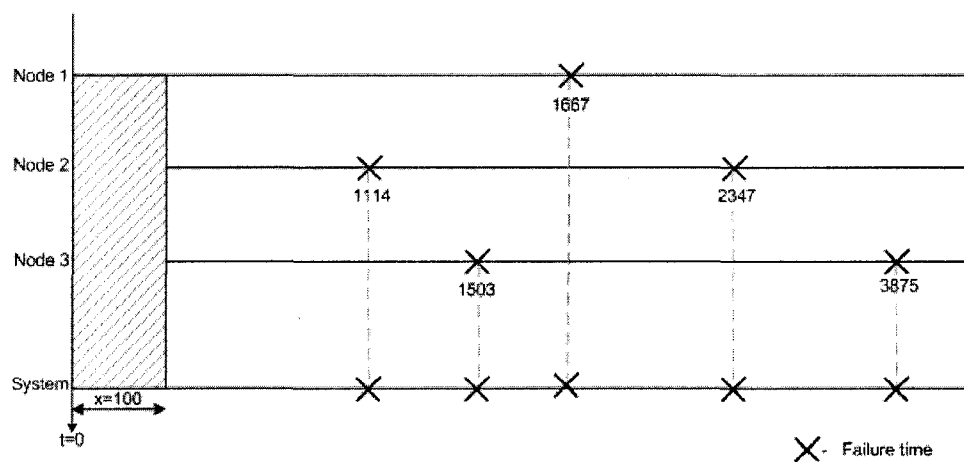


Figure 4.4. Job submitted at ($x=100$, $t=0$)

The system reliability (From Equation 4.10),

$$R_{sys}(100) = \prod_{i=1}^3 \left(1 - e^{-\left(\frac{0^{0.8606} - (0+100)^{0.8606}}{1542^{0.8606}} \right)} \right) = 0.7521$$

The system CDF (From Equation 4.9 and Equation 4.10) =

$$F_{sys}(100) = 1 - R_{sys}(100) = 0.2479$$

The system failure rate (From Equation 4.11),

$$\lambda_{sys}(100) = \frac{3 * \left(\frac{0.8606}{1542^{0.8606}} (0+100)^{0.8606-1} e^{-\frac{0^{0.8606} - (0+100)^{0.8606}}{1542^{0.7}}} \left(1 - e^{-\left(\frac{0^{0.8606} - (0+100)^{0.8606}}{1542^{0.8606}} \right)} \right) \right)}{\prod_{i=1}^3 \left(1 - e^{-\left(\frac{0^{0.8606} - (0+100)^{0.8606}}{1542^{0.8606}} \right)} \right)} = 0.0025$$

The system MTTF (From Equation 4.12),

$$E(100) = \int_{100}^{\infty} 3 * \left(\frac{\beta}{\alpha^{\beta}} (t+x)^{\beta-1} e^{-\frac{t^{\beta} - (t+x)^{\beta}}{\alpha^{\beta}}} \right) dx = 464.4902$$

The system MTTF is integrated using the Composite Simpson's approximation function

[54]. The Matlab [54] programs to calculate the above are given in Appendix 2.

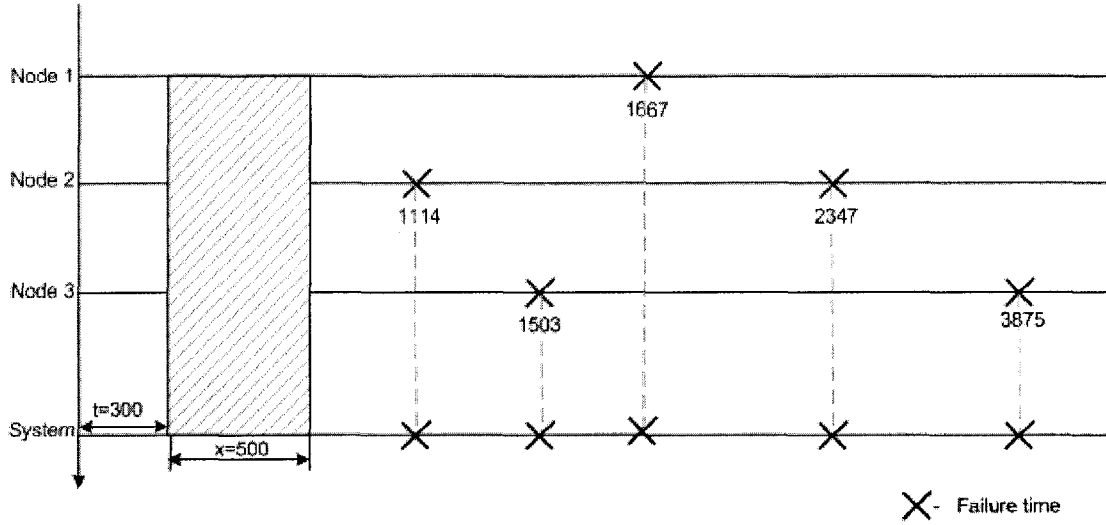


Figure 4.5 Job submitted at $(x=500, t=300)$

Figure 4.5 shows the case when the system was surviving until time $t=300$, and a job is with execution time $x=500$ is submitted at t . Here t is the survival time, and x is the excess life or the job running time. We show the calculation of reliability, CDF and failure rate for this case.

The system reliability (From Equation 4.10),

$$R_{sys}(500) = \prod_{i=1}^3 \left(1 - e^{-\left(\frac{300^{0.8606} - (0+500)^{0.8606}}{1542^{0.8606}} \right)} \right) = 0.3782$$

The system CDF (From Equation 4.10 and 4.9) = $F_{sys}(500) = 1 - R_{sys}(500) = 0.6218$

The system failure rate (From Equation 4.11),

$$\lambda_{sys}(500) = \frac{3 * \left(\frac{0.8606}{1542^{0.8606}} (300 + 500)^{0.8606-1} e^{-\frac{300^{0.8606} - (300+500)^{0.8606}}{1542^{0.8606}}} \left(1 - e^{-\left(\frac{300^{0.8606} - (300+500)^{0.8606}}{1542^{0.8606}} \right)} \right) \right)}{\prod_{i=1}^3 \left(1 - e^{-\left(\frac{300^{0.8606} - (300+500)^{0.8606}}{1542^{0.8606}} \right)} \right)} = 0.0018$$

The system MTTF (From Equation 4.12),

$$E(500) = \int_{500}^{\infty} 3 * \left(\frac{\beta}{\alpha^{\beta}} (t+x)^{\beta-1} e^{-\frac{t^{\beta}-(t+x)^{\beta}}{\alpha^{\beta}}} \right) dx = 536.8430$$

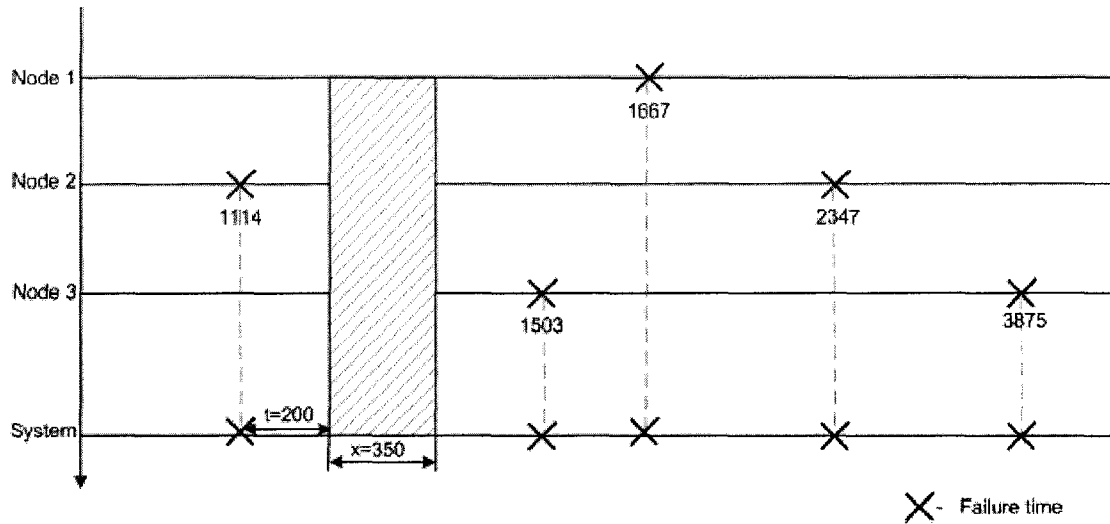


Figure 4.6 Job submitted at $t=200$, $x=350$ when the system failed at 1114 Hrs

In Figure 4.6 we show the job submission when the job running time is $x=350$, the system survived until time $t=200$ after the system failed at 1114 Hrs

The system reliability (From Equation 4.10),

$$R_{sys}(350) = \prod_{i=1}^3 \left(1 - e^{-\left(\frac{200^{0.8606} - (200+350)^{0.8606}}{1542^{0.8606}} \right)} \right) = 0.4877$$

The system CDF (From Equation 4.10 and 4.9), $F_{sys}(350) = 1 - R_{sys}(350) = 0.5123$

The system failure rate (From Equation 4.11),

$$\lambda_{sys}(350) = \frac{3 * \left(\frac{0.8606}{1542^{0.8606}} (200+350)^{0.8606-1} e^{-\frac{200^{0.8606} - (200+350)^{0.8606}}{1542^{0.8606}}} \left(1 - e^{-\left(\frac{200^{0.8606} - (200+350)^{0.8606}}{1542^{0.8606}} \right)} \right) \right)}{\prod_{i=1}^3 \left(1 - e^{-\left(\frac{200^{0.8606} - (200+350)^{0.8606}}{1542^{0.8606}} \right)} \right)} = 0.0019$$

The system MTTF (From Equation 4.12),

$$E(350) = \int_{350}^{\infty} 3 * \left(\frac{\beta}{\alpha^{\beta}} (t+x)^{\beta-1} e^{-\frac{t^{\beta} - (t+x)^{\beta}}{\alpha^{\beta}}} \right) dx = 522.4005$$

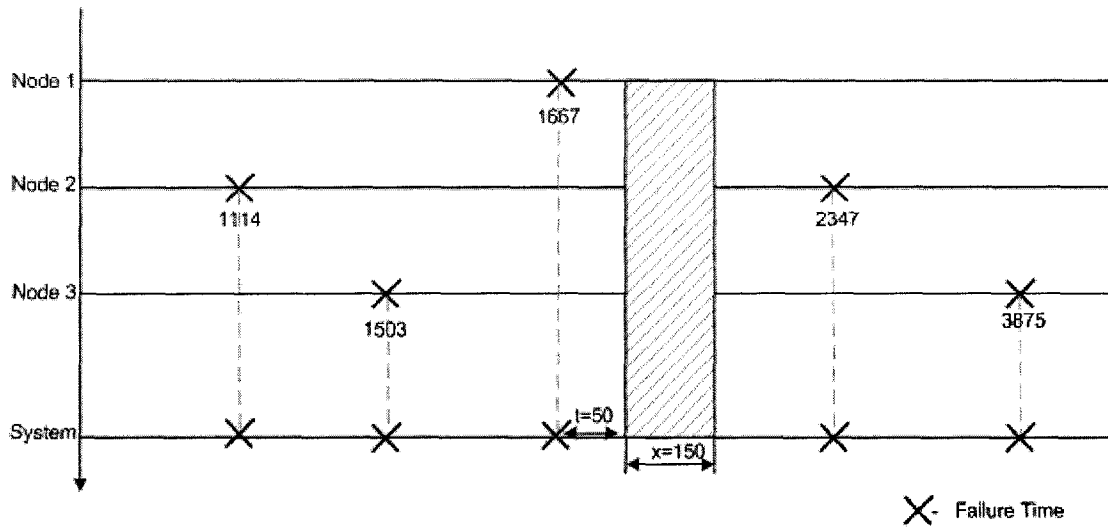


Figure 4.7 Job submitted at $t=50$, $x=150$ when the system failed at 1167 Hrs

In Figure 4.7, the system starts functioning at time $t=50$. We want to calculate the system reliability metrics for the future time $x=150$.

The system reliability (From Equation 4.10),

$$R_{sys}(150) = \prod_{i=1}^3 \left(1 - e^{-\left(\frac{50^{0.8606} - (50+150)^{0.8606}}{1542^{0.8606}} \right)} \right) = 0.6974$$

The system CDF (From Equation 4.10 and 4.9), $F_{sys}(150) = 1 - R_{sys}(150) = 0.3026$

The system failure rate (From Equation 4.11),

$$\lambda_{\text{sys}}(100) = \frac{3 * \left(\frac{0.8606}{1542^{0.8606}} (50+150)^{0.8606-1} e^{\frac{50^{0.8606} - (50+150)^{0.8606}}{1542^{0.7}}} \left(1 - e^{\left(\frac{50^{0.8606} - (50+150)^{0.8606}}{1542^{0.8606}} \right)} \right) \right)}{\prod_{i=1}^3 \left(1 - e^{\left(\frac{50^{0.8606} - (50+150)^{0.8606}}{1542^{0.8606}} \right)} \right)} = 0.0022$$

The system MTTF (From Equation 4.12),

$$E(100) = \int_{150}^{\infty} 3 * \left(\frac{\beta}{\alpha^{\beta}} (t+x)^{\beta-1} e^{\frac{t^{\beta} - (t+x)^{\beta}}{\alpha^{\beta}}} \right) dx = 489.5752$$

Parallel applications are allocated to a system of k nodes. Therefore, it is important to study the system reliability failure rate and MTTF with increasing the number of k nodes.

We use the same example where $\alpha_i = 1542$, $\beta_i = 0.8606$ and vary i from to 50 nodes.

Figure 4.8(a) shows the system failure rate for various values of 'k'. We observe that increasing the number of nodes increases the failure rate. Figure 4.8(b) shows the system MTTF with the increase in number of nodes. We observe that increasing the number of nodes decreases the MTTF. Figure 4.9 shows the effect of system reliability with the increase in the number of nodes. We observe that the system reliability decreases with the increase in number of nodes. We study Reliability-Aware optimal k node allocation in more detail in Chapter 6.

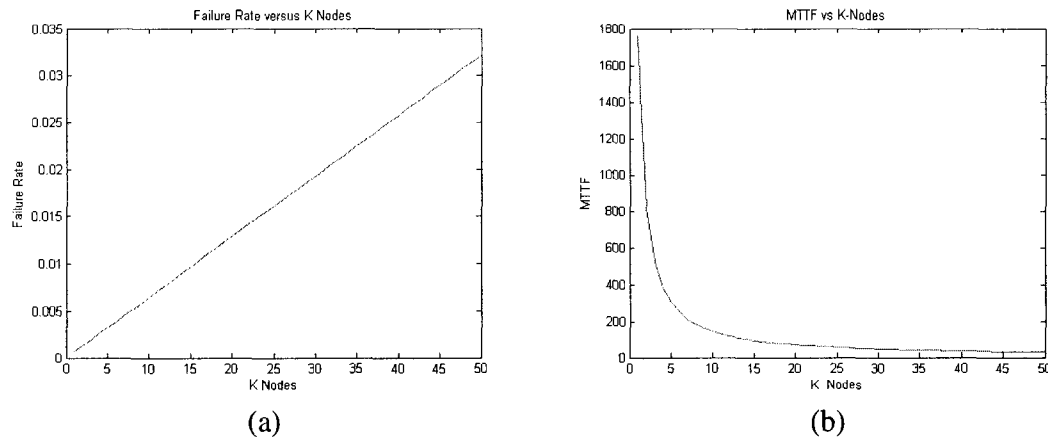


Figure 4.8. The effect of failure rate and MTTF with the increase in number of nodes (k)

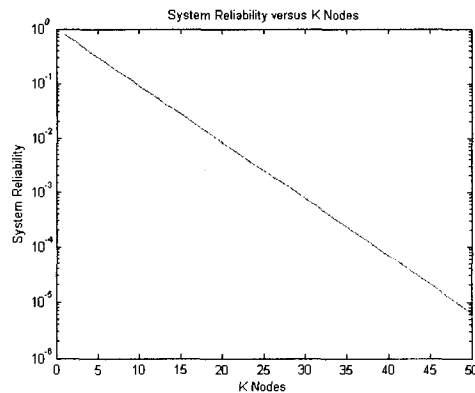


Figure 4.9 The effect of system reliability with the increase in the number of nodes

4.5 Conclusion

Estimating system reliability of a parallel program among a system of k nodes is a challenging problem when there are multiple components, with different failure behaviors. When components are independent and have a constant failure rate (exponential model), estimating the k node system reliability is simple because there is no dependency on the previous failure time. However, recent studies on HPC system suggest that individual compute nodes follow a time varying distribution like the Weibull [16], [17], [18], [19], [20] instead of the exponential. In this chapter, we developed a TTF distribution model for a system of k nodes for a parallel program, where individual nodes

follow a Weibull distribution. Based on the distribution function of the proposed model, we developed the algorithm to obtain the system TTF distributions, the reliability of a system of k nodes, the failure rate, and the MTTF. We also validated the model using a simulation based approach. The proposed model exhibits more accuracy than previous models in the literature since it considers the excess life (future running time given that the nodes survived until time ' t ') in order to estimate system reliability.

CHAPTER 5

RELIABILITY-AWARE RESOURCE ALLOCATION IN HPC SYSTEMS

In this chapter, we propose a reliability-aware resource allocation model for parallel programs based on a time-varying distribution and present reliability-aware resource allocation algorithms to minimize the performance loss due to failures. We also study the effectiveness of the proposed allocation algorithms based on the actual failure logs and parallel workloads. The failure data are obtained from the 512 node ASCI White system from Lawrence Livermore National Laboratory (LLNL), and the parallel workloads are obtained from Los Alamos National Laboratory (LANL) and San Diego Supercomputer Center (SDSC).

5.1 Introduction

Typically, parallel applications like MPI have multiple processes that concurrently run on several nodes. MPI processes running on these nodes intercommunicate by passing messages between the processes that span several nodes. Unfortunately, a failure on any nodes running these processes can cause the overall application outage and requires restarting the entire application. The computation time lost for a parallel program due to failures is called the waste time. An ideal model would predict the exact time and event of failure, which would enable the applications to

checkpoint right before the failure or a resource manager to avoid allocating the job to a node that may fail in future. However, current failure prediction techniques are still unrealistic to be applied in real systems.

We study stochastic techniques to determine the reliability of computation nodes and apply heuristics based on workload properties to minimize the overall waste time. We present reliability-aware resource management algorithms and compare with the existing techniques.. To understand the effectiveness of the proposed reliability-aware algorithms, we use the production parallel workloads available at [15] on failure data of ASC white. We first discuss the reliability functions for estimating the reliability of a parallel job. Then, we establish a reliability model for job allocation, discuss reliability-aware resource allocation algorithms, waste time metrics, and the effect of waste time metrics for various job types.

The rest of the chapter is organized as follows. Section 5.2 introduces the time varying reliability function, reliability-aware resource allocation model, and reliability-aware resource allocation algorithms for parallel programs. Section 5.3 describes the simulation framework for reliability prediction; parallel workloads used, and waste time of various reliability-aware resource allocation algorithms for resource allocation. Section 5.4 finally summarizes the contributions.

5.2 Reliability Model for a Parallel Application

5.2.1 Reliability of Job Completion Time

Let x denote the job runtime, and t the time since the most recent failure. (t is basically the failure-free runtime or survival time of a particular node). The reliability of job run time, conditioned on the failure free runtime t is given by

$$R(t + x | t) = \frac{R(t + x)}{R(t)} \quad (5.1)$$

Where, $R(t + x)$ is the reliability during the time $(t+x)$ and $R(t)$ is the reliability of the failure free running time. We use exponential and one of the time varying CDF to develop reliability cost functions. We choose Weibull because it is relatively easier to develop cost function for estimating the reliability of job completion time as compared to gamma and lognormal.

If we apply the above Equation (5.1) for the CDF of Weibull:

$$R(t + x | t) = \frac{e^{-\left(\frac{t+x}{\alpha}\right)^\beta}}{e^{-\left(\frac{t}{\alpha}\right)^\beta}} = e^{-\left(\left(\frac{t+x}{\alpha}\right)^\beta - \left(\frac{t}{\alpha}\right)^\beta\right)} \quad (5.2)$$

Applying Equation (5.1) for the CDF of Exponential

$$R(t + x | t) = \frac{e^{-\lambda(t+x)}}{e^{-\lambda t}} = e^{-\lambda x} = R(x) \quad (5.3)$$

Exponential model is well-known, and we observe from Equation (5.3) that for exponential distribution there is no memory on previous failure, and the reliability of job run time $R(x)$ is the same during the life time of the node. It does not matter when the previous failure happens. But in the case of Weibull from Equation (5.2), the reliability of job completion time decreases as a job is submitted away from the most recent failure (for shape parameter $m < 1$). To determine how a distribution performs in selecting reliable nodes to avoid failures and minimizes waste time, we can compare the reliability-aware resource allocation algorithms that apply reliability functions on the actual failure data and parallel job workloads.

5.2.2 Reliability Model for a Parallel Application

For a parallel program like MPI, a single node failure interrupts the entire program running on all k processors. In this reliability model, we assume that nodes have identical processing times, and nodes are statistically independent [25]. If x is the job runtime and $R_1(t_1 + x | t_1)$ is the reliability of node₁, where t_1 is its failure free running time, $R_2(t_2 + x | t_2)$ is the reliability of node₂, where t_2 is its failure-free running time and $R_k(t_k + x | t_k)$ is the reliability of node_k, where t_k is its failure free running time. Because node failures are independent from another, and the entire parallel program can be interrupted if any of the k nodes fail, the system reliability model is a series connection. The system reliability for a parallel program allocated on k nodes is therefore given by

$$R_{sys} = \prod_{i=1}^k R_i(t_i + x | t_i) \quad (5.4)$$

Note that the above reliability function is discussed in Chapter 4 may also be obtained from Equation (4.9) and the proof is given in Theorem (2) in the Appendix.

We define a heuristic of Reliability-aware Scheduling (RAS) Algorithm as follows:

“For a job that requires k out of n available processors, the job is allocated to k adjacent processors and nodes such that R_{sys} is maximized.”

5.2.3 Reliability-Aware Resource Allocation Algorithms

The RAS algorithm is given in Figure 5.1. For an exponential RAS algorithm (EXP) R_i is obtained from Equation (5.3). Exponential reliability algorithm requires the failures rate λ of each processor and the jobs run time. For a Weibull RAS algorithm (WEIB) R_i is obtained from Equation (5.2). Weibull reliability algorithm requires the shape (α) and scale (β) parameter values of each processor, the time since previous

failure (t), and the job running time (x). All the three parameter values (λ, m, c) are obtained from the failure history of each node.

The following performance metrics were considered to evaluate the reliability algorithms.

%MWT (Mean Waste Time): The ratio of waste time to the actual run length of the job.

Total Waste Time: The total waste time for each category of job. The category of jobs form a workloads are either the run lengths or the number of processors

The total waste time per category gives the amount of waste time for various categories of jobs. The %MWT gives a more normalized metric on the percentage of time lost relative to the job run length.

RAS (Reliability-aware Scheduling) Algorithm

```

N : Maximum Number of Jobs
M : Maximum Number of Processors
1: for ( each job ready in job queue J (i=1 to N))
      processors Idle (j=1 to M)
2:   select the first job (Ji) in the queue
3:   if ( job (Ji.no_of_procs) available==true)
4:     select k adjacent reliable processors
       from nodes out of M such that Rsys is
       maximized
5:   endif
6:   if any of the selected k processors failed
7:     requeue the job at the head of job queue
8:   endif
9: endfor
Note: Rsys for Reliability-aware algorithm is computed using
Equation (5.4)

```

Figure 5.1. The Reliability-Aware Scheduling Algorithm

5.2.4 A Study of Waste Time for a Parallel Program

Figure 5.2 and Figure 5.3 shows the effect of MWT with respect to job run length and number of processors. In the ideal case we want to avoid allocating job to nodes that would fail, but present mechanisms disallow prediction of the exact failure event and time. An accurate reliability prediction function should, however, minimize the failures and waste time with reliability-aware resource allocation policies. There could be several factors that affect the waste time in the presence of failures like the arrival sequence of jobs, type of jobs, and availability of resources.

To understand the effectiveness of a reliability prediction function we first compare how individual jobs are affected due to waste time. Figure 5.2(a) and Figure 5.2(b) illustrate the workloads categorized by the number of processors and Figure 5.2 (c) and Figure 5.2(d) show the comparison of total waste time with respect to number of processors when an individual job is allocated using three policies. We can observe that %MWT is minimal with reliability-aware resource allocation for jobs requiring different number of processors. When the number of processors required by a job increase, the failure probability also increases. Thus, the %MWT is higher for jobs that require more number of processors. In Figure 5.2 (c), for LANL workload there are 6 jobs (about 0.00005% of the entire jobs) that require 1024 processors and for SDSC in Figure 5.2 (d) there are 0.00002% of the entire jobs that require 2048 nodes. Because the percentage of the jobs that required many processors is smaller, we chose not to plot these values. The effect of waste time due to job run lengths is shown in Figure 5.3(b). The graphs in Figure 5.3(c) and Figure 5.3(d) show that the waste time increases when jobs have longer

job run lengths; however, the graphs do not show clear trends for much longer jobs because of the difference in the percentage of jobs with different run lengths.

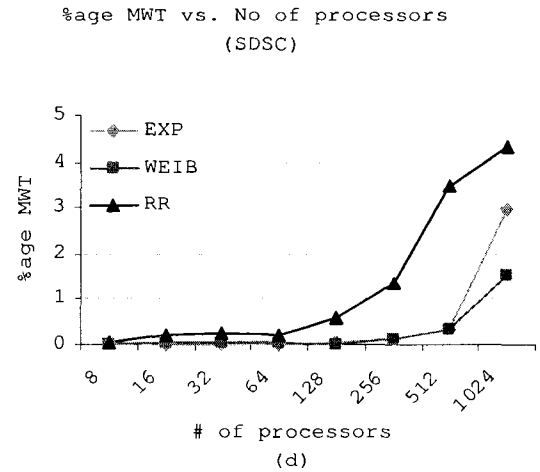
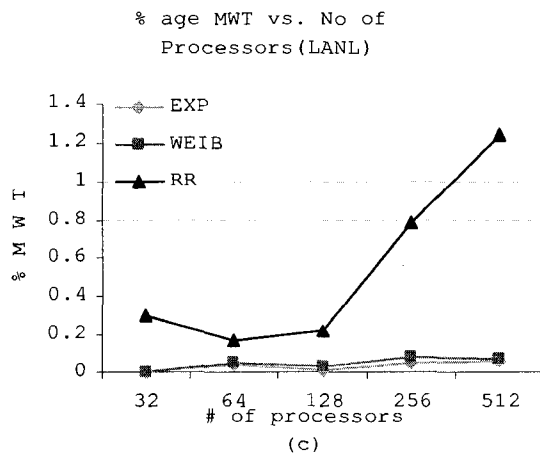
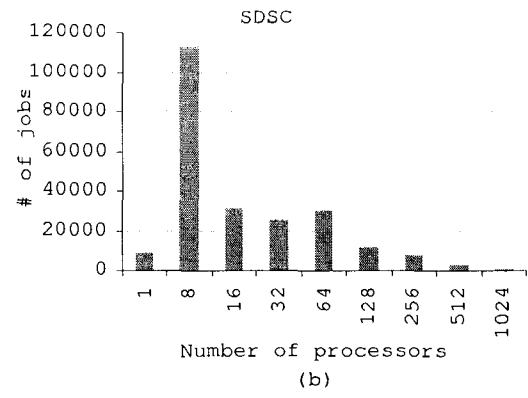
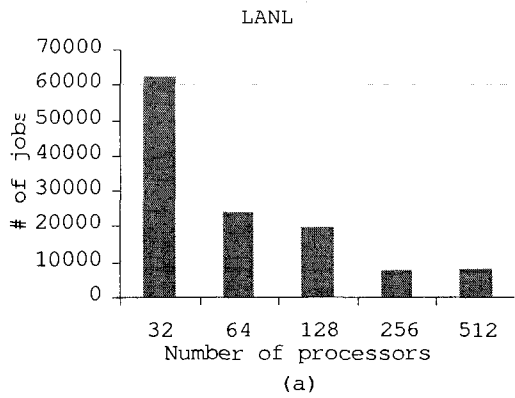
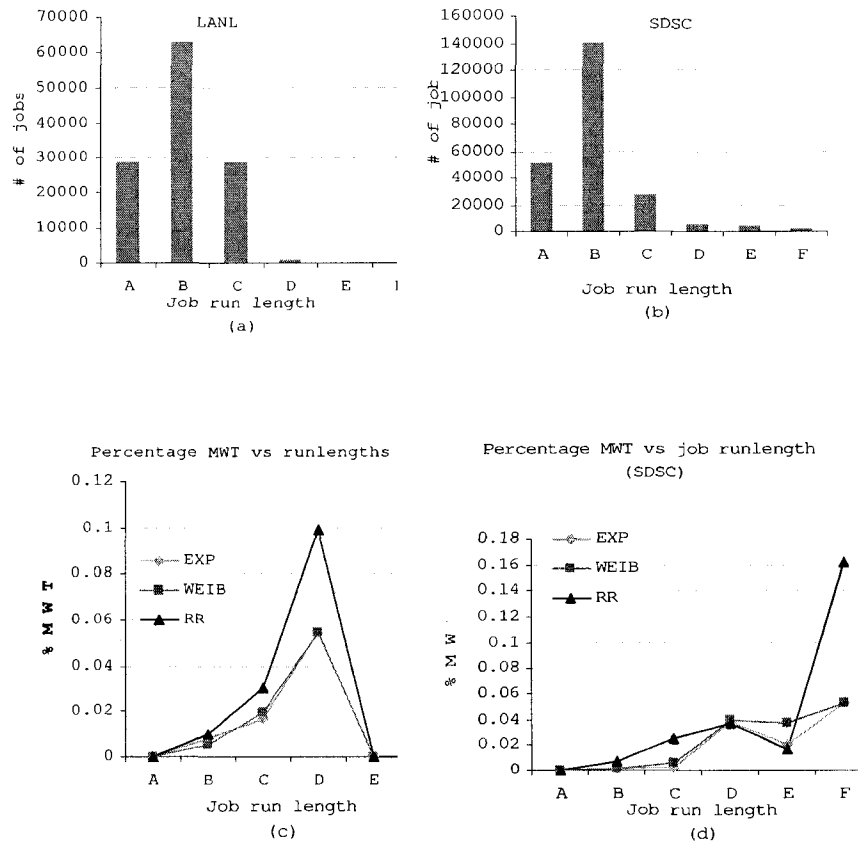


Figure 5.2 Description of workloads and comparison of Waste Times for reliability-aware policies for individual jobs requiring different number of processors.



A	B	C	D	E	F
<1Min	1Min<X≤1Hr	1Hr<X≤6Hrs	6Hrs<X≤12Hrs	12Hrs<X≤24Hrs	>1Day

Figure 5.3 Description of workloads and comparison of Waste Times for reliability-aware policies for individual jobs requiring different job run lengths

5.2.5 Heuristics for Reliability-Aware Resource Allocation

In an actual job scheduling scenario, we have a mixture of jobs based on arrival sequence with different run lengths, and number of processors. We explore the following heuristics that can aim to minimize the overall waste time:

- (A) Resubmit the job immediately after the failure

- (B) Allocate longer jobs to reliable processors
- (C) Allocate shorter jobs to unreliable processors

(A) Resubmit the job immediately after the failure

A job should not be penalized for a resource failure by queuing it at the end of the job queue. Hence, a job that has failed is immediately scheduled on the most reliable node.

(B) Allocate longer jobs to reliable processor

During a job execution, the waste time depends on what point of runtime the failure occurred. If a job fails just before completion, it has higher waste time as compared to when it fails in the beginning due to rollback time. In general, longer jobs are more likely to have more waste time as compared with shorter jobs, thus allocating longer jobs to more reliable processor reducing the failure chances for longer jobs, thereby minimizing the overall waste time.

(C) Allocate shorter jobs to unreliable processors

If both short and long jobs are allocated to reliable processors, the additional (or new) longer jobs coming later in the queue would get assigned to less reliable processors in the system, therefore allocating shorter jobs to less reliable nodes makes reservations for longer jobs. Because allocating shorter jobs to unreliable processors has less waste time as compared to longer jobs, the overall waste time can be reduced.

With the above mentioned heuristics, we develop an algorithm called the LJRAS (Longest Jobs Reliable Aware Allocation) that is given in Figure 5.4. The algorithm basically allocates longer jobs to reliable nodes, shorter jobs to unreliable nodes and resubmits the job immediately after failure. The reliability-aware simulation framework,

workloads and discussion on comparison of reliability-aware resource allocation techniques are discussed in the Section 5.3.

LJRAS (Longest Job Reliability-aware Scheduling) Algorithm

```

N : Maximum Number of Jobs
M : Maximum Number of Processors
1: for(each job ready in job queue J (i=1to N))
    && Each processor Idle(j=1 to M)
2:   select the first job (Ji) in the queue
3:   if ( job (Ji.no_of_procs) available==true)
4:     if ji.run_length > 1day
5:       select k adjacent reliable processors
       from nodes out of M such that Rsys is
       maximized
6:     else
7:       select k out of M adjacent unreliable
       processors such that Rsys is
       minimum
8:     endif
9:   endif
10:   if any of the k processors failed
11:     requeue the job at the head of job
       queue
12:   endif
13: endfor
Note: Rsys for Reliability-aware algorithm is computed using
Equation (5.4)

```

Figure 5.4 The Longest Job Reliability-Aware Scheduling Algorithm

5.3 Comparison of Reliability-Aware

Resource Allocation Algorithms

In this section we discuss the simulation study of applying various resource allocation algorithms namely, RAS, LJRAS and round robin with the parallel jobs on the actual failure data, and parallel workloads. We compared the waste time metrics for reliability-aware resource allocation algorithms with exponential, Weibull and round-robin techniques.

5.3.1 Simulation Study

We simulate the 8196 processor cluster from the ASC White failure logs and run parallel workloads in the presence of failures. The system failure logs (number of processors, their failure trace which include uptime and downtime) and parallel job workloads trace (job submit time, running time, and number of processors) are inputs to the simulator. Each parallel job requires a certain number of processors, and each job runs on not more than one processor at the same time. The parallel job continues to run until either the job is completed, or any node has failed. Jobs are scheduled based on First Come First Serve policy. If the job is failed, it is given highest priority.

For reliability-aware resource allocation, the simulator uses the most recently available failure parameters obtained from nodes in order to obtain the reliability of a parallel job. Failure rate, shape and scale parameters are evaluated using the MLE (Maximum Likelihood Evaluator) and updated periodically every 1000 minutes.

5.3.2 Parallel Workloads

The workload for the ASC white was unavailable. We chose two workloads from [53] that are significant on terms of jobs that required more number of processors, jobs, longer run lengths, and has more number of jobs. LANL workload is from a 1024 processor system and has 122,060 jobs, and SDSC BLUE workload is from an 1152 processor system with 243,314 jobs. We use the initial 6 months failure data of nodes from ASCI White to calculate the failure parameter values, we therefore assume jobs start executing at 250000 minutes. The number of processors in the failure data is 8196. Using the actual submit times would make most of the processors idle. Therefore to utilize the

system to maximum and compare the waste time for various techniques, we assume that all jobs are available at the same time.

5.3.3 Comparison of Reliability Prediction

We present the comparison results of among exponential RAS, exponential LJRAS, Weibull RAS, and Weibull LJRAS allocation policies and round robin (RR) technique. A round robin algorithm (RR) selects k adjacent processors in the order of the node numbers. If the last node number is reached, the RR algorithm starts selecting nodes beginning with the first node.

We first study the overall waste time and then discuss the waste time metrics with respect to various run length of jobs. The total waste time of each technique for the two workloads is shown in Figure 5.5. We observe that (1) reliability-aware techniques based on two variations, exponential and Weibull result in lesser waste time, and (2) allocating longer jobs on reliable processors and shorter jobs on less reliable processors further reduces the waste time. For SDSC workloads, the total waste time by RAS is reduced as much as 10percent and as much as 33.8 percent with LJRAS technique.

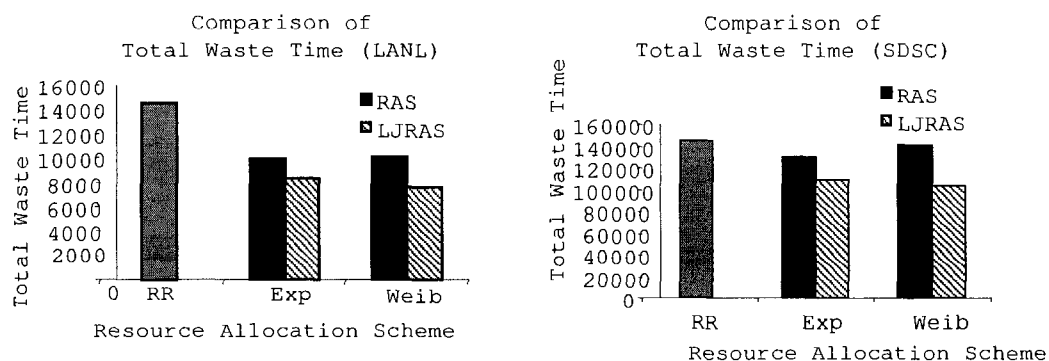


Figure 5.5 The comparison of overall waste times for the two workloads namely LANL and SDSC.

In the case of LANL, the waste time is reduced with RAS by as much as 30 percent and by LJRAS by 53 percent. The longer the jobs are, the more likely they will encounter failures during execution. Therefore, longer jobs have less reliability and more waste time as compared to shorter jobs. In RAS algorithm, allocating short jobs to reliable nodes makes the reliable nodes unavailable to longer jobs. The LJRAS takes advantage of the fact that shorter jobs have less failure probability, hence shorter jobs may be allocated to less reliable processors and thus reliable processors are reserved to longer jobs.

In general, the waste time is affected by three factors namely (1) the time of failure (2) the job run length and (3) the availability of reliable processors. For longer jobs, Weibull based reliability function performs better than exponential and exponential performs better than round robin. For example, the total waste time for LANL in Figure 5.6 (a), and total waste time for SDSC in Figure 5.6 (b) shows that for longer jobs (category D,E and F), the Weibull-based LJRAS performs as good as or better than the exponential-based LJRAS.

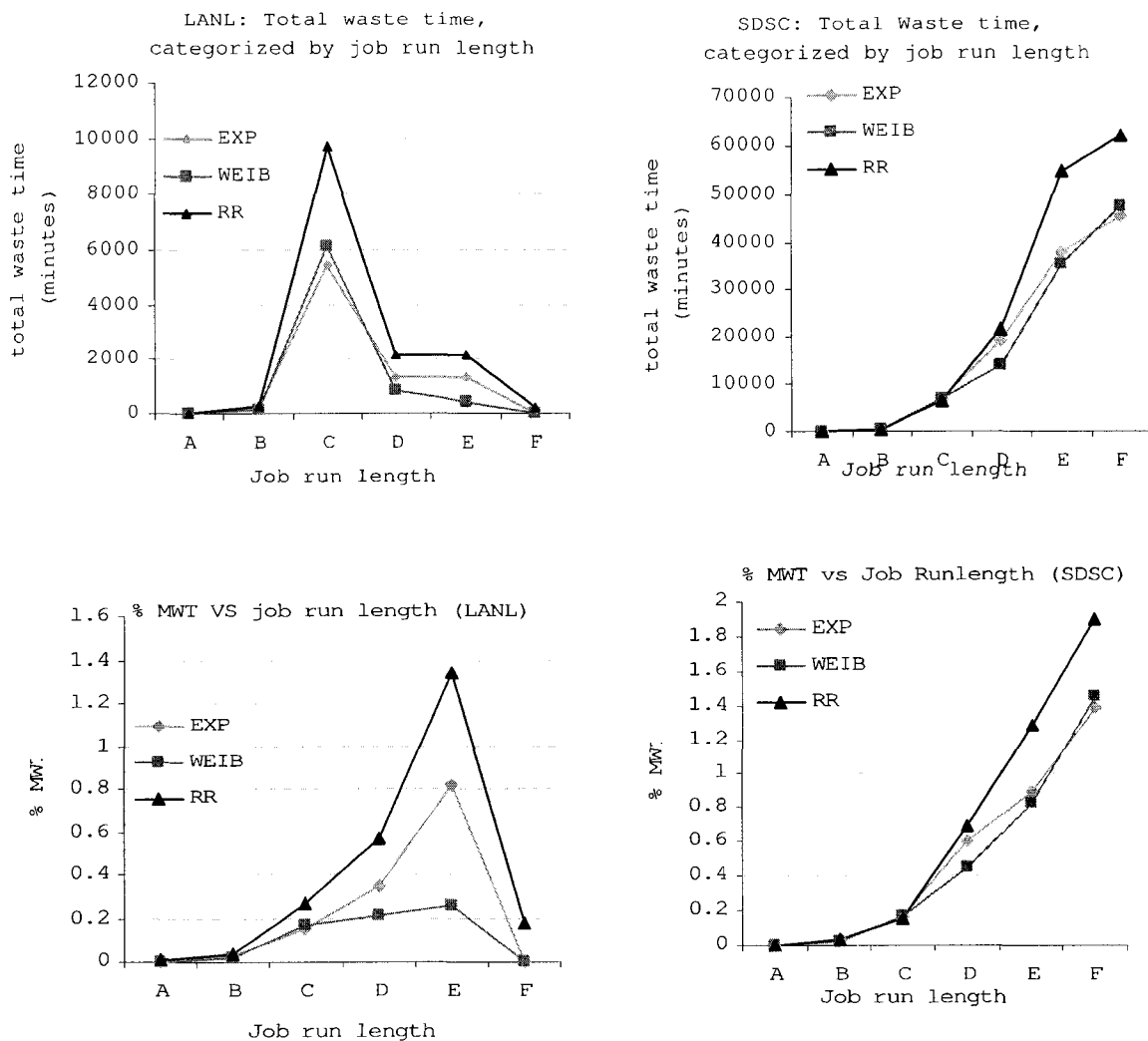


Figure 5.6 Description of LANL and SDSC workloads based on run lengths and comparison of Total Waste Time and % average Waste Time for each run lengths of jobs.

In our experiments, the failure data belongs to a Weibull population, which is one of the reasons why it performs better than exponential. A more accurate reliability prediction function may perform better than Weibull. With reliability-aware resource allocation, the waste time is also affected by several factors like the arrival rates, and system utilization. For instance, if arrival rate or system utilization is smaller, with reliability-aware allocation, most of the unreliable nodes are left idle and very few jobs

fail. Incorporating reliability-aware resource management in the presence of failures is a hard problem both because of the unpredictability of workload properties and failure behavior of HPC systems. We observed different distribution of job sizes in the workloads and failure properties of nodes to develop some heuristics to minimize the overall waste time.

5.4 Conclusions

Failures and downtimes are a growing concern for large- scale HPC systems. Future HPC systems require an integrated RAS framework [7] for providing reliability information of resources for runtime mechanisms such as checkpoint managers and resource managers. In this chapter, we proposed a time varying failure rate based reliability model for parallel applications and evaluated effectiveness of a reliability-aware resource allocation as compared to round-robin for exponential and Weibull reliability functions with two policies namely Reliability-Aware Scheduling (RAS) and Longest Job based Reliability-Aware scheduling (LJRAS). Our results indicate that LJRAS technique with Weibull function minimizes the waste time, and performs better than or equal to exponential in most cases.

CHAPTER 6

RELIABILITY-AWARE OPTIMAL K NODE ALLOCATION OF PARALLEL APPLICATIONS

In an ideal case, scalability by increasing the number of nodes would reduce the completion time. However, some important system factors limit the scalability to a certain threshold to achieve better performance[59][60][61][62]. Current and future parallel applications demand significant computing resources, and thus system reliability becomes a major scalability issue. In this chapter, we first study the effect of job completion time with respect to scalability. Our findings suggest that in the presence of failures, increasing the number of nodes for a parallel application would start to increase the completion time after a certain threshold. In addition, there is an optimal number of nodes the parallel application can scale, and the overall completion time can be minimized. Based on this observation, we propose a reliability-aware optimal k-node allocation algorithm and compare with existing resource allocation algorithms.

6.1 Introduction

There are two types of scaling, namely weak and strict. Strict scaling involves increasing the processor count to reduce the completion time. With weak scaling, the processor count is increased proportional with the input (i.e. throughput computing). We

study the performance aspect relevant to job completion time (strict scaling). Reliability has been mentioned as an important challenge for large scale computational applications. In Chapter 5 we observed that reliability-aware resource allocation can improve the performance loss. Though reliability decreases with increasing number of nodes, reliability-aware resource allocation in the context of scalability has not been given much attention. We also observed in Section 3.7, Chapter 3 that individual nodes may possess different reliabilities over time. Figure 6.1 shows the effect of system reliability from ASCI White with the increase in number of nodes. We observe that the system reliability decreases with the increase in the number of nodes.

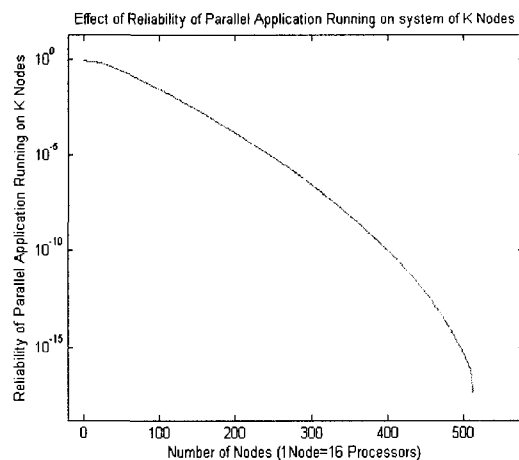


Figure 6.1 Effect of reliability with the increase in number of nodes.

Considering reliability as an important performance metric for resource managers, enables to develop heuristics that minimize waste time and reduce the job completion times. In this chapter, we study how reliability affects job completion time while scaling up the number of nodes and propose a reliability-aware optimal k node allocation algorithm based on the expected completion time of a parallel program.

The rest of the chapter is organized as follows. Section 6.2 discusses the expected completion time of a job without failures. The effect of reliability as an important performance and scalability metric is discussed in Section 6.3. Section 6.4 describes the reliability-aware optimal k node selection algorithm and compares it with other resource allocation algorithms for various types of jobs. Section 6.5 discusses the conclusions and future work.

6.2 The Expected Completion Time of a Parallel Program

To estimate the completion time of a parallel program in the presence of failures, we first derive the expected completion time on k nodes. Figure 6.2 shows the completion time of a parallel program when there are failures and repairs. The actual completion time $T_{c(k)}$ is an estimated running time of a parallel program on k nodes when there are no failures. In the event of a failure, the un-checkpointed parallel program running on set of k nodes is interrupted and has to be restarted from the beginning. The time until the failure which is wasted, is called the waste time (w_{ki}). The expected waste time is the MTTF of the given set of k nodes that we denote by M . The time the application takes to restart from the i^{th} failure is called the repair time r_{ki} . We denote the expected recovery time by R . The probability of system failure (F_k) Equation (4.9) in Chapter 4.

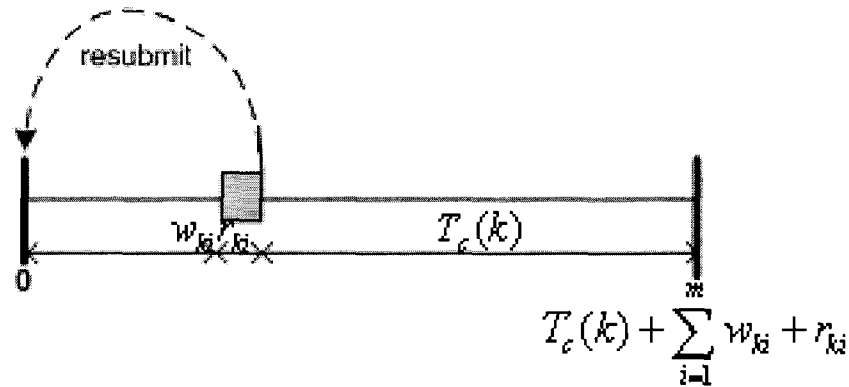


Figure 6.2 An un checkpointed parallel application with failures.

Theorem: The Expected Completion time of a parallel program is given by

$$E(T_{c(k)}) = T_{c(k)} + (M + R) \left[\frac{F_k}{1 - F_k} \right]. \quad (6.1)$$

Where $T_{c(k)}$ is the completion time of a job on k nodes without failures, M is the expected waste time due to failures, R is the recovery time, and F_k is the failure probability of the system.

Proof:

The expected completion time in the presence of multiple failures and repairs is given by:

$$E[T_{c(k)}] = (1 - F_{k1})T_{c(k)} + F_{k1}[w_{k1} + r_{k1} + (1 - F_{k2})T_{c(k)} + F_{k2}[w_{k2} + r_{k2} + (1 - F_{k3})T_{c(k3)} + \dots$$

In the above equation, F_{k1} is the failure probability of k nodes selected first time the job is allocated. If the job fails, the job may be allocated to a different set of k -nodes where the failure probability is F_{k2} and so on. Since the job is not checkpointed after a failure the job has to be restarted from the beginning. Therefore $T_{c(k)}$ still remains the same and has reliability of completion $(1 - F_{k2})$. For a Weibull-based distribution function the failure probability may change over time. For simplicity, we consider a special case where

$F_{ki}=F_k$ we assume average system reliability for all the nodes. Similarly, the average waste time for k nodes (M) and the average repair time for k nodes R . Therefore,

let $F_k = F_{k1} = F_{k2} = F_{k3} = \dots F_{km}$, $M = M_{k1} = M_{k2} = M_{k3} = \dots M_{km}$ and

$R = w_{k1} = w_{k2} = w_{k3} = \dots w_{km}$ for 'm' failures.

Therefore,

$$E[T_{c(k)}] = (1 - F_k)T_{c(k)} + F_k[M + R + (1 - F_k)T_{c(k)}] + F_k[M + R + (1 - F_k)T_{c(k)}] + \dots$$

We can rewrite the above equation as follows:

$$\begin{aligned} E[T_{c(k)}] &= (1 - F_k)T_{c(k)} + FM + FR + F(1 - F)T_{c(k)} + F^2M + F^2R + F^2(1 - F)T_{c(k)} \\ &\quad + F^3M + F^3R + F^3(1 - F)T_{c(k)} + \dots \end{aligned}$$

After factoring out M , R and $T_{c(k)}$ the above equation becomes

$$\begin{aligned} E[T_{c(k)}] &= (1 - F_k)T_{c(k)} + T_{c(k)}(F(1 - F) + F^2(1 - F) + F^3(1 - F) + \dots) \\ &\quad + M(F + F^2 + F^3 + \dots) + R(F + F^2 + F^3 + \dots) \end{aligned}$$

We have from geometric series that $\sum_{i=0}^{\infty} r^i = \frac{1}{1 - r}$.

Therefore, $\sum_{i=1}^{\infty} r^i = \frac{1}{1 - r} - 1 = \frac{r}{1 - r}$.

Using the above sum, we obtain, $E[T_{c(k)}] = (1 - F_k)T_{c(k)} \left[\frac{1}{1 - F_k} \right] + (M + R) \left[\frac{1}{1 - F_k} \right]$,

This implies $E[T_{c(k)}] = T_{c(k)} + (M + R) \left[\frac{F_k}{1 - F_k} \right]$.

The expected completion time considering the reliability of nodes on different scale systems is shown in Figure 6.3. We observe that the expected completion time increases after a reliability threshold is reached. The expected completion times for

different shape and scale parameters are shown in Figure 6.3. Figure 6.3 (a) shows the expected completion time for different values of shape parameter b .

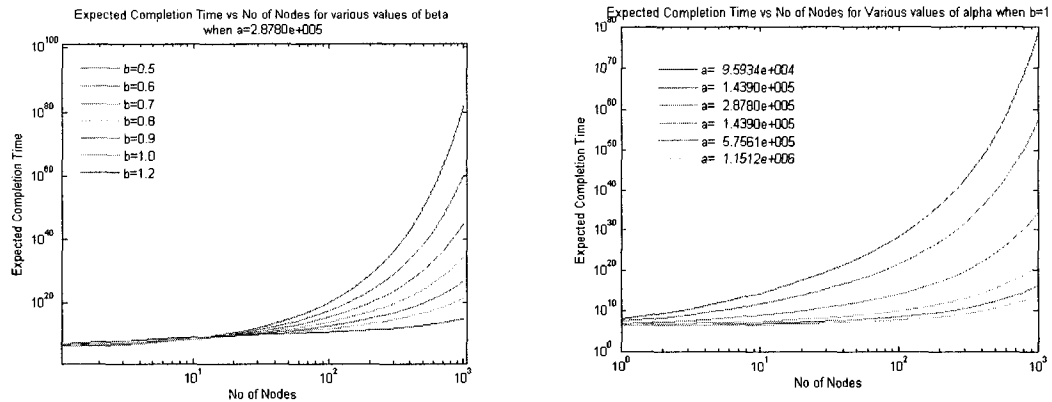


Figure 6.3 The expected completion times for various values of the shape and scale parameters for a Weibull distribution.

We observe that increasing the number of nodes increases the expected completion time, and smaller values of beta have higher expected completion time in the case of large number of nodes. Figure 6.3(b) shows the expected completion time for various number of nodes for different values of the scale parameters a . We also note that for different values of the scale parameter a , increasing the number of nodes increases completion time. Furthermore, lower the value of alpha, the higher is the expected completion time.

6.3 Performance and Scalability of Parallel Programs

In this section, we discuss the expected completion time with various scalability models. Amdahl's law and Gustafson's law are some of the widely discussed scalability models with respect to the completion time of a job.

6.3.1 Amdahl's Law

According to Amdahl's Law, the maximum achievable speedup is limited by the serial part of the application. The "speedup" of a parallel program is defined as the ratio of the rate that a job is running on k processors to the rate at which the same job is executed on one node [49]. The speedup $S(k)$ is given by

$$S(k) = \frac{1}{\frac{p}{k} + (1-p)} \quad \text{and} \quad T_{c(k)} = \frac{T_{c(1)}}{S(k)}$$

,where p is the fraction of code that can be parallelized and $1-p$ is the code that has to be executed sequentially and k is the total number of nodes. $T_{c(k)}$ is the estimated completion time on k nodes and $T_{c(1)}$ is the expected completion time on one node. Amdahl's law gives an upper bound on the amount of scalability that can be achieved for a program with a certain degree of parallelism. The expected completion time is therefore given by

$$E[T_{c(k)}] = T_c(k) * \left(\frac{p}{k} + (1-p) \right) + (M + R) \left[\frac{F_k}{1 - F_k} \right]. \quad (6.2)$$

We present an example program with an execution time of 10^5 hrs to study the scalability effect with and without reliability. Figure 6.4(a) shows that the job completion time tends to decrease in the beginning when increasing the number of nodes. However, after a certain point the job completion time remains constant because the improvement in scalability is negligible. Figure 6.5(a) shows the comparison of expected job completion time with respect to the number of nodes with and without reliability. When the reliability of nodes is considered, we observe that the expected completion time decreases in the beginning but starts to increase after a certain point.

6.3.2 Gustafson's Law

According to Gustafson's Law, the time needed to execute the serial fraction of the program may be overlapped with some other operations, unlike Amdahl's law, which imposes a restriction that the sequential part of the program is completely disjointed to the parallel counterpart. The scalability of According to Gustafson's Law [37] the speedup and the completion time are given by:

$$S(k) = (1 - p) + k * p$$

$$T_{c(k)} = \frac{T_c(1)}{S(k)}$$

The expected completion time with Gustafson's model is given by:

$$E[T_{c(k)}] = \frac{T_{c(1)}}{(1 - p) + k * p} + (M + R) \left[\frac{F_k}{1 - F_k} \right] \quad (6.3)$$

The example program with an execution time of 105 Hrs is used to study the effect of scalability for Gustafson's model, with and without reliability. Figure 6.5(b) shows the completion times of parallel application with three degrees of scalability. We observe that the actual completion time decreases linearly on a log-log scale. Also, Figure 6.5(b) shows that the expected completion time by considering reliability decreases in the beginning, but starts to increase after a certain point.

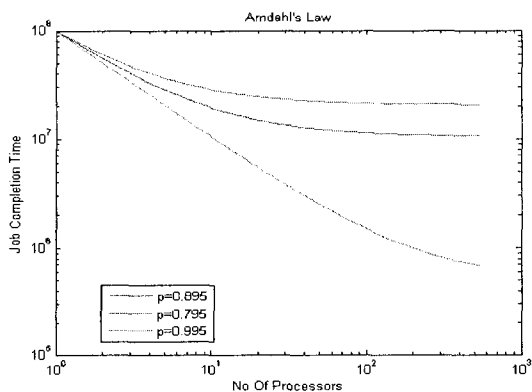


Figure 6.4 (a) Amdahl's Law

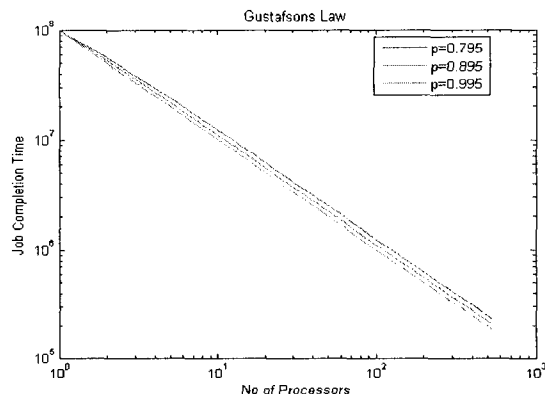


Figure 6.4(b) Gustafson's Law

Figure 6.4 The scalability effect with respect to job completion time for different performance models (Amdahl's Law and Gustafson's Law)

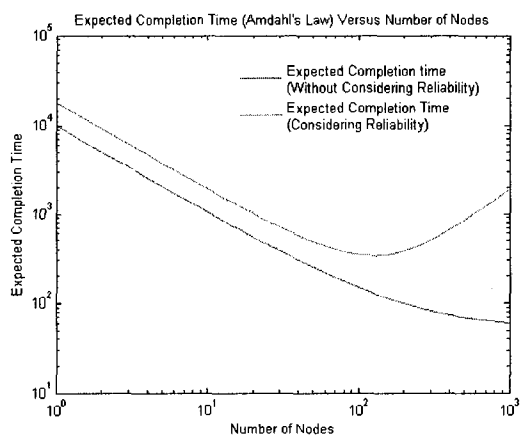


Figure 6.5(a)

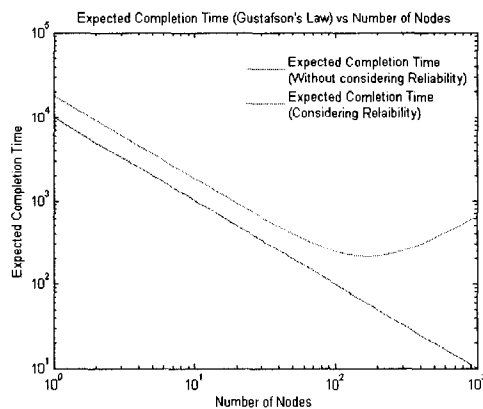


Figure 6.5(b)

Figure 6.5 The expected completion times of parallel programs considering reliability for Amdahl's Law (a) and Gustafson's law (b).

6.4 Reliability-Aware Resource Allocation

To study the effect of reliability in selecting the optimal number of k nodes, we consider reliability and job completion time as important metrics for space sharing. In this section, we discuss existing relevant resource allocation algorithms, the optimal k

node allocation algorithm and compare these algorithms by simulating the resource allocation of parallel jobs on the generated failure data representing ASCI White system, and synthetic workload representing the parallel workload properties.

6.4.1 Resource Allocation Algorithms

The main objective of the resource allocation algorithm is to select the optimal number of nodes that results in minimal overall completion time of a parallel application. A node may contain more than one processors. In our study, we assume a node has a single processor, but the resource allocation may be extended to an SMP system. Therefore, allocating a job to a node means allocating to a processor and vice-versa.

6.4.1.1 All Nodes (ALL)

This technique selects all the available nodes in the system. In the absence of failures, selecting all the nodes gives the minimum job completion time. However, if any of the allocated node fails before the job is completed, it would be resubmitted to all the nodes after the node is up.

6.4.1.2 Round Robin Allocation (RR)

The Round Robin allocation technique allocates the job to k' adjacent nodes based on the round-robin policy of node ids. When the last node number is reached, nodes are allocated beginning from the first node id. The k' number of processors required for the parallel application (which means k' node) is given by the user and the value of k' does not change. The RR policy does not take into account the node reliabilities or job run-lengths before allocating the job.

6.4.1.3 Reliability-aware Allocation (RA)

Here, the k' number of nodes for a job is given by the user, and the k' value is fixed. The algorithm selects the k' most reliable nodes for every job. Reliability-aware resource allocation of parallel applications [55] reduce the overall waste time as compared to Round-Robin and has been discussed in Chapter 5. The reliabilities are calculated using a Weibull distribution and the system reliability for k nodes is given in Equation (5.4) from Chapter 5.

For simulation, we randomly generate the workloads with the number of processors required by the user's applications. Several studies have shown that the number of processors selected by the user follows a two-phase log uniform distribution. We generate the number of processors for each job using the two-phase log uniform distribution. Further discussion on workload and failure data is given in section 6.4.3.

6.4.2 Reliability-Aware Optimal K Node Allocation

In an HPC system the reliability of each individual node is calculated based on the failure parameters obtained from the failure history of the nodes. Each node may have different reliability, and an optimal k node allocation algorithm considers the following three factors.

(A) The Number of Processors

Increasing the number of processors reduces job completion time. On the other hand, increasing the number of nodes will also increase the failure probability. This requires resubmission of jobs which increases the overall completion times and waste times.

(B) Job Run-length

Longer jobs have more chances to encounter failures, as compared with shorter jobs. Therefore, for a given set of nodes, longer jobs may have more waste time as compared with shorter jobs.

(C) Reliability of k nodes

The reliability of a selected node affects the chances that the node will fail in future. Also, failures increase the waste time and completion time of a job.

We define the scheduling problem may be as follows:

“Given a parallel application, and a HPC system that contains m nodes $n_1, n_2, n_3 \dots n_m$ with reliabilities $r_1, r_2, r_3 \dots r_m$, find k out of m nodes such that the overall completion time is minimized”

Figure 6.6 gives the algorithm for selecting optimal k out of the ‘ m ’ nodes in the system. The expected completion time on k nodes $ECTime(k,j)$ in the algorithm is calculated using Equation (6.1), where k is the number of nodes and j is the job run length on a single node. It may not possible to accurately estimate the completion time on a given k nodes (i.e $T_c(k)$ in Equation (6.1)); therefore we use some standard scalability/performance models discussed in Section 6.3 to study the effectiveness of k node allocation algorithm. The job completion time on k nodes without considering reliability may be calculated from one of the performance models (In Equation 6.2 or 6.3). The RA-Opt algorithm basically calculates the reliability of each individual node and incrementally allocates k -nodes such that the expected completion time is minimal.

Algorithm: Optimal K-Node allocation

```

1.  $k=0$  // The number of nodes selected
    $N[J]$  //contains all the node id's
    $J$  //contains the job runlength
2. for  $i=1:\max(\text{size}(N))$  nodes
3.     calculate  $R(i)$  //Equation (6.2)
4. end
5.  $M[J] = \text{sortdesc}(N,R)$  //sort the nodes based on descending order
6.     //of reliabilities
7.  $KNList.add(M(1))$ ; //add the node with highest reliability
8. while ( $ECTime(k+1,J) \leq ECTime(k,J)$ ) /Equation (6.3)
9.      $k=k+1$ 
10.     $KNList.add(M(k+1))$ 
11. end while
12.  $opt\_k = k$ 
13. allocate application to  $KNList$  nodes
    • Allocate ( $k$ ) Nodes to the Application

```

Figure 6.6 The optimal k node allocation algorithm.

Figure 6.7(a) shows a sample case where the optimal number of k nodes is selected based on the minimum expected completion time. The expected completion time decreases further and starts to increase at a certain point when it reached an optimal number of processors that are allocated for a given job. We show the results from the selection of k nodes randomly and RA-Opt algorithm based on the job run-lengths in

Figure 6.7(b). We observe that the number of nodes selected increases proportionally with the job run-length. The effectiveness of various resource allocation techniques with k node selection can be seen only when compared with completion times and waste times. The simulation results of applying various resource allocation techniques are discussed in Section 6.4.5.

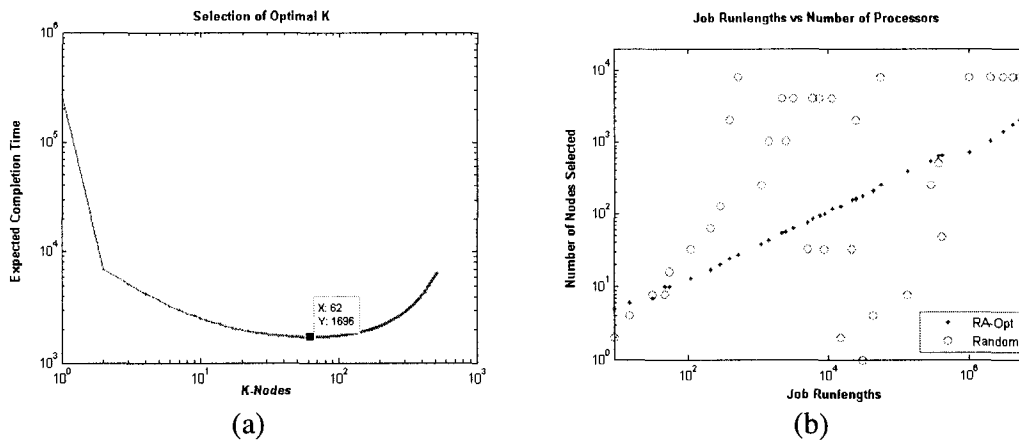


Figure 6.7 The Optimal k node selection by the algorithm. (a) shows the optimal k nodes at a point where the expected completion time is minimum and (b) shows the comparison of optimal k node selection vs. k nodes selected randomly.

6.4.3 Numerical Example

We demonstrate the Reliability-Aware optimal k node algorithm using an example shown in Table 6.1. The first column k is the number of selected nodes, $\lambda(t)$ is the failure rate of an individual nodes (given in Equation (4.11)). The second column s(t)

is the speedup factor $S(k) = \frac{1}{\frac{p}{k} + (1-p)}$, and $T_{c(k)} = \frac{T_{c(1)}}{S(k)}$ where $T_c(k)$ is the running

time on k nodes, which is discussed in Section 6.3.1. In this example, we assume the amount of code that can be parallelized, $p=0.895$ and k is the number of nodes. $R_{sys}(t)$ is the system reliability of k nodes, M is the MTTF. EC(t) is the expected completion time on k nodes.

Table 6.1 A Numerical Example showing the expected completion time of a parallel program on k nodes

k	$S(t)$	$T_{c(k)}$	$R_{SYS}(t)$	M	$E(T_{c(k)})$
1	1	1000	0.971864	3.50E+04	2014.406
2	1.809955	552.5	0.968956	1.75E+04	1113.804
3	2.479339	403.3333	0.966057	1.17E+04	813.7115
4	3.041825	328.75	0.954172	7.01E+03	665.333
5	3.521127	284	0.944844	5.01E+03	576.211
6	3.934426	254.1667	0.930032	3.50E+03	517.7785
7	4.294479	232.8571	0.899131	2.19E+03	478.5445
8	4.610951	216.875	0.861962	1.46E+03	450.6858
9	4.891304	204.4444	0.820061	1.03E+03	430.5775
10	5.141388	194.5	0.774655	7.62E+02	416.0872
11	5.365854	186.3636	0.734563	6.04E+02	404.6705
12	5.568445	179.5833	0.684369	4.74E+02	397.9672
13	5.752212	173.8462	0.633531	3.81E+02	394.1615
14	5.919662	168.9286	0.582774	3.13E+02	392.9122
15	6.072874	164.6667	0.513084	2.47E+02	398.8412
16	6.213592	160.9375	0.436384	1.94E+02	411.6036
17	6.343284	157.6471	0.367344	1.57E+02	428.7588
18	6.463196	154.7222	0.306048	1.31E+02	451.0254
19	6.574394	152.1053	0.252351	1.10E+02	479.3888
20	6.677796	149.75	0.209033	9.57E+01	511.7624

In the Table 6.1 we can observe that the expected running time initially decreases with the increase in number of nodes, and starts to increase at a certain point as the system reliability decreases. The Reliability-Aware optimal k node algorithm selects k nodes such that the expected completion time is minimum (k=14 in our case).

6.4.4 Simulation Study

The system failure logs and parallel job workloads are inputs to the simulator. Each job has a *job id*, *job run-length*, and *number of processors* required for the job. The failure logs have *node ids*, *failure times*, *down times*, and *reliability* of the nodes. We simulate a 10,000 node system using the failure properties of compute nodes obtained from ASCI White system logs. The processing times of each node are identical, however

the reliability of individual nodes may differ. We generate the failure data for the 10,000 system by using the ASCI White failure properties.

In this study we are interested in a large scale system. However, the parallel workloads available at [15] are not suitable for our purpose. Therefore, a synthetic workload was generated using the distribution of the number of processors and the job run-lengths [59]. We use the uniform-log distribution to generate the number of processors, and two stage hyper exponential distribution to generate job run-lengths [53][62][63][64]. In addition to the actual workload, we also injected some jobs with very long run-lengths to test the effectiveness of our techniques.

6.4.5 Performance Metrics

We consider the following performance metrics in our study:

Average Completion Time (ACT) is the ratio of the total completion time of a particular category of jobs to the total number of jobs.

Average Waste time (AWT) is the ratio of the total waste time to the total number of jobs.

Mean Completion Time (MCT) is the ratio of the total completion time to the unit job run-length (unit job run-length = job-run-length/number of processors)

Mean waste time (MWT) is the ratio of the total waste time to the unit job run-length.

Relative Percentage Difference RPD ($= 100 * \left(\frac{T_1 - T_0}{T_0} \right)$), where T_0 is the performance metric for the most optimal technique and T_1 is the performance metric for one of the three compared techniques (RR, RA or ALL). The positive value of percentage difference gives the percentage of improvement of T_0 over T_1 and a negative value indicates the percentage improvement of T_1 over T_0 .

6.4.6 Experimental Results

We assume that the MWT and MCT are affected by the number of processors, the job run-lengths and reliability of the selected k' nodes. The All-nodes technique selects all the nodes, which in an ideal case, should reduce the total completion time, but increases the system failure probability. Since failures may happen multiple times as any one of the nodes fail, the MCT and MWT is higher for the All-nodes technique as shown in (Figure 6.8(a) and Figure 6.8(c)). The RR technique allocates nodes based on round-robin policy, and no reliability is considered, therefore the MWT and MCT are higher. The RA technique allocates the most reliable nodes, but the number of nodes is fixed by the user similar to RR. Therefore, though RA technique performs better than RR and All-nodes techniques, the MWT and MCT are higher than RA-Opt. For RA-Opt technique the optimal number of nodes is selected such that the expected completion time is minimal.

Figure 6.8 (a) shows the comparison of MTA, for various techniques and Figure 6.8(b) shows the corresponding MWT. We can observe that the MCT of RA-Opt is 364 percent better than All technique, 1100 percent better than RR, and 33 percent better than RA algorithms. Figure 6.8(c) shows MWT and Figure 6.8(d) the percentage difference of MWT when RA-Opt is compared to other techniques. We observe that the MWT of RA-Opt is 306.39 percent better than the All technique, 156.64 percent better than RR and 44.3 percent better than RA technique. For the ALL technique, jobs fail more often contributing to waste time, however jobs also complete faster because all the nodes are available. Therefore, we observe in Figure 6.8 (a) that the MCT is lower for ALL technique as compared to RR, whereas Figure 6.8 (c) the MWT is higher.

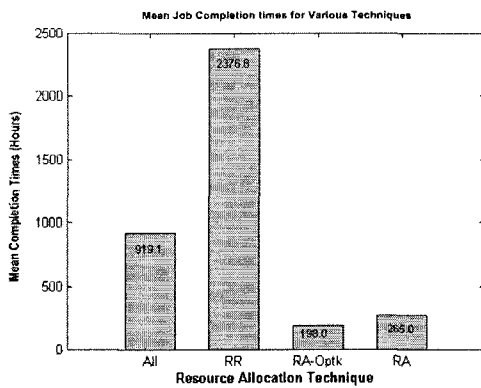


Figure 6.8(a)

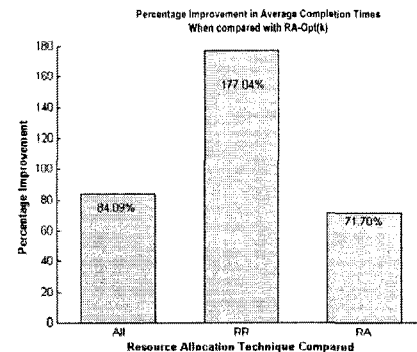


Figure 6.8 (b)

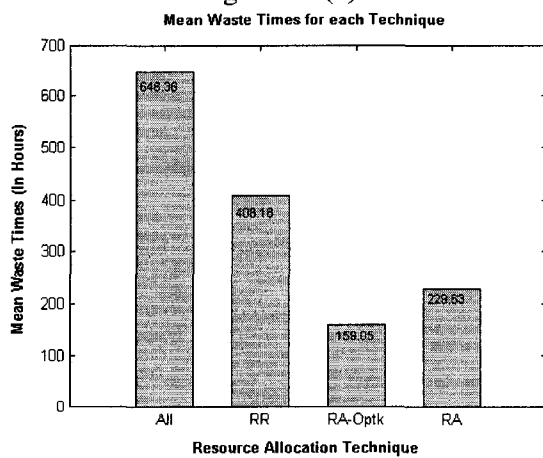


Figure 6.8 (c)

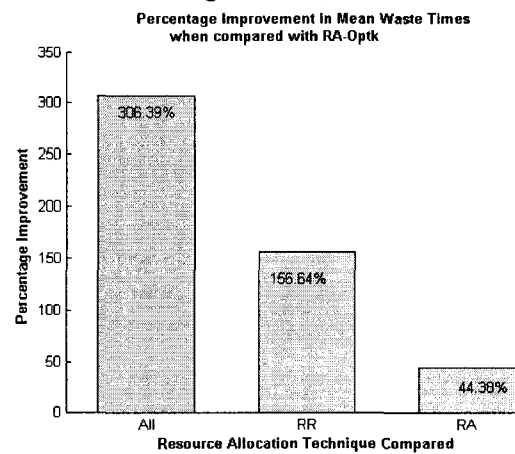


Figure 6.8(d)

Figure 6.8 Comparison of MTA and WTA for various resource allocation algorithms. (a) shows the MTA of the three techniques and (b) shows the corresponding percentages.(c) shows the MWT of each technique and (d) shows the corresponding improvement in percentages of RA-Opt over other techniques.

It is also important to compare the performance metrics with respect to job run-lengths and to especially understand how well the algorithm performs with respect to job run-lengths. Figure 6.9 shows the ACT and AWT metrics with respect to job run-lengths.

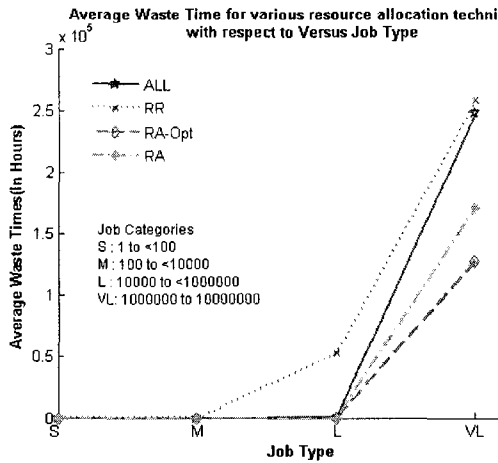


Figure 6.9(a)

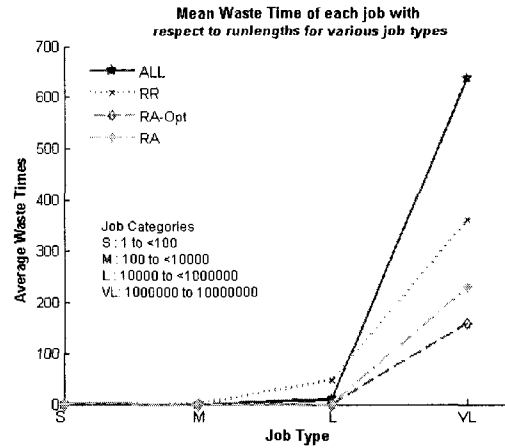


Figure 6.9(b)

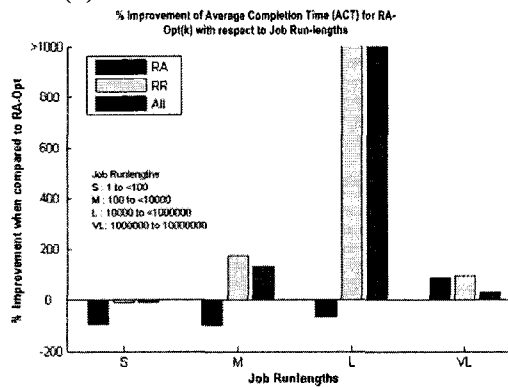


Figure 6.9(c)

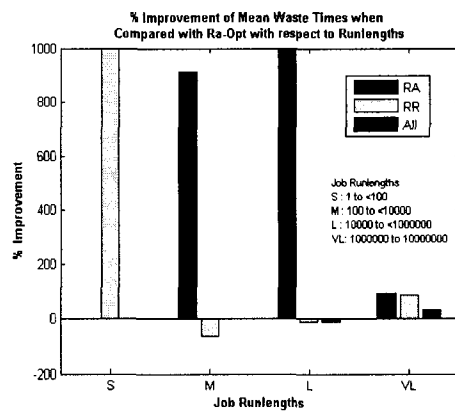


Figure 6.9(d)

Figure 6.9 The comparison of ATA and AWT with respect to run-lengths. Figure 6.9(a) and Figure 6.9(b) show the AWT and the MWT with respect to run-lengths and Figure 6.9(c) and Figure 6.9(d) show the ATA and the AWA with respect to job run-lengths.

For short and medium jobs, the AWT and MWT can be ignored (Figure 6.9(a), and Figure 6.9(b)) for all the four techniques, and the waste time increases with the increase in job run-lengths. The waste time for very long jobs is higher even for RA-Opt because the optimal number of selected nodes $k' \geq m$, where m is the total number of nodes available in the system. However, the RA-Opt technique has more flexibility in deciding the number of nodes and determining if it is worth adding another.

Further, we can observe in Figure 6.9(a, b, c, d) that RA-Opt produces the AWT, MWT, ACT and MCT for very longer jobs that are smaller as compared with other job types. Therefore, the RA-Opt technique performs especially well for longer jobs, where reliability becomes the most crucial.

6.6 Conclusion and Future Work

Increasing the number of nodes in HPC systems for solving ultrascale computational problems will decrease reliability, which presents new challenges in resource management. Several factors affect the completion time of a parallel program as nodes are scaled higher, and reliability becomes a major factor in deciding the optimal number of nodes to minimize completion time. In this chapter, we discuss the effect of reliability on job completion time as scalability and performance metrics for large scale parallel applications. Then, we developed an expected completion time function of parallel programs based on the system reliability. This function is used to develop the algorithm that selects an optimal number of nodes for minimizing the completion time. Our simulation results indicate that long jobs can especially benefit with the reliability-aware optimal k node allocation algorithm to steer away from failures thereby minimizing the completion time and waste time for jobs.

This work has several scopes for improvement. The reliability-aware optimal k-node allocation can be combined with various scheduling algorithms to explore if further improvement is possible. The importance of processor selection for checkpoint based jobs was discussed by Plank[43]. Checkpoint/restart and reliability-aware resource allocation can be optimized together to minimize the overall completion time of the

parallel programs. We also plan to extend the present model to deal with different F_{ki} 's, i.e when the failure probability, MTTF may change over time.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

Future computational platforms may have hundreds and thousands of processors that aim to deliver peta-scale performance. Resource management of such a large scale system would become a major challenge because of the presence of multiple hardware and software components, diverse applications with different workload requirements and users with different priorities. Therefore, reliability would be a major performance hindrance factor, especially for time critical applications that demand QOS from the computational service provider.

This dissertation presents a system TTF model based on the time varying failure distribution of individual nodes, and proposes reliability-aware resource allocation algorithms for parallel applications. First, we demonstrate the TTF distributions and correlations of a production HPC system, the LLNL based ASCI White system. Then we present the system TTF distribution model, failure rate, reliability and the MTTF when the TTF's of individual nodes have a time-varying failure rate. The effectiveness of proposed reliability-aware resource allocation algorithms were evaluated on the actual failure data and standard workloads. We observe that applying a time-varying failure rate based reliability-aware resource allocation algorithms reduces the overall performance loss by as much as 53 percent. Finally, we study the effect of reliability with scaling up the number of nodes and proposed reliability-aware optimal k node allocation algorithm.

The comparison results of the proposed optimal k nodes versus existing resource allocation algorithms suggest that giving flexibility for the scheduler in determining the optimal number of nodes based on reliability is especially beneficial for large scale parallel applications.

This work has a broad scope to be extended for both reliability prediction, and reliability-aware resource allocation. The reliability prediction approach we used is based on well known statistical distributions observed from the TTF data. In addition to the TTF history, the failure properties of nodes like the usage history, CPU load and motherboard temperature could be incorporated into the reliability model to improve the prediction accuracy.

The proposed reliability-aware resource allocation algorithms can be further investigated with various scheduling and queuing policies and reliability-aware resource allocation can be incorporated into currently available resource managers. In addition, reliability of nodes becomes a very crucial factor for time sharing applications because in the event of a failure, several applications running on a single node are simultaneously affected. Therefore, reliability-aware resource allocation of time sharing applications is another extension to this work as well.

APPENDIX A

PROOFS FOR SYSTEM RELIABILITY MODEL

Theorem 1:

The system TTF function $s(x)$ is a distribution function, i.e. .

We verify the above equation for two cases, when $k=1$ and $k=2$, where, k is the number of nodes.

Proof:**Case k=1:**

For single node, the TTF distribution of the node is a Weibull, and we know that $w(t)$ is a distribution function. Therefore, for the system Pdf $s(x)$ we get,

$$\int_0^{\infty} s(x) dx = \int_0^{\infty} w(x) dx = 1.$$

Case k=2:

For two nodes,

$$\begin{aligned} \int_0^{\infty} s(x) dx &= \int_0^{\infty} w_1(x) [1 - H_2(t_1 + x | t_1)] dx + \int_0^{\infty} h_2(t_1 + x | t_1) [1 - W_1(t)] dx \\ &= [1 - H_2(t_1 + x | t_1)] W_1(x) \Big|_0^{\infty} + \int_0^{\infty} W_1(x) h_2(t_1 + x | t_1) dx + \int_0^{\infty} h_2(t_1 + x | t_1) [1 - W_1(x)] dx \\ &= \int_0^{\infty} h_2(t_1 + x | t_1) dx = 1, \end{aligned}$$

Where $H_2(x)$ is the CDF of Excess Weibull, and $W_1(x)$ is the CDF of Weibull.

Theorem 2:

The system reliability when the TTF distribution of individual nodes is Weibull is given

$$\text{by } R_j(x) = 1 - S_j(x) = \prod_{i=1}^k [1 - F_i(x)].$$

Proof:

We verify the above equation for the case where $\alpha_1 = \alpha_2 = \dots = \alpha$ and

$$\beta_1 = \beta_2 = \dots = \beta .$$

From the theory of the first order statistic, the TTF distribution of the system is given by,

$$S_j(x) = \int_0^x k[1 - F(t)]^{k-1} f(t) dt ,$$

$$S_j(x) = -[1 - F(t)]^k \Big|_0^x$$

Therefore the system reliability is given by $R_j(x) = 1 - S_j(x) = [1 - F(x)]^k$.

Theorem 3:

Given a random variable U, drawn from a uniform distribution, on (0, 1) we observe that

$H_i^{-1}(t+x|t) = (t^\beta - \alpha^\beta \cdot \ln(1-U))^{1/\beta} - t$ has an excess Weibull distribution.

Proof:

We use a similar idea given in [65]

Let $U = H(t+x|t)$.

Hence , $U = 1 - e^{-\frac{1}{\alpha^\beta}(t^\beta - (t+x)^\beta)}$ and $\ln(1-U) = \frac{1}{\alpha^\beta}(t^\beta - (t+x)^\beta)$

$$\begin{aligned} \alpha^\beta \cdot \ln(1-U) &= t^\beta - (t+x)^\beta \\ t^\beta - \alpha^\beta \cdot \ln(1-U) &= (t+x)^\beta \\ (t^\beta - \alpha^\beta \cdot \ln(1-U))^{1/\beta} &= (t+x) \\ x &= (t^\beta - \alpha^\beta \cdot \ln(1-U))^{1/\beta} - t. \end{aligned}$$

APPENDIX B

MATLAB PROGRAMS FOR THE SYSTEM TTF MODEL

```

1. Program to calculate the system MTTF
function y= sysMTTF(t,s,A,B)
% usage sysMTTF(10,50,[1000],[0.7])
% t=10;
% s=50;
% A=[1000 1000];
% B=[0.7 0.7];
%t is the survival time
%s is the excess life
%A is an array of alpha/scale parameter/characteristic lifes for each node
%B is the corresponding beta/shape parameter for each Node
%Note that A and B have to be of the same size
y=1;
cnt=1;
val=[];

for(i=1:max(size(A)))
    %(1) Build the expression as a string
    term2=['s.* ewpdfstr(A(cnt),B(cnt),t)];
    test2='epdf()';
    j=1;
    while(j<=max(size(A)))
        if(cnt ~= j)
            term2=[term2 .* '(1-( ewcdfstr(A(j),B(j),t) ))'];
            test2=[test2 #' (1-ewcdf())'];
        end
        j=j+1;
    end
    syms s;
    a=A(cnt);
    b=B(cnt);
    term2 =['@(s)' term2];
    yy=eval(term2)
    %the following steps find the upper limit to integrate
    inc=10;
    v1=1;v2=10;
    tol=0.001; %you may adjust the tolerance as needed
    while(abs(v1-v2)>tol)
        v1=v2;
        inc=inc*10;
        v2=quad(yy,0,inc);
    end
    %basically inc is the higher order limit to integrate.
    cdf=v2;
    val=[val cdf];
end

```

```
y=sum(val);
```

2. Program to calculate the system reliability of k nodes
- ```
function y= sysRel(t,s,A,B)
%t is the survival time
%s is the excess life
%A is an array of alpha/scale parameter/characteristic lives for each node
%B is the corresponding beta/shape parameter for each Node
%Note that A and B have to be of the same size
y=1;
for(i=1:max(size(A)))
 R(i)=1-eweibcdf(t,s,A(i),B(i));
 y=y*R(i);
end
```
- 

3. Program to calculate the system failure rate of k nodes
- ```
function y= sysFrate(t,s,A,B)
cnt=1;
%For each node...
while(cnt<=max(size(A)))
    P(cnt)=eweibpdf(t,s,A(cnt),B(cnt)); % the Pdf of each node
    C(cnt)=eweibcdf(t,s,A(cnt),B(cnt)); % the CDF of each node
    cnt=cnt+1;
end
cnt1=1;
cnt2=1;
sum1=0;sum2=0;
while(cnt1<=max(size(A)))
    cnt2=1;
    sum1=P(cnt1);
    while(cnt2<=max(size(A)))
        if(cnt1~=cnt2)
            sum1=sum1*(1-C(cnt2));
        end
        cnt2=cnt2+1;
    end
    sum2=sum2+sum1;
    cnt1=cnt1+1;
end
y=sum2;
```
-

4. function y= eweibcdf(t,s,a,b)
- ```
% This function calculates the Weibull CDF
%t is the survival time
```

```

% s is the excess life
% a is alpha or c
% b is beta or m
y=1-exp(-power((t+s)/a,b)+power(t/a,b));

```

---

```

5. function y=eweibpdf(t,s,a,b)
% t is the survival time
% s is the excess life
% a is alpha or c
% b is beta or m
y=((b*power(t+s,b-1))/power(a,b))*exp((power(t,b)-power(t+s,b))/power(a,b));

```

---

```

6. function S= wcdfstr(x,y)
a=num2str(x);
b=num2str(y);
s='s';
S=['(1-exp(-power(s./ a ', ' b ')))]';

```

---

```

7. function S= wpdfstr(x,y)
a=num2str(x);
b=num2str(y);
s='s';
S=['((' b ' ./s).*power(s./ a ', ' b ').*exp(-power(s./ a ', ' b ')))]';

```

---

```

8. function F= ewcdfstr(x,y,z)
% a - alpha or scale parameter
% b - beta of the shape parameter
% t - survival time of node
% s - The excess life
a=num2str(x);
b=num2str(y);
t=num2str(z);
s='s';
F=['1-exp((1./power(' a ', ' b ')).*(power(' t ', ' b ')-power(' t '+' s ', ' b ')))]';

```

---

```

9. function F= ewpdfstr(x,y,z)
% a - alpha or scale parameter
% b - beta of the shape parameter
% t - survival time of node
% s - The excess life
a=num2str(x);
b=num2str(y);
t=num2str(z);
s='s';
F=['((' b ' .*power(' t '+' s ', ' b '-1))./power(' a ', ' b ')).*exp((power(' t ', ' b ')-
power(' t '+' s ', ' b '))./power(' a ', ' b ')))]';

```

---

## REFERENCES

- [1] Y. Liang, Y. Zhang, A. Sivasubramaniam, M. Jette, and R.Sahoo, "BlueGene/L Failure Analysis and Prediction Models," pp. 425-434, *International Conference on Dependable Systems and Networks (DSN'06)*, 2006
- [2] Fullop, J, "Clumon - The cluster monitoring system," <http://clumon.ncsa.uiuc.edu/>
- [3] M. L. Massie, B.N. Chun and D.E. Culler, "The GAnglia Distributed Monitoring System: Design, Implementation, and Experience," *Parallel Computing*, vol 30, iss 7, pp. 817-840, July 2004.
- [4] E. Galstad, NAGIOS: Host and Service Monitor, 2005. Available at <http://www.nagios.org>.
- [5] C. Engelmann and G. A. Geist. "Super-Scalable Algorithms for Computing on 100,000 Processors," *Proceedings of International Conference on Computational Science (ICCS)*, Atlanta, GA, USA, May 2005.
- [6] 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, pp. 97-108, April-June, 2004.
- [7] H. Song, C. Leangsuksun, N. Gottumukkala, R. Nassar, S. L. Scott, and Andy Yoo, "Near-Real-time Availability Monitoring and Modeling for HPC/HEC runtime systems," *Symposium of Los Alamos Computer Science Institute*, Santa Fe, New Mexico, October 2005.
- [8] C. Engelmann, S. L. Scott, D. E. Bernholdt, N. R. Gottumukkala, C. Leangsuksun, J. Varma, C. Wang, F. Mueller, A. G. Shet, and P. Sadayappan. "MOLAR: Adaptive runtime support for high-end computing operating and runtime systems,". *ACM SIGOPS Operating Systems Review*, vol. 40, pp. 63-72, 2006.
- [9] D. G. Feitelson and L. Rudolph, "Toward convergence in job schedulers for parallel supercomputers: Job Scheduling Strategies for Parallel Processing," vol. 1162, *Lecture Notes in Computer Science*, Springer-Verlag, pp. 1-26, 1996.



- [10] Gaj, K., El-Ghazawi, et al., "Performance Evaluation of Selected Job- Management Systems," *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002*, pp.254-261, 2002.
- [11] Jackson, D. B., Snell, Q., and Clement, M. J., "Core Algorithms of the Maui Scheduler," In Revised Papers *From the 7th international Workshop on Job Scheduling Strategies For Parallel Processing*, Lecture Notes In Computer Science, vol. 2221. Springer-Verlag, London, pp. 87-102, 2001
- [12] Gentsch, W., "Sun Grid Engine: Towards creating a compute power grid, Cluster Computing and the Grid," *Proceedings. First IEEE/ACM International Symposium*, pp. 35 – 36, May 2001.
- [13] A. Yoo, M. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," *Lecture Notes in Computer Science*, pp. 44-60, 2003.
- [14] M. F. Buckley and D. P. Siewiorek., "Vax/vms Event Monitoring and Analysis," *In FTCS-25, Computing Digest of Papers*, pp. 414–423, June 1995.
- [15] D. G. Feitelson. Parallel workloads archive. <http://cs.huji.ac.il/labs/parallel/workload/index.html>, 2001.
- [16] B. Schroeder and G.A Gibson, "A large-scale study of failures in high-performance computing systems," *In Proceedings of the international Conference on Dependable Systems and Networks*, June 2006.
- [17] Heath, T., Martin, R. P., and Nguyen, T. D. 2002. "Improving cluster availability using workstation validation," *SIGMETRICS Perform. Eval. Rev.* vol. 30, pp. 217-227, June 2002.
- [18] J. Xu, Z. Kalbarczyk, and R.K Iyer, "Networked Windows NT system field failure data analysis," *Pacific Rim International Symposium on Dependable Computing*, pp.178-185, 1999.
- [19] N. R. Gottumukkala, C. Leangsuksun, Y. Liu, R. Nassar, and S. L. Scott, "Reliability analysis in HPC clusters," in *Proceedings of High Availability and Performance Workshop (HAPCW) 2006, in conjunction with Los Alamos Computer Science Institute (LACSI) Symposium 2006*, Santa Fe, NM, USA, October 17, 2006.
- [20] R. K. Iyer, D. J. Rossetti, and M. C. Hsueh, "Measurement and modeling of computer reliability as affected by system activity," *ACM Trans. Comput. Syst.*, vol. 4, no 3, 1986.
- [21] R.K Sahoo, M.S. Squillante, A. Sivasubramaniam, and Y. Zhang, "Failure data analysis of a large-scale heterogeneous server environment," *International Conference on Dependable Systems and Networks*, pp. 772-781, July 2004.

- [22] T. Lin, and D.P. Siewiorek, "Error log analysis: statistical modeling and heuristic trend analysis," *IEEE Trans on Reliabilit*, vol.39, no.4, pp.419-432, October 1990.
- [23] D.K. Dey, and L.R. Jaisingh, "Estimation of system reliability for independent series components with Weibull life distributions," *IEEE Transactions on Reliability*, vol.37, Iss.4, pp. 401-405, October 1988.
- [24] T. F. Hassett, D. L. Dietrich, and F. Szidarovszky, "Time-varying failure rates in the availability & reliability analysis of repairable systems," *IEEE Trans.on Reliability*, vol. 44, no. 1, pp. 155–160, March 1995.
- [25] 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.
- [26] Oliner, A.J., R.K. Sahoo, J.E. Moreira, M. Gupta, and A. Sivasubramaniam, "Fault-aware job scheduling for BlueGene/L systems," *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pp. 26-30 April 2004.
- [27] A. J. Oliner, R. Sahoo, J. E. Moreira, M. Gupta, and A.Sivasubramaniam, "Fault-aware Job Scheduling For BlueGene/L Systems," *In Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, 2004.
- [28] W. Linping, M. Dan, et al, "A Failure-Aware Scheduling Strategy in Large-Scale Cluster System," *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, pp. 645-648, 2006.
- [29] S.M. Shatz, J.P. Wang, M. Goto, "Task Allocation for Maximizing Reliability of Distributed Computer Systems," *IEEE Transactions on Computers*, vol. 41, no. 9, pp. 1156-1168, September 1992.
- [30] S. Kartik and C. Siva Ram Murthy, "Task Allocation Algorithms for Maximizing Reliability of Distributed Computing Systems," *IEEE Trans. on Computers*, vol. 46, no. 6, pp. 719-724, 1997.
- [31] S. Srinivasan, N.K. Jha, "Safety and Reliability Driven Task Allocation in Distributed Systems," *IEEE Trans. Parallel Distributed Systems*. vol.10, no. 3, pp.238-251,1999.
- [32] M. Wu, X.H. Sun and H. Jin, "Performance under Failures of High-End Computing". *In Proceedings of the 2007 ACM/IEEE Conference on Supercomputing Reno, Nevada*, 2007.
- [33] G. M. Amdahl, "Validity of the single processor approach to achieving large-scale computing capabilities," *in Proceedings of the AFZPS*, pp. 483-485, vol. 30, 1967.

- [34] N. J. Davies. "The Performance and Scalability of Parallel Systems", PhD thesis, Department of Computer Science, Faculty of Engineering, University of Bristol, UK, December 1994.
- [35] Kumar, V. and A. Gupta, "Analysis of Scalability of Parallel Algorithms and Architectures: A survey," *In Proceedings of the 5th international Conference on Supercomputing*, Cologne, Germany, pp.396-405, 1991.
- [36] K., Hwang, "Advanced Computer Architecture: Parallelism, Scalability, and Programmability," New York, McGraw-Hill, Inc, 1993.
- [37] V. Kumar and A. Gupta, "Analyzing scalability of parallel algorithms and architectures," *Journal of Parallel and Distributed Computing*, vol. 22, no.3, pp.379-391, 1994.
- [38] Gustafson, J. L. "Reevaluating Amdahl's law," *Communications of the ACM*, vol.31, pp. 532-533, May 1988.
- [39] Nicol, D. M. and Willard, F. H. "Problem size, parallel architecture, and optimal speedup." *Journal of Parallel & Distributed Computing*. Vol. 5, no. 4 pp. 404-420, August 1988.
- [40] Reed, D. A., Lu, C., and Mendes, C. L. 2006. "Reliability Challenges in Large Systems," *Future Generation Computing. Systems*, vol. 22, no. 3, pp. 293-302, Febraury 2006.
- [41] J.S. Plank, W.R. Elwasif, "Experimental Assessment of Workstation Failures and Their Impact on Checkpointing Systems," The Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing, 1998.
- [42] N. R. Gottumukkala, C. Leangsuksun, and S. L. Scott, "Reliability-aware approach to improve job completion time for large-scale parallel applications," *In Proceedings of 2nd Workshop on High Performance Computing Reliability Issues (HPCRI) 2006*, Austin, TX, USA, February 11-15, 2006.
- [43] 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*, pp. 250-259, June 1999.
- [44] C. Weaver and T. M. Austin, "A Fault Tolerant Approach to Microprocessor Design," *In Proceedings of the 2001 international Conference on Dependable Systems and Networks*, pp. 411-420, June 2001.
- [45] Lawrence Livermore National Laboratory Trace Logs: url: <http://www.llnl.gov/asci/platforms/white/>

- [46] A. C. Cohen and B.J. Whitten, 1988. *Parameter Estimation in reliability and Life Span Models*, Marcel Dekker, New York.
- [47] T T. Soong "Model Verification," in *Fundamentals of Probability and Statistics for Engineer*, John Wiley & Sons Ltd, Chichester, UK, p. 327, 2004.
- [48] S. Nath, H. Yu, P.B. Gibbons, and S. Seshan, S., "Subtleties in Tolerating Correlated Failures in Wide-Area Storage Systems," In *Proceedings of the 3rd Conference on 3rd Symposium on Networked Systems Design and Implementation*, vol. 3, USENIX Association, Berkeley, CA, May 2006.
- [49] R. Jain, "The Art of Computer Systems Performance Analysis", J. Wiley & Sons, Inc., 1991.
- [50] B. Chun and A. Vahdat, "Workload and failure characterization on a large-scale federated testbed," Technical Report IRB-TR-03040, Intel Research Berkeley, November 2003.
- [51] D. Tang, R. K. Iyer, and S. S. Subramani, "Failure analysis and modelling of a vaxcluster system," In *Proceedings of 20th. Intl. Symposium on Fault-tolerant Computing*, pp. 244–251, 1990.
- [52] R.V. Hogg, and E.A. Tanis. 1983, "Probability and Statistical Inference," 2nd edition. Macmillan Publishing Co., Inc. New York.
- [53] B. Schroeder, and G. A. Gibson, "Disk Failures in the Real World: What does an MTTF of 1,000,000 Hours Mean to You? " *the 5<sup>th</sup> USENIX Conference on File and Storage Technologies (FAST 2007)*, San Francisco, CA, Febraury. 2007.
- [54] R. L. Burden and J. Douglas Faires. *Numerical Analysis*. PWS-Kent, Boston, fourth edition, 1989.
- [55] N.R. Gottumukkala, C. Leangsuksun, R. Nassar, and S.L Scott. "Reliability-Aware Resource Allocation in HPC Systems," *Proceedings of the IEEE International Conference on Cluster Computing 2007*, Austin Texas, September 2007.
- [56] Yudan Liu; C.B. Leangsuksun, H. Song; and S.L. Scott, "Reliability-aware Checkpoint/Restart Scheme: A Performability Trade-off, " *Cluster Computing, 2005. IEEE International*, pp.1-8, September 2005.
- [57] K.G. Popstojanova, K.and K.S. Trivedi, "Failure Correlation in Software Reliability Models," *IEEE Transactions on Reliability*, vol.49, iss.1, pp. 37-48, March 2000.
- [58] MATLAB.2006, MATLAB. Version 6.0. MathWorks, Natick, Massachusetts, USA.

- [59] S. Sahni, V. Thanvantri, "Performance Metrics: Keeping the Focus on Runtime," *IEEE Parallel and Distributed Technology*, vol. 04, no. 1, pp.43-56, Spring, 1996.
- [60] D. Eager, J. Zahorjan, and E. Lazowska, "Speedup versus Efficiency in Parallel Systems," *IEEE Transactions on Computers*, vol. 38, pp.408-423, March 1989.
- [61] J. Worlton, "Toward a Taxonomy of Performance Metrics", in *Computer Benchmarks*, J. J. Dongarra and W. Gentsch, Eds. Elsevier Advances In Parallel Computing Series, vol. 8. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, 1993.
- [62] A.B. Downey, "A Parallel Workload Model and its Implications for Processor Allocation," In *Proceedings of the 6th IEEE international Symposium on High Performance Distributed Computing*, High Performance Distributed Computing. IEEE Computer Society, Washington, DC, August 1997.
- [63] D.G. Feitelson, "Packing Schemes for Gang Scheduling," In *Proceedings of the Workshop on Job Scheduling Strategies For Parallel Processing* D. G. Feitelson and L. Rudolph, Eds. Lecture Notes In Computer Science, vol. 1162. Springer-Verlag, London, pp. 89-110, 1996.
- [64] U. Lublin and D. G. Feitelson, "The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs", *Journal of Parallel & Distributed Computing*. Vol. 63, no.11, pp. 1105-1122, November 2003.
- [65] S. Ross, *A First Course in Probability*. New York: Macmillan, 1976.