Spring 2014

Reliability Guided Resource Allocation for Large-scale Supercomputing Systems

Shruti Umamaheshwaran
Purdue University

Graduate School ETD Form 9
(Revised /)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By SHRUTI UMAMAHESHWARAN

Entitled Reliability Guided Resource Allocation for Large-scale Supercomputing Systems

For the degree of Master of Science

Is approved by the final examining committee:

Dr. Thomas J. Hacker

Dr. Eric T. Matson

Dr. John A. Springer

Dr. Tomasz W. Wlodarczyk

To the best of my knowledge and as understood by the student in the
C Disclaimer (Graduate School Form), this thesis/dissertation
Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Dr. Thomas J. Hacker

Approved by: Prof. Jeffrey L. Whitten 04/09/2014

Head of the Graduate Program Date

RELIABILITY GUIDED RESOURCE ALLOCATION FOR LARGE-SCALE SUPERCOMPUTING SYSTEMS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Shruti Umamaheshwaran

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

May 2014

Purdue University

West Lafayette, Indiana

Dedicated to my family.

ACKNOWLEDGMENTS

My sincere thanks to my thesis committee chair and academic advisor, Dr. Thomas J. Hacker for giving me an opportunity to work on this interesting research topic. His constant support and guidance in polishing my research skills has helped me understand the research problem well and work towards the development of the analysis methodology, detailed in this thesis.

I also wish to express my gratitude to my committee members: Dr. Eric T. Matson, Dr. John A. Springer and Dr. Tomasz W. Wlodarczyk for their valuable comments and suggestions. I thank all my committee members for their assistance in finalizing the thesis proposal, plan of study and the final thesis.

I would also like to thank my family for their cooperation and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

λ_r	Average arrival rate of resource requests for resource class r
t_r	Average service time for resource requests for resource class r
α_r	Normalized offered workload for resource class r in Erlangs
C_r	Number of resources used by resource class r
n_r	Number of processors per resource class r
B_{p_r}	Probability of blocking a resource request for resource class r
P_{c_r}	Probability of queuing a resource request for resource class r
F_r	Queue length for resource class r
W_r	Waiting time for jobs in the queue for resource class r

LIST OF ABBREVIATIONS

MTTF	Mean Time to Failure
MTBF	Mean Time Between Failures
CAC	Connection Admission Control
CS	Complete Sharing
CP	Complete Partitioning
LWF	Lease Workload Format
RAS	Reliability Availability Serviceability
ANL	Argonne National Laboratory

GLOSSARY

MTBF	In the case of repairable systems, ”MTBF” stands for ”mean time between failures.” This average time excludes the time spent waiting for repair, being repaired, being re-qualified, and other downing events such as inspections and preventive maintenances and so on; it is intended to measure only the time a system is available and operating. Whereas, in the case of non-repairable systems, MTBF stands for mean time before failure and is represented by the mean life value for a failure distribution of non-repairable units. (Weibull.com, 2013)
MTTF	MTTF stands for ”mean time to failure” and is represented by the mean life value for a failure distribution of non-repairable units. (Weibull.com, 2013)
MTTR	MTTR stands for ”mean time to repair” and is represented by the mean life value for a distribution of repair times. (Weibull.com, 2013)
System Reliability	The reliability of an entire system, as opposed to the reliability of its components. The system reliability is defined by the reliability of the components as well as the way the components are arranged reliability-wise. (Weibull.com, 2013)

Erlang	Offered traffic in Erlangs can be determined by computing the product of the call arrival rate, λ, and the average call-holding time, h , $E = \lambda * h$. Erlang is a dimensionless unit since it is the product of average call arrival rate (T^{-1}) and average call holding time (T^{1}). Traditionally, it was used to compute the number of switching elements for the carried load. (Kobayashi & Mark, 2009)

ABSTRACT

Shruti Umamaheshwaran M.S., Purdue University, May 2014. Reliability Guided Resource Allocation for Large-scale Supercomputing Systems. Major Professor: Thomas J. Hacker.

In high performance computing systems, parallel applications request a large number of resources for long time periods. In this scenario, if a resource fails during the application runtime, it would cause all applications using this resource to fail. The probability of application failure is tied to the inherent reliability of resources used by the application. Our investigation of high performance computing systems operating in the field has revealed a significant difference in the measured operational reliability of individual computing nodes. By adding awareness of the individual system nodes' reliability to the scheduler along with the predicted reliability needs of parallel applications, reliable resources can be matched with the most demanding applications to reduce the probability of application failure arising from resource failure. In this thesis, the researcher describes a new approach developed for resource allocation that can enhance the reliability and reduce the costs of failures of large-scale parallel applications that use high performance computing systems. This approach is based on a multi-class Erlang loss system that allows us to partition system resources based on predicted resource reliability, and to size each of these partitions to bound the probability of blocking requests to each partition while simultaneously improving the reliability of the most demanding parallel applications running on the system. Using this model, the partition mean time to failure (MTTF) is maximized and the probability of blocking of resource requests directed to each partition by a scheduling system can be controlled. This new technique can be used to determine the size of the system, to service peak loads with a bounded probability of blocking to resource requests. This approach would be useful for high performance computing system operators seeking to improve the reliability, efficiency and cost-effectiveness of their systems.

CHAPTER 1. INTRODUCTION

1.1 Background

Large-scale high performance computing systems built from tens of thousands of processors are reaching into the peta- and exascale in terms of raw performance and capability. These systems are built up from electrical and software components, any of which may fail and consequently cause the failure of jobs that depend on the reliable operation of all of these elements.

To avoid and react to faults when they occur, there are several proactive and reactive fault tolerance strategies that have been devised that seek to reduce the probability of failures, the costs of failures, or to recover from failures when they occur. One common reactive strategy is checkpointing, in which a parallel application saves the current computational state so that if the computation is restarted, work can progress from the last saved state. Reactive reliability operations such as checkpointing, however, are failing to scale as system sizes increase, and themselves can place a tremendous burden on the system, inducing further failures and decreasing the fraction of productive time spent on performing useful work rather than defensive operations for the eventuality of a failure (Hacker & Mahadik, 2011).

As an alternative to reactive operations such as checkpointing, *proactive* fault tolerance strategies can help prevent failures in the first place. By avoiding a failure, a system can significantly increase the useful work extracted from the system by avoiding the need for frequent defensive operations.

Scheduling systems for large-scale supercomputing systems in use today offer an excellent place to introduce proactive fault tolerance strategies. Since the scheduler actively manages the work allocation to nodes, it acts as a control system that can be easily modified to actively manage the reliability profile of a system and to modulate the flow of resource requests and jobs that use those systems.

Scheduling systems in use today, such as Torque (Adaptive Computing, 2013), SLURM (SLURM,2013), PBS (Henderson, 1995), and LoadLeveler (Kannan et al., 2001), do not take into account the inherent reliability characteristics of the systems they manage - they treat all computational resources as homogeneous and replaceable elements. Consequently, they do not attempt to guide jobs to reliable resources based on the reliability needs of those jobs. As a result, specific jobs using large number of processors or having long runtimes that would benefit by using high reliability resources may have to suffer from the assignment of poor reliability system resources when in fact there are high reliability resources idle in the cluster. This underutilization of reliable resources is wasteful and inefficient, and leads to unnecessary low jobs and system reliability.

To improve this situation, this research presents a new proactive fault tolerance approach that the researcher has developed, which exploits the available information, that includes: 1) predicted node reliability; 2) historic resource use patterns from the workload offered to the system; 3) information about jobs waiting for resources in a system queue; and 4) the desired reliability and queuing characteristics of these systems provided by system administrators.

1.2 Scope

This research focuses on studying the impact of node reliability on the overall reliability and efficiency of jobs in high performance computing systems. This study also concentrates on determining the number of nodes to be provisioned in a cluster to service workloads at busy periods by offering a minimum probability of blocking or queuing to the incoming resource requests.

Another aim of this research work is to develop a partition strategy by determining the number of partitions and the size of each partition, such that the overall reliability of partition in terms of partition MTTF (Mean Time to Failure) can be improved. The scope of this research involves implementing this analysis methodology on an open-source scheduler Haizea, which is an infrastructure manager used for scheduling resources in an Open Nebula cluster (Sotomayor, 2009).

1.3 Significance

This reliability analysis approach can improve scheduling decisions for large parallel jobs submitted to the system, improve hardware reliability experienced by the parallel applications, and improve the overall system reliability. In this thesis, the researcher shows that through the use of this new approach, the overall reliability and efficiency of jobs in the system can be increased.

The benefits of this new approach can be immediately realized without the need to purchase additional hardware or to reconfigure the system architecture. This approach simply exploits information already available on the system and synthesizes this information to manage the scheduling process to significantly improve the reliability of jobs, especially those that suffer the most from the inherent poor reliability of components that make up most petascale systems today and exascale systems tomorrow.

As most systems today are moving towards the era of cloud and cluster-based computing, it is essential for cluster providers to provide reliable resources and make systems more available by keeping the blocking probability and queue wait times as low as possible. The methodology developed in this study is generic and can be applied to any system irrespective of the platform, scheduling policy implemented in the cluster and type of incoming workload.

1.4 Problem Statement

Given the observed reliability features of a system, how can the system resources be allocated to resource requests such that the reliability of the jobs is maximized and at the same time specify a bound on blocking probability of resource requests, for a given workload at peak periods?

1.5 Assumptions

The assumptions for this study include:

- The request site nodes are logically classified into R resource classes, where the number of nodes n_i in each class r_i is 2^{i-1} where $i \epsilon \{1, ..., R\}$
- The system is an asymptotically large network thus allowing us to use approximations for Erlang Loss function
- The simulation uses a hypothetical cluster with all uni-processor nodes and each node has an associated reliability in terms of MTTF in hours
- The reliability need of every resource request is mentioned as an attribute of every request in the lease workload file
- The terms MTBF and MTTF are used interchangeably in this research and the MTBF is considered the observed reliability of nodes
- The ANL Intrepid BlueGene workload logs and system failure logs of the Coates cluster are accurate and not corrupt
- The reliability pattern of the system nodes follows a Weibull distribution
- The Haizea scheduler, which is the scheduler of the OpenNebula toolkit, functions properly and emulates a real scheduler

1.6 Limitations

The limitations for this study include:

- Network characteristics (like bandwidth, switch latencies, etc.) and memory factors, though vital are not considered in determining the resource occupancy in the cluster
- The Haizea simulation workload consists of tasks submitted only during the busy period
- The system logs have been filtered to remove false alarms and warnings before computing mean time between failures

1.7 Delimitations

The delimitations for this study include:

- The simulation results are based on only Haizea simulator
- The model and scheduling policies framework have been tested on workloads from ANL Intrepid BlueGene supercomputer only
- The simulation is developed only for Type I scheduling policy
- The system logs for determining the observed node reliability have been obtained from the Coates Cluster at Purdue University

1.8 Summary

Chapter 1 gives an overview of the research project by explaining the problem statement, scope, significance and other background information related to this research. The next chapter provides a review of literature, outlining the motivation for this research.

CHAPTER 2. REVIEW OF LITERATURE

The primary issue faced by most large-scale supercomputing and cloud computing systems is the occurrence of numerous system failures. Too many system failures in large datacenters can adversely affect the overall reliability of the applications running on them, culminating in low system utilization and low efficiency of the computing system. Poor usage of system resources would hamper not only the speeds and performance of user jobs, but also draw umpteen monetary and energy resources.

There are two ways to tackle this problem, one is the hardware approach and the other is a less expensive software approach. The hardware approach considers replacement or addition of new components to existing system hardware when system failures occur. The hardware method is generally not feasible and would certainly be worth its weight in gold! Considering the second approach - the software approach - is comparatively a practical solution and can be used by almost all large-scale cluster system administrators. In this method the system administrators neither have to modify their existing system hardware nor do they have to pay a pretty penny.

The software method, which is developed in this research, adjusts and refines specific attributes in the configuration files and adds reliability determinants in task schedulers of the computing systems, thus enhancing a reliability guided scheduling decision. These reliability determinants are 'machine learned', from observed reliability characteristics from historic system logs and failure trends in service logs.

In order to propose the new reliability guided approach described in this research, an in depth understanding of the following points is essential. They are:

- Types of resource allocation policies in use today
- Resource Availability in large-scale systems
- System Reliability

2.1 Resource Allocation Policies

In the most general case of resource allocation, all resource requests are admitted simply if resources are available at the time a connection is requested. This is commonly called a complete sharing (CS) admission policy where the only constraint on the system is the overall system capacity. In a CS policy, connections that request fewer resource units are more likely to be admitted. A CS policy does not consider the importance of a connection when resources are allocated (Beard, 2001). In a complete partitioning (CP) policy, every class of resource is allocated a set of resources that can only be used by that class. An upper limit (UL) policy places upper limits on the numbers of connections possible from each class to ensure that no one class dominates the system resources.

The study of resource allocation policies is one of the primary steps to develop the new approach described in this research work. The selection of the resource allocation policy is important as it determines the partitioning strategy to be used for distributing system nodes, which is directly linked with the reliability of system. Every node in the system has a reliability characteristic associated with it usually measured in Mean Time to Failure (MTTF). MTTF is a basic measure of reliability for nodes in the system and is the mean time expected until the first failure of a node. This is a statistical value and is meant to be the mean over a long period of time.

The right selection of resource allocation policy should ensure that the combination of nodes in a partition does not deteriorate the partition reliability, thus improving the overall reliability of the partition as well as the jobs running in that partition. Selection of the resource allocation policy will help us determine the number of partitions and the size of each partition, such that the probability of blocking of resource requests during the busy period is minimum thus eliminating resource wastage issues due to overprovisioning. The stochastic knapsack approach would be used in this research to distribute nodes into system partition based on node reliability.

2.2 Availability in Large-scale Systems

The previous work by Hacker and Mahadik (Hacker & Mahadik, 2011), models availability of resources in a cloud computing system based on an offered workload. Their work presents a model for predicting probability of blocking service requests of an N node cloud computing service during the busy period, and also provides a technique to determine the number of hot-spare nodes needed to provide a reliable cloud computing service. This research is an extension to the work mentioned above and focuses on the reliability aspect in a cluster computing system. In the approach described in this research, the researcher concentrates on studying the effect of number of nodes per resource class on the overall job and partition reliability. In this thesis the selected resource allocation policy uses system node reliability as a criterion for resource allocation.

The analysis model used for understanding the impact of scheduling policies is based on the tele-traffic theory called Erlang theory (Rappaport, 1996). The Erlang computations are calculated using the two Erlang formulas - Erlang B (loss function) and Erlang C (queuing model). Erlang calculations are widely used in circuit switched telephone networks to measure the offered load to provide adequate service trunks/call lines, to minimize the number of blocked calls. However due to high time complexity of computing the blocking probability by the traditional Erlang formula, the analysis model that this research uses is an approximation of the Erlang formula. Beard (Beard, 2001) described the approximation to the Erlang formula for stressed ATM networks, and the results of the approximations are almost exact to the actual computations at peak/busy periods.

In a related article by Beard (Beard, 2001) , Beard derived a linear approximation equation for estimating probability of blocking for a class of network traffic. His work uses the method of upper limit policy to impose limits on the highest and lowest blocking probability influenced by each class of traffic, thus guaranteeing a minimum bandwidth during the busy period. This reserach uses this tele-traffic theory approximation equation for cluster computing systems to partition the system nodes based on node reliability characteristics, such that most of the resource classes in the system are serviced with a minimum blocking probability during busy periods.

2.3 System Reliability

The importance of cloud computing and cluster based systems is significantly increasing. There have been articles on improving reliability in distributed and cluster-based systems. The articles on improving reliability in heterogeneous distributed systems by Tang et.al (Tang, 2010) and Shatz et.al (Shatz, 1992), uses the concept of duplicating task modules on multiple components during task allocation. Their reliability analysis of the scheduling attributes keeps the hardware resources fixed and computes task paths and completion metrics.

Another related article by Tang et.al, (Tang, 2012) describes a hierarchical reliability driven scheduling approach in grid systems by implementing a local and a global scheduling algorithm in combination. The significant improvement in terms of system reliability, schedule length and speedup are depicted in graphs and the data for their analysis has been taken from real-time applications. This article measures task reliability by a reliability probability metric equal to the probability of all its data that successfully transfers from its immediate parent tasks and successful executions on the processor it is assigned to (Tang, 2012). The three articles listed above explains the effect of scheduling policy in terms of task scheduling modules, which includes Directed Acyclic Graphs (DAG), scheduling path, schedule length etc., but in this thesis the resource allocation policies and its effects on reliability are discussed.

Running tasks on high reliability nodes can save time and compute power, by minimizing checkpointing. Checkpointing is a fault tolerance strategy, where the progress and current state of the system is regularly saved at fixed intervals, called the checkpointing interval, so that the system is resilient to system failures. The optimal check pointing interval is given by Daly's checkpointing formula (Daly, 2003). This formula helps to measure the failure rates in the system and is a powerful factor for determining system reliability. High reliability nodes are usually assigned to long running tasks or tasks using large number of processors, so that the impact of system failure and overhead due to checkpointing can be minimized.

2.4 Summary

The review of literature has been able to raise probing questions, facilitating a deeper understanding of existing schedulers and the policies they follow along with their advantages and disadvantages. The study of previously published relevant articles have led to the following research questions:

- Does the resource allocation policy have a significant impact on reliability of jobs submitted to the system?
- Does blocking probability affect the reliability of jobs?
- What should be the ideal number of partitions and partition size for a system with a given workload to maximize reliability?
- Can awareness of node reliability be added in a scheduler to aid node selection decision?

With these questions in mind, the researcher provides an insight into the research framework and research methodology in the next chapter.

CHAPTER 3. PROCEDURES AND METHODOLOGY

This chapter discusses the theoretical framework and emphasizes the research methodology and design. The nature of research, hypothesis, sample set, variables and approach are also detailed in this chapter.

3.1 Research Methodology

The purpose of this research is to improve the overall reliability of jobs in the system by understanding the impact of scheduling and resource allocation policies when specified the observed reliability of nodes in the system. This research focuses on determining the effect of size of the provisioned cluster system on the probability of blocking and queue wait time for resource requests. This research also aims to estimate the optimal size of a system partition by selecting the best partitioning strategy, which is decided based on the offered workload during the peak/busy period and resource allocation policy used.

The goal of this developed methodology is to ensure that the overall reliability of the partition in terms of the partition MTTF (Mean Time To Failure) and reliability of jobs submitted to the system (especially the large and long-running jobs) increases significantly. This reliability guided scheduling approach is based on machine learning by analyzing historic workload and system failure log data. The improvement of this new methodology would be evaluated by comparing the results obtained from analysis and simulations. Simulations can help us understand the methodology in a controlled environment where parameters can be modified and the effect of the independent variables on the dependent ones can be studied. The nature of this research is quantitative, thus this research follows a systematic empirical investigation of the methodology using statistical, mathematical or computational techniques (Given, 2008).

3.2 Research Framework

In this section, the researcher describes the system and a few assumptions, which are used in this research. The total number of nodes in the system is N. On the request site the nodes are logically classified into R resource classes, where number of nodes n_i contained in each class is 2^{i-1} (Hacker & Mahadik, 2011). Each resource class can have a range of nodes bounded by a minimum $C_{r_{min}}$ and maximum $C_{r_{max}}$ number of nodes to service the offered incoming load during the busy period of system use. This range of the number of nodes that will need to be allocated to each resource class is computed based on a range of blocking probability values B_p in the interval $[0, 1]$. Every class has an associated average arrival rate λ_i and average service time per resource class is t_i. On the system side, the system can be divided into S partitions with the number of nodes in each partition C_s.

Most of the work on estimating blocking probabilities for connection admission control (CAC) policies is based on the Erlang loss function, which allows the researcher to exactly compute blocking for different policies under Markov connection arrival assumptions (Key, 1990), but this can be used only when networks are of medium size (less than 1000 units of capacity) (Beard, 2001). As systems grow large and powerful, approximations for the Erlang loss function are more helpful. In such large networks, load and capacity asymptotically approach infinity at a constant ratio of load to capacity greater than 1 (i.e., an overloaded condition). Beard's derivation (Beard, 2001) for approximating blocking probability, uses the complete partitioning (CP) policy, where every resource request can use the group of resources that have been rationed explicitly for that class.

As per Kelly's derivation (Kelly, 1986), the CP policy is considered to be equivalent to the complete sharing (CS) policy- where all system resources are completely shared among all resource classes- at overloaded conditions as the most likely state of the two policies coincide with each other at peak times. Kelly's derivation (Kelly, 1986) also justifies their equivalence by stating that the state space of the CP policy is a subset of the CS policy. Therefore, considering an asymptotically large network and using a combination of complete partitioning and complete sharing resource allocation policies, the results from Beard's derivation of Erlang loss function approximation to estimate probability of blocking resource requests are used in this research.

As derived by Beard (Beard, 2001) the blocking probability of each class r_i in the system is denoted as Bp_i, and can be represented by the following linear approximation

$$Bp_r = 1 - \frac{C_r}{\lambda_r b_r} \tag{3.1}$$

subject to

$$0 \leq C_r \leq \lambda_r b_r \tag{3.2}$$

and

$$N = \sum_{i=1}^{R} C_i \tag{3.3}$$

where C_r is number of nodes per resource class r. The average arrival rate for each resource class is denoted by λ_r and b_r is a constant such that

$$An \leq C$$

$$\begin{bmatrix} b_1 & 0 & \ldots & 0 \\ 0 & b_2 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & b_r \end{bmatrix} \begin{bmatrix} n_1 \\ n_2 \\ \vdots \\ n_r \end{bmatrix} \leq \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_r \end{bmatrix}$$

where the matrix A is defined by thresholds from complete partitioning policy (Beard, 2001).

The product $\lambda_r b_r$ denotes the offered workload during busy period for resource class r, where b_r is computed by the product of the number of processors per resource class n_i and the average service time t_r. Thus the offered load α_r per resource class r, measured in Erlangs, is given by $n_i * \lambda_i * t_i$. The average service time t_r and average arrival rate λ_r for every resource class r can be obtained easily from the analysis of computational workload logs. In order to completely satisfy the resource requests of a resource class, the number of nodes in that resource class r (C_r) must be equal to the offered load of that resource class. Using this approximation equation, the researcher determines the capacity of the system, for different values of blocking probability and for the same given workload.

For the reliability analysis, the expected reliability for every resource class is determined using the Weibull distribution described by Hacker (Hacker, 2010), as listed below. The probability of node failure is given by,

$$F(\Delta t, \beta, \tau) = 1 - e^{-\left(\frac{\Delta t}{\tau}\right)^{\beta}} \tag{3.4}$$

The reliability equation is complementary to the equation to compute probability of failure, thus reliability for any resource class i is given by

$$R(\Delta t, \beta, \tau) = e^{-\left(\frac{\Delta t}{\tau}\right)^{\beta}} \tag{3.5}$$

where τ is the scale parameter which is computed using,

$$\tau = \frac{MTTF}{\Gamma\left(1 + \left(\frac{1}{\beta}\right)\right)} \tag{3.6}$$

β is the shape parameter, and $[0, T]$ is the time interval over which the value is computed. In this research analysis this value is the measured node reliability for each resource class r, where $1 \leq r \leq R$.

Substituting Equation 3.6 in Equation 3.5,

$$R(T, \beta, \tau) = e^{-\left(T * \left(\frac{\Gamma\left(1+\left(\frac{1}{\beta}\right)\right)}{MTTF}\right)\right)^{\beta}} \tag{3.7}$$

Here, $R(T, \beta, \tau)$ is the reliability probability and is a real value between 0 and 1. For simplicity, this term is denoted as R_p.

By solving the Equation 3.7, an equation to compute the expected reliability is obtained as follows,

$$\left(\frac{T * \Gamma\left(1+\left(\frac{1}{\beta}\right)\right)}{MTTF}\right)^{\beta} = ln\left(\frac{1}{R_p}\right)$$

$$\left(\frac{T * \Gamma\left(1+\left(\frac{1}{\beta}\right)\right)}{MTTF}\right) = ln\left(\frac{1}{R_p}\right)^{\frac{1}{\beta}}$$

$$MTTF = \frac{T * \Gamma\left(1+\left(\frac{1}{\beta}\right)\right)}{\left(ln\left(\frac{1}{R_p}\right)\right)^{\frac{1}{\beta}}} \tag{3.8}$$

To obtain the observed reliability of nodes in the system in terms of Mean Time To Failure (MTTF), the system logs collected from the Coates compute cluster at Purdue University is used, by measuring the time between reboots of each node of the cluster as a proxy for time to failure for each node. Most system logs may not have failure information from the start of system installation. This makes the determination of MTTF over the operational lifetime of the system complex. The MTTF is usually the sum of the mean time between failures (MTBF) and mean time to repair (MTTR).

As many system logs fail to provide the MTTR information, the only information that is available from most real-time system logs is the MTBF. Therefore, this reliability analysis uses MTBF and MTTF interchangeably and considers the MTBF as the observed reliability of nodes.

3.3 Analysis Methodology

The first step of this analysis begins with the selection of a resource allocation policy to guide the allocation of nodes to job resource requests that suit most cluster and cloud systems. Some of the resource allocation policies common today are *complete sharing*, *complete partitioning*, *upper limit policy* and *guaranteed minimum* (Beard, 2001). The researcher explores each of these four policies to select a suitable policy for further analysis.

The complete sharing policy treats the entire system as a single common pool of computing resources, where any resource request submitted to the system can be admitted as long as there are free nodes available. Though this policy is simple to implement, this policy makes it difficult to limit the allocation of low reliability nodes to tasks that require high reliability. Thus this policy is not effective in managing the allocation of reliable nodes to jobs.

Next, the researcher evaluates the complete partitioning policy. In this policy, the entire system is divided into non-overlapping partitions. A partition is dedicated to each resource class, so that a submitted resource request is admitted into the system only if there are free nodes available in the partition dedicated to its resource class. The drawback of this policy is that if the partition becomes full, the incoming tasks are blocked or queued even if there are idle nodes available in other partitions. Thus, the blocking probability of tasks seeking resources or queue lengths for some resource classes increases significantly, which is not an agreeable situation, especially during peak workloads.

The researcher next evaluates the upper limit policy and guaranteed minima policies. In these policies, an upper and lower bound is placed on the number of resources that can be allocated to tasks admitted into the system. Once these bounds are reached, requests are blocked or queued even though there are idle nodes available in the system. Unfortunately, these two policies would force reservation of high reliability resources in the hope of admitting jobs demanding high reliability to those partitions, in which case, these nodes could have otherwise been used to improve overall reliability of lower reliability tasks.

After analyzing these resource allocation policies as discussed above, the researcher selected a combination of the complete sharing policy and the complete partitioning policy to take advantage of the benefits of both these allocation policies. Here, using the complete partitioning policy, one partition per resource class is allocated for the high reliability demanding resource classes, with each partition having enough nodes so as to minimize the probability of blocking resource requests. The other resource classes which do not have dedicated partitions of their own are first directed towards the low reliability partition and based on availability of nodes are allocated to the higher reliability partitions. Within each partition, the nodes can be shared and are allocated to resource requests using the complete sharing policy.

Conversely if a task requesting the highest reliability nodes finds the partition full, the task is blocked or queued until the high reliability nodes are available, even if there are low reliability nodes idle at that time. Using this approach, one ensures that the jobs that need high reliability nodes will be allocated nodes with comparable or greater reliability and at the same time the jobs with low reliability demand can also be allocated the high reliable system nodes if those nodes are idle. Using this approach improves the overall reliability of all jobs submitted to the system.

Once the resource allocation policy is selected, the second step is to compute the reliability need of every resource class based on the service time and resource class size. The researcher determines an expected reliability range (in MTTF, units in hours) for every resource class in the system, using the reliability equation (Equation 3.6). This is the reliability range for every task submitted to the corresponding resource class within a range of reliability a minimum reliability of $R_{p_{min}}$ to a maximum reliability of $R_{p_{max}}$

In the third step, using Beard's approximation of the Erlang loss function in Equation 3.1, the minimum number of nodes $C_{r_{min}}$ and maximum number of nodes $C_{r_{max}}$ required to satisfy a given workload is derived. This range of number of nodes per resource class depends on a range of blocking probability values that can be set by the system administrator based on the incoming offered workload. The minimum value for blocking probability is 0 (i.e. no job requests are blocked), and maximum value is B_p which can be a real number between 0 and 1 (i.e. $(B_p * 100)\%$ of tasks are blocked due to insufficient resources).

The ideal system size is the total number of nodes per resource class $C_{r_{max}}$ when blocking probability is 0. There will be adequate resources in the system to satisfy all resource requests at peak load at this system size. Therefore, this equation is an important tool to help cluster-based and cloud system providers decide the maximum system size that should be provisioned to handle large incoming workloads at peak periods.

Fourth, after determining $C_{r_{max}}$ and the expected reliability for each resource class, the resource classes are then sorted in descending order of their reliability need. To determine the resource class ranking, a cost function for each resource class is computed, which is the product of service time t and resource class size $(n_i t_i)$. The classes with higher computed cost are the classes whose jobs have higher service time and require large number of nodes to execute. In other words, the resource classes with higher rankings are affected the most when node failures occur.

Fifth, the equations used to determine the number of nodes per resource class C_r, can be generalized to a set of four different scheduling policies. Most cluster-based computing systems today use one of these four scheduling techniques depending on the applications they execute and the number of resources that they can provide. These policies, described in detail by Hacker (Hacker & Mahadik, 2011), use blocking or queuing techniques to allocate resources in the system based on node availability. The sum of the number of resources for each partition represent the entire system size needed to provide the desired probability of blocking for each resource partition. These scheduling strategies are-

- Type I - All or nothing allocation policy

  The Type I scheduling model uses the 'all or nothing' policy, to allocate resources. The resources are either allocated immediately or completely blocked if resources are not available in the system. Most cloud computing systems that provide on demand software and infrastructure as a service use this policy (Hacker & Mahadik, 2011). The number of nodes that would be required per resource class for a system using the Type I scheduling is given by rearranging the terms Equation 3.1,

$$B_{p_r} = 1 - \frac{C_r}{n_r \lambda_r t_r} \tag{3.9}$$

To determine the range of the number of nodes required to service a request of resource class r, where $1 \leq r \leq R$, a probability of blocking $B_p = 0$ is used to compute the ideal number of nodes needed to satisfy all requests within a resource class,

$$B_{p_r} = 1 - \frac{C_r}{n_r \lambda_r t_r} = 0$$

Thus,

$$C_r = n_r \lambda_r t_r \tag{3.10}$$

where C_r is the total number of nodes needed by resource class r, n_r is the resource class size, t_r is the average service time for class r, and λ_r is the average arrival rate for each resource class.

To compute the number of nodes in any other case where blocking probability B_p is non-zero, using

$$B_{p_r} = 1 - \frac{C_r}{n_r \lambda_r t_r} = B_p$$

Thus,

$$C_r = (1 - B_p) * n_r \lambda_r t_r \tag{3.11}$$

From Equation 3.10 and Equation 3.11 the range of the number of nodes for all resource classes r is obtained, where $1 \leq r \leq R$, which can be the generalized representation for Type I scheduling policy for resource class capacity which is offered a blocking probability B_p,

$$C_{r_{min}} \leq C_r \leq C_{r_{max}}$$

$$(1 - B_p) n_r \lambda_r t_r \leq C_r \leq n_r \lambda_r t_r \tag{3.12}$$

Using Equation 3.12 different values of $C_{r_{min}}$ for different values of blocking probabilities B_p are computed. This equation can be used to determine the maximum number of nodes required in every resource class during peak load, needed to service all job requests as well as the number of nodes required with a non-zero blocking probability B_p associated with each resource class.

- Type II - Partial allocation with blocking

  A Type II scheduling strategy allows complete or partial allocation. This means that a partial set of requested resources are satisfied while the remaining fraction of resource requests are denied (Hacker & Mahadik, 2011).

  A Type II scheduling policy allocates a fraction of requested resources and denies the remainder without queuing. The equations used in Type I scheduling model can be extended for Type II schedulers. In the event that a resource request is blocked with a blocking probability B_{p_r} for a class r, then the task may respond with request for a smaller allocation satisfied with lower reliability nodes.

  The resource classes are ranked in the descending order of computed expected reliability, where each resource class's reliability need depends on the service time and the size of resource request. Thus, when a resource request for a resource class r is denied, the request is directed to the next resource class $(r-1)$ in the ranked list, which has lower expected reliability than nodes in resource class r. The resource class $(r-1)$ may have same or different blocking probability Bp_{r-1}.

  Using Beard's linear approximation (Beard, 2001) of Erlang B formula for computing the blocking probability, the offered load α_r for every class is the sum of the offered load of the class r and load that was blocked by the previous resource class. To compute the offered load for each of the resource classes $1 \leq r \leq R$ following equation is derived,

$$\alpha_r = n_r\lambda_r t_r + Bp_{r-1}(n_{r-1}\lambda_{r-1}t_{r-1} + Bp_{r-2} \\ (n_{r-2}\lambda_{r-2}t_{r-2}\cdots + Bp_2(n_2\lambda_2 t_2 + Bp_1 n_1\lambda_1 t_1))) \tag{3.13}$$

Once the offered load for each class is obtained, the number of nodes required per resource class C_r for each resource class is computed iteratively using,

$$C_r = (1 - Bp_r) * \left(n_r\lambda_r t_r + \frac{Bp_{r-1}}{(1 - Bp_{r-1})}C_{r-1}\right) \tag{3.14}$$

where $1 \leq r \leq R$ and initial condition for the iteration $C_0 = 0$.

For each resource class r, the number of nodes required when blocking probabilities are 0 is computed using the following equation.

$$C_r = n_r\lambda_r t_r \tag{3.15}$$

This equation is similar to the linear approximation equation for Type I scheduling policy (Equation 3.10), which infers that the maximum number of nodes for each resource class, at blocking probability $B_p = 0$, is same for all the four scheduling policies, implying that system administrators can use this value to determine the maximum capacity of a system that can be provisioned to service busy periods for a given workload.

- Type III - Partial allocation and waiting

  The scheduling model of Type III provides the option to partially fulfill the request immediately and add the remaining part of a resource request to a queue instead of blocking the resource requests (Hacker & Mahadik, 2011). As the Type III scheduling policy maintains a queue for partially allocated resources, the probability of queuing P_c, the queue waiting time W and the length of queue F for this policy are also computed.

The Type II scheduling policy equations for computing number of nodes per resource class can be extended to Type III scheduling policy as both these policies use the concept of partial request allocation.

$$C_r = (1 - Bp_r) * \left(n_r \lambda_r t_r + \frac{Bp_{r-1}}{(1 - Bp_{r-1})} C_{r-1} \right) \tag{3.16}$$

where $1 \leq r \leq R$ and $C_0 = 0$.

Using Equation 3.16 iteratively (which is the same as Equation 3.14), the number of nodes for each resource class C_r is computed. Further, using the Erlang C formula and considering $\alpha_r = \lambda_r t_r$, the equation to compute the probability of queuing is derived using,

$$P_{c_r} = \frac{Bp_r}{1 - \left(\frac{\alpha_r}{C_r}\right)(1 - Bp_r)} = \frac{C_r Bp_r}{(C_r - \alpha_r) + (\alpha_r Bp_r)} \tag{3.17}$$

where α_r is the traffic intensity of resource class r and C_r is the number of system nodes required per resource class, computed from Equation 3.16. Statistical equilibrium is obtained only for $\alpha_r < n$, else the queue increases towards infinity (Hacker & Mahadik, 2011).

Once the probability of queuing for each resource class is obtained using Equation 3.17, the average queue length F described by Zeng (Zeng, 2003) is obtained using the following equation,

$$F_r = \frac{\alpha_r}{(C_r - \alpha_r)} Pc_r \tag{3.18}$$

The mean waiting time W for the jobs in queue is computed after the probability of queuing Pc_r is obtained by Little's Law,

$$W_r = Pc_r \frac{t}{(C_r - \alpha_r)} \tag{3.19}$$

where t is the mean service time for the resource class r and α_r is the traffic intensity of each resource class.

- Type IV- Queue based allocation

The Type IV scheduling strategy uses the complete queuing model where all requests that require complete or partial allocation of resources are added to queues. Most traditional HPC systems use this strategy for their resource allocations where the entire request is queued until resources are made available. The equations derived for Type IV use equations derived for Type I and Type III scheduling policies. In this scheduling policy, the job requests that do not have resources available immediately are added to a queue.

Following the derivation of Mahadik and Hacker (Hacker & Mahadik, 2011), the number of nodes per resource class C_r using the Erlang B approximation formula derived for Type I scheduling policy is determined.

$$C_r = (1 - B_p) * n_r \lambda_r t_r \tag{3.20}$$

Further, using the Erlang C formula and considering $\alpha_r = \lambda_r t_r$, the equation to compute the probability of queuing is derived.

$$Pc_r = \frac{Bp_r}{1 - \left(\frac{\alpha_r}{C_r}\right)(1 - Bp_r)} = \frac{C_r Bp}{(C_r - \alpha_r) + (\alpha_r Bp)} \tag{3.21}$$

where α_r is the traffic intensity of resource class r and C_r is the number of system nodes required per resource class, computed from Equation 3.20. Statistical equilibrium is obtained only for $\alpha_r < n$, else the queue increases towards infinity (Hacker & Mahadik, 2011). The queue length and the mean wait time can be computed using Equations 3.18 and 3.19.

The next step in this analysis determines the size and number of partitions in the system. The number of partitions in the system S is the number of higher ranking resource classes which have been assigned C_s number of nodes, where C_s is the maximum of $C_{r_{max}}$ and resource class size n. When job requests of these higher ranking resource classes are submitted to the system, the scheduler directs these requests to the partition dedicated to them for resource allocation. For resource requests of other resource classes with lower expected node reliability that have not been allocated partitions of their own, the resource requests will first be directed to the lowest reliability partition in the system. If the requested partition is full then the scheduler directs their request to the immediately next resource partition with higher reliability. The scheduler checks for idle nodes in the subsequent partitions with higher reliability, until the scheduler finds available resources in higher reliability partitions or finds all partitions full. If all the high reliability partitions are full, the request is blocked or queued until the high reliability nodes are available, even though there are few low reliability nodes idle. This strategy ensures that resource requests of every class are scheduled on nodes that have comparable or higher reliability than requested, thus improving overall job reliability.

The next step is to determine the sorted list of physical nodes, ranked by their observed reliability. For this, the researcher assesses the reliability of nodes based on observations of node reliability derived from historic system logs. These values are the mean time between failures for each system node. The nodes are ranked in the descending order of this measured MTBF (in hours). Based on node reliability history, Hacker and Romero (Hacker & Romero, 2009) describe an efficient reliability estimation approach based on a discrete semi-markov model that can also be used to estimate node MTTF.

The final step in this analysis is to map each physical node to a resource class. Based on the resource class ranking computed using the resource cost function, C_s system nodes are distributed to each partition starting from highest ranking resource class, moving down the ranks as long as nodes exist in the reliability-wise sorted pool of resources.

In this analysis, there are S system partitions, each partition s where $0 \leq s \leq S$ has an associated weight W_s, which is the total number of nodes in the partition needed to service the given workload at a specified blocking probability B_p. Each system partition has an associated rank based on the reliability need of resource classes submitted to the partition. The goal is to maximize the average reliability of the partition using the maximizing condition given in Equation 3.22.

Thus the problem is to maximize

$$MTTF_s = \frac{\sum_{i=1}^{C_s} M_r}{C_s}$$

Subject to the constraint

$$0 \leq W_s \leq C_s \tag{3.22}$$

An algorithm has been formulated that distributes N system nodes into S system partitions with each partition s of size C_s, such that the average partition reliability (in MTTF) is maximized.

This algorithm takes as input a set of **N** system nodes and an array **LIST** of size **N**, which stores the associated reliability $\boldsymbol{M_i}$ of each node. The system is divided into **S** partitions. An array **SIZE** of size **S** stores the desired size of each partition. $\boldsymbol{C_s}$ computed using Equation 3.11, 3.14 or is selected by an administrator (depending on the desired allocation policy for each resource class). The output of the algorithm is a list **MTTF** of size **S**, representing the computed average mean time to failure of each partition, sorted in descending order of computed average partition reliability such that

$$MTTF[1] \geq MTTF[2] \geq ... \geq MTTF[S]$$

where partition 1 corresponds to the highest reliability partition and partition S corresponds to the least reliability partition.

TEMPORARY VARIABLES: **i** (iteration variable) , **j** (iteration variable) , **temp** (temporary assignment for swapping values), $\boldsymbol{C_{rem}}$ (the current number of system available nodes), **offset** (marker variable to indicate first node of each partition), **sum** (stores the sum of reliabilities of all nodes in a partition)

```
1. begin pseudocode
2.     //sort the list of node reliabilities in descending order
3.     for i = 1 to N do
4.         for j = i to N do
5.             if LIST[ i ] $\leq$ LIST[ j ] then
```

```
6.                  temp ← LIST[ i ]
7.                  LIST[ i ] ← LIST[ j ]
8.                  LIST[ j ] ← temp
9.              end if
10.           end for
11.       end for
12.       //compute the average reliability of each partition
13.       $C_{rem}$ ← N
15.       offset ← 0
16.       for i = 1 to S do
17.           if $C_{rem} \geq$ SIZE[ i ] then
18.               sum ← 0
19.               for j = 1 to SIZE[ i ]
20.                   sum ← sum + LIST[ j + offset ]
21.               end for
22.               MTTF[ i ] ← sum / SIZE[ i ]
23.               offset ← offset + SIZE[ i ]
24.               $C_{rem}$ ← $C_{rem}$ - SIZE[ i ]
25.           end if
26.       end for
27.   end *pseudocode*
```

This algorithm has a total time complexity $O(N^2) + O(N)$, where N is the total number of nodes in the system. This is the total time required to sort all system nodes N in the descending order of their associated node reliabilities, assign C_s nodes to each partition and then compute the average mean time to failure for each of the S partitions.

3.4 Variables

To assess the effects of probability of blocking requests for each resource class, the researcher uses a set of blocking probability values such that $B_p \epsilon [0, 1]$, i.e. $B_p = \{0, 0.1, 0.2, 0.3, 0.4\}$, to compute the number of nodes per resource class C_r for each resource classes and for all scheduling policies. The maximum number of nodes needed per resource class $C_{r_{max}}$ is computed for a blocking probability $B_p = 0$. In addition, the probability of queuing, queue waiting time, queue length is computed for scheduling policies Type III and Type IV using the same set of blocking probability values. Here the blocking probability B_p is an independent variable used to compute dependent variables the number of nodes per resource class C_r, probability of queuing P_{c_r}, the average queue wait time W_r, and queue length F_r.

The offered workload for the analysis is the computational workload log submitted to the ANL BlueGene Intrepid supercomputer. These job submission logs help derive the arrival times and mean service times of the submitted jobs for each resource class.

The mean time to failure information for the system nodes, is the observed reliability obtained from system logs. The scheduler attempts to assign hardware based on the expected reliability need of jobs submitted to the system. The expected reliability for every resource class is a dependent variable computed based on the average service time t_r and the size n of each resource class.

3.4.1 Hypothesis

The null and alternate hypothesis for this research is as follows:

$H1_o$: There is no improvement in reliability for large and long-running jobs by shifting assignment of high reliability nodes to these resource requests.

$H1_a$: There is an increase in reliability of jobs with high reliability need by assignment of high reliability nodes to these resource requests.

$H2_o$: There is no change in system efficiency, as the total checkpointing operation cost of the system nodes does not change after using the new methodology.

$H2_a$: There is an increase in system efficiency due to decrease in total checkpointing operation cost of all system nodes after using the new methodology.

The statistical analyses to validate these hypotheses are shown in Appendix A.

3.4.2 Measure of Success

For the first hypothesis there would be an increase in reliability of jobs with high reliability need by assignment of high reliability nodes to these resource requests. To measure the success of this hypothesis, the reliability of jobs of these high reliability-demanding jobs is measured in MTTF (in hours). If the reliability of jobs after using the new node assignment is greater than the job reliability before using this approach, the null hypothesis can be rejected. The acceptance of the first alternate hypothesis indicates that the new resource allocation policy described in this research is successful in improving reliability of the jobs with high reliability demand.

For the second alternate hypothesis to be true, the total cost of checkpointing operation before using the new approach should be greater than the total cost of checkpointing operation after using reliability guided resource allocation. To measure this cost, the frequency of checkpointing operations is computed using Daly's optimal checkpointing interval, before and after applying the new methodology. Then the total time spent in checkpointing, the checkpointing operation cost, is calculated for all system nodes in units *node-min.*). If the computed cost before using reliability awareness is greater than the computed cost after using this methodology, the new technique described in this research is successful in improving overall system reliability.

3.5 Summary

This chapter explained the framework, design methods, approach and the assessment instruments useful for this research. The next chapter details the evaluation results from the analytical model.

CHAPTER 4. EVALUATION OF ANALYSIS METHODOLOGY

4.1 The System and Workload Description

In order to understand the effect of this developed methodology on reliability and probability of blocking, the researcher evaluated this approach on a hypothetical cluster using historic logs from large-scale supercomputers to study the overall impact of this approach on real systems. To keep this analysis as real as possible, the computation workload and Reliability Availability and Serviceability (RAS) logs used in this research are of the ANL Blue Gene/P system *Intrepid* (ANL Intrepid Log, 2009). To understand the observed system node reliability pattern, the system failure logs of the Coates cluster at Purdue University are analyzed.

The reference system in this research is comprised of a 40960 uni-processor cluster with 2TB total memory. The computation workload used in this analysis contains 8 months of accounting records of all jobs submitted to the *Intrepid* from January 1, 2009 and finished before September 1, 2009. These logs mainly consist of jobs for scientific and engineering computing applications. As the minimal partition size on the system *Intrepid* is 64 nodes, the smallest resource class in this reference system is used as 64 (ANL Intrepid Log, 2009).

The fields of interest observed from the log file are:

- Job number
- Submit time (in seconds)
- Running time (in seconds)
- Requested number of processors

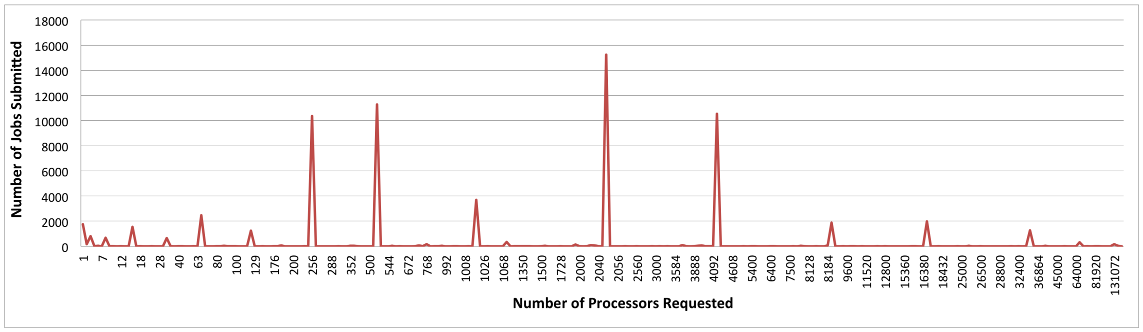


Figure 4.1.: Load distribution based on resource requests

Analyzing the values in the log, a graph of the number of processors requested versus the number of jobs submitted (these values are derived from the log) as shown in Figure 4.1 is plotted based on the data shown in Table 4.1. The researcher observed that 95.89% of the 68936 resource requests submitted to the system obey the pattern of powers of two, which verifies one of our fundamental assumptions, thus enabling us to make logical partitions called resource classes as powers of two.

The computed parameters shown in Table 4.2 include the average arrival rate λ, the average service time t per resource class and the total number of jobs J submitted to the resource class. Table 4.2 represents the stochastic workload parameters derived from Intrepid's scheduling logs. The parameters computed here are for the period from 1^{st} April, 2009 to 15^{th} June, 2009. This peak load period has been provided in the documentation on the Parallel Workloads Archive website (ANL Intrepid Log, 2009) in the graph of offered load (on y-axis) versus time in months (on x-axis).

4.2 Reliability Computation

After computing the workload parameters for the resource classes, the expected reliability for each class is computed using (Hacker & Meglicki, 2007)

$$MTTF = \frac{T * \Gamma\left(1 + \left(\frac{1}{\beta}\right)\right)}{(ln(1/R_p))^{(1/\beta)}} \tag{4.1}$$

Table 4.1: Distribution of Number of Requested Processors from ANL-Intrepid logs

Number of Processors Requested	Number of Jobs Submitted	Percentage of total jobs
1	1749	2.54
2	179	0.26
4	821	1.19
8	684	0.99
16	1558	2.26
32	647	0.94
64	2464	3.57
128	1242	1.80
256	10381	15.06
512	11284	16.37
1024	3686	5.35
2048	15239	22.11
4096	10540	15.29
8192	1882	2.73
16384	1981	2.87
32768	1268	1.84
65536	328	0.48
131072	171	0.25
Total	66104	95.89

Table 4.2: Workload parameters

Resource Class	Resource Class Size (n)	Average Service Time t (hrs)	Average Arrival Rate λ (1/hrs.)	Number of Jobs Submitted (J)
1	64	0.476	4.327	8102
2	128	0.257	0.936	1242
3	256	0.347	0.699	10381
4	512	2.074	3.912	11284
5	1024	1.608	2.239	3686
6	2048	2.152	1.238	15239
7	4096	1.679	0.726	10540
8	8192	1.999	0.690	1882
9	16385	1.207	0.549	1981
10	32768	0.896	0.510	1268
11	40960	0.504	0.625	499

where MTTF is the mean time to failure, β is the shape parameter of the Weibull reliability distribution, R_p is the reliability probability and T is the service time of 95% of the jobs of each resource class.

The value of β is obtained from field data and taken as 0.7 (Hacker, 2010). The expected reliability MTTF is computed for a range of reliability probabilities $R_p \, \epsilon \, \{0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95, 0.99\}$, as shown in Table 4.3. This computed value is the reliability needed for any node in the system to execute a job completely, across N nodes to achieve a reliability of R_p. This also implies that the average MTTF of the nodes that could be allocated to the job (across all the nodes in the partition) need to be greater than or equal to the computed MTTF for a given reliability R_p.

In the next step the resource classes are sorted based on the resource cost function, which is the product of service time T and the resource class size n. This ranking represents the relative reliability need of the resource classes based on the number of processors and running time needed for jobs assigned to each resource class. Table 4.4 lists the computed resource cost and the resource class ranks. Higher resource costs indicate greater reliability need for the resource class.

Next, the number of nodes needed per resource class to satisfy the offered workload is determined as described in Table 4.2 constrained by a blocking probability B_p for the four scheduling policies using Equation 3.11 for Type I and Type IV and using Equation 3.14 for Type III and Type IV. In all these computations a range of blocking probabilities $B_p \epsilon \{0, 0.1, 0.2, 0.3, 0.4\}$ respectively is used to understand the minimum and maximum capacity of the system.

Figure 4.2 depicts the number of nodes per resource class needed to achieve the blocking probability B_p for the Type I and Type IV scheduling policies. This graph also displays the maximum system capacity required at different blocking probabilities (B_p). Similarly Figure 4.3 shows the number of nodes per resource class for the Type II and Type III scheduling policies along with the maximum system capacity for the specified blocking probability.

Table 4.3: Node MTTF required to achieve job reliability R_p for each resource class

Class Size n	Time T (in hrs.)	Node MTTF for R_p=0.6	Node MTTF for R_p=0.65	Node MTTF for R_p=0.70	Node MTTF for R_p=0.75	Node MTTF for R_p=0.80	Node MTTF for R_p=0.85	Node MTTF for R_p=0.90	Node MTTF for R_p=0.95	Node MTTF for R_p=0.99
64	1.007	212.98	271.69	355.79	483.69	695.31	1093.61	2031.22	5680.06	5.83E+04
128	0.878	371.39	473.77	620.42	843.45	1212.47	1907.03	3542.03	9904.86	1.02E+05
256	1.014	857.84	1094.31	1433.05	1948.20	2800.56	4404.85	8181.36	2.29E+04	2.35E+05
512	8.37	1.42E+04	1.81E+04	2.37E+04	3.22E+04	4.62E+04	7.27E+04	1.35E+05	3.78E+05	3.88E+06
1024	7.139	2.42E+04	3.08E+04	4.04E+04	5.49E+04	7.89E+04	1.24E+05	2.30E+05	6.44E+05	6.61E+06
2048	7.727	5.23E+04	6.67E+04	8.74E+04	1.19E+05	1.71E+05	2.69E+05	4.99E+05	1.39E+06	1.43E+07
4096	7.041	9.53E+04	1.22E+05	1.59E+05	2.16E+05	3.11E+05	4.89E+05	9.09E+05	2.54E+06	2.61E+07
8192	10.701	2.90E+05	3.70E+05	4.84E+05	6.58E+05	9.46E+05	1.49E+06	2.76E+06	7.73E+06	7.93E+07
16384	6.013	3.26E+05	4.15E+05	5.44E+05	7.39E+05	1.06E+06	1.67E+06	3.10E+06	8.68E+06	8.91E+07
32768	3.005	3.25E+05	4.15E+05	5.44E+05	7.39E+05	1.06E+06	1.67E+06	3.10E+06	8.68E+06	8.91E+07
40960	1.274	1.72E+05	2.20E+05	2.88E+05	3.92E+05	5.63E+05	8.85E+05	1.64E+06	4.60E+06	4.72E+07

Table 4.4: Reliability cost function

Resource Class	Resource Class Size	Service Time T (in hrs.)	Cost Function (node hrs.)	New Class Rank
1	64	1.007	64.448	11
2	128	0.878	112.384	10
3	256	1.014	259.584	9
4	512	8.37	4285.440	8
5	1024	7.139	7310.336	7
6	2048	7.727	15824.896	6
7	4096	7.041	28839.936	5
8	8192	10.701	87662.592	3
9	16384	6.013	98516.992	1
10	32768	3.005	98467.840	2
11	40960	1.274	52183.040	4

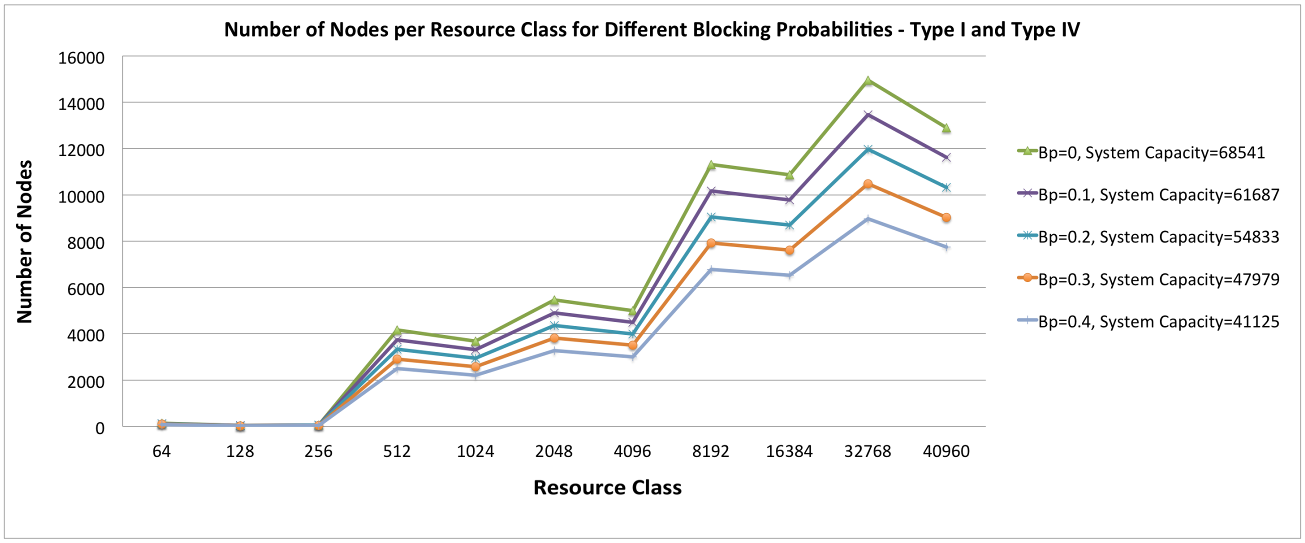


Figure 4.2.: Number of nodes for each resource class for Type I and Type IV needed to achieve blocking probability B_p

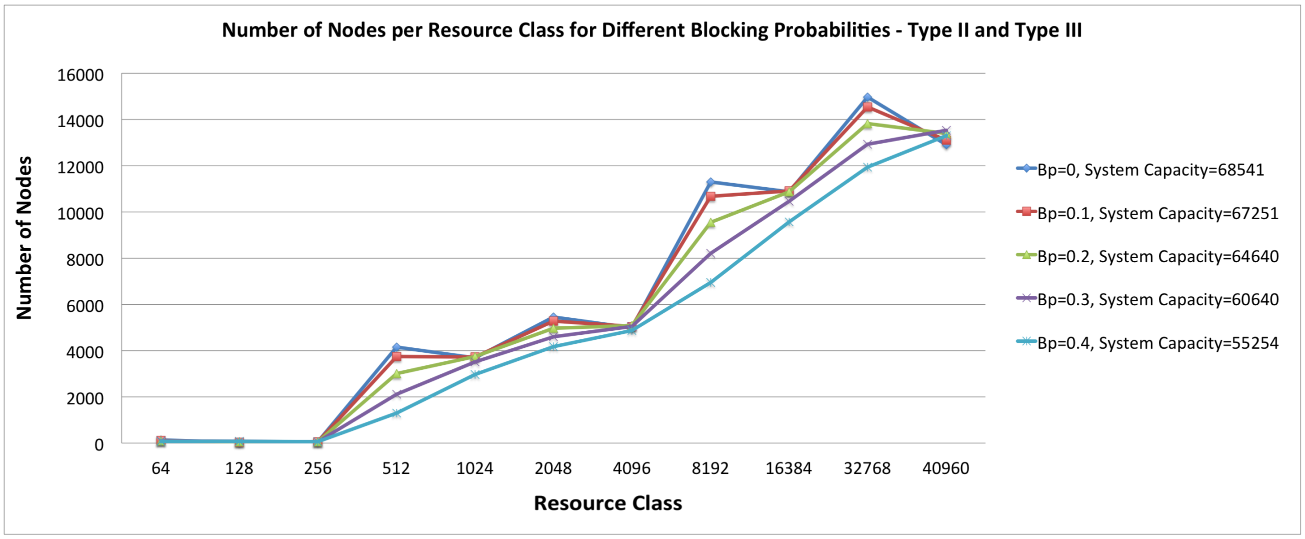


Figure 4.3.: Number of nodes for each resource class for Type II and Type III needed to achieve blocking probability B_p

An important inference from Figures 4.2 and 4.3 is that, as the probability of blocking requests increases for a resource class, the number of nodes required by the resource class decreases. This observation leads to infer that the blocking probability B_p and the number of nodes per resource class C_r have an inverse relationship. This is also verified by observing the minus sign in Equations 3.11 and 3.14, which represents a linear relation between B_p and C_r with a negative slope. In systems having different blocking probabilities for different resource classes, an increase in blocking probability in a higher ranking resource class increases the job reliability in lower resource classes, because the higher reliability nodes in the system will now be available and can be assigned to resource requests from lower resource classes.

Once the number of nodes per resource class C_r is obtained for all the four scheduling policies, the queuing characteristics for Type III and Type IV scheduling policies are computed. The three main queuing characteristics include probability of queuing P_c, the queue length F and the queue waiting time W. The graphs in Figure 4.4 and Figure 4.5 show that the effect of blocking probability on the probability of queuing a task in the queue, the queue length and the queue waiting time for a range of blocking probabilities are directly related. This indicates that, as the blocking probability B_p of a resource class increases, the number of resource requests that would be satisfied decreases and forces job requests to be added to the system queue. Further, more jobs in the queue results in larger queue lengths and consequently impacts the waiting time of jobs in the queue.

The next step is to determine the number of system partitions S and the size of each partition C_s. To find the size of each partition, the number of nodes needed per resource class is computed at a specified blocking probability B_p. Table 4.5 lists the maximum number of nodes $C_{r_{max}}$ i.e. C_r at blocking probability $B_p = 0$, the total number of nodes needed per resource class C_r for blocking probabilities 0.1 and 0.2 for the four scheduling policies and the new class ranking based on computed resource cost. For Type I and Type IV scheduling policies, using Equation 3.11 the number of

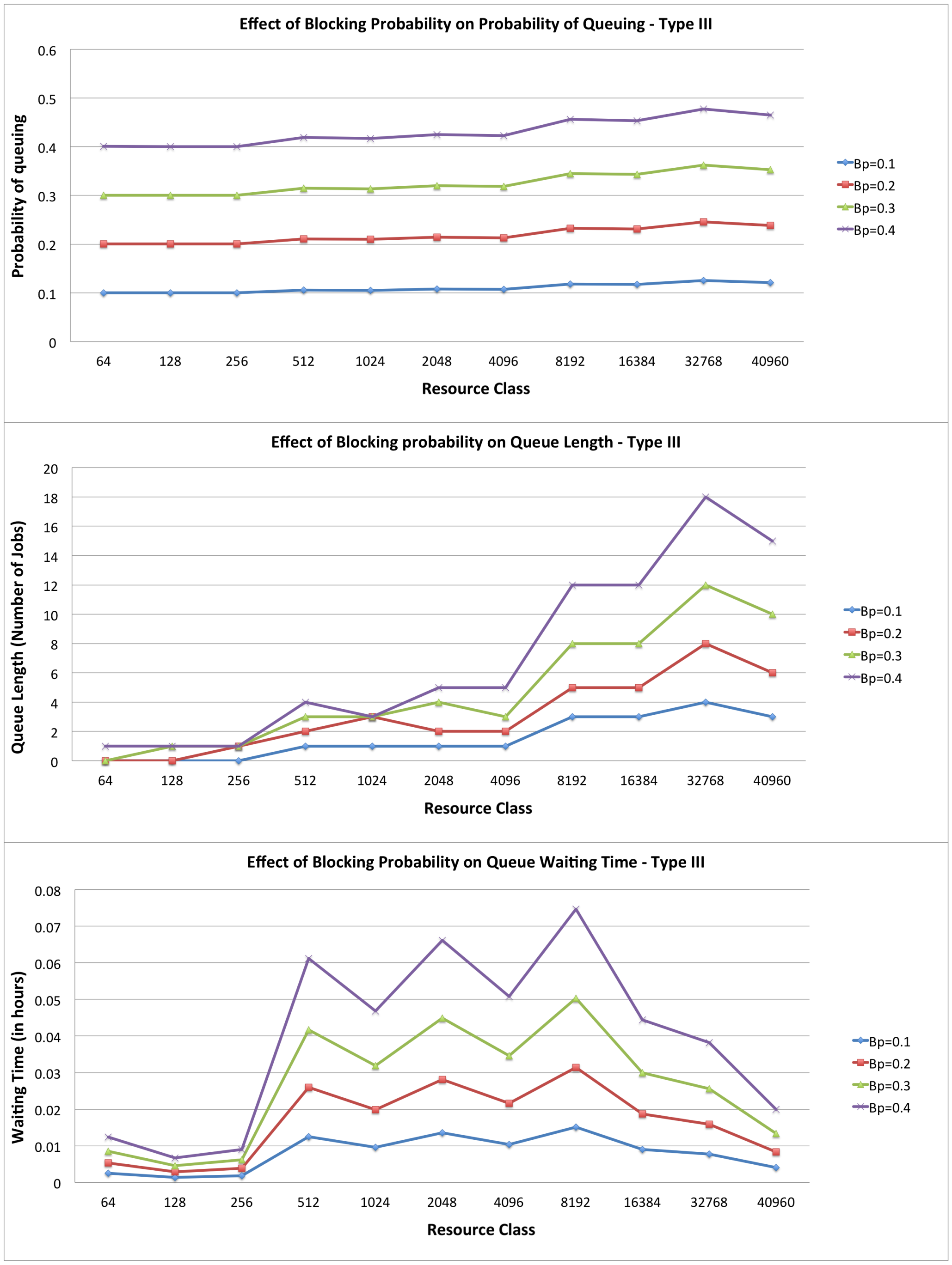


Figure 4.4.: Effect of Blocking Probability on Queue Characteristics for Type III

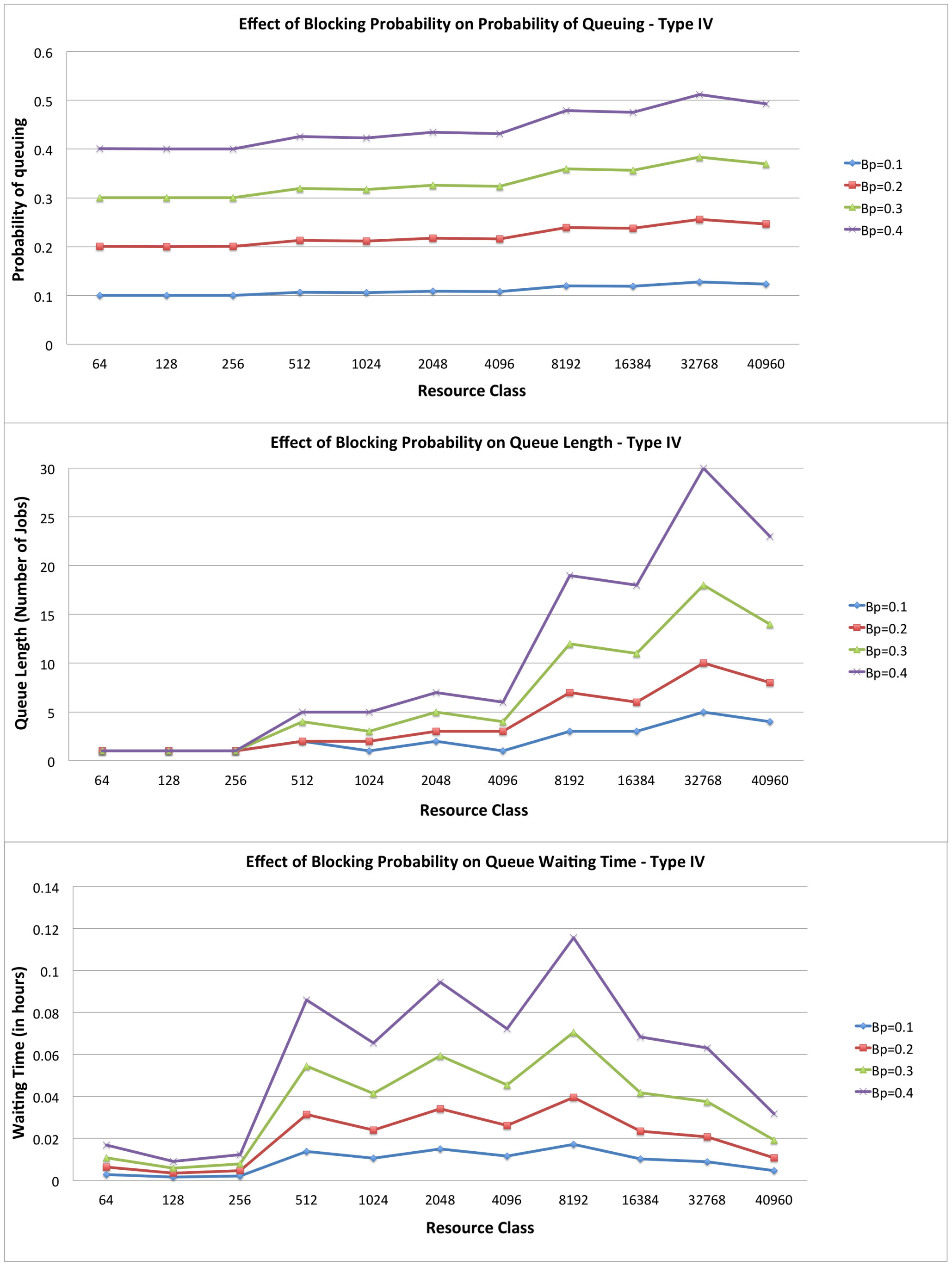


Figure 4.5.: Effect of Blocking Probability on Queue Characteristics for Type IV

resources per resource class C_r is computed and for Type II and Type III Equation 3.14 are used.

The C_r values in Table 4.5 are the number of nodes required by a resource class in a system using a pure complete sharing resource allocation policy, in which case all resources are shared and resource requests can be accepted into the system as long as there are system nodes available. In this research, the new resource allocation policy is used which is a combination of complete sharing and complete partitioning policy. Thus the number of nodes in each partition C_s is the maximum of number of nodes required by the resource class C_r at the specified blocking probability B_p or the resource class size n. Hence, the total number of partitions in the system S is the total number of resource classes that have been assigned nodes. As the total number of nodes per system partition C_s varies based on blocking probability B_p, the number of system partitions S also varies.

Note that the allocation in Table 4.5 is based on a reference system size of 40960 with an offered workload during the busy period distilled from a system. In some cases, the offered workload for some of the partitions is low and the partition size is less than the resource class size. These partition sizes can be scaled up as a function of the overall system size, assuming that the aggregate stochastic reliability characteristics of the increased node population are similar. If there is a need to increase the partition size for a resource class, the overall system size can be scaled up by the same fraction to provide a larger population of nodes to assign to the resource class. Since the added nodes (assuming a sufficiently large number of new nodes) are samples from a large population, the distribution of node MTTFs should follow the distribution of the entire system. Thus, as the overall system size increased, partition sizes can be scaled up, and if the sum of the partition sizes is less than overall system size, the scheduler will be able to avoid scheduling the worst nodes to any partition.

Next, the analysis proceeds to the mapping of resource classes to the actual physical nodes. The larger resource classes and long-running jobs usually demand high reliability nodes in the system, as the impact of system failures on these applications is drastic. System failures can increase overall runtime and checkpointing effort needed for long running jobs. Therefore, it is preferable to assign nodes with higher inherent reliability to the higher ranked resource classes. The node failure patterns from the system logs of the Coates Cluster at Purdue University is observed and the researcher assumes that the node failure pattern for the reference system of 40960 nodes follows that of Coates. As the average Mean Time to Failure obtained from the system logs is 10250.67 hours, the hypothetical cluster is also considered to have system MTTF as 10250.67 hours.

To understand the effect of blocking probability on partition reliability, the assigned reliability of each partition at blocking probability $B_p = 0.1$ and $B_p = 0.2$ is observed. Using the algorithm described in the analysis section of this reserach, the 40960 cluster nodes are distributed among each resource class such that each resource class is assigned C_s system nodes. Tables 4.6 and 4.7 list the system partitions, the partition sizes and their corresponding average mean time to failure for the four scheduling policies at blocking probability $B_p = 0$, $B_p = 0.1$ and $B_p = 0.2$. Separate partitions are not assigned for the two largest resource classes of size 32768 and 40960 as they require a minimum of all system nodes for each job. Since these two large partitions would select from the entire system, the node assignment begins with the most reliable node in the system for the 16384-resource class size.

From these tables it is evident that as the probability of blocking increases the reliability of resource classes also increases. However, in further analysis and computations the partition size and the assigned reliability corresponding to a blocking probability $B_p = 0$ is used, so that the minimum reliability improvement this methodology can guarantee for the given workload can be observed.

4.3 Analysis Results

After the new assigned reliability for every resource class is obtained, which is achieved by directing the resource classes to reliable partitions, the overall reliability

Table 4.5: Number of required Nodes per Resource Class for different blocking probabilities based on new class ranks

Resource Class	Resource Class Size	$C_{r_{max}}$ $B_p = 0$	C_r at $B_p = 0.1$ Type I & Type IV	C_r at $B_p = 0.1$ Type II & Type III	C_r at $B_p = 0.2$ Type I & Type IV	C_r at $B_p = 0.2$ Type II & Type III	New Class Rank
1	64	132	119	119	105	105	11
2	128	31	28	41	25	56	10
3	256	62	56	59	50	55	9
4	512	4154	3739	3745	3323	3008	8
5	1024	3688	3319	3735	2950	3737	7
6	2048	5454	4909	5278	4363	4969	6
7	4096	4993	4494	5039	3994	5087	5
8	8192	11305	10174	10674	9044	9547	3
9	16384	10865	9778	10908	8691	10861	1
10	32768	14956	13460	14547	11965	13819	2
11	40960	12902	11612	13108	10322	13396	4

Table 4.6: Number of Partitions S , Partition Size C_s and Average MTTF of a node in each Partition obtained from observed reliability of system logs

Partition S	Class Size	Partition Size at $B_p = 0$	Average MTTF of Partition (in hrs.)	Partition Size at $B_p = 0.1$ Type I & Type IV	Average MTTF of Partition (in hrs.)	Partition Size at $B_p = 0.1$ Type II & Type III	Average MTTF of Partition (in hrs.)
1	16384	16384	12102.93	16384	12102.93	16384	12102.93
2	8192	11305	9826.90	10174	9884.47	10674	9858.49
3	4096	4993	8907.62	4494	9103.56	5039	8992.86
4	2048	5454	8168.40	4909	8422.93	5278	8254.23
5	1024	2824	7669.14	3319	7917.12	3585	7717.33
6	512					1680	7545.23

Table 4.7: Number of Partitions S , Partition Size C_s and Average MTTF of a node in each Partition obtained from observed reliability of system logs

Partition S	Class Size	Partition Size at $B_p = 0.2$ Type I & Type IV	Average MTTF of Partition (in hrs.)	Partition Size at $B_p = 0.2$ Type II & Type III	Average MTTF of Partition (in hrs.)
1	16384	16384	12102.93	16384	12102.93
2	8192	9044	9941.54	9547	9916.62
3	4096	4096	9272.03	5087	9141.48
4	2048	4363	8679.65	4969	8423.73
5	1024	2950	8163.62	3737	7904.80
6	512	3323	7860.93	1236	7445.44
7	256	256	7277.21		
8	128	128	7272.14		
9	64	105	7261.69		
10	Extra Partition	311	7258.35		

improvement achieved when using this methodology is measured. For this, the reliability probability R_p for the old approach, using Equation 3.7, at time T, which is the service time of 95% of jobs submitted per resource class, β is 0.7 and using MTTF as 10250.67 hours is computed first. Then, the reliability probability R_p for the new approach using the same equation and parameters but with the new MTTF obtained by directing resource requests with high reliability need to the nodes with high reliability is computed.

Table 4.8 lists the reliability probability R_p values for the current complete sharing policy (old) and the new mixed approach (new). From this table, it can be observed that in the old approach, resource classes demanding higher reliability suffered reduced average node reliability due to the assignment of lower reliability nodes to the resource classes. Using the new approach, the resource classes demanding higher reliability have a resulting higher R_p value from avoiding relatively low reliability nodes. This verifies that the researcher's approach has been successful in improving the reliability of resource classes with higher reliability requirement. Although these percentages seem small, recall that this difference reflects a difference in two power functions. Thus, a small percentage difference can make a significant impact on actual reliability experienced by a parallel application.

To understand the impact of this difference on the running costs of large parallel applications, this methodology is evaluated using the effects on Daly's checkpoint interval (Daly, 2003). The process of checkpointing involves taking snapshots of applications that run on large systems to help fault tolerance and system recovery. Low reliability increases checkpointing frequency especially when the task is large, making this an expensive and time-consuming operation (Naksinehaboon et.al, 2008). Using this new approach of reliability guided resource allocation policy, the reliability of these large tasks can be improved by spending less time on checkpointing operations. To understand the effects of the reliability change in this analysis, the researcher

Table 4.8: Reliability of each resource class using new Average MTTF of partitions - New Approach

Resource Class Rank	Resource Class Size	Time T (in hrs.)	Assigned Partition Reliability (Old)	Reliability at Time T R_p (Old)	Assigned Partition Reliability (New)	Reliability at Time T R_p (New)	Reliability Change
1	16384	6.013	10250.67	0.0032	12102.93	0.0060	0.28% Inc.
2	32768	3.005	10250.67	0.0032	10250.67	0.0032	No Change
3	8192	10.701	10250.67	0.0050	9826.90	0.0043	0.07% Dec.
4	40960	1.274	10250.67	0.0251	10250.67	0.0251	No Change
5	4096	7.041	10250.67	0.0878	8907.62	0.0683	1.95% Dec.
6	2048	7.727	10250.67	0.2022	8168.40	0.1536	4.87% Dec.
7	1024	7.139	10250.67	0.3942	7669.14	0.3197	7.45% Dec.
8	512	8.37	10250.67	0.5270	7669.14	0.4562	7.08% Dec.
9	256	1.014	10250.67	0.9140	7669.14	0.8956	1.83% Dec.
10	128	0.878	10250.67	0.9512	7669.14	0.9405	1.07% Dec.
11	64	1.007	10250.67	0.9666	7669.14	0.9593	0.74% Dec.

to analyzed the impact on the optimal checkpoint interval (Daly, 2003), (Hacker & Meglicki, 2007)

$$\tau_{opt} = \sqrt{2\delta MTTF} - \delta \tag{4.2}$$

where δ is checkpoint latency and MTTF is the mean time to failure.

To determine the improvement achievable using this allocation strategy, the optimal time interval between checkpointing operations is computed using a system of equations with the expected reliability ($MTTF_1$) and the new assigned reliability ($MTTF_2$) possible from selecting a node from any node in the system. In the equations δ_1 and δ_2 is the checkpointing latency using the expected and assigned reliability.

$$\tau_1 = \sqrt{2\delta_1 MTTF_1} - \delta_1$$

$$\tau_2 = \sqrt{2\delta_2 MTTF_2} - \delta_2$$

The difference between the two equations above gives the improvement in optimal checkpointing interval. Assuming $\delta_1 = \delta_2 = \delta$, as the same workload is being used and the checkpoint interval is being computed for the same resource classes, the change in checkpointing interval is computed as follows,

$$\tau_2 - \tau_1 = \sqrt{2\delta MTTF_2} - \sqrt{2\delta MTTF_1} \tag{4.3}$$

Using Equation 4.3 to measure reliability improvement in this analysis, this methodology is tested if it has succeeded in increasing the interval between two checkpointing operations thus minimizing time to perform expensive checkpointing operations. A decrease in checkpointing intervals indicate significant improvement in reliability of jobs and saves precious computation time.

For a positive improvement in overall reliability,

$$\tau_2 - \tau_1 > 0$$

$$\sqrt{2\delta MTTF_2} > \sqrt{2\delta MTTF_1}$$

$$MTTF_2 > MTTF_1$$

$$\frac{MTTF_2}{MTTF_1} > 1 \tag{4.4}$$

Using Equation 4.4 for comparing reliabilities before and after using this new methodology, the overall reliability improvement of our methodology is measured. The reliability improvement is represented as a scale factor by computing the ratio of new assigned reliability ($MTTF_2$) from the approach to the reliability assigned to each resource class ($MTTF_1$) prior to using the new strategy. The assigned reliability for each resource class without reliability guidance, i.e. $MTTF_1$ is the system MTTF value 10250.67 hours. Ratios greater than one indicate a significant increase in reliability by using our approach, ratios equal to one indicate no significant change in reliability and ratios less than one do not benefit much from this methodology. Table 4.9 lists the ratio of assigned reliability $MTTI_2$ for blocking probability B_p values 0, 0.1 and 0.2. In this table, the higher resource classes have ratios greater than one, implying that the large and long-running jobs benefit from our reliability guided node assignment. However, this reliability improvement is achieved at the cost of decreased reliability in the lower resource classes.

After the improvement in optimal checkpointing interval by using our methodology is computed, the actual cost or gain to parallel applications from this change needs to be computed. If an optimal checkpoint interval τ (in minutes) is computed, which is the time elapsed between two checkpointing events for a resource class, with a given checkpointing latency δ (in minutes), which is the time required to save a checkpoint, over N nodes, then the cost of a checkpoint operation can be quantified as $\delta * N$, which would be the overall work required for one checkpoint operation. Given a fixed δ, the change in the optimal checkpoint interval τ that results from a change in the MTTF experienced by a parallel application can be computed.

If checkpointing occurs several times per day (or per hour), the overall cost difference for checkpointing using the new approach would then be

$$\Delta Cost = \frac{(60min.)}{\tau_2}\delta N - \frac{(60min.)}{\tau_1}\delta N$$

$$\Delta Cost = 60\delta N\left(\frac{1}{\tau_2} - \frac{1}{\tau_1}\right)$$

Table 4.9: Reliability Improvement for Each Resource Class by computing the ratio $MTTF_2$ to $MTTF_1$

Class Rank	Class Size	$MTTF_1$ (Old) (in hrs.)	$MTTF_2$ at B_p=0 (New) (in hrs.)	Ratio	$MTTF_2$ at B_p=0.1 (New) Type I & Type IV	Ratio	$MTTF_2$ at B_p=0.1 (New) Type II & Type III	Ratio	$MTTF_2$ at B_p=0.2 (New) Type I & Type IV	Ratio	$MTTF_2$ at B_p=0.2 (New) Type II & Type III	Ratio
1	16384	10250.67	12102.93	1.18	12102.93	1.18	12102.93	1.18	12102.93	1.18	12102.93	1.18
2	32768	10250.67	10250.67	1.00	10250.67	1.00	10250.67	1.00	10250.67	1.00	10250.67	1.00
3	8192	10250.67	9826.90	0.96	9884.47	0.96	9858.49	0.96	9941.54	0.97	9916.62	0.97
4	40960	10250.67	10250.67	1.00	10250.67	1.00	10250.67	1.00	10250.67	1.00	10250.67	1.00
5	4096	10250.67	8907.62	0.87	9103.56	0.89	8992.86	0.88	9272.03	0.90	9141.48	0.89
6	2048	10250.67	8168.40	0.80	8422.93	0.82	8254.23	0.81	8679.65	0.85	8423.73	0.82
7	1024	10250.67	7669.14	0.75	7917.12	0.77	7717.33	0.75	8163.62	0.80	7904.80	0.77
8	512	10250.67	7669.14	0.75	7545.23	0.74	7717.33	0.75	7860.93	0.77	7445.44	0.73
9	256	10250.67	7669.14	0.75	7545.23	0.74	7717.33	0.75	7277.21	0.71	7445.44	0.73
10	128	10250.67	7669.14	0.75	7545.23	0.74	7717.33	0.75	7272.14	0.71	7445.44	0.73
11	64	10250.67	7669.14	0.75	7545.23	0.74	7717.33	0.75	7261.69	0.71	7445.44	0.73

Table 4.10: Checkpointing Interval results of each resource class after using the new methodology

Resource Class	Resource Class Size	Service Time T per job (in hours)	Change in Optimal Checkpoint Interval	Checkpointing Operation Cost (in node-min. per hour)
1	16384	6.013	+ 39.21	- 1806.09
2	32768	3.005	No Change	No Change
3	8192	10.701	- 9.46	+ 242.29
4	40960	1.274	No Change	No Change
5	4096	7.041	- 30.70	+ 413.54
6	2048	7.727	- 48.60	+ 342.13
7	1024	7.139	- 61.14	+ 222.30
8	512	8.37	- 61.14	+ 111.15
9	256	1.014	- 61.14	+ 55.58
10	128	0.878	- 61.14	+ 27.79
11	64	1.007	- 61.14	+ 13.89

$$\Delta Cost = 60\delta N \left(\frac{\tau_1 - \tau_2}{\tau_1 \tau_2} \right)$$

$$\Delta Cost = 60\delta N \left(\frac{\sqrt{MTTF_1} - \sqrt{MTTF_2}}{(\sqrt{MTTF_1} - \sqrt{\frac{\delta}{2}})(\sqrt{MTTF_2} - \sqrt{\frac{\delta}{2}})} \right) \tag{4.5}$$

Thus, as N or δ increases, the cost of a single checkpoint operation increases. If τ decreases, which means more frequent checkpoint operations, the cost increases. Now, if the checkpoint interval τ increases by some fraction Q, the corresponding work and costs for the checkpoint operation also decreases by a proportional amount.

To compute this checkpointing cost the value of δ and τ is to be obtained. For all practical purposes, the value of the checkpointing latency δ should be less than the service time of the job, so that the computing power of the processors is spent in doing useful work than in checkpointing operations. In most large cluster-based computing systems in use today, the checkpointing latency is of the order of 4 to 5 minutes. Considering an upper bound for δ as 10 minutes in this analysis, the checkpointing interval τ is computed using Equation 4.2.

Table 4.10 highlights the change in the optimal checkpoint interval using Equation 4.3 and the change in time required for a checkpointing operation for each resource class computed using Equation 4.5. From this table, a significant increase in the optimal checkpointing interval and the total node time saved by minimizing checkpointing frequency for higher resource classes is observed. The total time spent on checkpointing operations for the highest ranking resource class decreases by *1806.09 node-min.* per hour, but at the cost of increasing the total checkpointing time for all lower resource classes by *1428.67 node-min.* per hour. From these two values, the total time saved by reducing frequency of checkpointing operations of higher resource classes is approximately *377.42 node-min.* per hour. This precious computation time can be used by the system to perform useful work.

Using these evaluation techniques it can be observed that by using our methodology, it is possible to achieve a significant improvement in reliability of jobs requesting large number of resources and having long service times. This methodology also helps cluster providers to determine the maximum capacity of their systems that should be provisioned to have negligible probability of blocking resource requests, especially at busy periods.

4.4 Summary

This chapter explained the steps and effects of the researcher's methodology on a computational workload from the ANL BlueGene supercomputer *Intrepid.* The next chapter simulates this analysis on a hypothetical cluster using the Haizea scheduler for resource allocation.

CHAPTER 5. SIMULATION

5.1 Simulation Approach

To measure the benefit of the new methodology from simulation, the effect of system failures on jobs of each resource class were observed. The simulated cluster consisted on 40960 uniprocessor nodes, each node having reliability similar to the reliability characteristics observed in the system logs from the Coates Cluster.

To ensure consistency and comparability, the same offered workload logs used in the analysis phase of this research is used in the simulation. For the simulation the new partitioning policy, which is a combination of complete sharing and complete partitioning policy is implemented. Thus, a total of five system partitions was used, with each partition size corresponding to blocking probability $B_p = 0$ as shown in Table 4.6.

To measure the effect of failures on resource classes, the researcher generated these simulated failures. For this purpose, the discrete-event simulation to model the operation of the system as a discrete sequence of events in time was used. Each event occurs at a particular instant in time and marks a change of state in the system (Robinson, 2004). Between consecutive events, no change in the system is assumed to occur; thus the simulation can directly jump in time from one event to the next.

5.2 Simulation Software

In order to simulate the scheduling decisions of a real-time computing system, the Haizea (Sotomayor, 2009) scheduler was used. It is an open-source virtual infrastructure manager to control resource requests in cluster-based systems, in the simulated mode. In Haizea, resources are allocated through protocols called leases for a wide range of resource class sizes in the system.

Haizea's architecture is comprised of (Hacker & Mahadik, 2011):

- Request frontend, that is responsible for accepting lease requests, be it from Open Nebula (Milojicic et.al, 2011) , interactively from a command-line interface, or from a trace file
- Scheduling core, for processing leases and scheduling decisions
- Enactment module which communicates with the cluster by providing sequence of instructions that are generated by the scheduler

In this simulation, the unattended mode of Haizea was used for which we converted the computational workload logs of the ANL Blue Gene system *Intrepid* into a trace file using a Perl script. The trace file is an XML file containing a list of job requests written in the form of leases, where each lease contains information about the number of nodes required, the start time, the duration of the task and lease id as XML elements. The Haizea scheduler requires the Haizea configuration file (specifying the configuration options for the simulation) and the request trace file as inputs to generate scheduling decisions for the given workload as an output.

The source code was modified to add simulated node failure by altering the list of available resources in the resource pool during the simulation run at discrete time steps to give an illusion of an increase or decrease in node availability. As a result, the scheduler alters scheduling decision of resource requests submitted to the system, which is studied by the researcher.

To simulate node failures, the first task was to determine the system nodes that were going to fail. Based on the understanding of node reliabilities of each system node from the analysis phase of this research, a file containing the list of system nodes with least node reliability was created. By storing the failure prone node list in a file, this experiment guaranteed persistence, repeatability of experiment and ensured that same nodes fail in all simulation runs.

After creating this list, the researcher determined the time between these simulated failures. In real systems, node failures usually show spatial and temporal clustering and the time between system failures follows a Weibull distribution. However, due to limitations in Haizea to run large simulations, the researcher used a time interval of one simulated hour between simulated system failures to ensure that our simulation completes within reasonable time. Hence the node failures in our simulation do not show temporal clustering. If we used a Weibull distribution of time between failures within the limitations of the Haizea simulation, the failure density over time would be much less and the simulation would take excessively long time to compute. Hence the node failures in this simulation do not show temporal clustering.

Using the node failure information file, the simulation was run two times - once without using reliability guided scheduling, and the second time after adding reliability mappers in the code. In the first simulation run, as the scheduler was not aware of system reliability information the simulator treated all nodes as identical and replaceable components, hence selected system nodes randomly from the resource pool for resource allocation.

For the second simulation run (using our methodology), the trace file was modified to add the reliability characteristics of every node as well as added the reliability requirement of every resource request. The second simulation used the same node failure distribution file used in the previous run, but this time along with the node reliability information of each node. This information was given to the scheduler in the form of a node map as shown in Table 5.1. This table contains the system node information along with their reliability range (which groups system nodes based on reliability in bins of 500 hours), the total number of system nodes in each reliability range and the actual mapping of system nodes based on reliability. This reliability node classification helps the scheduler to assign the reliable nodes to jobs with high reliability demand.

The scheduler's source code was modified to add reliability constraints to ensure that though lower reliability nodes are idle, the resource requests are satisfied by

Table 5.1: Reliability Range and Site Node mapping for simulation to facilitate reliability guided approach

	Reliability Range (in hours)	Number of nodes in each range	Physical System Nodes in reliability range
1	14000-13500	1470	40960-39490
2	13500-13000	2520	39489-36970
3	13000-12500	1995	36969-34975
4	12500-12000	2940	34974-32035
5	12000-11500	1785	32034-30250
6	11500-11000	2625	30249-27625
7	11000-10500	3255	27624-24370
8	10500-10000	4200	24369-20170
9	10000-9500	4305	20169-15865
10	9500-9000	4620	15864-11245
11	9000-8500	3237	11244-8008
12	8500-8000	3536	8007-4472
13	8000-7500	3640	4471-832
14	7500-7000	832	831-1
	Total	40960	

allocation of only those nodes with comparable or higher reliability. This allowed the scheduler to assign reliable system nodes to resource requests that have a high reliability need, thus ensuring reliability guided scheduling decision. Figure 5.1 shows the frequency histogram for each reliability ranges of system nodes.

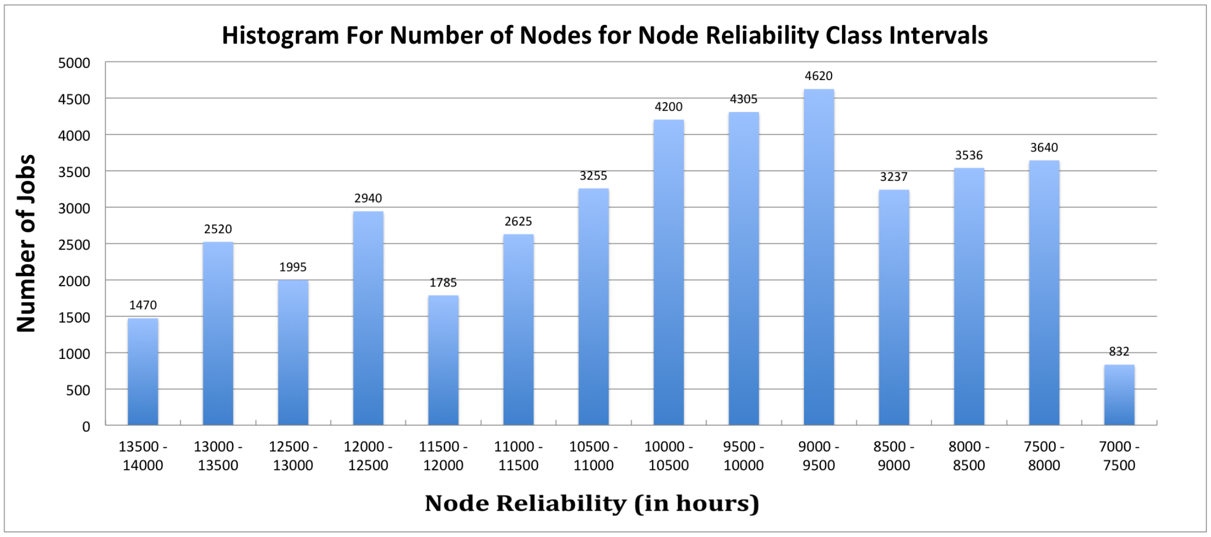


Figure 5.1.: Frequency histogram for reliability range of system nodes

5.3 Simulation Results

After the simulation runs as per the simulation methodology discussed above, the scheduling decision and failure summary report stored in the log files of the first and second simulation run were analyzed. As the total number of node failures for both simulation runs were kept the same, the only difference observed between the two log files was a different failure pattern in resource requests or job failures. Table 5.2 summarizes the total number of failures using the old and new strategies.

From Table 5.2, the researcher observed that number of node failures occurring during the lifetime of higher resource class jobs is higher when using the old approach. The results for the simulation run after using our methodology improves reliability of higher resource classes by enabling scheduling decisions taking into account the reliability need of resource classes as well as the node reliability on which these requests are scheduled.

From the simulation runs the researcher observed several characteristics. First, the researcher observed a decrease in the total number of jobs failed for higher resource classes on using the new methodology as compared to the number of job failures for the same resource classes without our approach. Secondly, the total number of node failures in partitions have shifted from large long running resource classes to lower resource classes by using our new partitioning strategy, that is a combination of complete sharing and complete partitioning policy. Thirdly, a significant positive impact was observed in higher resource classes that have very high cost of application failure, by noticing a considerable decrease in failure rate for these resource classes, however at the cost of an increased failure rate in the lower resource classes.

Thus from these observations it is evident that this methodology is successful in improving the reliability of large and long-running jobs by adding reliability awareness.

Table 5.2: Number of Failures before and after using the new methodology

Partition S	Partition Size	Partition Reliability from Simulation (in hours)	Resource Classes Affected	Number of Node Failures (Before)	Number of Node Failures (After)	Number of Jobs Failed (Before)	Number of Jobs Failed (After)	Impact of new Reliability Based Scheduling Approach on Node Failure
1	16384	12174.33	16384	927	783	149	126	+ 15.53%
2	11305	9841.57	8192	387	305	150	119	+ 21.19%
3	4993	9101.44	4096	148	161	62	68	- 8.78%
4	5454	8123.67	2048	162	172	150	160	- 6.17%
5	2824	7602.69	1024,512, 256,128,64	200	403	2103	4237	- 100%
Total	40960	10255.04		1824	1824	2615	4709	

5.4 Summary

This chapter justifies that this methodology can be implemented on a scheduler and highlights reliability improvement based on simulation results. The next chapter concludes this research thesis and explains the major benefits of using this newly developed methodology.

CHAPTER 6. CONCLUSION

Resource allocation in cluster computing systems with node reliability awareness can improve overall system reliability by making clusters resilient to system failures. Adding reliability-based resource mappers in schedulers, taking into account the reliability need of admitted jobs as well as the hardware reliability of the resources on which these jobs are scheduled, can have a significant impact in improving the overall reliability and efficiency of the cluster computing system.

This research has developed a new approach that uses a combination of complete sharing policy and a selective complete partitioning strategy. This strategy gives the advantage of high system utilization - a characteristic of complete sharing policy - and the minimized blocking probability benefit of complete partitioning policy. This partitioning policy ensures that resource requests of classes with high reliability need are assigned to system nodes with comparable or higher reliability value, thus guaranteeing them an improvement in job reliability, however at the cost of increased failure rate for lower resource classes. This scheduling policy provides flexibility. Resource requests of different resource classes can be combined from a combined partition to improve overall job reliability for the lower resource classes and to provide a large pool of resources. However such combinations should be selected carefully, since the average reliability may be reduced.

This approach has shown significant improvement in the reliability of jobs, especially the large and long running jobs, and resulting in an increase in the optimal checkpoint interval for these jobs. It is important to note that all our results have been determined for a system size of 40960 nodes, for a larger system size there will be greater improvement in reliability. Large-scale computing system providers to determine the size of the system that should be provisioned to service job requests at busy periods can use this methodology. This new algorithm can be immediately incorporated into existing schedulers without the need to invest in new hardware or architecture modifications.

LIST OF REFERENCES

LIST OF REFERENCES

American Psychological Association. (2010). *Publication manual of the American Psychological Association (6th ed.)*. Washington, DC: Author.

Adaptive Computing. Intelligent HPC Workload Management. *Retrieved on Oct 12, 2013 from http://old.adaptivecomputing.com/docs/490*

ANL Intrepid Log. Parallel Workload Archive. *Retrieved on 02/02/2013 http://www.cs.huji.ac.il/labs/parallel/workload/l_anl_int/index.html*

Beard, C. C., & Frost, V. S. (2001). Prioritized resource allocation for stressed networks. *Networking, IEEE/ACM Transactions* on, 9(5), 618-633.

Choudhury, G. L., Leung, K. K., & Whitt, W. (1995). An algorithm to compute blocking probabilities in multi-rate multi-class multi-resource loss models. *Advances in Applied Probability*, 1104-1143.

Daly, J. (2003). A model for predicting the optimum checkpoint interval for restart dumps. *In Computational Science—ICCS* 2003 (pp. 3-12). Springer Berlin Heidelberg.

Given, Lisa M. (2008). The Sage encyclopedia of qualitative research methods. *Los Angeles, Calif.: Sage Publications.*

Hacker, T. (2010). Toward a reliable cloud computing service. *Cloud Computing and software services: Theory and Techniques,* 139.

Hacker, T., Mahadik, K. (2011). Flexible resource allocation for reliable virtual cluster computing systems. *In High performance computing, networking, storage and analysis* (SC), 2011 International conference for Nov, 11(pp. 1-12).

Hacker, T. J., & Meglicki, Z. (2007, June). Using queue structures to improve job reliability. *In Proceedings of the 16th international symposium on High performance distributed computing* (pp. 43-54). ACM.

Hacker, T. J., Romero, F., & Carothers, C. D. (2009). An analysis of clustered failures on large supercomputing systems. Journal of Parallel and Distributed Computing, 69(7), 652-665.

Henderson, R. L. (1995, January). Job scheduling under the portable batch system. *In Job scheduling strategies for parallel processing* (pp. 279-294). Springer Berlin Heidelberg.

Kannan, S., Roberts, M., Mayes, P., Brelsford, D., & Skovira, J. F. (2001). Workload management with loadleveler. *IBM Redbooks, 2, 2.*

Kelly, F. P. (1986). Blocking probabilities in large circuit-switched networks. *Advances in applied probability,* 473-505.

Key, P. B. (1990). Optimal control and trunk reservation in loss networks. *Probability in the Engineering and Informational Sciences*, 4(2), 203-242.

Kobayashi, H., & Mark, B. (2009). System modeling and analysis: foundations of system performance evaluation. *Pearson Prentice Hall.*

Milojičić, D., Llorente, I. M., & Montero, R. S. (2011). Opennebula: A cloud management tool. *Internet Computing,* IEEE, 15(2), 11-14.

Naksinehaboon, N., Liu, Y., Leangsuksun, C., Nassar, R., Paun, M., & Scott, S. L. (2008, May). Reliability-aware approach: An incremental checkpoint/restart model in hpc environments. *In Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE International Symposium* on (pp. 783-788). IEEE.

Rappaport, T. S. (1996). Wireless communications: principles and practice (Vol. 2). *New Jersey: Prentice Hall PTR.*

Robinson, S. (2004). Simulation: the practice of model development and use. *John Wiley \& Sons.*

Shatz, S. M., Wang, J. P., & Goto, M. (1992). Task allocation for maximizing reliability of distributed computer systems. *Computers, IEEE Transactions on*, 41(9), 1156-1168.

SLURM: A Highly scalable resource manager. *Retrieved on Oct 12, 2013 from https://computing.llnl.gov/linux/slurm/documentation.html*

Sotomayor, B. (2010 (accessed May 5, 2010)). The Haizea manual. *Available from http://haizea.cs.uchicago.edu*

Tang, X., Li, K., Qiu, M., & Sha, E. H. M. (2012). A hierarchical reliability-driven scheduling algorithm in grid systems. *Journal of Parallel and Distributed Computing,* 72(4), 525-535.

Tang, X., Li, K., Li, R., & Veeravalli, B. (2010). Reliability-aware scheduling strategy for heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing,* 70(9), 941-952.

Zeng, G. (2003). Two common properties of the erlang-B function, erlang-C function, and Engset blocking function. *Mathematical and computer modelling,* 37(12), 1287-1296.

Zheng, Z., Yu, L., Tang, W., Lan, Z., Gupta, R., Desai, N., ... & Buettner, D. (2011, May). Co-analysis of RAS log and job log on Blue Gene/P. *In Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International* (pp. 840-851). IEEE.

APPENDICES

CHAPTER A. STATISTICAL ANALYSIS

Table A.1: Expected and observed assigned reliability from analysis for each resource class at blocking probability $B_p = 0$

Resource Class	Resource Class Size	Observed reliability from analysis	Expected reliability from analysis
1	16384	12102.93	10250.67
2	32768	10250.67	10250.67
3	8192	9826.9	10250.67
4	40960	10250.67	10250.67
5	4096	8907.62	10250.67
6	2048	8168.4	10250.67
7	1024	7669.14	10250.67
8	512	7669.14	10250.67
9	256	7669.14	10250.67
10	128	7669.14	10250.67
11	64	7669.14	10250.67

Table A.2: Expected and observed assigned reliability from simulation for each resource class at blocking probability $B_p = 0$

Resource Class	Resource Class Size	Observed reliability from simulation	Expected reliability from simulation
1	16384	12174.33	10255.04
2	32768	10255.04	10255.04
3	8192	9841.57	10255.04
4	40960	10255.04	10255.04
5	4096	9101.44	10255.04
6	2048	8123.67	10255.04
7	1024	7602.69	10255.04
8	512	7602.69	10255.04
9	256	7602.69	10255.04
10	128	7602.69	10255.04
11	64	7602.69	10255.04

Table A.3: Checkpointing operation cost difference from analysis and simulation for each resource class at blocking probability $B_p = 0$

Resource Class	Resource Class Size	Checkpointing Operation Cost Simulation (in node-minute)	Checkpointing Operation Cost Analysis (in node-minute)
1	16384	1862.38	1806.09
2	32768	0.00	0.00
3	8192	-236.06	-242.29
4	40960	0.00	0.00
5	4096	-349.37	-413.54
6	2048	-351.52	-342.13
7	1024	-229.81	-222.31
8	512	-114.91	-111.15
9	256	-57.45	-55.58
10	128	-28.73	-27.79
11	64	-14.36	-13.89

Table A.1 lists the observed and expected reliability from analysis and the Table A.2 lists the observed and expected reliability from simulation. The P value for the analysis data is 0.003775 and for simulation data the p-value is 0.004358, which makes both these results significant at $p < 0.05$. As the p-values for the analysis and simulation are less than 0.05, we can reject the null hypothesis, and accept the alternate hypothesis that there is an increase in reliability of jobs with high reliability need by shifting assignment of high reliability system nodes to these resource classes.

Table A.3 lists the checkpointing operation cost for each resource class from simulation and analysis. The total cost benefit i.e. the total time saved in by reducing checkpointing operations is *481.18* node-minute per hour from simulation and *377.42* node-minute per hour from analysis data. These results show a significant improvement in system efficiency, thus we reject the second null hypothesis and accept the alternate hypothesis that our methodology is successful in increasing system efficiency by decreasing total checkpointing operation cost.

CHAPTER B. DATA PROCESSING SCRIPTS

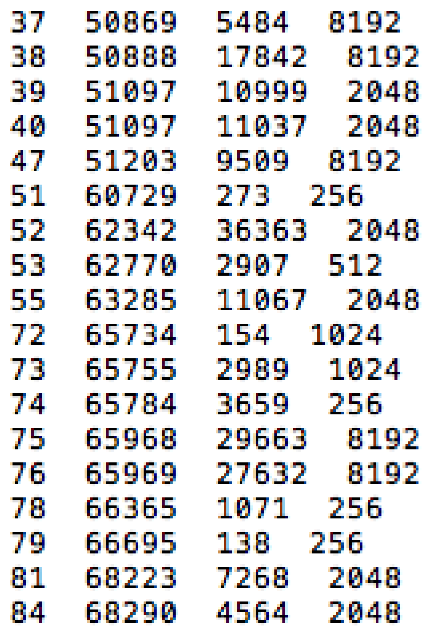
First the computational workload input file *ANL-Intrepid-2009-1.swf* from the ANL BlueGene Intrepid supercomputer is converted to job_log.txt using the *awk* command on the terminal. The contents of this job_log.txt file are shown in Figure B.1.

This file is formatted to represent the time fields in suitable format (time in seconds is converted to DD: HH:MM:SS format) using the Perl script - log_to_lwf.pl, as shown below,

```
#!/usr/bin/perl


open SOURCE, "< job_log.txt" or die "Could not open sample data input file";
open OUTPUT, "> data_in_lwf.txt" or die "Could not open sample output file";


while(<SOURCE>)
{
($JOB_ID,$ARR_TIME, $SER_TIME,$NUM_NODES) = split (/ \s/,$_);
printf OUTPUT "%d ",$JOB_ID;
$sec = $ARR_TIME;
$d = int(($sec/3600)/24);
$h = ($sec/3600)%24;
$m = ($sec/60)%60;
$s = $sec%60;
printf OUTPUT "%.2d:%.2d:%.2d:%.2d ",$d,$h,$m,$s;
$sec = $SER_TIME;
$d = int(($sec/3600)/24);
```



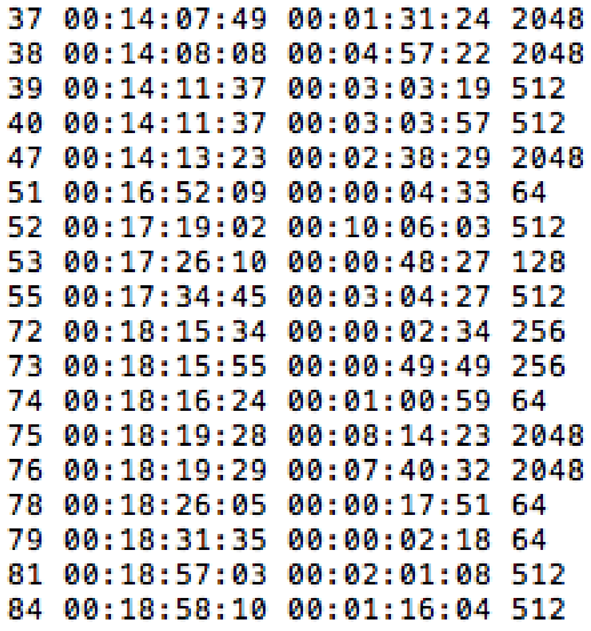
```
37  50869  5484   8192
38  50888  17842  8192
39  51097  10999  2048
40  51097  11037  2048
47  51203  9509   8192
51  60729  273   256
52  62342  36363  2048
53  62770  2907   512
55  63285  11067  2048
72  65734  154   1024
73  65755  2989   1024
74  65784  3659   256
75  65968  29663  8192
76  65969  27632  8192
78  66365  1071   256
79  66695  138   256
81  68223  7268   2048
84  68290  4564   2048
```

Figure B.1.: Input computational workload file with job id, submitted time, service time and job size (job_log.txt file)

```
$h = ($sec/3600)%24;
$m = ($sec/60)%60;
$s = $sec%60;
printf OUTPUT "%.2d:%.2d:%.2d:%.2d ",$d,$h,$m,$s;
$num = $NUM_NODES ;
printf OUTPUT "%d\n",int($num/4);
}

close SOURCE;
close OUTPUT;
```

The above script generates the output file data_in_lwf.txt, shown in Figure B.2. This file is the lease workload format (LWF) and is converted into a trace file using the Perl script lwf_in_xml.pl, shown below.



```
37 00:14:07:49 00:01:31:24 2048
38 00:14:08:08 00:04:57:22 2048
39 00:14:11:37 00:03:03:19 512
40 00:14:11:37 00:03:03:57 512
47 00:14:13:23 00:02:38:29 2048
51 00:16:52:09 00:00:04:33 64
52 00:17:19:02 00:10:06:03 512
53 00:17:26:10 00:00:48:27 128
55 00:17:34:45 00:03:04:27 512
72 00:18:15:34 00:00:02:34 256
73 00:18:15:55 00:00:49:49 256
74 00:18:16:24 00:01:00:59 64
75 00:18:19:28 00:08:14:23 2048
76 00:18:19:29 00:07:40:32 2048
78 00:18:26:05 00:00:17:51 64
79 00:18:31:35 00:00:02:18 64
81 00:18:57:03 00:02:01:08 512
84 00:18:58:10 00:01:16:04 512
```

Figure B.2.: Lease Workload Format file created using a Perl script having job id, submitted time in human timestamp format, job duration and job size (data_in_lwf.txt)

```
#!/usr/bin/perl

use XML::Writer;
use IO::File;

my $output = new IO::File("> lwf_to_xml.xml");
open(FILE, "< data_in_lwf.txt ") || die("Can't open text file");
my $writer = new XML::Writer(OUTPUT => $output,NEWLINES => 1);
   $writer->xmlDecl("UTF-8");

   $writer->startTag("lease-workload","name"=>"sample");
   $writer->startTag("description");
   $writer->characters("This is a trial conversion of .txt to .xml");
   $writer->endTag("description");

   #Site Information tags
   $writer->startTag("site");
   $writer->startTag("resource-types","names"=>"CPU Memory Reliability");
   $writer->endTag("resource-types");
   $writer->startTag("nodes");
   $writer->startTag("node-set","numnodes"=>"40960");
   $writer->startTag("res","type"=>"CPU","amount"=>"4");
   $writer->endTag("res");
   $writer->startTag("res","type"=>"Memory","amount"=>"2");
   $writer->endTag("res");
   $writer->startTag("res","type"=>"Reliability","amount"=>"95");
   $writer->endTag("res");
   $writer->endTag("node-set");
   $writer->endTag("nodes");
   $writer->endTag("site");

   #Lease Requests
   $writer->startTag("lease-requests");
   while (<FILE>)
   {
# ANL Intrepid Logs (69436 job requests)
($JOB_ID,$ARR_TIME, $SER_TIME, $NUM_NODES) = split(/ /, $_);
($myduration,$tchr)=split(/\n/,$SER_TIME);
$writer->startTag("lease-request","arrival"=>$ARR_TIME);
$writer->startTag("lease","preemptible"=>"true");
   $writer->startTag("nodes");
   $writer->startTag("node-set","numnodes"=>int($NUM_NODES));
   $writer->startTag("res","type"=>"CPU","amount"=>"4");
   $writer->endTag("res");
   $writer->startTag("res","type"=>"Memory","amount"=>"2");
   $writer->endTag("res");
   $writer->startTag("res","type"=>"Reliability","amount"=>"3");
$writer->endTag("res");
$writer->endTag("node-set");
   $writer->endTag("nodes");
$writer->startTag("start");
$writer->endTag("start");
$writer->startTag("duration","time"=>$myduration);
$writer->endTag("duration");
$writer->startTag("software");
$writer->startTag("disk-image","id"=>"foobar.img","size"=>"1024");
$writer->endTag("disk-image");
$writer->endTag("software");
        $writer->endTag("lease");
   $writer->endTag("lease-request");
  }
  $writer->endTag("lease-requests");
  $writer->endTag("lease-workload");
  $writer->end();
  $output->close();

close FILE;
exit;
```

The generated trace file - lwf_to_xml.xml, is an XML document containing all resource requests in the form of leases. The next appendix explains the contents of a trace file.

CHAPTER C. SAMPLE TRACE FILE

The trace file is an important document submitted to the Haizea scheduler, while running simulations in the unattended simulation mode. The trace file contains information about every resource request, in the form of leases. The following trace file lwf_to_xml.xml, is used in the first run of our simulation i.e. without reliability guided scheduling.

```
<?xml version="1.0" encoding="UTF-8"?>
<lease-workload name="sample">
<description>This is a conversion of .txt to .xml</description>
<site>
<resource-types names="CPU Memory Reliability"></resource-types>
<nodes>
<node-set numnodes="40960">
<res type="CPU" amount="4"></res>
<res type="Memory" amount="2"></res>
</node-set>
</nodes>
</site>
<lease-requests>
<lease-request arrival="00:07:56:57">
<lease preemptible="true">
<nodes>
<node-set numnodes="512">
<res type="CPU" amount="4"></res>
<res type="Memory" amount="2"></res>
```

```
</node-set>
</nodes>
<start></start>
<duration time="00:02:06:06"></duration>
<software>
<disk-image id="foobar.img" size="1024"></disk-image>
</software>
</lease>
</lease-request>

<lease-request arrival="00:08:02:29">
<lease preemptible="true">
<nodes>
<node-set numnodes="64">
<res type="CPU" amount="4"></res>
<res type="Memory" amount="2"></res>
</node-set>
</nodes>
<start></start>
<duration time="00:00:05:35"></duration>
<software>
<disk-image id="foobar.img" size="1024"></disk-image>
</software>
</lease>
</lease-request>
</lease-requests>
```

The above sample trace file shows the site configuration information in the *site* element, and describes the resource request requirements for two resource requests. The resource requests submitted to the system are called leases in Haizea. A lease id (assigned by the scheduler), the amount of memory, number of nodes, and amount of CPU needed for its execution, describes each lease. Each lease has an associated arrival time and time duration for which the lease will be holding resources. The following trace file is used for the second simulation, i.e. when using reliability guided scheduling.

```
<?xml version="1.0" encoding="UTF-8"?>
<lease-workload name="sample">
<description>This is a conversion of .txt to .xml</description>
<site>
<resource-types names="CPU Memory Reliability"></resource-types>
<nodes>
<node-set numnodes="16384">
<res type="CPU" amount="4"></res>
<res type="Memory" amount="2"></res>
<res type="Reliability" amount="1"></res>
</node-set>
</nodes>
<nodes>
<node-set numnodes="11305">
<res type="CPU" amount="4"></res>
<res type="Memory" amount="2"></res>
<res type="Reliability" amount="2"></res>
</node-set>
</nodes>
<nodes>
<node-set numnodes="4993">
<res type="CPU" amount="4"></res>
<res type="Memory" amount="2"></res>
<res type="Reliability" amount="3"></res>
```

```
</node-set>
</nodes>
<nodes>
<node-set numnodes="5454">
<res type="CPU" amount="4"></res>
<res type="Memory" amount="2"></res>
<res type="Reliability" amount="4"></res>
</node-set>
</nodes>
<nodes>
<node-set numnodes="2824">
<res type="CPU" amount="4"></res>
<res type="Memory" amount="2"></res>
<res type="Reliability" amount="5"></res>
</node-set>
</nodes>
</site>
<lease-requests>
<lease-request arrival="00:07:56:57">
<lease preemptible="true">
<nodes>
<node-set numnodes="512">
<res type="CPU" amount="4"></res>
<res type="Memory" amount="2"></res>
<res type="Reliability" amount="5"></res>
</node-set>
</nodes>
<start></start>
<duration time="00:02:06:06"></duration>
```

```
<software>
<disk-image id="foobar.img" size="1024"></disk-image>
</software>
</lease>
</lease-request>

<lease-request arrival="00:08:02:29">
<lease preemptible="true">
<nodes>
<node-set numnodes="16384">
<res type="CPU" amount="4"></res>
<res type="Memory" amount="2"></res>
<res type="Reliability" amount="1"></res>
</node-set>
</nodes>
<start></start>
<duration time="00:00:05:35"></duration>
<software>
<disk-image id="foobar.img" size="1024"></disk-image>
</software>
</lease>
</lease-request>
</lease-requests>
```

The contents of this file are the same as the trace file for the first run, but the only difference is that, this file contains reliability information of system nodes as well as the expected reliability of leases.

CHAPTER D. SIMULATION CONFIGURATION FILE

To run the simulation we must provide a configuration file to the scheduler so that the scheduler can run in simulated time instead of using real time. The configuration file also sets parameters like log location, trace file location, wake-up interval etc. The following code shows the contents of the configuration file used in this simulation.

```
[general]
loglevel: INFO
logfile: simulation_log.log
mode: simulated
lease-preparation: unmanaged
persistence-file: none

[scheduling]
wakeup-interval: 10
backfilling: aggressive
suspension: all
suspend-rate: 32
resume-rate: 32
migration: yes

[simulation]
clock: simulated
starttime: 2014-02-01 00:00:00
resources: in-tracefile
```

```
imagetransfer-bandwidth: 100
status-message-interval: 30

[accounting]
datafile: /var/tmp/haizea/results.dat
probes: immediate cpu-utilization

[deploy-imagetransfer]
transfer-mechanism: multicast
avoid-redundant-transfers: True
diskimage-reuse: none
diskimage-cache-size: 20480

[tracefile]
tracefile: lwf_to_xml.xml
```

CHAPTER E. HAIZEA SOURCE CODE MODIFICATIONS

E.1 Reliability Mappers

The *site* tag in the trace file describes the entire system using the XML elements *node-sets*. By using multiple node sets, the researcher adds node reliability information so that the scheduler can consider the node reliability before scheduling a lease that has a higher reliability demand. To add this reliability information, the researcher added the a *res* tag with two attributes *type* and *amount*. The *type* attribute takes the value *res* to indicate that this is a reliability characteristic of the node. The *amount* attribute is an integer between 1 and 10 that specifies the reliability need, where 1 indicates highest reliability and 10 indicates least reliability. The following example explains a node-set with CPU, memory and reliability as three characteristics that define a system node.

```
Example-
<nodes>
<node-set numnodes="3">
<res type="CPU" amount="512"/>
<res type="Memory" amount="2"/>
<res type="Reliability" amount="1"/>
</node-set>
</nodes>
```

The researcher modified the scheduler's source code to add reliability awareness of system nodes. The code modifications are as shown below.

1. [core/enacted/simulated.py in line 30]

```
if not ("CPU" in site. resource_ types and "Memory" in site. resource_ types
and "Reliability" in site. resource_ types):
# CPU and Memory must be specified
             raise
```

2. [core/enacted/simulated.py in line 81]

```
#action.vnodes[vnode].resources.get_by_type(constants.RES_MEM)
rel = 3 #action.vnodes[vnode].resources.get_by_type(constants.RES_REL)
self.logger.debug("Received request to start VM for L%iV%i on host %i,
image=%s, cpu=%i, mem=%i, reliability=%i" %
(action.lease_haizea_id, vnode, pnode, image, cpu, memory, rel))
```

3. [common/constants.py in line 22]

```
RES_MEM = "Memory"
RES_REL = "Reliability"
RES_NETIN = "Net-in"
```

4. [cli/commands.py in line 385]

```
        cpu = int(fields[5])
        mem = int(fields[6])
        rel = int(fields[7])
        disk = int(fields[8])
        vm_image = fields[9]
```

5. [cli/commands.py in line 415]

```
        res = ET.SubElement(node_set, "res")
        res.set("type", "Reliability")
        res.set("amount", ‘rel‘)
```

E.2 Failure Generator

This code was added in the *simulator_clock* class, *run* function in the *manager.py* file of the Haizea scheduler. At line 745 the following lines were added:

```
f = open('node_list.txt')
lines = f.readlines()
f.close()
i=0
start_flag= 0
```

At line 751 following lines were added:

```
if self.time.minutes == 0:
      if start_flag==1:
      prev_failed_node = self.manager.scheduler.vm_scheduler.
resourcepool.get_node(self, val[i-1])
      prev_failed_node.capacity = stored_cap
      self.logger.status("Node repaired: "+prev_failed_node.id)
failed_node = self.manager.scheduler.vm_scheduler.resourcepool.
get_node(self, val[i])
stored_cap= self.manager.scheduler.vm_scheduler.
resourcepoolnode.get_capacity(failed_node)
failed_node.capacity=0
self.logger.status("Node failed: "+failed_node.id)
i=i+1
start_flag=1;
```