Summer 2012

Reliability models for HPC applications and a Cloud economic model

Thanadech Thanakornworakij
Louisiana Tech University

RELIABILITY MODELS FOR HPC APPLICATIONS AND A CLOUD ECONOMIC MODEL

by

Thanadech Thanakornworakij, B.S., M.S.

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

COLLEGE OF ENGINEERING AND SCIENCE
LOUISIANA TECH UNIVERSITY

August 2012

LOUISIANA TECH UNIVERSITY

THE GRADUATE SCHOOL

3/12/2012
Date

We hereby recommend that the dissertation prepared under our supervision by THANADECH THANAKORNWORAKIJ

entitled

RELIABILITY MODELS FOR HPC APPLICATIONS AND A CLOUD ECONOMIC MODEL

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

[illegible]
Supervisor of Dissertation Research

[illegible]
Head of Department

Department

Recommendation concurred in:

[illegible]

[illegible]

[illegible]

[illegible]

Advisory Committee

Approved:
[illegible]
Director of Graduate Studies

[illegible]
Dean of the College

Approved:
[illegible]
Dean of the Graduate School

GS Form 13a
(6/07)

ABSTRACT

With the enormous number of computing resources in HPC and Cloud systems, failures become a major concern. Therefore, failure behaviors such as reliability, failure rate, and mean time to failure need to be understood to manage such a large system efficiently.

This dissertation makes three major contributions in HPC and Cloud studies. First, a reliability model with correlated failures in a *k*-node system for HPC applications is studied. This model is extended to improve accuracy by accounting for failure correlation. Marshall-Olkin Multivariate Weibull distribution is improved by excess life, conditional Weibull, to better estimate system reliability. Also, the univariate method is proposed for estimating Marshall-Olkin Multivariate Weibull parameters of a system composed of a large number of nodes. Then, failure rate, and mean time to failure are derived. The model is validated by using log data from Blue Gene/L system at LLNL. Results show that when failures of nodes in the system have correlation, the system becomes less reliable.

Secondly, a reliability model of Cloud computing is proposed. The reliability model and mean time to failure and failure rate are estimated based on a system of *k* nodes and *s* virtual machines under four scenarios: 1) Hardware components fail independently, and software components fail independently; 2) software components fail independently, and hardware components are correlated in failure; 3) correlated software failure and independent hardware failure; and 4) dependent software and hardware failure. Results show that if the failure of the nodes and/or software in the system possesses a degree of dependency, the system becomes less reliable. Also, an increase in the number of computing components decreases the reliability of the system.

Finally, an economic model for a Cloud service provider is proposed. This economic model aims at maximizing profit based on the right pricing and rightsizing in the Cloud data center. Total cost is a key element in the model and it is analyzed by considering the Total Cost of Ownership (TCO) of the Cloud.

APPROVAL FOR SCHOLARLY DISSEMINATION

Author [illegible]

Date 6/6/2012

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

First of all, I would like to thank Dr. Chokchai Box Leangsuksun, my advisor, for recruiting me to the CAM Ph.D. program. He guides and gives me valuable feedback and comments on my dissertation during my Ph.D. program. Secondly, I would like to thank Dr. Raja Nassar who provides me time and patience to work with me and gives valuable idea on mathematical and statistical modeling in all of my work. He also corrects and improves my writing. Thirdly, I sincerely thank Dr. Mihaela Paun for her valuable guidance throughout my dissertation. Finally, I would like to thank my committee members, Dr. Zeno Greenwood and Dr. Weizhong Dai also for their valuable feedbacks on this dissertation.

I would like to thank my dear Thai friends. Phongphonkit Phadungsak has helped me on many occasions since I arrived in Ruston until he left. I also thank Nichamon Naksinehaboon for helping and guiding me on Mathematics and Statistics in this dissertation. I also thank Narate Taerat for answering my computer science problems and for giving me Blue Gene/L log data used in this dissertation. I also thank my roommate Mungtra Chusilp for helping me personally and for cooking good food. I would like to thank also my Chinese friend Wang Liqun who helps me on many occasions and tutors me for the CAM qualifying exam. Finally, I would also like to thank Waraporn Jindarat, who always gives me love and moral support.

I would like to dedicate this dissertation to my family, my parents: Thanasub and Vipada Thanakornworakij, my brothers and sister: Apicha, Manop, and Nuchanat Thanakornworakij, for their endless and unconditioned love and for always supporting me in everything in my life.

CHAPTER 1

INTRODUCTION

In today's computing, significant HPC systems have become larger with more and more components, such as CPUs, storages and network cards. These systems support large computational scientific or mission critical business applications, which run long and are computationally intensive. The Top 500 [1] lists the top 500 most powerful High Performance Computing (HPC) systems. The list is compiled twice a year by high performance computing experts. In November 2011, The fastest supercomputer in the Top 500 list was the K computer from Japan, which comprises 88,128 SPARC64 8-core CPUs or 705,024 cores with the performance of 8.162 petaflops. The increase in computing component counts significantly escalates the chance of failures. Furthermore, any failure may interrupt applications, especially MPI applications. Hence, the system failure behavior of computing must be studied and well-understood in order to deploy the right fault tolerant mechanisms to handle mission critical applications effectively.

1.1 Overview of HPC systems

High performance computing (HPC) is the collection of computers tied together with network to solve large complex applications. There are software management programs for HPC systems, such as Rocks [2] and OSCAR [3], that provide the ability for users or administrators to build and maintain an HPC cluster. To utilize parallel computing, Message Passing Interface (MPI) is a standard library that allows parallel applications in sending and receiving messages between processes. How best to divide a complex parallel job to maximize HPC performance is a challenging area of study.

With the advent of multicore processors, OpenMP [4], [5] becomes a popular parallel programming paradigm. OpenMP is an application programming interface (API) used to support the execution of multi-thread in share memory architecture. OpenMP is flexible and yet simple for programing. In addition, a hybrid programing model [6], [7], which employs both MPI and OpenMP in the same application, is an interesting programing approach. Performance comparisons among MPI, OpenMP, and hybrid (MPI and OpenMP) have been extensively studied and evaluated [6], [8] . The studies showed that applications may or may not gain performance improvement from the hybrid model, depending on some applications characteristic and parameters [8], [9], [10].

When HPC systems become larger, failures become a major concern. For example, the Blue Gene/L system at Lawrence Livermore National Laboratory (LLNL) is comprised of 131,072 dual core processors. The Blue Gene/L system failed every other day [11]. Hence, the HPC systems need to have effective fault tolerant mechanisms to mitigate failures. A checkpoint/restart mechanism is a common fault tolerant technique that creates software states of computing. When any failure happens, users can restore the system from the last checkpoint file. When an HPC system performs checkpointing, the system has to spend time, which is called checkpoint overhead, to synchronize application processes and pause to save the computing information. If applications perform checkpoints too often, they will waste considerable time and the applications will delay the completion time. However, if checkpoints are performed too infrequently, when failures happen, application recovery will spend considerable re-computing time, which is the time since the last checkpoint. To optimize HPC performance, the checkpoint scheduling or the appropriate time for checkpointing has been studied [12], [13], [14].

An incremental checkpoint mechanism has been proposed to reduce checkpoint overhead. In this mechanism, the user does not need to save the whole application every time. The first checkpoint is normally a full checkpoint. The incremental checkpoint saves only the part of the application that has changed from the previous checkpoint. This can reduce the checkpoint overhead significantly. However, recovery time from the incremental checkpoint after a failure is quite complex and costly. It involves loading the full checkpoint with every incremental checkpoint since the last full checkpoint and then combining them together. Hence, the number of incremental checkpoints ,between full checkpoints, must be optimized and has been studied in [15], [16]

Redundancy [17], [18] is used to handle the failure in HPC systems. This technique uses more than one component to work on the same task, which increases the likelihood that at least one component will not fail. The number of redundant components is still a question. If one uses too many resources to do the same task, a lot of HPC resources are wasted. On the other hand, if not enough redundant components are used, failures may stop the entire application.

Fault tolerant mechanisms need accurate reliability estimation to be used efficiently. Therefore, failure behavior in HPC systems needs to be understood.

1.2 Overview of Cloud Computing

Cloud computing provides play-as-you-pay services for customers over the internet. There are three types of Clouds. First, a public Cloud allows users to select services and pay per usage, like a public utility. Second, a private Cloud is a Cloud infrastructure that serves mission critical services within an organization. Finally, a Hybrid Cloud is a combination between private and public Clouds. The hybrid Cloud provides services in organization and when it has excessive demand beyond its capacity, the excessive jobs will be submitted to run on the public Cloud.

Virtualization is a core technology in Cloud computing environment. It allows a machine to run more than a Virtual Machine and Operating System (OS). A machine can also runs multiple types of OSes, such as Windows and Linux. Virtualization normally consists of Hypervisor, also called a Virtual Machine Manager (VMM), and Virtual Machines (VM). Hypervisor virtualizes hardware to allow multiple OSes in a machine. There are two types of hypervisors: a Bare metal hypervisor which runs directly on a physical machine and hosted Hypervisor which runs on the host OS. Virtual machine is abstract software that runs a guest OS as if it were a real machine. It contains virtual CPU, RAM, and Network. Well-known commercial virtualization products are VMware [19], Citrix [20], and Microsoft Hyper-V [21]. Open source virtualization products are Virtual Box [22], Xen [23], and OpenVZ [24]. To deploy and maintain a Cloud system, there are many Cloud management systems available such as VMware vCloud [19], OpenNebula [25], Eucalyptus [26], OpenStack [27], and Nimbus [28].

Cloud computing services can be categorized into three main groups:

Infrastructure-as-a-Service (IaaS): Cloud service providers allow users to lease their infrastructures. The providers can then create virtual servers which define the number of cores, the amount of RAM, the size of storage, etc., based on the need of customers and charge them for the usage time. Examples of IaaS Cloud providers are Amazon Elastic Compute Cloud (Amazon EC2) [29], and GoGrid [30], Rackspace [31].

Software-as-a-Service (SaaS): SaaS provides specific applications via the Internet and charges the customer for a subscription fee. SaaS can upgrade an application without the user interference. Users always use the newest version of services. Salesforce [32] is an example of SaaS that provides a Customer Relationship Management (CRM) application.

Platform-as-a-Service (PaaS): PaaS is a Cloud computing service that provides facilities to develop and deploy applications over the Internet. Users do not need to own hardware and software that they will run their applications. Google App Engine [33], Bungee [34], and Heroku [35] are examples of PaaS providers.

With the increasing demand of HPC, Cloud service providers have offered HPC-as-a-Service. Organizations that need to run their jobs on HPC systems do not need to worry about maintaining an HPC system, upgrading computing resources, or paying power and cooling cost. Examples of HPC-as-a-Service providers are Penguincomputing [36] and Amazon EC2. Amazon EC2 [29] also provides a General Purpose Graphic (GPU) cluster, which is the recent technology of parallel computing.

To deal with failures in Cloud systems, there are several fault tolerant techniques. A redundant technique or High Availability (HA) is commonly used. A Cloud system can create redundant virtual machines in different physical machines. The Cloud system will monitor primary VMs. If a failure happens to the primary VMs, the system will automatically fail over to the redundant VMs and fail back when the primary VMs resume their operation. Furthermore, live migration is a pro-active fault tolerant technique that allows virtual machines to move from the physical machines that are likely to fail without any interruptions. Moreover, checkpoint/restart [37] can be deployed to save a state of a VM or application. When a failure happens, it will restart the VM from the last saved state. To optimize waste time and deploy fault tolerant techniques efficiently, understanding reliability information of Cloud systems is crucial. In this dissertation, system reliability, failure rate, and Mean-time-to-failure (MTTF) of Cloud system will be discussed in Chapter 5.

1.3 Organization of Dissertation

This dissertation is organized as follows: Chapter 2 discusses related work on a reliability model with correlated failures in an HPC environment, a reliability model of Cloud computing for HPC applications, and an economic model for a Cloud service provider. Marshall-Olkin multivariate and parameter estimation techniques for correlated failure problem are discussed in Chapter 3. Chapter 4 presents a novel reliability model with correlated failures in a *k*-node system for HPC applications. Chapter 5 proposes a new reliability model of Cloud computing for HPC applications. An economic model for Cloud service providers is described in Chapter 6. The conclusion and future work are discussed in Chapter 7.

CHAPTER 2

RELATED WORK

This chapter discusses the related work on a reliability model of HPC systems as well as a reliability model of Cloud computing. The related work on an economic Cloud model is also discussed here.

2.1 A Reliability Model of HPC Systems

There are many studies in the literature that are concerned with the reliability behaviors of HPC systems. Results in [38][39][40][41], showed that the time to failure of an HPC system is best fitted by models with a time-varying failure rate distribution, such as the Weibull, gamma and Lognormal distributions. However, previous researches [42], [43] assume the time to failure to be the exponential distribution with constant failure rate. Heath [38] studied the time between failures (TBF) of workstations and found that the failures follow a Weibull distribution as showed in Equation (2.1). Xu [39] showed that the TBF's of individual Windows NT Servers also fit the Weibull distribution well. Recently, in [44], the authors developed a TTF distribution model for a *k*-node system, where individual nodes follow a Weibull with independent failures assumption among nodes. The Probability Distribution Function (PDF) of the Weibull distribution [44] with scale (α) and shape (β) is given by:

$$f(x) = \frac{\beta}{\alpha}\left(\frac{x}{\alpha}\right)^{\beta-1} e^{-\left(\frac{x}{\alpha}\right)^{\beta}}. \tag{2.1}$$

The Cumulative Distribution Function (CDF) of the Weibull is given by

$$F(x) = 1 - e^{-\left(\frac{x}{\alpha}\right)^{\beta}}. \tag{2.2}$$

The PDF of the conditional Weibull distribution is given by

$$f(t+x|t) = \frac{\beta}{\alpha^{\beta}}(t+x)^{\beta-1}e^{\frac{t^{\beta}-(t+x)^{\beta}}{\alpha^{\beta}}}. \tag{2.3}$$

The CDF of the conditional Weibull is given by

$$F(t+x|t) = 1 - e^{\frac{t^{\beta}-(t+x)^{\beta}}{\alpha^{\beta}}}. \tag{2.4}$$

Then, the authors [44] developed the PDF of the j^{th}Time to Failure (TTF) for a *k*-node system expressed as

$$s_j(x_j) = \sum_{i=1}^{k} f_i \prod_{\substack{l=1 \\ l\neq i}}^{k}(1 - F_l), \tag{2.5}$$

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

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

The CDF of the system TTF of a *k*-node system is given by

$$S_j(a) = P(t \le a) = \int_0^a \sum_{i=1}^{k} f_i(\tau) \prod_{\substack{l=1 \\ l\neq i}}^{k}\big(1 - F_l(\tau)\big)d\tau. \tag{2.6}$$

The *k*-node system reliability is given by

$$R_j(a) = 1 - S_j(a) = \prod_{l=1}^{k}(1 - F_l(a)). \tag{2.7}$$

The *k*-node failure rate is given by

$$\lambda_j(x) = \frac{\sum_{i=1}^{k} f_i \prod_{\substack{l=1 \\ l\neq i}}^{k}(1-F_l)}{1-\int_0^x \sum_{i=1}^{k} f_i(\tau) \prod_{\substack{l=1 \\ l\neq i}}^{k}(1-F_l(\tau))d\tau}. \tag{2.8}$$

The *k*-node mean time to failure (MTTF) is given by

$$E(x_j) = \int_0^\infty x_j \sum_{i=1}^{k} f_i \prod_{\substack{l=1 \\ l\neq i}}^{k} (1 - F_l). \tag{2.9}$$

Some existing studies [41], [44] assume that there is no correlation among nodes with regard to time to failure. However, a number of studies have shown that nodes in the same system can be correlated. Xu [39] showed that there is failure dependency of Windows NT Servers across the network. Lin [45] studied error log analysis and showed that there are correlated times to failure among nodes in the systems. Also, Gottumakkala [46] analyzed failure correlation using the Spearman correlation coefficient. The Spearman correlation coefficient is given by:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2-1)}, \tag{2.10}$$

where d is the difference in paired ranks, and n is number of cases.

He classified the correlation coefficient values to define the significant of correlations as shown in Table 2.1. The result showed that 10 percent of the Time to failures (TTF) have a strong correlation, 30 percent have a high correlation, and 60 percent have a weak correlation. Moreover, the related research [47] analyzed the data of 22 high-performance computing (HPC) systems collected over a nine year period at the Los Alamos National Laboratory (LANL). The study showed that more than 30% of nodes failed at the same time. Therefore, in order to estimate more accurately HPC system reliability, the model should consider the possibility of simultaneous failures among nodes. The fact that nodes can fail simultaneously gives rise to correlations among them with regard to time to failure. This dissertation proposes a reliability model of a system of k nodes with correlated failures for HPC applications in Chapter 4.

Table 2.1 Significance of correlation with respect to correlation coefficient values [46]

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

2.2 A Reliability Model of Cloud Computing

Many researchers have studied Cloud computing reliability. Kashi [48] studied characteristics of hardware failure and hardware repair in Cloud computing. The result showed that when a server failed, it has a greater chance of failing again. Also, successive failures best fit an inverse curve. Moreover, Dai [49] introduced Cloud service reliability. They classified failures into two groups, request stage failures and execution stage failures. The request stage failures are calculated from overflow failures and timeout failures by a Markov process. The reliability of request stage is given by:

$$R_{request} = \sum_{n=0}^{S-1} q_n + \sum_{n=S}^{N-1} q_n \int_0^{T_d} f_n(t)dt, \tag{2.11}$$

where $f_n(t)$is the PDF of waiting time to finish the n requests, q_n is the steady probability that the system stays at state n, S is the number of homogeneous schedule servers. N is the capacity of the request queue, $T_d$is the due time.

The execution stage failures included data resource missing, computing resource missing, software failure, database failure, hardware failure, and network failure. To derive the reliability of the execution stage, the new evaluation based on Bayesian theorem and Graph theory was proposed. A minimal combination of availability elements, links and nodes, is called a Minimal Execution Spanning Tree (MEST). The reliability of a Cloud service that has *N* MESTs is given by:

$$R_{execute} = Pr(\cup_{i=1}^{N} MEST_i) = \sum_{j=1}^{N_i} \Pr(E_i) \Pr(\bar{E}_1, \dots, \bar{E}_{j-1} | E_j), \qquad (2.12)$$

where E_j is the successful operation of the $MEST_j$ and $\bar{E}_j$ is the unsuccessful operation of the $MEST_j$. The reliability of a Cloud service can be calculated by:

$$R_{Service} = R_{request} \cdot R_{execute}. \qquad (2.13)$$

Hacker, in [50], considered hardware reliability and assumed a virtual machine per server. The proposed model was based on the Weibull distribution. However, this work did not incorporate software reliability into the model. None of the work considers reliability of Cloud computing for HPC applications. Chapter 5 discusses the reliability of a Cloud computing application that includes both hardware and software.

Many studies have shown that the time to system failure (TTF) can be described by a Weibull distribution [39], [50], [51]. Hacker [50] studied the impact of reliability rates among individual components on the HPC system reliability. He also showed that the time between failures for an HPC system followed a Weibull distribution. Xu [39] also showed TTF's of Windows NT servers follow a Weibull distribution. Gottumukkala et. al, [44] developed the reliability model of a *k*-node system for HPC applications when individual TTF follows a Weibull distribution. They considered the excess life or time since the last failure of an individual node in their reliability model to gain a more accurate estimation of reliability of the system based on an assumption that nodes fail independently. However, time to failure of some computer systems may not be independent [39], [47]. Xu [39] showed that there is failure dependency of Windows NT Servers across the network. Schroeder [47] presented a study of failures in an HPC system at Los Alamos National Laboratory (LANL) that found that nodes were correlated with regard to failure and that two or more nodes may fail at the same time. chapter 5 considers the excess life as well as correlation due to possible simultaneous failures of nodes in the system.

For software reliability, there are many studies undertaken to understand the characteristics of a software failure [52], [53], [54]. Alan [52] was interested in understanding software reliability models and their utility. He evaluated nine different software reliability models. He showed that a simple exponential model performs as well or better than complex models, and the simple model outperformed the other models in terms of both stability and predictive ability. Musa [53] evaluated seven model groups on 15 data sets. Based on the evaluation, the exponential and logarithmic models were recommended for modeling software reliability. Thirumurugan [54] studied software reliability modeling in the testing and operational phases. In the operational phase, software fault was not removed. Failures occurred at a constant rate over time [54], [55]. He considers an exponential software reliability model. Results showed that, at any given time, the reliability in the operational phase was less than that in the testing phase.

The majority of software reliability models make the assumption that failures occur independently. However, evidently in [56], the failures are not always independent. G-Popstojanova [57] extended the classic software reliability theory, Markov renewal modeling, in order to formulate software reliability models that considered dependent failures and time to failure following the exponential distribution.

2.3 An Economic Model for Cloud Computing Environment

Pricing and economic analysis have been previously applied in grid and Cloud computing. Abdelkader [58] studied an economic model for resource allocation in grid computing and treated computational and storage resources as interchangeable. Price was determined by demand and supply at the equilibrium point. Dash [59] proposed a self-tuned economic model for a query service of scientific information based on the quality of service and on profit guarantee. In his work, the economic model accounted for CPU time, bandwidth, network, and disk space. Mihailescu [60] presented a dynamic pricing or an auction model for allocating resources in large distributed systems. Woitaszek and Tufo [61] analyzed an economic charging model in the perspective of a supercomputing service provider. They examined IBM Blue Gene/L system, considered its cost and applied a charging model that is used in Amazon EC2 to calculate a break-even point. Results showed that pricing is not competitive with the commodity system in Amazon EC2. Xinhui [62] considered not only Cloud Total Cost of Ownership (TCO) [62] but utilization cost as well. The utilization cost is the cost calculated from using part of the resources in order to evaluate efficiency of the Cloud. The utilization cost is calculated according to the number of VMs. Walker [63] presented the real cost of a CPU hour for three cases: purchase, lease and purchase-upgrade. He calculated the Net Present Value and the Net Present Capacity of each case. Niyato [64] proposed economic models to determine an optimal strategy between private Cloud and Cloud service providers for a monopoly market, Nash equilibrium for competitive market, and bargaining for cooperative market. Chapter 6 presents an economic model study, based on market share and TCO, for a Cloud service provider in order to maximize profit.

CHAPTER 3

MARSHALL-OLKIN MULTIVARIATE MODEL AND PARAMETER ESTIMATION

This chapter discusses Marshall-Olkin Multivariate Exponential and Weibull distribution. Marshall-Olkin Multivariate models are used to model the correlation between components. Marshall-Olkin Multivariate models account the components failing at the same time, which is applicable to model correlated failures as in the litelature [47] that 30% of the failures in LLNL systems failing at the same time. Furthermore, the Marshall-Olkin Weibul distribution has Weibul marginals, which appropriate to model failures of an individual node following the Weibul distribution. Also introduced is an univariate method for the parameter estimation of Marshall-Olkin Multivariate models. The advantage of this parameter estimation technique is that it is simple to estimate the parameters when the number of nodes is large while an existing Parameter Estimation technique does not easily extend to estimate the parameters when the number of nodes is larger than three. In addition, simulations to validate the parameter estimations are also examined.

Acronyms

CDF	Cumulative Distribution Function
PDF	Probability Density Function

TTF	Time to Failure
LLNL	Lawrence Livermore National Laboratory
LANL	Los Alamos National Laboratory
HPC	High Performance Computing
GOF	Goodness-of-fit
MOMED	Marshall-Olkin's Multivariate Exponential Distribution

Notations

$Z_j(t, \lambda_i)$	independent Poisson process of shock to component i
t_{ij}	Survival time of the ith node to the jth re-start of the system
$R_j(x)$	reliability of the system after the j^{th} restart of the system
$\lambda_j(x)$	failure rate of the system after the j^{th} restart of the system

3.1 Marshall-Olkin Multivariate Exponential Model

This multivariate survival expression in Equation (3.1) has k components and $2^k - 1$ parameters, and it is based on the fatal shock model [65], [66]. Suppose that a component fails after receiving a fatal shock. The occurrence of shocks is based on independent

$$\bar{F}_{Y_1 \dots Y_k}(y_1, \dots, y_k) = \exp\{-\sum_{i=1}^{k} \lambda_i y_i - \sum_{i<s} \lambda_{i,s} \max(y_i, y_s) - \sum_{i<s<l} \lambda_{i,s,l} \max(y_i, y_s, y_l)$$

$$- \dots - \lambda_{1,2,\dots,k} \max(y_i, y_2, \dots, y_k)\}. \tag{3.1}$$

Poisson process $Z_i(t, \lambda_i)$, with i = 1 2… k. In each Poisson process, t >0 and lambda is the Poisson parameter. The events in the Poisson process, $Z_i(t, \lambda_i)$ are shocks to component i, the events in the process, $Z_{i,s}(t, \lambda_{i,s})$ are simultaneous shocks to both components i and s, the events in the process $Z_{i,s,l}(t, \lambda_{i,s,l})$ are simultaneous shocks to components i, s and l and the events in the process $Z_{1,2,\dots,k}(t, \lambda_{1,2,\dots,k})$ are simultaneous shocks to all k components. The Marshall-Olkin Multivariate Exponential Distribution (MOMED) is the only multivariate exponential distribution model that has a marginal exponential distribution, which the failures of the individual node still exponential distribution.

Equation (3.1) can be used to obtain the reliability model for a system of k after the j^{th}re-start (j = 1,2,3,…). Here, re-start refers to the case when a failed node or nodes are replaced to re-start the system. In the bivariate case, the general case presented in Equation (3.1) reduces to

$$\bar{F}_Y(y_i, y_s) = P[Y_i > y_i, Y_s > y_s] = \exp\{-\lambda_i y_i - \lambda_s y_s - \lambda_{1,2} Max(y_i, y_s)\}. \qquad (3.2)$$

Hence, the PDF is obtained as:

$$f(y_i, y_s) = (-1)^2 \frac{\partial \bar{F}(y_i, y_s)}{\partial y_i \partial y_s} =$$

$$\begin{cases} (\lambda_i + \lambda_{1,2})\lambda_s \exp\{-(\lambda_i + \lambda_{1,2})y_i - \lambda_s y_s\}, where\ y_i\ is\ max \\ (\lambda_s + \lambda_{1,2})\lambda_i \exp\{-(\lambda_s + \lambda_{1,2})y_s - \lambda_i y_i\}, where\ y_s\ is\ max. \end{cases} \qquad (3.3)$$

where $i, s = 1,2\ and\ i \neq s$.

As known from the literature [53], an exponential distribution is appropriate for modeling software reliability (Applications, VM's and Hypervisors), which will be discussed more in Chapter 5. Software failures may be independent or dependent. Independent software failures, for example, can occur in the case of embarrassingly parallel jobs, such as biological sequence and video encoding. In case of dependent failure, system configuration and operation environment may cause dependent software failures [38]. Moreover, applications may fail at the same time, due to certain situation such as communication outages between processes. Blocking communication, for example, on an MPI application may cause simultaneous process failures. Another example is simultaneous failures due to the fact that applications may have to wait on data to become available.

3.2 Marshall-Olkin Multivariate Weibull Model

The Marshall-Olkin multivariate Weibull model is derived from the multivariate exponential model by a variable transformation technique used in [65]. In the survival function of the multivariate exponential (Equation (3.1)), consider the transformation $Y_i = X_i^c$, c>0, $i = 1,...,k$. Using the transformation of variable technique in [67], one can obtain the survival function of the general multivariate Weibull (MVW) which is given by:

$$\bar{F}_{X_1 \ldots X_k}(x_1, \ldots, x_k) = \exp\{-\sum_{i=1}^{k} \lambda_i x_i^c - \sum_{i<s} \lambda_{i,s} \max(x_i^c, x_s^c) - \sum_{i<s<l} \lambda_{i,s,l} \max(x_i^c, x_s^c, x_l^c)$$

$$- \cdots - \lambda_{1,2,\ldots,k} \max(x_i^c, \ldots x_k^c)\}. \tag{3.4}$$

Hanagal [65] and Prochan [68] considered parameter estimation for the special case of the model in Equation (3.4). A special case of the Multivariate Weibull model presented below.

$$\bar{F}_{X_1 \ldots X_1}(x_1 \ldots x_k) = \mathrm{P}(X_1 > x_1, \ldots, X_k > x_k)$$

$$= \exp\{-\lambda_1 x_1^c - \cdots - \lambda_k x_k^c - \lambda_{1,2,\ldots,k} Max(x_1^c, \ldots, x_k^c)\}. \tag{3.5}$$

The model considers only the $Z_{1,2,\ldots,k}(t)$, which is the Poisson process governing simultaneous shocks to components 1,2,...,k. The model is justifiable in an environment where all components are simultaneously exposed to shocks.

The marginal function of X_i, i = 1,..., k is a Weibull distribution and can be expressed as:

$$P(X_i > x_i) = \bar{F}_x(0, \ldots, x_i, \ldots, 0) = exp\{-(\lambda_i + \lambda_{1,2,\ldots,k})x_i^c\}, i = 1, \ldots, k. \qquad (3.6)$$

A common c assumption has been used in the literature by Proscan [68], D. Kundu and A. K. Dey [69] and Hangal [65]. One advantage of a common c value is that it gives a multivariate Weibull which marginal is a Weibull and agrees with literatures [39], [44], [46] that an individual node's failures follow the Weibull distribution.

3.3 Parameter Estimation for Marshall-Olkin Weibull Distribution

Denote a PDF of the Weibull distribution by W(c,λ). Consider three independent processes, $U_{12} \sim W(\lambda_{12}, c)$, $U_1 \sim W(\lambda_1, c)$, and $U_2 \sim W(\lambda_2, c)$. Here, U_{12} is the distribution governing the time to simultaneous failure of the two nodes, U_1 is that for time to failure of node 1, and U_2 that of node 2. Let $X_1 = \min\{U_{12}, U_1\}$, and $X_2 = \min\{U_{12}, U_2\}$. The bivariate vector (X_1, X_2) has the bivariate Weibull distribution with the parameters $\lambda_1, \lambda_2, \lambda_{12}, c$ [69].

This can be expressed as:

$$f(x) = c^2(\lambda_i + \lambda_{12})\lambda_j x_i^{c-1} x_j^{c-1}) * \exp\{-(\lambda_i + \lambda_{12})x_i^c - \lambda_j x_j^c\},\ i, j = 1,2 \text{ and } i \neq j. \qquad (3.7)$$

For parameter estimation of the multivariate Weibull, not much work has been done due to the complexity of the problem. There is no explicit form for the Maximum Likelihood Estimators (MLE) of the parameters. Moreover, The MLE of the unknown parameters does not always exist [69]. Kundu [69] proposed an EM algorithm to estimate the Multivariate Weibull when k = 2 or 3. This method is difficult to extend to estimate the parameters of the Multivariate Weibull in the general case of k nodes.

A new method of estimation is proposed that can be easily extended to multivariate cases. For two nodes, this method (referred to as the univariate method) considers the observed time to failure for U_{12}, which is the failure time that both nodes fail together, U_1 or U_2, which is the failure time that node 1 or node 2 fail independently. Maximum Likelihood Estimates for λ_1, λ_2 , λ_{12} and c can then be obtained from the Weibull distributions W(λ_1, c), W (λ_2, c), and W (λ_{12}, c). The likelihood function is expresses as:

$$L(x_1, \dots, x_n; \lambda, c) = \prod_{j=1}^{n} c\lambda x^{c-1} \exp\{-\lambda x^c\}. \tag{3.8}$$

In this case three estimates for c are obtained. One can then take the average of the three estimates as an estimate for c. For cases that more than two nodes (k nodes), Maximum Likelihood Estimates for $\lambda_i$and c for node i, where i = 1, 2, ..., k, failing independently can be obtained from the Weibull distribution W(λ_i, c), which is estimated from the observed time U_i. Maximum Likelihood Estimates for $\lambda_{12\ldots k}$ can be obtained from the Weibull distribution W ($\lambda_{12\ldots k}$, c), which is estimated from the observed time $U_{12\ldots k}$. Parameter c can be obtained by averaging from every Maximum Likelihood Estimate.

3.4 Simulation of Parameter Estimation

The simulation technique is used to validate the two estimation methods, the EM algorithm and the proposed univariate method described above. The procedure that was used to run simulation in order to generate time to failure for the case of two nodes was defined as follows [69]. Consider the processes $U_{12} \sim W(0.3,1.2)$, $U_1 \sim W(0.5,1.2)$, and $U_2 \sim W(0.3,1.2)$. Here, X_1 and X_2 are the time to failure for node 1 and node 2, respectively. Next, the parameters were estimated using the EM algorithm and the univariate method. The number of observations (sample size) was 10, 25, 50 and 100 per replication. There were 100 replications for each sample size. Tables 3.1 and 3.2 present the estimates for the two methods. Also, Tables 3.3 and 3.4 give the variance of estimates for the two methods. A comparison of the average of each estimate over 100 replications to its parameter shows that both methods gave accurate results. As expected, the estimates became closer to their parameter values as the sample size increased from 10 to 100. Also, the variance of the estimates decreased with an increase in sample size. Both methods seem to be equally good. However, the estimation method, based on the univariate distribution, is simple to use, especially for any number of nodes while the existing EM parameter estimation does not easily extend for more than four computing nodes. Tables 3.5 and 3.6 shows an example of the parameter estimation and variance of the univariate method for five nodes with initial parameter $\lambda_1 = 0.5, \lambda_2 = 0.3$, $\lambda_3 = 0.3, \lambda_4 = 0.7$, $\lambda_5 = 0.5$, $\lambda_{12345} = 0.5$, $C = 1.2$. The results show that the estimates approach to the initial parameter when increasing the sample size and variance decreases as the sample size increases.

Table 3.1 Average parameter estimates, over 100 replications, using the EM algorithm for different sample sizes

Parameters / Sample size	$\bar{\lambda}_1 = 0.5$	$\bar{\lambda}_2 = 0.3$	$\bar{\lambda}_{12} = 0.3$	$\bar{c} = 1.2$
N = 10	0.5032	0.3512	0.3213	1.272
N = 25	0.5079	0.3033	0.3130	1.2377
N = 50	0.4993	0.3102	0.3065	1.2210
N = 100	0.4990	0.3070	0.3138	1.2014

Table 3.2 Average parameter estimates, over 100 replications, using the univariate method for different sample sizes.

Parameters / Sample size	$\bar{\lambda}_1 = 0.5$	$\bar{\lambda}_2 = 0.3$	$\bar{\lambda}_{12} = 0.3$	$\bar{c} = 1.2$
N = 10	0.5119	0.2973	0.2747	1.4087
N = 25	0.4995	0.2978	0.3030	1.2753
N = 50	0.5057	0.3057	0.2989	1.2329
N = 100	0.5032	0.2933	0.3004	1.2162

Table 3.3 Variances of the parameter estimates for each sample size using the EM algorithm

Parameters / Sample size	var(λ_1)	var(λ_2)	var(λ_{12})	var(c)
N = 10	0.0351	0.0409	0.0231	0.0640
N = 25	0.0227	0.01209	0.0145	0.0204
N = 50	0.0094	0.0074	0.0054	0.0083
N = 100	0.0058	0.0036	0.0027	0.0046

Table 3.4 Variances of the parameter estimates for each sample size using the univariate method

Parameters / # of Observation	var(λ_1)	var(λ_2)	var(λ_{12})	var(c)
N = 10	0.0590	0.0246	0.0248	0.18040
N = 25	0.0227	0.0111	0.0074	0.03526
N = 50	0.0085	0.0045	0.0044	0.19733
N = 100	0.0048	0.0025	0.0018	0.00906

Table 3.5 Average parameter estimates, over 100 replications, using the univariate method for different sample sizes for *k*=5.

Parameters / Sample size	$\bar{\lambda}_1 = 0.5$	$\bar{\lambda}_2$=0.3	$\bar{\lambda}_3$=0.3	$\bar{\lambda}_4$=0.7	$\bar{\lambda}_5$=0.4	$\bar{\lambda}_{12345}$=0.5	$\bar{c} = 1.2$
N = 10	0.5144	0.2992	0.2752	0.7882	0.3988	0.4981	1.39015
N = 25	0.4954	0.2889	0.3023	0.7262	0.4135	0.4955	1.2708
N = 50	0.5087	0.3067	0.3167	0.6983	0.4057	0.4873	1.2266
N = 100	0.4972	0.2976	0.3055	0.6955	0.3998	0.4894	1.2201

Table 3.6 Variances of the parameter estimates for each sample size using the EM algorithm for *k*=5

Parameters / Sample size	$var(\lambda_1)$	$var(\lambda_2)$	$var(\lambda_3)$	$var(\lambda_4)$	$var(\lambda_5)$	$var(\lambda_{12})$	$var(c)$
N = 10	0.0458	0.0231	0.0221	0.1283	0.0359	0.0385	0.1313
N = 25	0.0166	0.0084	0.0092	0.0354	0.0147	0.0148	0.0442
N = 50	0.0090	0.0051	0.0051	0.0156	0.0042	0.0093	0.0210
N = 100	0.0047	0.0021	0.0022	0.0063	0.0036	0.0039	0.0096

CHAPTER 4

A RELIABILITY MODEL WITH CORRELATED FAILURES FOR HPC APPLICATIONS

In the previous work, there is only an independent case of reliability estimation for HPC systems. This chapter will develop a reliability estimation with correlated failures for HPC applications. The reliability model is validated by log data from Blue/GeneL system from LLNL with K-S goodness-of-fit test. A system failure rate and mean time to failure of such applications are also presented.

4.1 Introduction

Large scale High Performance Computing (HPC) systems play a critical role in computational science. To meet the demand for faster computing, HPC systems are composed of thousands of computing nodes. For example, the number of nodes of the IBM Blue Gene/L at Lawrence Livermore National Lab is 65,536 nodes or 131,072 cores. However, in many cases, when a system has a single node failure, it results in the total application failure [44]. Therefore, it is important to study the reliability of such large HPC systems

A runtime scheduler or resource manager can make use of system reliability information to improve the operation of an HPC system. For example, resource managers can allocate a job (in an application), depending on the reliability information of the nodes, in order to minimize performance loss in the considered system. The intelligent job scheduler may allocate the long-running jobs to the more reliable nodes and may schedule the short-running jobs to the less reliable nodes. Therefore, one can consider system reliability in order to effectively utilize HPC resources to maximize performance. Furthermore, the system reliability information of a system is an important factor to use in fault tolerance performance models that help in mitigating system failures.

Time to Failure (TTF) provides important information on the reliability of a system. There are several studies in the literature [38],[39],[40] which show that the distribution of the time to failure (TTF) of a computer system is in accord with a time-varying failure rate distribution such as the Weibull. In the previous study [46], a reliability model was developed for an HPC system composed of k independent nodes, each having a Weibull or a conditional Weibull TTF distribution. It has been shown in [46] that the Weibull distribution gives the best fit to the TTF of each node in the LLNL ASC white system. However, nodes in a computer system may also exhibit correlations with regard to their TTF [39], [45]. Schroeder and Gibson [47], in a study of failures in an HPC system at Los Alamos National Laboratory (LANL), showed that nodes were correlated with regard to failure and that some nodes may fail at the same time. Hence, it is necessary to develop a system reliability model taking into account simultaneous failure of nodes.

In this chapter, a reliability study developed in [46] is extended, based on independence among nodes, to model the reliability of an HPC system where nodes are correlated in their time to failure due to the occurrence of simultaneous failures. It is assumed that the system of k nodes fails when any of the nodes fail. When a node fails, it is replaced or renewed, and the system is re-started again. The Weibull is considered as the distribution of time to failure and uses the multivariate Weibull distribution, based on the well-known Marshall–Olkin multivariate exponential model described in Chapter 3, to account for simultaneous failures among nodes.

4.2 System Reliability Model, Failure Rate and Mean Time to Failure

This dissertation considers a system of k nodes. Nodes can fail at the same time which leads to nodes being correlated with regard to their times until failure. The previous work [46] supports the fact that the time to failure of a node was best described by a Weibull distribution. Hence, without loss of generality, the Weibull is considered as the distribution of time to failure in this study. It is assumed that the system of k nodes fails when any of the nodes fail.

The focus in this chapter is on a model where the nodes are not repaired, but rather replaced. This is a realistic scenario in practice and has an advantage over repair concerning delay in operation time and therefore cost of operation. In HPC systems there are a relatively large number of nodes, and one does not wait to repair, but instead replace using a spare node from the resource pool. It is not feasible to delay the computation operation for the node to be repaired. The practical and efficient approach is to replace the node and continue the operation, and in the meantime repair the failed node(s) if necessary.

The interest at this study is in determining the reliability of the system after the time it is re-started. It is important to note that the distribution of the time to failure is

Weibull for the node that is replaced and conditional Weibull for the other nodes. Note that because of simultaneous failures, more than one node may be replaced at the same time. The conditional Weibull is defined as the probability that a node will fail at time x_i, given that the node has survived until time t_{ij}, where i is the node that fails and j is the j^{th} system re-start. The conditional life PDF for a Weibull distribution can be expressed as:

$$f\left(t_{ij} + x_i \middle| t_{ij}\right) = \lambda c(t_{ij} + x_i)^{c-1} e^{-\lambda(-t_{ij}^c + (t_{ij}+x_i)^c}. \tag{4.1}$$

The CDF is given as:

$$P\{X \le t_{ij} + x_i \mid t_{ij}\} = 1 - e^{-\lambda(-t_{ij}^c + (t_{ij}+x_i)^c)} \ , \tag{4.2}$$

The k-node system reliability model can be expressed as follows:

$$R_j(x) = \bar{F}(x) = P(X_1 > x, X_2 > x, X_3 > x, \ldots, X_k > x)$$

$$= \exp\{-\textstyle\sum_{i=1}^{k} \lambda_i * x_i' - \sum_{\substack{i,s=1 \\ i<s}}^{k} \lambda_{i,s} \max\{x_i', x_s'\} - \sum_{\substack{i,s,l=1 \\ i<s<l}}^{k} \lambda_{i,s,l} \max\{x_i', x_s', x_l'\} - \ldots -$$

$$\lambda_{1,2,\ldots,k} \max\{x_1', \ldots, x_k'\}, \tag{4.3}$$

where

$$x_i' = \begin{cases} x^c & , if\ the\ i^{th} node\ is\ replaced\ at\ j^{th} restart \\ -t_{ij}^c + (t_{ij} + x)^c & , if\ the\ i^{th} node\ is\ not\ replaced\ at\ j^{th} restart. \end{cases}$$

Similarly for $x_s', x_l', \ldots, x_k'$.

Note that after the j^{th} failure of the system, node i that failed has a Weibull distribution (since it was replaced at the time t_{ij} when it failed) and the other nodes have conditional Weibull distributions since they survived beyond time t_{ij}.

Equation (4.3) is readily derived from Equation (3.1) using the transformation of variable technique with the transformation $Y_i = -t_i^c + (x_i + t_i)^c, t_i \ge 0$. In practice, t_i can be equal to zero, which corresponds to a node with a Weibull distribution or t_i larger than zero corresponding to a node with a conditional Weibull distribution.

From Equation (4.3), it is seen that the PDF can be expressed as follows:

$$f_j(x) = (-1)\frac{dR_j(x)}{dx}$$

$$= [c\sum_{i=1}^{k}\lambda_i * x_i'' + \sum_{\substack{i,s=1\\ i<s}}^{k}\lambda_{i,s} * x_{is}'' + \sum_{i,s,l=1}^{k}\lambda_{i,s,l} * x_{i,s,l}'' + \cdots + \lambda_{1,2,\dots,k} * x_{1,2,\dots,k}''] *$$

$$\exp\{-\textstyle\sum_{i=1}^{k}\lambda_i * x_i' - \sum_{\substack{i,s=1\\ i<s}}^{k}\lambda_{i,s}\max\{x_i', x_s'\} - \sum_{i,s,l=1}^{k}\lambda_{i,s,l}\max\{x_i', x_s', x_l'\} - \cdots -$$

$$\lambda_{1,2,\dots,k}\max\{x_1', \dots, x_k'\}, \tag{4.4}$$

where $x_i'' = x^{c-1}$ if the ith node is replaced at the j^{th} re-start

$= (t_{ij} + x)^{c-1}\ if$ the $i^{th}\ node$ is not replaced at the j^{th} re-start.

$x_{is}'' = \frac{d\max\{x_i', x_s'\}}{dx}$, $x_{isl}'' = \frac{d\max\{x_i', x_s', x_l'\}}{dx}$,..., and $x_{123\dots k}'' = \frac{d\max\{x_1', \dots, x_k'\}}{dx}$.

From Equation (4.3), it is seen that for the special case when one component is shocked or all components are shocked simultaneously, the reliability can be expressed as:

$$R_j(x) = \bar{F}(x) = P(X_1 > x, X_2 > x, X_3 > x, \dots, X_k > x)$$

$$= \exp\{-\textstyle\sum_{i=1}^{k}\lambda_i * x_i' - \lambda_{1,2,\dots,k}\max\{x_1', \dots, x_k'\}, \tag{4.5}$$

where $x_i' = x^c\ if\ the\ i^{th}\ node$ is replaced at the j^{th} re-start

$= -t_{ij}^c + (t_{ij} + x)^c\ if\ the\ i^{th}\ node$ is not replaced at the j^{th} re-start.

The PDF of the system for the special case described in Equation (4.5) is expressed by:

$$f_j(x) =$$

$$[c\textstyle\sum_{i=1}^{k}\lambda_i * x_i'' + \lambda_{1,2,\dots,k} * x_{1,2,\dots,k}'']*\exp\{-\sum_{i=1}^{k}\lambda_i * x_i' - \lambda_{1,2,\dots,k}\max\{x_1', \dots, x_k'\}, \tag{4.6}$$

where $x_i'' = x^{c-1}\ if\ the\ i^{th}\ node$ is replaced at the j^{th} re-start

$$= (t_{ij} + x)^{c-1} \ if \ the \ i^{th} \ node \text{ is not replaced at the } j^{th} \text{ re-start,}$$

and $x''_{123\ldots k} = \frac{d\max\{x'_1,\ldots,x'_k\}}{dx}$.

From Equations (4.4) and (4.5), one may define the failure rate "$\Gamma_j(x)$" after the j^{th} re-start of the system as:

$$\Gamma_j(x) = \frac{f_j(x)}{R_j(x)}$$

$$= c\sum_{i=1}^{k} \lambda_i * x''_i + \sum_{\substack{i,s=1 \\ i<s}}^{k} \lambda_{i,s} * x''_{i,s} + \sum_{i,s,l=1}^{k} \lambda_{i,s,l} * x''_{i,s,l} + \cdots + \lambda_{1,2,\ldots,k} * x''_{1,2,\ldots,k}. \quad (4.7)$$

For the special case, the failure rate is

$$\Gamma_j = \frac{f(x)}{R_j(x)} = c\sum_{i=1}^{k} \lambda_i * x''_i + \lambda_{1,2,\ldots,k} * x''_{1,2,\ldots,k}. \quad (4.8)$$

From the definition of expectation, the MTTF is given by

$$E(x) = \int_0^{\infty} x * f_j(x)dx, \quad (4.9)$$

where $f_j(x)$ is the PDF of an univariate distribution for the general case presented in Equation (4.5) or the special case presented in Equation (4.6) for a system of k nodes.

4.3 Goodness-of-fit Test for System Reliability

The system reliability model is validated by fitting it to experimental data using the Kolomogorov-Smirnov (K-S) goodness-of-fit test. The failure logs are obtained from the BLUE Gene/L system, located at Lawrence Livermore National Laboratory (LLNL). The BLUE Gene/L logs are from June 3rd, 2005 to January 4th, 2006. The system is comprised of 65,536 nodes. There were 99 nodes that failed more than 15 times within a six months period. Moreover, some of the nodes failed simultaneously, which implies a correlation among nodes with regard to time to failure.

Consider a system comprised of two nodes that are correlated with regard to their TTF's. The times to failure of the two nodes give rise to a bivariate distribution $(\mathbf{X_1}, \mathbf{X_2})$. The system reliability function can be obtained from Equation (4.5), for k=2, by letting $\mathbf{X_1} = \mathbf{X_2} = \mathbf{X}$. This gives

$$R_j(x) = \exp\{-(\lambda_1 + \lambda_2 + \lambda_{1,2})x^c\}. \tag{4.10}$$

In this case, $t_{ij} = 0$ since the bivariate data observed are the times to failure of each node after the two nodes are put into operation at the same time. It is important to point out that a general expression for t_{ij} not zero can be readily obtained from Equation (4.5). Only the case when $t_{ij} = 0$ is used in order to agree with the observed data. Using the EM algorithm on the bivariate TTF data, the parameters λ_1, λ_2 , $\lambda_{1,2}$ and c are estimated. These estimates were used to fitted the cumulative distribution function of the Weibull $(1 - R_j(x))$ to the observed Min(X_1, X_2) using the K-S goodness-of-fit test.

Table 4.1 presents the goodness-of-fit of the modified K-S test [70] to a Weibull distribution (Equation (4.10)) for 25 non-overlapping pairs of nodes. For each test, the K-S test statistic and the corresponding critical value of rejection at the 5% level [70] is presented. In this case, the modified K-S test is used since the Weibull parameters are estimated from data. From Table 4.1, it shows that the Weibull distribution gave a good fit to 23 pairs of nodes (critical value > K-S statistic). Figure 4.1 is representative of the observed fit to the experimental data. It is known that the K-S test is a sensitive test. Hence, even when the K-S test rejected the null hypothesis of a Weibull distribution, the fit seems to be still adequate, as seen graphically in Figure 4.2.

Table 4.1 K-S statistic and critical value of K-S goodness-of-fit test to the TTF of 25 systems, each represented by 2 nodes

System	Critical value	KS statistic	System	Critical value	KS statistic	System	Critical value	KS statistic
1	0.188	0.337	10	0.213	0.207	19	0.239	0.232
2	0.222	0.220	11	0.231	0.221	20	0.222	0.211
3	0.231	0.200	12	0.248	0.238	21	0.337	0.281
4	0.248	0.322	13	0.213	0.205	22	0.222	0.221
5	0.248	0.247	14	0.274	0.213	23	0.265	0.241
6	0.256	0.207	15	0.231	0.210	24	0.231	0.223
7	0.265	0.247	16	0.248	0.244	25	0.256	0.244
8	0.256	0.230	17	0.239	0.236			
9	0.222	0.220	18	0.314	0.290			

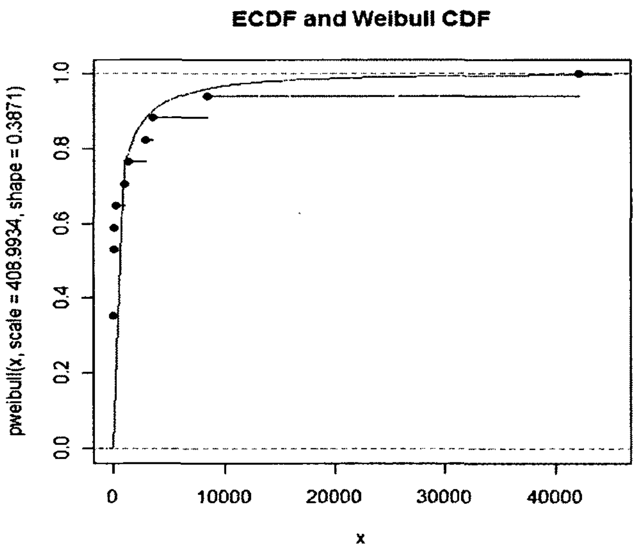

Figure 4.1 The empirical CDF (ECDF) and the Weibull CDF of a 2-node system, K-S statistic = 0.244 and its critical value = 0.256 from the K-S test.

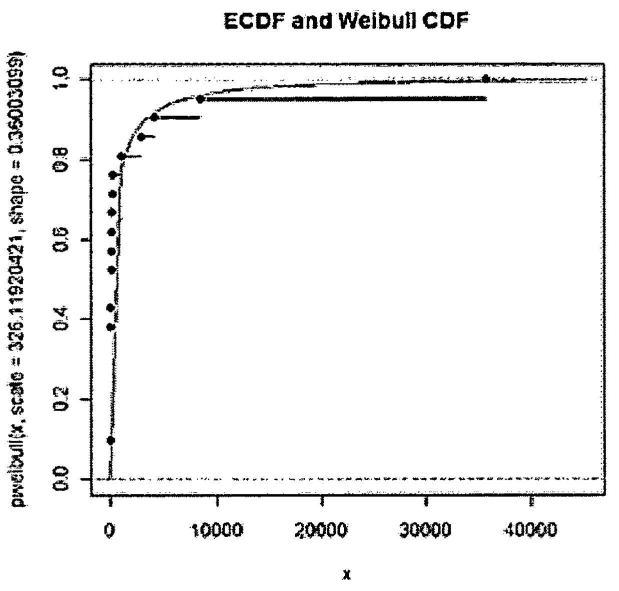

Figure 4.2. The empirical CDF (ECDF) and the Weibull CDF of a 2-node system, K-S statistic= 0.3371 and its critical value=0.188 from the K-S test.

4.4 Numerical Result for Correlated Failure Model

This section exemplifies, for k = 2, 3, and 50 the effect of survival time (T) and parameters for joint node failures ($\lambda_{i,j,...}$) on system reliability, from Equation (4.3), and on the failure rate, Equations (4.7) and MTTF, Equation (4.9). For k = 2, an example is used where job run length (x) equals to 200 hours and the Multivariate Weibull parameters equal to λ_1=0.00003, λ_2=0.00005, C=1.5, with $\lambda_{1,2}$={0, 0.00003, 0.00005, 0.00007}. For k=3, assume $\lambda_{1,2}, \lambda_{1,3}, \lambda_{2,3}$ = 0.00005, and $\lambda_{1,2,3}$= {0, 0.00003, 0.00005, 0.00007}. For k=50, define λ_i=0.000004 and C=1.5 with $\lambda_{i,j,...}$={$3x10^{-19}$, $5x10^{-19}, 7x10^{-19}$} It is obvious from Equation (4.3) that when the joint failure rates ($\lambda_{i,j,...}$) increase, the reliability of the system decreases. This implies that a system with independent nodes ($\lambda_{i,j,...}$= 0) is more reliable than one where the nodes are correlated in failure. Also, it is seen that the general model in Equation (4.3) is less reliable than the special case (k > 2) in Equation (4.5). Also, as the number of nodes increases, the reliability of the system decreases. Furthermore, reliability of the system decreases as survival time T increases. These facts are demonstrated in Figures 4.3, 4.4 and 4.5.

Figures 4.6, 4.7, and 4.8 show, as expected, that the MTTF decreases when the parameters for joint failure($\lambda_{i,j,\ldots}$) increase or as the survival time T increases. Moreover, Figures 4.9, 4.10 and 4.11 show that the failure rate increases as the joint parameters ($\lambda_{i,j,\ldots}$) increase in value or the survival time increases. Furthermore, one can observe that when the number of nodes (k) increases, MTTF decreases and the failure rate increases as expected.

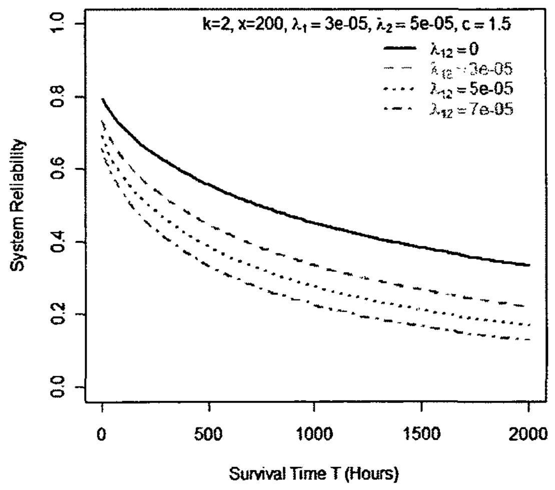

Figure 4.3. System reliability of a system of 2 nodes

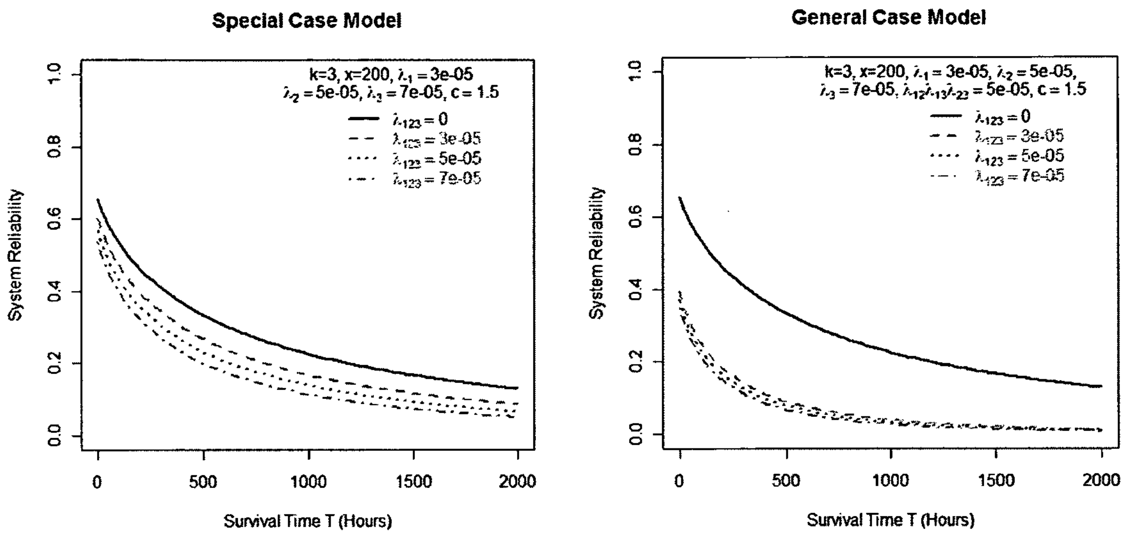

Figure 4.4 System reliability of a 3-node system in the special case and general case models

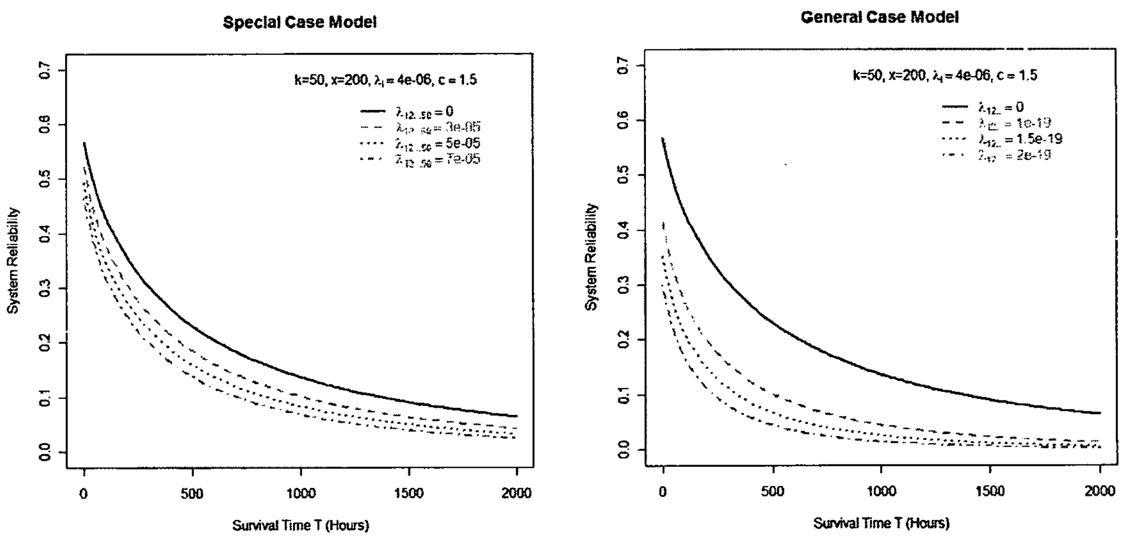

Figure 4.5 System reliability of a 50-node system in the special case and general case models

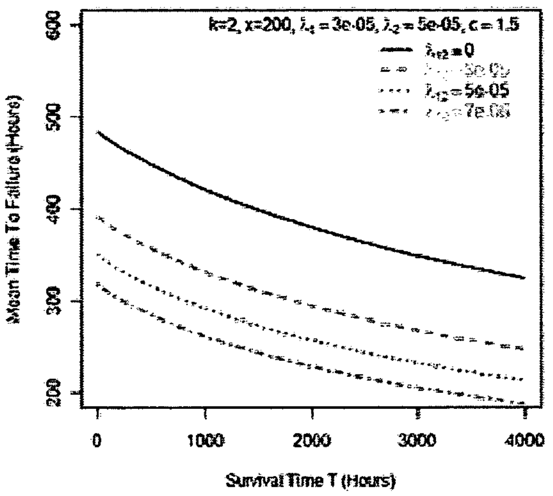

Figure 4.6 Mean time to failure of a system of 2 nodes

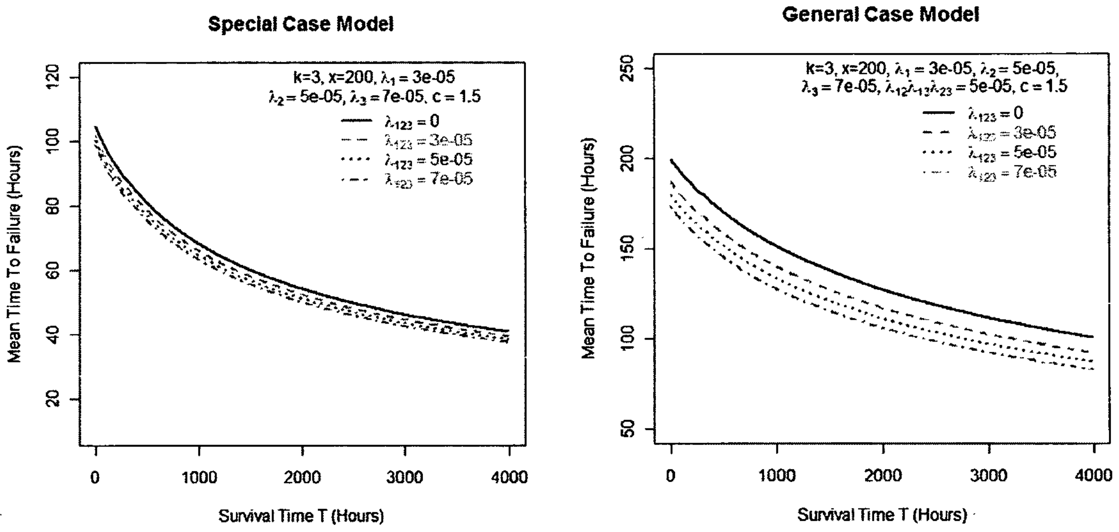

Figure 4.7 Mean time to failure of a 3-node system for the special case and the general case models

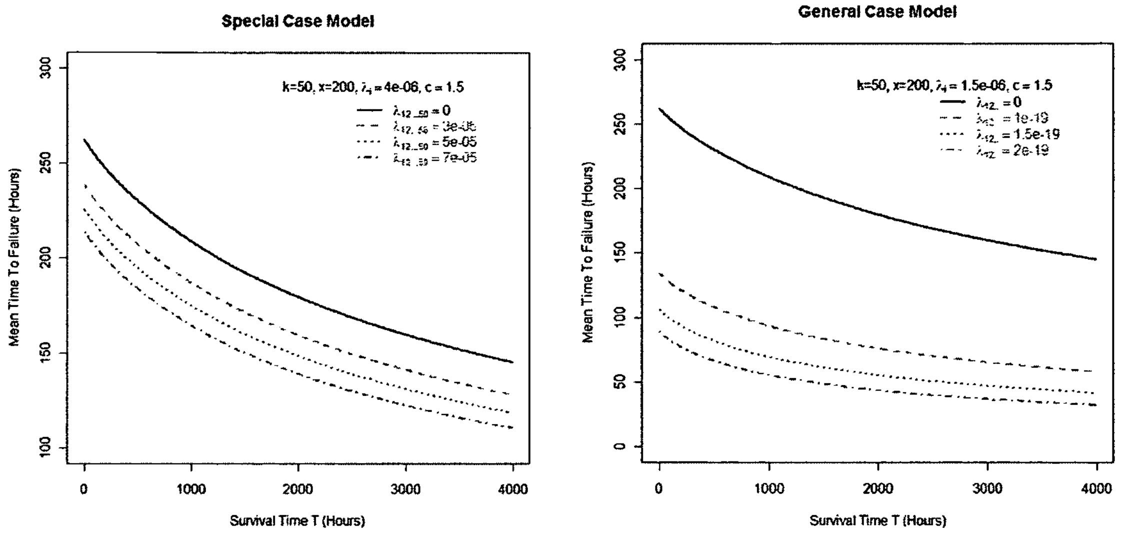

Figure 4.8. Mean time to failure of a 50-node system for the special case and the general case models

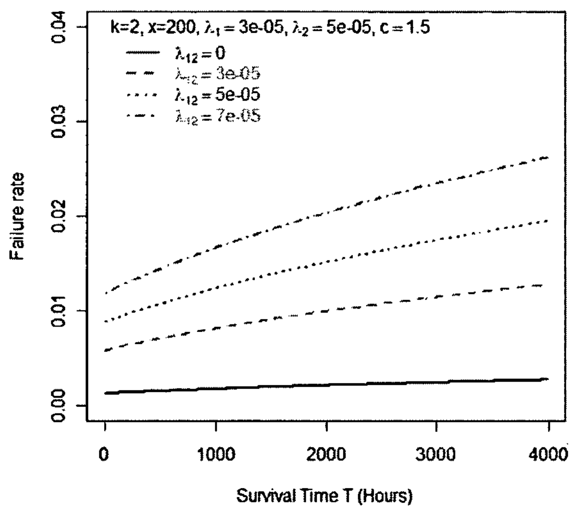

Figure 4.9 Failure rate of a system of 2 nodes

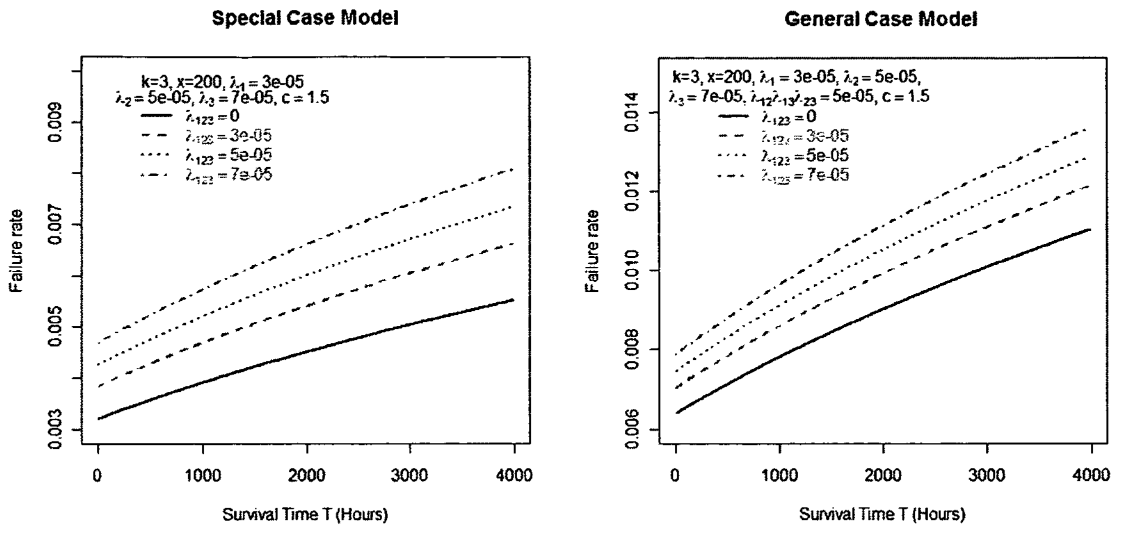

Figure 4.10 Failure rate of a 3-node system for the special case and the general case models

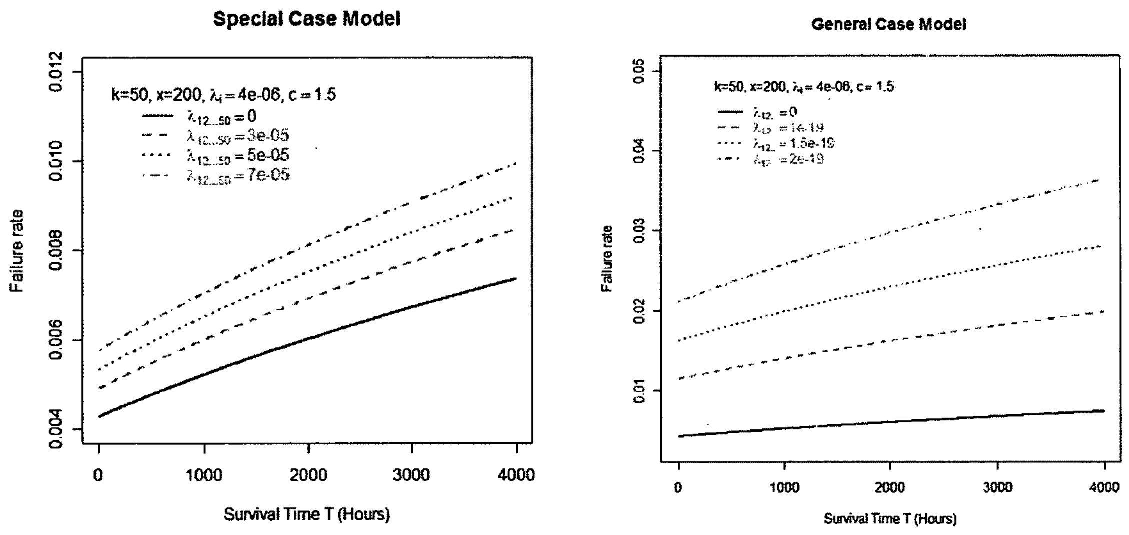

Figure 4.11 Failure rate of a 50-node system for the special case and the general case models

4.5 Conclusion

Reliability information is important for improving HPC system utilization. In order to mitigate the interruption of an application because of a failure, it is of importance to know how reliability, mean time to failure, and failure rate of an HPC system are affected by joint node failures. Many studies have shown the existence of failure correlations among nodes in computer systems [39], [45] and [47]. In this chapter, a general reliability model is proposed that accounts for joint failures among nodes and developed the reliability, failure rate, and mean time to failure of a system based on k nodes. Also validated is the model using observed times to failure from BlueGene/L system. Results show that if the failures of the components (nodes) in the system possess a degree of dependency, the system becomes less reliable. Moreover, when the failures of components are independent, the proposed system reliability model is the same as the one in [46]. Chapter 5 will extend this system reliability to model the reliability of Cloud computing system.

CHAPTER 5

A RELIABITLITY MODEL OF CLOUD COMPUTING FOR HPC APPLICATIONS

Cloud computing has become popular for HPC applications. Many public Clouds, such as Amazon Elastic Computing Cloud (EC2), provide their services specifically for HPC users. Therefore, reliability information is necessary for administrators and users to manage their systems and their jobs efficiently. This chapter presents reliability models of Cloud computing accounting for correlation failure in software and hardware.

5.1 Introduction

Cloud computing allows businesses to rent computing resources from Cloud service providers, such as Amazon, Rackspace, etc., instead of investing to acquire computing facilities. Users can increase or decrease the amount of resources- -such as storage, memory, and central processing unit (CPU) – as they need and pay per usage according to the amount of services or resources they need, like in a traditional public utility. When companies decide to use a Cloud service, they have in mind a Service Level Agreement (SLA), which involves the reliability and performance of the Cloud service within a desired time frame. Reliability of Cloud systems directly relates to its performance. When a system fails, applications that are running on the Cloud can be interrupted. If the system does not have any fault tolerant mechanisms, such as live migration and checkpoint/restart, failed jobs have to be re-run. If the system provides fault tolerant mechanisms, there is still some performance loss, re-computing time and the time it takes to restart the system to function again. Therefore, it is important to accurately estimate the reliability of a Cloud computing system in order to better mitigate faults and therefore effectively utilize its performance to achieve the SLA of Cloud users.

The demand for High Performance Computing (HPC) is increasing for solving advanced scientific researches and some mission critical applications. Many scientific and HPC applications are running on the Cloud. For instance, CernVM [71] is the Virtual machine image for running LHC experiment of High Energy Physics. In addition, Amazon EC2 presents HPC case studies [72]. Recently, Penguincomputing [36] has provided Cloud services customized for HPC, called HPC as-a Service [73].The benefit of Cloud computing is cost-efficient even though it has scarified performance trade-off [74].

In the Cloud environment, service providers must manage numerous computing components such as processors, memory modules, storage, network switches, etc. The more components, the more likely failures are to occur. A failure may interrupt an entire application, for instance Message Passing Interface (MPI) applications [44]. MPI has been used in most parallel scientific applications [75]. MPI on Cloud is active research area. For example, Ying [76] presents MPI algorithm model and Raihan [74] analyzed HPC applications and tested an MPI performance on the Cloud. If the service providers could obtain the failure characteristic of the Cloud computing environment, they can better manage the computing resources to tolerate the failures and sustain better performance [77], [78].

Reliability information is one of the key factors to consider in the Cloud computing environment. As such, many studies considered reliability as a main factor in their research on performance and usage cost. The work of Artur [79] focused on monetary versus reliability balance based on Spot Instances in the Amazon EC2. The Spot Instances are idle resources. Users can bid a price for idle resources. Whenever the price of resources is equal or less than the bid price, the Spot instances are allocated to the users. On the other hand, if the price goes above the bid price, the Spot instances are deallocated without warning. This leads to the question of how to bid the Spot Instances with given SLA constraints. Prasad [80] presented a resource allocation method for data processing, considering not just CPU speed, memory usage, data throughput, and network speed, but also reliability. In his work, Prasad presented algorithms for partitioning data in order to gain better performance and resource allocation for optimal pricing and for meeting SLA constraints.

This chapter proposes a reliability model for a Cloud computing system that considers software, application, Virtual Machine, Hypervisor, and hardware failures as well as correlation of failures within the software and hardware. The accuracy of the estimation is also improved by introducing excess life in order to account for the aging of hardware.

5.2 TTF Distribution of a Cloud System

This section considers the reliability of an application that has ℓ processes (App1-App ℓ) in a Cloud system composed of s virtual machines deployed in k nodes as shown in Figure 5.1, where each node can have a different number of virtual machines. An assumption is made that it is an MPI application. A system fails when any one of the k

nodes fails or any software components fails. In the hardware case, when any node fails, it is replaced with a new node and the system is re-started. In the software case, the VM is re-started and the system operates again.

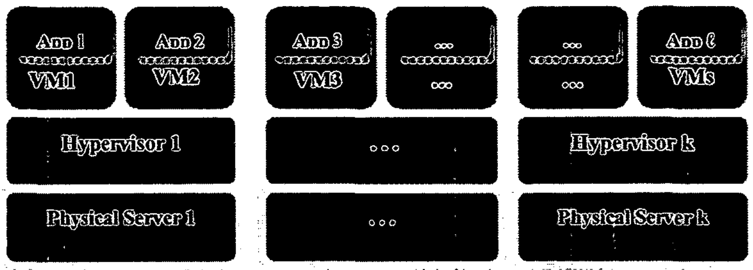

Figure 5.1 Cloud Computing Architecture

The interest is in determining the Probability Density Function (PDF) for the time to the first failure of the system after the j^{th} node replacement or a VM re-start due to a failure. This time is referred to as the time to failure (TTF). Therefore, the following assumptions can be made concerning the failure properties of individual nodes in a system,

1) TTF of an individual node follows a Weibull distribution [41], [46]

2) TTF of an application that is running on a particular VM has an exponential distribution [53], [55].

3) The first failure interrupts the entire application.

4) After a failure, the node is replaced with a new node at the next time instant, and hence the system returns to a normal operational mode.

Consider a k-node Cloud system with n ($n = \ell+s+k$) software components (App, VM and Hypervisor), each with an exponential time to failure), where any failure in any one of the n components interrupts the entire application. If any of the software components fails, the component (and hence the system) will be re-started. On the other hand, if a node fails, the node is replaced and the system re-started.

In case of node failures, the distribution of the time to failure is Weibull for the node that is replaced and excess Weibull for the other nodes. Note that because of simultaneous failures, more than one node may be replaced at the same time. The excess life is defined as the probability that a node will fail at time x given that the node has survived until time t. The excess life PDF for a Weibull distribution can be expressed as

$$f(t_{ij} + x \mid t_{ij}) = \lambda c(t_{ij} + x)^{c-1} e^{-\lambda(-t_{ij}{}^{c} + (t_{ij}+x)^{c})}, \tag{5.1}$$

with the CDF given as:

$$P\{X \le t_{ij} + x \mid t_{ij}\} = 1 - e^{-\lambda(-t_{ij}^{c} + (t_{ij}+x)^{c})} , \tag{5.2}$$

where i is the node that fails and j is the j^{th} system re-start.

5.3 Cloud System Reliability

In general, there could be many possible combination of failure dependencies among components. However, this study practically focuses on the four major scenarios described as follows.

Four possible scenarios are

(1) *Software failures occur independently.* Also, *hardware failures occur independently.*

From Equations (3.1) and (3.4), one can show that the reliability model of n software components $(X_{k+1,\ldots,}X_{k+n})$ running on k physical nodes $(X_1, \ldots, X_k)$ can be expressed as follows:

$$R_j(x) = \bar{F}(x) = P(X_1 > x, X_2 > x, \ldots, X_k > x, X_{k+1} > x, X_{k+2} > x, \ldots, X_{k+n} > x)$$

$$= \exp\{-\sum_{i=1}^{k} \lambda_i x_i' - \sum_{v=1}^{n} \gamma_v x\}, \qquad (5.3)$$

where $x_i' = x^c$ *if the* i^{th} *node* is replaced at the j^{th} re-start

$= -t_{ij}^c + (t_{ij} + x)^c$ *if the* i^{th} *node* is not replaced at the j^{th} re-start.

Similarly, for $x_s', x_l', \dots, x_k'$. Note that for independence, only λ_i 's and γ_i 's are not equal to zero.

Expression (5.3) can be readily derived from Equation (3.1) by using the transformation $Y_i = X_i^c$, $c > 0$ for the nodes that have a Weibull time to failure (failed and were replaced to restart the system), $Y_i = -t_i^c + (t_i + X)^c$ for nodes that have an excess Weibull distribution (did not fail when the system was re-started) and Y = X for the *n* software components that have an exponential time to failure.

(2) *In case of correlated software failures,* and independent hardware failure, the reliability model can be expressed as:

$$R_j(x) = \exp\{-\sum_{i=1}^{k} \lambda_i x_i' - \sum_{v=1}^{n} \gamma_v x - \sum_{\substack{v,w=1 \\ v<w}}^{n} \gamma_{vw} x - \sum_{\substack{v,w,z=1 \\ v<w<z}}^{n} \gamma_{vwz} x - \dots - \gamma_{12\dots n} x\} \qquad (5.4)$$

(3) For a *system where software failures occur independently, but hardware failures are correlated,* the reliability model is given by:

$$R_j(x) = \exp\{-\sum_{i=1}^{k} \lambda_i x_i' - \sum_{\substack{i,s=1 \\ i<s}}^{k} \lambda_{is} \max\{x_i', x_s'\} - \sum_{\substack{i,s,l=1 \\ i<s<l}}^{k} \lambda_{isl} \max\{x_i', x_s', x_l'\} - \dots -$$

$$\lambda_{123\dots k} \max\{x_1', \dots, x_k'\} - \sum_{v=1}^{n} \gamma_v x\} \qquad (5.5)$$

(4) In the case of *both software and hardware correlated failures,* the reliability model of the Cloud system is given by

$$R_j(x) = \exp\{-\textstyle\sum_{i=1}^{k}\lambda_i x_i' - \sum_{\substack{i,s=1\\ i<s}}^{k}\lambda_{is}\max\{x_i', x_s'\} - \sum_{\substack{i,s,l=1\\ i<s<l}}^{k}\lambda_{isl}\max\{x_i', x_s', x_l'\} - \dots -$$

$$\lambda_{123\dots k}\max\{x_1', \dots, x_k'\} - \textstyle\sum_{v=1}^{n}\gamma_v x - \sum_{\substack{v,w=1\\ v<w}}^{n}\gamma_{vw}x - \sum_{\substack{v,w,z=1\\ v<w<z}}^{n}\gamma_{vwz}x - \dots - \gamma_{12\dots n}x\}$$

(5.6)

In all four scenarios, it is assumed that hardware and software fail independently from each other.

From Equation (5.3), the PDF of an application on Cloud can be derived as:

$$f(x) = [c\textstyle\sum_{i=1}^{k}\lambda_i x_i'' + \sum_{v=1}^{n}\gamma_v] * \exp\{-\sum_{i=1}^{k}\lambda_i x_i' - \sum_{v=1}^{n}\gamma_v x\}, \quad (5.7)$$

where $x_i'' = x^{c-1}$ *if the* i^{th} *node* is replaced at the j^{th} re-start

$= (t_{ij} + x)^{c-1}$ *if* the i^{th} *node* is not replaced at the j^{th} re-start.

From Equation (5.4), the PDF for this case is

$f(x) =$

$$[c\textstyle\sum_{i=1}^{k}\lambda_i x_i'' + \sum_{v=1}^{n}\gamma_v + \sum_{\substack{v,w=1\\ v<w}}^{n}\gamma_{vw} + \sum_{\substack{v,w,z=1\\ v<w<z}}^{n}\gamma_{vwz} + \dots + \gamma_{12\dots n}]*\exp\{-\sum_{i=1}^{k}\lambda_i x_i' -$$

$$\textstyle\sum_{v=1}^{n}\gamma_v x - \sum_{\substack{v,w=1\\ v<w}}^{n}\gamma_{vw}x - \sum_{\substack{v,w,z=1\\ v<w<z}}^{n}\gamma_{vwz}x - \dots - \gamma_{12\dots n}x\}. \quad (5.8)$$

The PDF that is derived from Equation (5.5) is given as:

$$f(x) = [c\textstyle\sum_{i=1}^{k}\lambda_i x_i'' + \sum_{\substack{i,s=1\\ i<s}}^{k}\lambda_{is}x_{is}'' + \sum_{i,s,l=1}^{k}\lambda_{isl}x_{isl}'' + \dots + \lambda_{123\dots k}x_{123\dots k}'' + \sum_{v=1}^{n}\gamma_v] *$$

$$\exp\{-\textstyle\sum_{i=1}^{k}\lambda_i * x_i' - \sum_{\substack{i,s=1\\ i<s}}^{k}\lambda_{is}\max\{x_i', x_s'\} - \sum_{i,s,l=1}^{k}\lambda_{isl}\max\{x_i', x_s', x_l'\} -$$

$$\dots - \lambda_{123\dots k}\max\{x_1', \dots, x_k'\} - \textstyle\sum_{v=1}^{n}\gamma_v x\}, \quad (5.9)$$

and $x_{is}'' = \frac{d\max\{x_i', x_s'\}}{dx}$, $x_{isl}'' = \frac{d\max\{x_i', x_s', x_l'\}}{dx}$, …, and $x_{123\dots k}'' = \frac{d\max\{x_1', \dots, x_k'\}}{dx}$.

The PDF from Equation (5.6) can be expressed as:

$$f(x) = [c \sum_{i=1}^{k} \lambda_i x_i'' + \sum_{\substack{i,s=1 \\ i<s}}^{k} \lambda_{is} x_{is}'' + \sum_{i,s,l=1}^{k} \lambda_{isl} x_{isl}'' + \cdots + \lambda_{123\ldots k} x_{123\ldots k}'' +$$

$$\sum_{v=1}^{n} \gamma_v + \sum_{\substack{v,w=1 \\ v<w}}^{n} \gamma_{vw} + \sum_{\substack{v,w,z=1 \\ v<w<z}}^{n} \gamma_{vwz} + \ldots + \gamma_{12\ldots n}] *$$

$$\exp\{-\sum_{i=1}^{k} \lambda_i * x_i' - \sum_{\substack{i,s=1 \\ i<s}}^{k} \lambda_{is} \max\{x_i', x_s'\} - \sum_{i,s,l=1}^{k} \lambda_{isl} \max\{x_i', x_s', x_l'\} - \ldots$$

$$-\lambda_{123\ldots k} \max\{x_1', \ldots, x_k'\} - \sum_{v=1}^{n} \gamma_v x - \sum_{\substack{v,w=1 \\ v<w}}^{n} \gamma_{vw} x - \sum_{\substack{v,w,z=1 \\ v<w<z}}^{n} \gamma_{vwz} x - \ldots -$$

$$\gamma_{12\ldots n} x\}. \tag{5.10}$$

After the reliability and PDF of an application are determined, the information is used to compute failure rate and mean time to failure of an application on Cloud.

5.4 Failure Rate and Mean Time To Failure

The failure rate "$\lambda_j(x)$" of independent failure of software and hardware after the j^{th} re-start of the system is given by:

$$\lambda_j = \frac{f(x)}{R_j(x)} = c \sum_{i=1}^{k} \lambda_i x_i'' + \sum_{v=1}^{n} \gamma_v. \tag{5.11}$$

For the case of dependent software failure and independent hardware failure, the failure rate is

$$\lambda_j = \frac{f(x)}{R_j(x)} = c \sum_{i=1}^{k} \lambda_i x_i'' + \sum_{v=1}^{n} \gamma_v + \sum_{\substack{v,w=1 \\ v<w}}^{n} \gamma_{vw} + \sum_{\substack{v,w,z=1 \\ v<w<z}}^{n} \gamma_{vwz} + \ldots + \gamma_{12\ldots n}. \tag{5.12}$$

The failure rate for the case of independent software failure and dependent hardware failure is

$$\lambda_j = \frac{f(x)}{R_j(x)} = c \sum_{i=1}^{k} \lambda_i x_i'' + \sum_{\substack{i,s=1 \\ i<s}}^{k} \lambda_{is} x_{is}'' + \sum_{i,s,l=1}^{k} \lambda_{isl} x_{isl}'' + \cdots + \lambda_{123\ldots k} x_{123\ldots k}'' +$$

$$\sum_{v=1}^{n} \gamma_v. \tag{5.13}$$

In the case of dependent software and hardware failures, the failure rate is given as:

$$\lambda_j = \frac{f(x)}{R_j(x)} = c\sum_{i=1}^{k} \lambda_i x_i^{''} + \sum_{\substack{i,s=1 \\ i<s}}^{k} \lambda_{is} x_{is}^{''} + \sum_{i,s,l=1}^{k} \lambda_{isl} x_{isl}^{''} + \cdots + \lambda_{123\ldots k} x_{123\ldots k}^{''} +$$

$$\sum_{v=1}^{n} \gamma_v + \sum_{\substack{v,w=1 \\ v<w}}^{n} \gamma_{vw} + \sum_{\substack{v,w,z=1 \\ v<w<z}}^{n} \gamma_{vwz} + \ldots + \gamma_{12\ldots n}. \tag{5.14}$$

The mean time to failure (MTTF) of an application on Cloud is given by:

$$E(x) = \int_0^{\infty} x * f(x)dx\,, \tag{5.15}$$

where f(x) can be the PDF of the any case presented in Equations (5.7), (5.8), (5.9), and (5.10) for a Cloud system.

5.5 Virtual Machine and Hypervisor Failure

There is some evidence indicating that Virtual machines, and hypervisors may exhibit an increase in the failure rate with time, referred to as software aging [81],[82]. The current model can be readily modified to accommodate such a scenario. In this situation, the aging software will be treated as a node, in which case the time to failure distribution for the software will be Weibull or excess Weibull instead of the exponential. More precisely, the TTF of a virtual machine or hypervisor follows a Weibull distribution when it failed and is restarted. On the other hand, it will have an excess Weibull if a virtual machine or hypervisor did not fail when the system failed. It is assumed that the failures of virtual machines and hypervisors are independent. In this case, the reliability model of *n* VMs and *k* hypervisors can be expressed as follows:

$$R_j^v(x) = P(X_1 > x, X_2 > x, \ldots, X_k > x, X_{k+1} > x, X_{k+2} > x, \ldots, X_{k+s} > x)$$

$$= \exp\{-\sum_{i=1}^{k} \lambda_i x_i^{'} - \sum_{v=1}^{s} \lambda_v x_v^{'}\}, \tag{5.16}$$

where $x_i' and\ x_v' = x^c\ if\ the\ i^{th}\ virtual\ machine\ \ or\ \ v^{th}$ hypervisor is re-started (rejuvenated) at the j^{th} re-start

$= -t_{ij}^c + (t_{ij} + x)^c\ \ if\ the\ i^{th}\ \ virtual\ machine\ or\ v^{th}$ hypervisor is not re-junivated at the j^{th} re-start.

One can obtain the reliability model of a Cloud computing system, including virtual machines or hypervisors, from Equation (5.17).

$$R_j^{sys}(x) = R_j(x) * R_j^v(x), \tag{5.17}$$

where $R_j(x)$ can be obtained from Equations (5.3), (5.4), (5.5), or (5.6). The PDF of system reliability, $f^{sys}(x)$, can be readily obtained from Equation (5.17) by differentiating -$R_j^{sys}(x)$ with regard to x.

The system failure rate is given by:

$$\lambda_j^{sys} = \frac{f^{sys}(x)}{R_j^{sys}(x)}. \tag{5.18}$$

The MTTF can be express as:

$$E^{sys}(x) = \int_0^{\infty} x * f^{sys}(x)dx. \tag{5.19}$$

5.6 Numerical Result Examples for The Proposed Cloud Reliability Model

This section demonstrates by examples, for $k = 3$ and n=12, the effect of survival time (T) and parameter values for joint node failures ($\lambda_{ij...}$) on system reliability for the four cases, Equations (5.3), (5.4), (5.5) and (5.6), in Figure 5.2 and on the failure rate (Equations (5.11), (5.12), (5.13) and (5.14)) and MTTF, Equation (5.15) in Figure 5.3. The case of an all-to-all communication of MPI applications is considered where every process sends information to every other process. If the data from a particular process could not reach other processes, every other process has to wait, and thus all fail together. In this case, the joint parameters reduce to only one parameter ($\gamma_{12\ldots n}$). An example is used where job run length (x) equals 100 hours and the node parameters $\lambda_1, \lambda_2, \lambda_3, =$ 0.00005, *the 2-node joint parameters* $\lambda_{12}, \lambda_{13}, \lambda_{23} = 0.00005$, *the 3-node joint parameter* $\lambda_{123} = 0.00003$, *C*=1.5, the 2 software-component joint parameters $\gamma_1, \ldots, \gamma_{12} = 0.00003, \mathit{the}\ 3\ \mathit{software\ component\ joint\ parameter}\ \gamma_{123} = 0.0007$, and the hypervisor and VM parameters λ=0.00001. It is obvious from the reliability Equations (5.3), (5.4), (5.5) and (5.6) that when the system has failures that are correlated it becomes less reliable. Moreover, when the joint failure rates ($\lambda_{ij\ldots}$) increase, the reliability of the system decreases. This implies that a system with independent nodes ($\lambda_{ij\ldots}, \gamma_{ij\ldots}$= 0) is more reliable than one where the nodes are correlated in failure. Furthermore, reliability of the system decreases as the survival time T increases, as demonstrated in Figure 5.2.

Figure 5.3 shows that the failure rate increases as the joint parameters ($\lambda_{ij\ldots}, \gamma_{ij\ldots}$) increase in value or the survival time increases. Figure 5.4 shows, as expected, that the MTTF decreases when the parameters for joint failure ($\lambda_{ij\ldots}, \gamma_{ij\ldots}$) increase or as the survival time T increases.

Figure 5.5 shows the Cloud system reliability as function of nodes. As the number of nodes increases, the reliability decreases. Figure 5.6 shows the Cloud system failure rate when the number of nodes increases. The failure rate increases when the system has more nodes. Figure 5.7 shows the Cloud system MTTF. When the number of nodes increases, Mean time to failure decreases. Also, when adding more components and the failures of components have correlation, the system becomes less reliable. That means less reliability, more failure rate, and shorter MTTF.

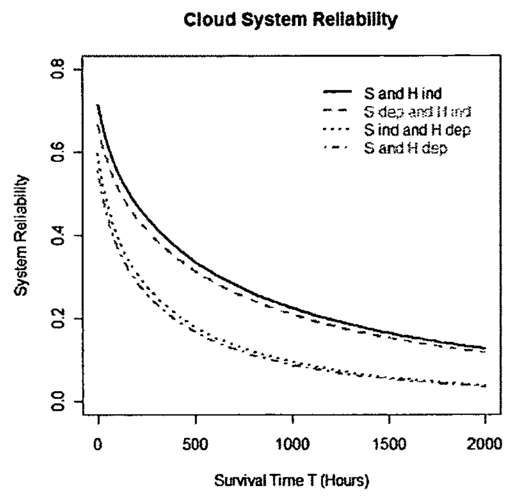

Figure 5.2 Cloud System Reliability, with k=3 and n=12, for the four scenarios in Equations. (5.3) – (5.6).

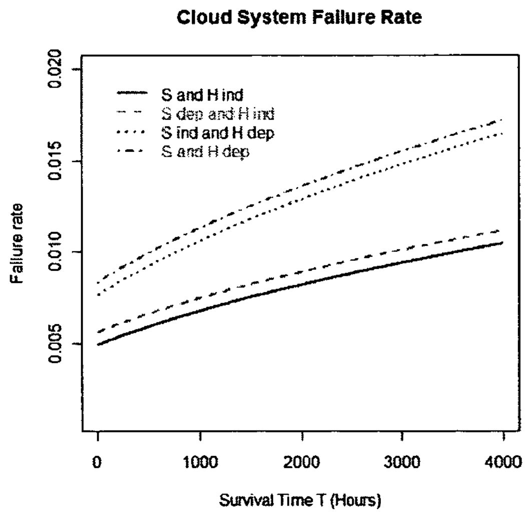

Figure 5.3 Cloud System Failure Rate, with k=3 and n=12, for the four scenarios in Equations. (5.11) - (5.14).

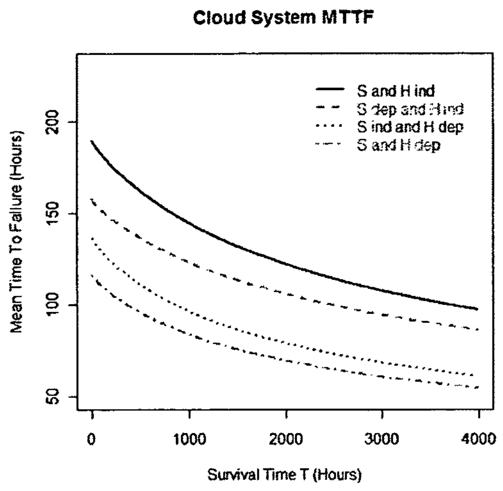

Figure 5.4 Cloud System MTTF, with k=3 and n=12, for the four scenarios.

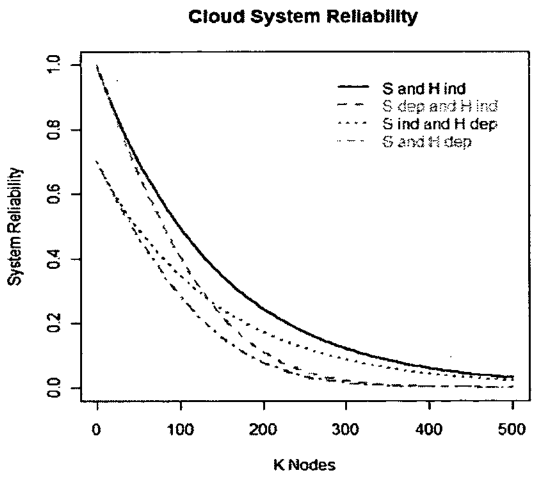

Figure 5.5 Cloud System Reliability as function of the number of nodes (k).

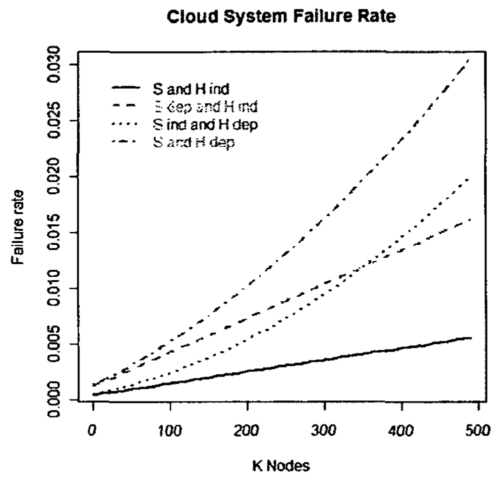

Figure 5.6 Cloud System Failure Rate as function of the number of nodes (*k*).

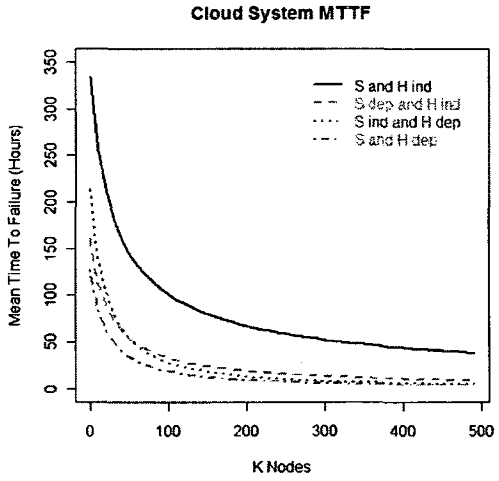

Figure 5.7 Cloud System MTTF as function of the number of nodes (*k*).

5.7 The Expected Completion Time with Checkpoint/Restart

There are studies on the implementation of checkpoint/restart [83], [84] for MPI applications in a Cloud environment. This section proposes a checkpoint/restart model and utilize the reliability information from the four scenarios described above in the checkpoint model. Figure 5.8 illustrates the checkpoint scheme used. Here, T_{re} is re-computing time, T_{rt} is recovery time, *C* is checkpoint cost, *CI* is checkpoint interval, and TTF is the time to failure of the system. The following equation gives the expected completion time.

$$E[T_{ck}] \approx T_{free} + C^{*}N + W, \tag{5.20}$$

where T_{ck} is the completion time, $W \approx \left\lfloor\frac{T_{free}}{MTTF}\right\rfloor * (E[T_{re}] + T_{rt})$, T_{free} is the fault-free execution time, *MTTF* is the mean time to failure, and *N* is the number of checkpoints.

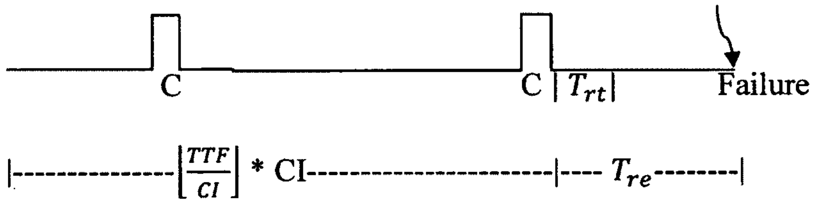

Figure 5.8 Parameters of a checkpoint model.

The parameters that are defined are based on the work in [24]. The expected job completion time, $E[T_{ck}]$, can be obtained from fault-free execution time and Wasted time, *CN+W*. From Figure 5.6, it is seen that the re-computing time can be expressed as:

$$T_{re} = \text{TTF} - \left\lfloor\frac{TTF}{CI}\right\rfloor * \text{CI}. \tag{5.21}$$

By taking expectation, users can obtain

$$E[T_{re}] = E[\text{TTF}] - E[\left\lfloor\frac{TTF}{CI}\right\rfloor]^{*}\text{ CI}, \tag{5.22}$$

where *TTF* is time to failure, *CI* is checkpoint interval, and

$$E[\left\lfloor\frac{\text{TTF}}{\text{CI}}\right\rfloor] = \int_0^{\infty}\left\lfloor\frac{\text{t}}{\text{CI}}\right\rfloor^{*}f(t)\,dt. \tag{5.23}$$

To find the waste time, the PDF and MTTF are used for the four scenarios described in Section 5.3. An example of an application that runs on 12 VMs and 3 physical machines is considered and assumes that hardware and software are independent of each other. The parameters that are used for PDF and MTTF are the same as in Section 5.6 for the case $k = 3$, and $n = 12$. Also, T_{rt}, C and CI were assumed to be 40 seconds, 20 seconds, and 5 hours, respectively.

Figure 5.6 shows the waste time ($CN + W$) as a function of the Fault-free job execution time. The different waste time of each cases comparing with the both software and hardware independent failures is about 15% for correlated software failures and independent hardware failures, 36% for independent software failures and correlated hardware failures, 47% for both hardware and software correlated failures. As can be seen, the smallest waste time is in the case when both the system software and hardware components are independent. This is due to the fact that the system is more reliable when the system components fail independently. The largest waste time is for the case when one has correlations among software as well as among hardware.

5.8 Conclusion

Reliability information is important for improving Cloud computing system utilization. In order to mitigate the interruption of an application because of a failure, it is of importance to know how the reliability, failure rate, and mean time to failure of a Cloud computing system are affected by joint node failures. Many studies have shown the existence of failure correlations among nodes in computer systems [39] and [47]. In this chapter, reliability models are proposed that consider software and hardware components, which account for joint failures among nodes as well as VMs, and developed the reliability, failure rate, and mean time to failure of a system based on *k* nodes and *s* VMs. Four major cases are considered that are combination of software and hardware failure correlation. Also, discussed is the reliability model for the case where VMs and Hypervisors exhibit an aging effect. The waste time derived for each of the four reliability scenarios is presented in Figure 5.9. Results show that if failures in the system possess a degree of dependency, the system becomes less reliable.

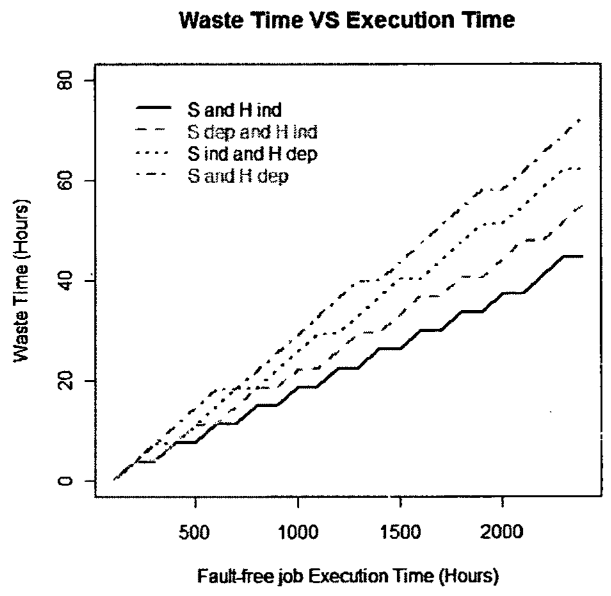

Figure 5.9 Waste Time versus Fault-Free Job Execution Time in Cloud System.

CHAPTER 6

AN ECONOMIC MODEL FOR CLOUD SERVICE PROVIDER

For Cloud service providers, profit maximization and customer satisfaction are the concerns. How to price and how large a data center is needed are important factors to their profit. This chapter presents an economic model for a Cloud service provider to maximize their profit and provide the information for their decision making for price of a VM hour and how large the system should be. This model also accounts for several different qualities of services.

6.1 Introduction

Cloud service providers offer resources and services over the Internet to users who pay for an amount of usage as if it were a traditional public utility, such as electric power and water. By utilizing the services of Cloud service providers, individual users or enterprises do not need to invest into large in-house computing resources. Due to the virtualization technology, enterprises can outsource excessive jobs by running on Virtual Machines (VM) in third-party computing resources. This allows them to scale up or down the amount of resources as needed and pay for only the time they use.

As demand for Cloud computing is increasing dramatically, new opportunities are created for running a Cloud business. To maximize profits, one has to be able to estimate the demand of the market in order to gain revenue and minimize cost. Based on supply and demand [85], if the company sets the price lower than market price, it can, in principle, increase its sales. On the other hand, if the price is set too high, sales may diminish. Hence, a good pricing strategy is important for increasing revenue. A successful pricing strategy can lead to a significant increase in profit [86]. If one can understand the demand of Cloud computing, one can define the right price to gain market share and increase revenue.

Another factor that impacts the profit of the Cloud service provider is cost. To understand the cost of a Cloud service provider, total cost of ownership (TCO) [62] is used in the analysis. The total cost of ownership analysis considers direct and indirect cost of owning a business over its life span. The direct cost consists of equipment and infrastructure cost, such as server, network, facility, etc. The indirect cost includes power, cooling, license, etc). When one understands the TCO, one can decide on the right size of a Cloud system based on demand in order to maximize profit.

This chapter proposes an economic model for a Cloud service provider, based on revenue and cost, by considering computing resources charged to customers over time. A logit model is considered for revenue analysis, and the total cost of ownership is used to analyze the cost of a Cloud system. Also considered is a Service Level Agreement (SLA), which guarantees a certain availability level and applies a probabilistic model to calculate the expected penalty cost when a Cloud service provider cannot meet the guaranteed level.

6.2 Economic Cloud Model

6.2.1 Revenue

Price influences the behavior of customers. If Cloud providers can set the right price, they can get higher revenue, and hence more profit. A typical pricing in Cloud services is directly related to the VM hours that Cloud system provides. If they can define the demand function of their company or market-share function, they can define the right price in order to have higher revenue. The revenue over a certain time period can be computed as:

$$\text{Revenue} = \text{Price} * \text{Quantity} \quad , \tag{6.1}$$

where in Equation (6.1) *Quantity* is the number of VM hours that run on the Cloud system and *Price* is that per VM hour. It is seen from Equation (6.1) that that the total revenue is determined by the demand function. A Cloud provider can choose either price or quantity. From the demand function shown in Figure 6.1, if the Cloud provider sets a price, he or she will know the number of VM hours. On the other hand, if a provider wants to sell a certain quantity (the number of VM hours), the Cloud provider has to set price according to the demand function. In this work, three services with different availability are considered. Three demand functions are utilized based on each service with a different price.

A logit model is commonly used for representing a demand function [87] or customer behavior [88], as shown in Figure 6.1. The logit model is the model of choice used to model customer behavior in the market. When a company sets its price very high, it can sell only to its loyal customers. Hence, setting the price at a high level will not change the quantity that they can sell by a significant amount. On the other hand, when one sets the price very low, one will gain considerably from demand. Changing the price at a low level will not affect the change in demand. However, if the company sets the price around the market price or the standard price, the demand will change dramatically. The logit model is given as:

$$D(p) = C * \exp\{-(a + bp)\}/(1 + \exp\{-(a + bp)\}), \tag{6.2}$$

where p is price, a, b, C are parameters with $C > 0$ $b > 0$, a can be either less than or more than 0, C refers to the maximum quantity of the market-share of that firm or the market size. The market price is approximately equal to $-a/b$. Therefore, the total revenue can be calculated by the formula

$$\text{Total Revenue} = p*D(p)\,, \tag{6.3}$$

where *D(p)* is the number of VM hours.

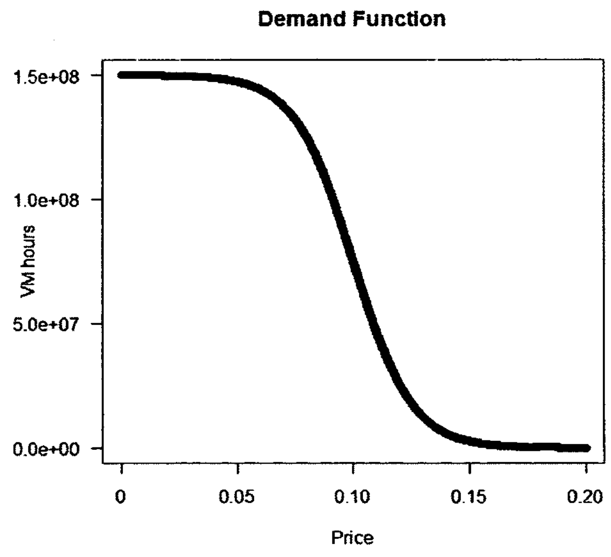

Figure 6.1 Logit demand function

6.2.2 Cloud Cost Analysis

Cloud TCO is defined in [62] as the cost spent for owning the Cloud system over its life span. It includes the cost of acquiring the Cloud system and operating it. The cost will be classified into eight categories according to [62]: server, network, software, power, cooling, facilities, real estate, and support and maintenance. In the model that is presented here, quantity is considered to be the number of servers. A server can run a certain amount of Virtual Machines (VM), called VM density [62]. This study is different from other studies in the literature [62] in that it considers costs due to system failure and repair, to maintain a certain level of availability, and to penalty cost for failing to meet the SLA. In the following, all the costs considered in the model are described.

Server Cost

The Cloud has usually homogenous servers tied into racks. The total cost of servers includes the original service price plus the upgrade cost to avoid performance degradation in a competitive market. The server cost per rack is considered. Server cost can be computed as:

$$\text{Scost} = \text{Nserv}*\text{ServC} + \text{Upgradecost}*\text{years}\,, \tag{6.4}$$

where *Nserv* is the number of servers, *ServC* is a unit server cost, *Upgradecost* is upgrade cost per year. In the server cost, the upgrade cost (*Upgradecost*) was considered to be of importance. The upgrade cost is defined as follows:

$$\text{Upgradecost} = \text{Nserv}\ *\text{ServC}*\text{percent}\,, \tag{6.5}$$

where *percent* is 0-100% of server cost.

Network Cost

The network cost is mainly the switch cost. The number of switches is based on the number of ports that are needed from the servers. A server can have more than one network interface card (NIC). An NIC may have more than one port. Therefore, network cost is defined as:

$$\mathrm{Ncost} = \mathrm{SwC} * \mathrm{Nswitch}\,, \tag{6.6}$$

where *SwC* is unit switch cost and *Nswitch* is the number of switches calculated as:

$$\mathrm{Nswitch} = \mathrm{Nport} * \mathrm{Nnic} * \mathrm{Nserv} / \mathrm{NSWport}\,, \tag{6.7}$$

where *Nport* is the number of ports per NIC, *Nnic* is the number of NIC per server, *NSWport* is the number of ports in a switch.

Software Cost

The software cost is mainly the cost for the license. There are Operating systems costs for VMs and Software costs for managing VMs. The operating system cost is applied per VM. Managing software is licensed by the number of processors in the system. Therefore, the software cost is

$$\mathrm{SoC} = \mathrm{Nvm} * \mathrm{Oprice} + \mathrm{Npps} * \mathrm{SUprice} * \mathrm{Nserv}, \tag{6.8}$$

where *Nvm* is the number of VMs, *Oprice* is the license price per VM. *Npps* is the number of processors per server, *SUprice* is software unit price per processor used for management of VMs.

Power Cost

The power cost is considered per server. In the current model, the power cost is calculated by:

$$\mathrm{PowC} = \mathrm{Nserv} * \mathrm{Powpr} * \mathrm{v}\,, \tag{6.9}$$

where *Powpr* is the power consumption required per server and *v* is the electric utility cost (kWh).

Cooling Cost

The cooling cost is the power consumption used for cooling the system. In this model, cooling cost is calculated by:

$$\text{CoolC} = \text{Nserv} * \text{Powpr} * \text{Powpw} * \text{v}, \tag{6.10}$$

where *Powpw* is the power consumption of cooling 1W of heat from the equipment.

Facilities Cost

Facilities cost is the cost paid for equipment such as KVM switch and cables. They are attached to racks. The facilities cost is calculated by:

$$\text{FacC} = \text{Cfac} * \text{Nrack}, \tag{6.11}$$

where *Cfac* is cost of facilities per rack and *Nrack* is the number of racks in the system.

Real Estate Cost

Real estate cost is the cost spent for leasing a room required for hosting the servers. Real Estate cost (*REC*) is computed by:

$$\text{REC} = \text{Csq} * \text{Sqr}. \tag{6.12}$$

Here, *Csq* is cost per square foot per year and *Sqr* is the number of square feet needed by the Cloud system. *Sqr* is calculated as:

$$\text{Sqr} = \text{Nrack} * \text{SqRack}, \tag{6.13}$$

where *SqRack* is the space in square feet needed by a rack.

Support and Maintenance Cost

This cost comes from the salary of the IT staff, performance maintenance, system configuration, and training. Support and maintenance cost is given by:

$$\text{SupC} = \text{Salary} + \text{training} + \text{CSLA} + \text{RC} + \text{AVC}, \tag{6.14}$$

where *Salary* is the wage of all IT staff members.

$$\text{Salary} = \text{wage}*\text{Nrack}/\text{rpIT}, \tag{6.15}$$

where *wage* is the salary per IT staff and *rpIT* is the number of racks that an IT staff handles.

Training is the cost spent for training the IT staff, *CSLA* is the cost spent for penalty due to Service Level Agreement (SLA), repair cost (*RC*) is cost due to failures in the Cloud and *AVC* is the cost of preparing the system for a certain level of availability.

To approximate the repairing cost, it is assumed that the server time to failure (x) follows a gamma distribution ($\Gamma(k_i, s_i)$) and failures are independent [44]. The gamma distribution fits well the TTF of HPC systems, [47], [46] and has a useful property in that the sum of independent gamma random variables is also a gamma random variable. Hence, the repair cost (*RC*) can be calculated by:

$$\text{RC} = \sum_{i=1}^{\text{Nserv}} NF_i * \text{AC}\ , \tag{6.16}$$

where *AC* is average repair cost per failure and *NF* is the expected number of failures per server. The expected number of failures can be obtained as $\text{E}[\frac{T}{x}]$, where T is the lifespan of the Cloud and x is the time to failure of a server. $\text{E}[\frac{T}{x}]$ can be approximated using the Taylor series expansion at $a=\mu$.

$$E[\frac{T}{x}] = E[\frac{T}{\mu} - \frac{T}{\mu^2}(x-\mu) + \frac{T}{\mu^3}(x-\mu)^2] = \frac{T}{\mu} + \frac{\text{T}*\text{Var(x)}}{\mu^3}. \tag{6.17}$$

Here, μ is mean time till failure and can be obtained from the gamma distribution.

Cloud customers may require different system availabilities. For example, mission critical applications may need 99.999% availability, while other not so critical applications may require a lower availability level. Cloud service providers have to engineer the process of repair time in order to meet a given level of availability. The costs that are usually encountered are the cost for reducing time-to-repair, the cost for redundant equipment such as servers and switches, and the cost for hiring and training. Scott [89] has shown that a higher cost is associated with achieving higher levels of availability. He also showed that cost rises exponentially with an increase in the availability level. This work follows Scott [89] in assuming an exponential relationship between availability and cost, as shown in Figure 6.2.

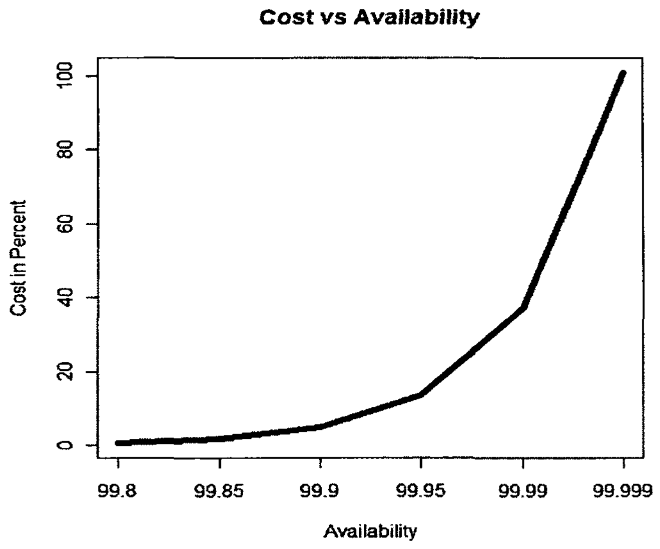

Figure 6.2 Exponential cost rise with an increase in availability level

The availability cost for each availability level is calculated as follows:

$$\text{AVC} = \exp\{\text{availability level}\} * \text{HAC} \ , \qquad (6.18)$$

where availability level is the level of availability such as 99.999, 99.99 or 99.95, as shown in Figure 6.2, *HAC* is the highest availability cost for engineering the Cloud system to have a 99.999% availability. To achieve 99.999% availability, one has to have redundant servers and sufficient well-trained IT staffs. Therefore, *HAC* is a cost to the system, which considers redundant server cost (*RScost*), availability software cost

(*ASoC*), additional switch cost (*NCost*) for redundant server, additional IT Staff (*salary*) cost, training cost, additional facility cost, and additional space cost (*AREC*) (for redundant servers). *HAC* is computed by:

$$\text{HAC} = \text{Nserv} * \text{RScost} + \text{ASoC} + \text{NCost} + \text{salary} + \text{training} + \text{AREC} + \text{FacC} + \text{SpaceC} + \text{PowC} + \text{CoolC}, \quad (6.19)$$

Availability software is the module that manages server redundancy for maintaining a certain level of availability in a Cloud system. Availability software cost (*ASoC*) is calculated by:

$$\text{ASoC} = \text{Npps} * \text{ASUprice} * \text{Nserv}, \quad (6.20)$$

where *Npps* is the number of processors per server, and *ASUprice* is availability Software price per processor.

For penalty cost, SLA is used for guaranteeing the satisfaction of a Cloud customer. Quality (throughput and response time) [90] and reliability [29], [31] of Cloud services are the main concern. Cloud service providers such as Amazon EC2 and Rackspace guarantee certain availability for customers. If availability is less than the point they claim, customers receive service credit back. Therefore, this analysis approximates the cost coming from SLA due to refunds to customers as a result of failure to meet the guaranteed level of availability.

Availability is uptime divided by the total operation time. Given A% availability of a VM, one can calculate downtime (t) per year. If downtime exceeds t hours per year, then the Cloud provider would encounter cost due to customer refunds. Therefore, one would like to know the probability (P_D) that the total downtime in a year is more than t

hours (that is the probability of encountering costs due to refunds). One can calculate this probability for a VM as follows:

$$P_D = \sum_{r=1}^{\infty}\{1 - \mathrm{P}[T_r \leq \mathrm{t}]\} * \mathrm{P}[\mathrm{N(y)} = \mathrm{r}] \quad , \qquad (6.21)$$

where r is the number of failures, T_r is the sum of repair times (t_1 +...+ t_r), and $N(y)$ is the number of failures in a year interval. Assume that repair time follows a gamma distribution Γ(α, β) [91]. Hence, T_r is the sum of r independent gamma random variables. Therefore, T_r has a gamma distribution, Γ(r*α, β). Hence, one can compute $\mathrm{P}[T_r \leq \mathrm{t}]$ from Γ(r*α, β). It is assumed that time to failure (TTF) follows a gamma distribution [47]. Hence,

$$\mathrm{P}[\mathrm{N(y)} = \mathrm{r}] = e^{-t/s} \sum_{i=ra}^{(r+1)(a-1)} \frac{t^i}{s^i i!} . \qquad (6.22)$$

In practice, the gammas parameters α, β can be estimated from data on repair time. Also, the parameters a and s can be estimated from data on time to failure of a server.

The expected penalty cost due to customer refunds (CSLA in Equation (6.14) is calculated by:

$$\mathrm{CSLA} = \sum_{i=1}^{\mathrm{Nserv*Nvm}} P_{D_i} * \mathrm{SC}, \qquad (6.23)$$

where *SC* is the service credit that is refunded to customers when availability of the VM is less than A%, and *Nvm* is the number of VMs that run on a physical machine.

Total cost is the summation of eight costs over the lifetime of the Cloud. This can be expressed as:

Total cost = Scost + Ncost + SoC + PowC + CoolC + FacC + REC + SupC (6.24)

The cost of a VM hour can be calculated by:

R = Cost/Total capacity, (6.25)

where *Total capacity* is calculated by:

$$\text{Total capacity} = D(p) * \text{lifespan}, \qquad (6.26)$$

where D(p) is the number of VM hours that the system run in a year.

6.3 Analysis and Result

This section shows with an example how to maximize the profit, based on demand and cost analysis, by choosing the right price and right size of a Cloud. After revenue and cost are analyzed, profit can be calculated from the following formula

$$\text{Profit} = \text{Revenue} - \text{Total Cost}, \qquad (6.27)$$

where *Revenue* is from Equation (6.3) and cost from Equation (6.24). Revenue as well as cost were calculated for each of the availability levels in Figure 6.3. In this Figure, the number of servers was determined by the level of demand in VM hours, (*D(p)*) for a given price, *p*. The number of servers was then used to determine the total cost from Equation (6.24).

To maximize profit, one can readily determine the number of servers that maximizes the difference between total revenue [*p(D(p)*] and total cost over the lifespan of a certain number of years.

Revenue and cost are considered over five years. Also considered are three types of services that have different Quality of Service (QoS), which guarantee 99.999%, 99.99% and 99.95% availability. It is assumed that the maximum market share (C) of the Cloud provider is 1.0×10^8, 1.25×10^8, and 1.5×10^8 VM hours per year and the market price is \$0.20, \$0.15, and \$0.10 per VM hour [63] for each of the availability levels, respectively. The server capacity is considered as the quantity in Eq (6.1). For the Cloud cost analysis, a 42 1U compute blade racks is assumed. A dual-processor compute blade server costs \$2,000 [63]. The upgrade cost is equivalent to the original server cost [63].

Also, the redundant server cost was 1.5 times the original server cost and the number of VM per server was 12. It is estimated that a server requires 0.6 kW to operate, and 0.6 W for cooling. It is assumed that it requires 1W to cool 1W of heat from the equipment (*Powpw*). A rack needs around 20 square feet (*SqRack*), which costs approximately $400 per square foot (*Csq*) [62]. A server has 2 NICs and each NIC has 2 ports. A 48-port Gigabit Router costs around $3,000. For the software cost, it is assumed that the Cloud uses Linux-based operating systems in VMs. The managing software cost and availability software per processor is $100. Four racks per IT staff are assumed. The salary for an IT staff is $5,000 a month. Training an IT staff per year costs $250. Repairing cost is calculated by Cost per time ($200) [92], [48] multiplied by the number of failures. Mean time-to-failure (MTTF) is 2184 hours with variance equals to 1657 [46]. No downtime is assumed for 99.999% availability service because of redundancy. Mean-time-to-repair is 7.5 and 30 minutes for 99.99% and 99.95%, respectively.

Results are presented in Figure 6.3 for each service level agreement. Figure 6.3 shows revenue and cost for each availability level. Revenue is based on price and the quantity (number of VMs) from the logit demand function in Equation (6.2). When quantity is high, the price is low. Therefore, the revenue is less at high quantity. Cost is a linear function because it is based only on the number of servers. When the number of servers increases, the cost increases. To find the number of servers that maximizes the profit, one chooses the number that maximizes the difference between revenue and cost, Figure 6.3. For 99.999% availability service, the maximum profit is attained for 798 servers with a price of $0.182 per hour and a cost of $0.11 per hour. For 99.99% availability service, 840 servers give the maximum profit with a price of $0. 143 per hour and a cost \$0.0773 per hour. For 99.95% availability service, the profit is maximized with 1004 servers and a price of \$0. 8919 per hour with a cost of \$0.052 per hour.

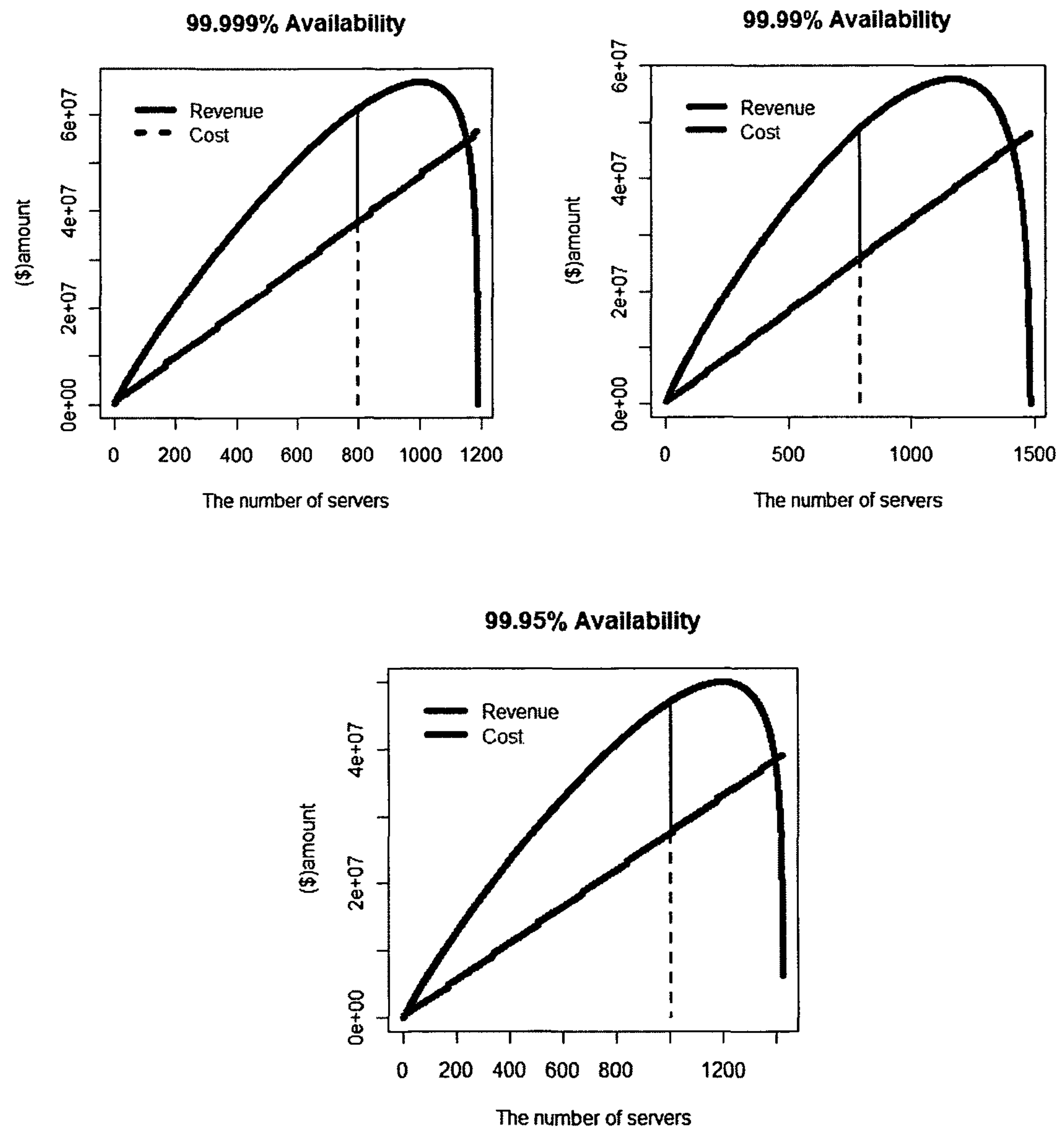

Figure 6.3. Total revenue [*pD(p)*, Equation (3)] versus total cost (Equation (24)) with 99.999, 99.99, and 99.95% availability level.

Figures 6.4 (a) and (b). present price per VM hour and number of servers that maximize profit for a given server cost and 99.95% availability. Figure 6.4 (c) shows the maximum profit (total revenue – total cost) for a given server cost for 99.95%

availability. Figures 6.5 (a) and (b) present price per VM hour and number of servers that maximize profit for a given server cost and 99.99% availability. Figure 6.5 (c) shows the maximum profit (total revenue – total cost) for a given server cost for 99.99% availability. Figures 6.6 (a) and (b) present price per VM hour and number of servers that maximize profit for a given server cost and 99.999% availability. Figure 6.6 (c) shows the maximum profit (total revenue – total cost) for a given server cost for 99.999% availability. As seen from these graphs, as cost per server increases, the number of servers that maximizes profit decreases, price per VM hour that maximize profit increases, and maximum profit decreases. This analysis helps a Cloud service provider determine price per VM hour and number of servers that would maximize profit, given a certain cost per server. For instance, for 99.95% availability shown in Figure 6.4 (a), (b), and (c), if server cost increases from 1,000 to 2,000 where the slope of price per VM hour and the number of servers are flat, the Cloud service provider should keep the same price per VM hour as well as the same number of servers in order to maximize profit. Notice, however, that maximum profit for $2000 cost is less than that for $1000 cost.

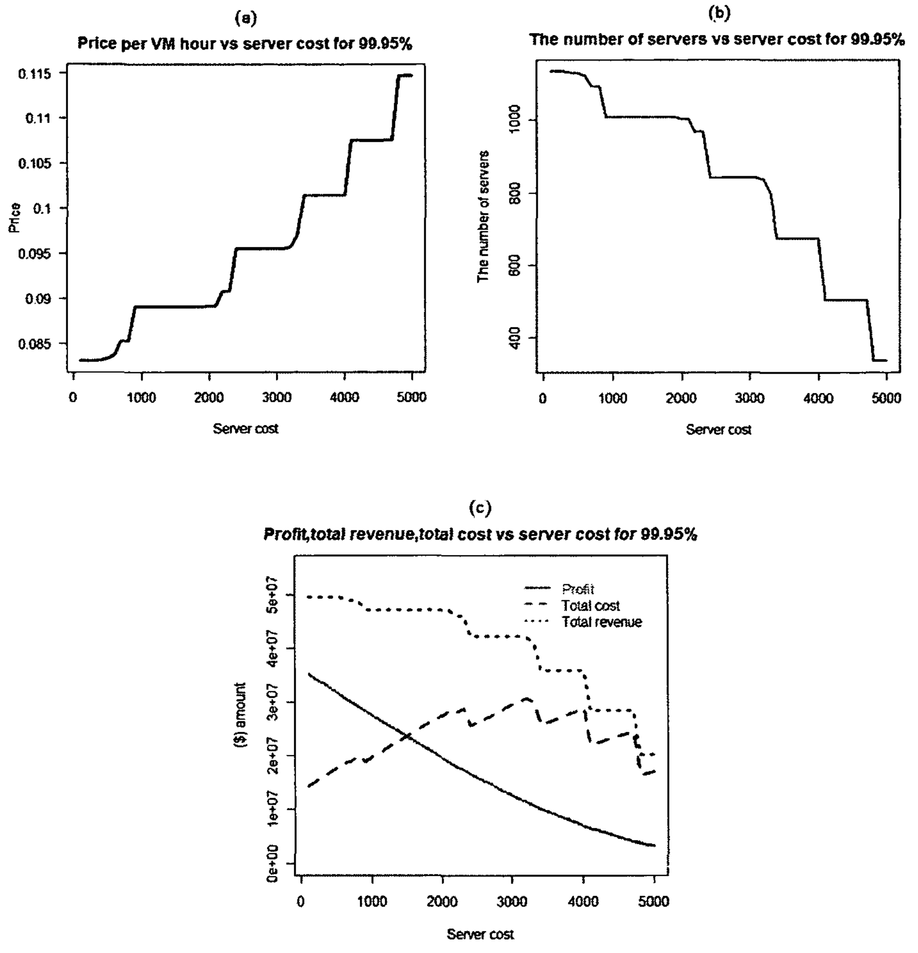

Figure 6.4 a, b, and c. (a) Price per VM hour that maximizes profit for a given server cost for 99.95% availability. (b) the number of servers that maximizes profit for a given server cost for 99.95% availability. (c) Maximum profit (revenue - total cost) for a given server cost for 99.95% availability.

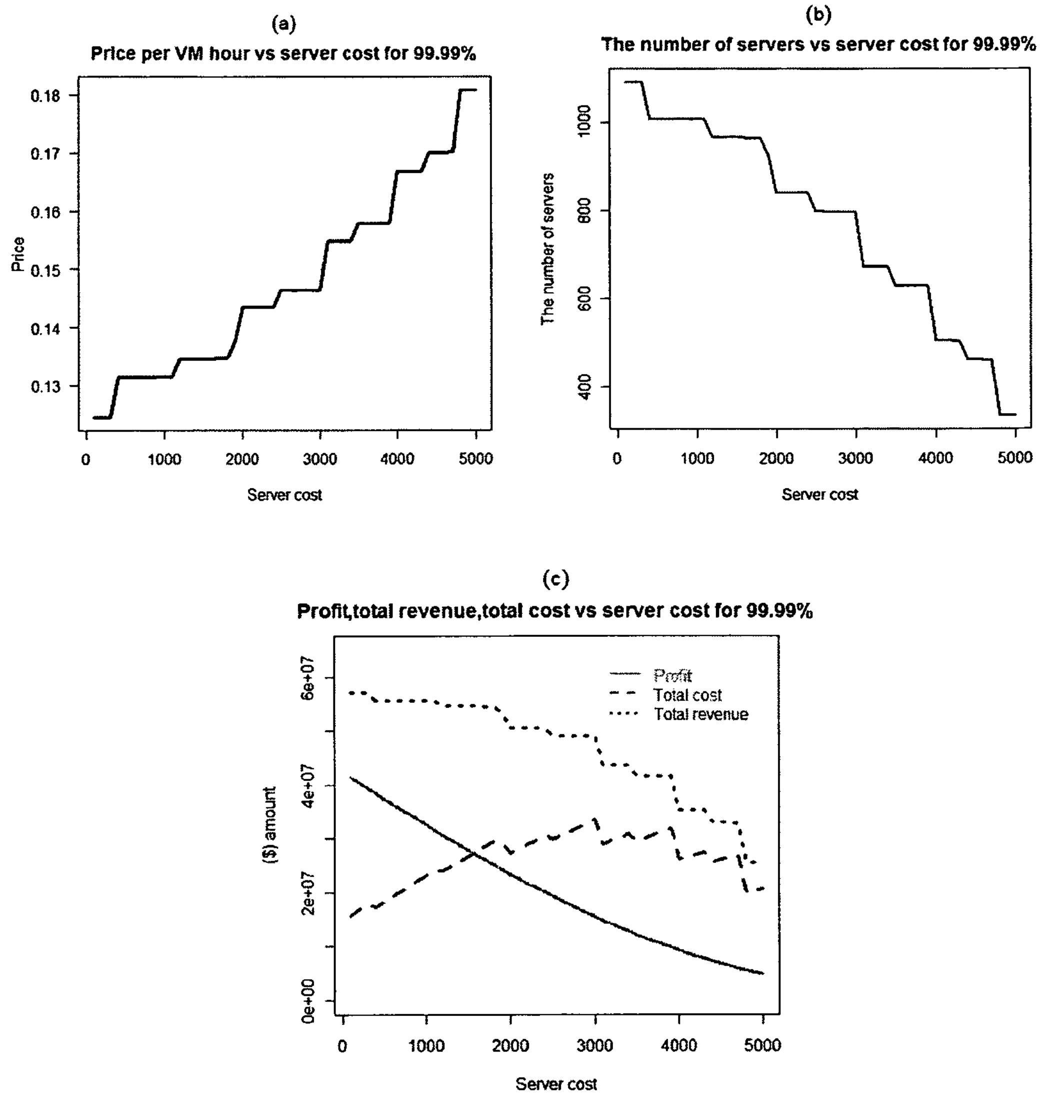

Figure 6.5 a, b, and c. (a) Price per VM hour that maximizes profit for a given server cost for 99.99% availability. (b) the number of servers that maximizes profit for a given server cost for 99.99% availability. (c) Maximum profit (revenue - total cost) for a given server cost for 99.99% availability.

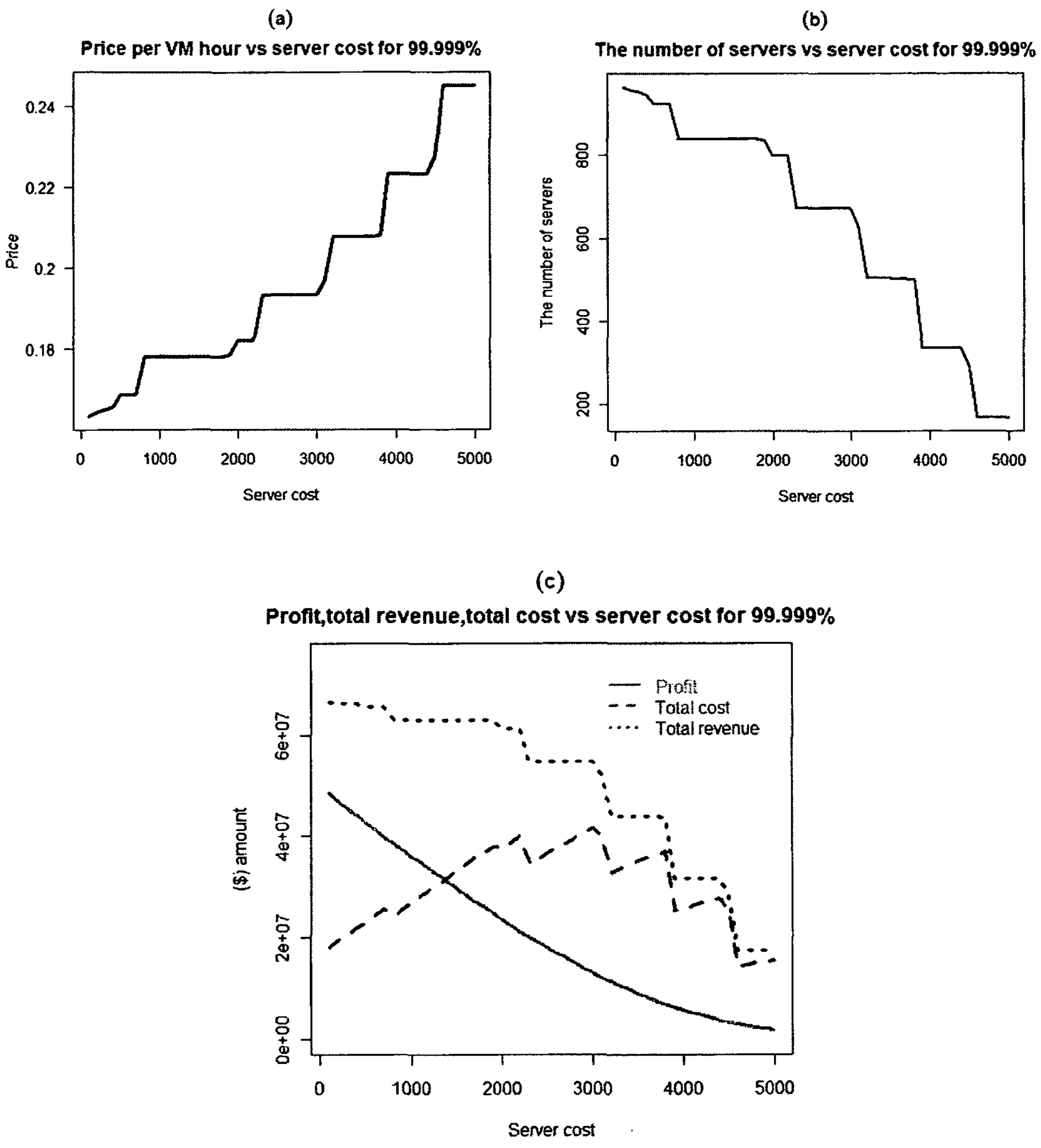

Figure 6.6 a, b, and c. (a) Price per VM hour that maximizes profit for a given server cost for 99.999% availability. (b) the number of servers that maximizes profit for a given server cost for 99.999% availability. (c) Maximum profit (revenue - total cost) for a given server cost for 99.999% availability.

6.4 Conclusion

Finding the right size and the right price is crucial for a Cloud service provider. The right sizing and right pricing of the Cloud is based on revenue and cost analysis.

Pricing is closely related to revenue. If the providers define the price per VM hour higher than the market price, fewer customers are willing to pay. On the other hand, if the price is very low, the cost will exceed the revenue. How to price a VM hour is important. The Total Cost of Ownership is a tool used to analyze the cost of owning the Cloud.

This chapter proposed an economic model for Cloud service providers that can be used to maximize profit based on choosing the right sizing in the Cloud data center for three different qualities of service. In addition, customer satisfaction can be explored as part of the future model. The demand is based on a logit model. TCO analysis is divided into eight categories: server, network, software, power, cooling, facilities, real estate and support and maintenance. The results show how to obtain the maximum profit based on right pricing and right sizing. This work can be extended by considering revenue from charging for storage and data transfer.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

HPC and Cloud computing are mainstream systems that have increasingly gained popularity in scientific and business community. The rise in computing demands and speeds for scientific and knowledge discovery requires new computing systems that are growing larger into peta-scale or even in exa-scale in the near future. When a large number of computing components work together, failures frequently occurs. In order to deal with failures, failure behaviors in such systems need to be studied and well-understood. In addition, there are also fault tolerant techniques, for instance checkpoint/restart, redundancy, or live migration to handle the failures. However, to use these fault tolerant techniques efficiently, failure behaviors, such as reliability function, mean time to failure or failure rate, must be taken into consideration during operation in the large scale environment.

This dissertation presents an improved reliability model for HPC applications in an HPC system and a Cloud computing system. The main contribution is a development of the first reliability model with correlated failures in a *k*-node system, which is an idea extended from an independent case into correlated failures. This dissertation also proposes the univariate parameter estimation technique for Marshall-Olkin Multivariate Weibull distribution that can easily estimate the parameters when a particular system has a large number of nodes while the existing EM parameter estimation does not easily extend for more than four components or computing nodes. Then, expressions are derived for the probability of system failure at any time *t*, for the failure rate, and for the mean time to failure. This work validates the reliability model by using log data from Blue Gene/L system at LLNL. The result shows that when the particular system posses the degree of dependence, the system become less reliable. Secondly, the proposed work extends the idea of the reliability model with correlation failures of an HPC system to model that of a Cloud computing system by modeling the software components using Marshall-Olkin Multivariate Exponential distribution. The empirical result shows that if failures of hardware and software behave dependently, it decreases the system reliability. Finally, this dissertation develops an economic model for Cloud service providers for their decision making in order to maximize profit. As such, the pricing strategy is very vital. How to price to maximize profit is then investigated. Cost is another important factor in profit. Thus, Cloud TCO helps to analyze the cost. In this work, analysis of the failure cost and the penalty cost for different needs of High Availability (HA) services are included in Cloud TCO. The result enables Cloud service providers to decide the right pricing and rightsizing strategy for the Cloud data center.

This dissertation can be further extended in several ways. The reliability models with correlated failures both in HPC and Cloud systems can be improved for the nodes in the system that employs redundant components. Moreover, this work can also be considered as to how it accounts for different failure dependent behaviors. For example, the failures in one node increase the probability of failures in other nodes. In Cloud economic model, revenue and cost for data transfer and storage can be enhanced in the model. Furthermore, analyzing the economic aspect for other Cloud platforms such as Software-as-a-Service and Platform-as-a-Service, which account for the cost parameters such as for application maintenance or development environments, is another potential extension for this work.

REFERENCES

[1] "TOP500 Supercomputing." [Online]. Available: http://top500.org/. [Accessed: 10-Nov-2011].

[2] "Rocks." [Online]. Available: http://www.rocksclusters.org/wordpress/. [Accessed: 10-Nov-2011].

[3] "OSCAR." [Online]. Available: http://svn.oscar.openclustergroup.org/trac/oscar. [Accessed: 10-Nov-2011].

[4] C. Terboven, D. Mey, and S. Sarholz, "OpenMP on Multicore Architectures," in *A Practical Programming Model for the Multi-Core Era*, vol. 4935, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 54–64.

[5] "OpenMP.org." [Online]. Available: http://openmp.org/wp/. [Accessed: 11-Feb-2012].

[6] R. Rabenseifner, G. Hager, and G. Jost, "Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-Core SMP Nodes," in *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, 2009, pp. 427–436.

[7] D. Li, B. R. de Supinski, M. Schulz, K. Cameron, and D. S. Nikolopoulos, "Hybrid MPI/OpenMP power-aware computing," in *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, 2010, pp. 1–12.

[8] B. J. Pope, B. G. Fitch, M. C. Pitman, J. J. Rice, and M. Reumann, "Petascale computation performance of lightweight multiscale cardiac models using hybrid programming models," in *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society,EMBC*, 2011, pp. 433–436.

[9] R. D. Loft, S. J. Thomas, and J. M. Dennis, "Terascale Spectral Element Dynamical Core for Atmospheric General Circulation Models," in *Supercomputing, ACM/IEEE 2001 Conference*, 2001, pp. 26 – 58.

[10] G. Wellein, G. Hager, A. Basermann, and H. Fehske, "Fast sparse matrix-vector multiplication for TeraFlop/s computers," in *Proceedings of the 5th international conference on High performance computing for computational science*, Berlin, Heidelberg, 2003, pp. 287–301.

[11] J. Varma, C. Wang, F. Mueller, C. Engelmann, and S. L. Scott, "Scalable, fault tolerant membership for MPI tasks on HPC systems," in *Proceedings of the 20th annual international conference on Supercomputing*, New York, NY, USA, 2006, pp. 219–228.

[12] Y. Liu, R. Nassar, C. Leangsuksun, N. Naksinehaboon, M. Paun, and S. L. Scott, "An optimal checkpoint/restart model for a large scale high performance computing system," in *IEEE International Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008*, 2008, pp. 1–9.

[13] S. W. Kwak and J. M. Yang, "Schedulability and optimal checkpoint placement for real-time multi-tasks," in *2010 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, 2010, pp. 778–782.

[14] T. Dohi, T. Ozaki, and N. Kaio, "Optimal Checkpoint Placement with Equality Constraints," in *2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing*, 2006, pp. 77–84.

[15] M. Paun, N. Naksinehaboon, R. Nassar, C. Leangsuksun, S. L. Scott, and N. Taerat, "Incremental Checkpoint Schemes for Weibull Failure Distribution," *Int. J. Found. Comput. Sci.*, 2010, pp. 329~344.

[16] S. Yi, J. Heo, and J. Hong, "Taking Point Decision Mechanism of Page-level Incremental Checkpointing based on Cost Analysis of Process Execution Time," *J. Inf. Sci. Eng.*, vol. 23, no. 5, pp. 1325–1337, 2007.

[17] C. Engelmann and S. Böhm, "Redundant Execution of HPC Applications with MR-MPI.," in *In Proceedings of the 10th IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN) 2011*, Innsbruck, Austria, 2011, pp. 31–38.

[18] C. Engelmann, "Redundancy for Soft-Error Resilience in Large-Scale HPC Systems," Schloss Dagstuhl, Wadern, Germany, 03-May-2009.

[19] "VMware Virtualization." [Online]. Available: http://www.vmware.com/. [Accessed: 15-Nov-2011].

[20] "Citrix." [Online]. Available: http://www.citrix.com/lang/English/home.asp. [Accessed: 15-Nov-2011].

[21] "Microsoft Hyper-V." [Online]. Available: http://www.microsoft.com/en-us/server-cloud/hyper-v-server/default.aspx. [Accessed: 15-Nov-2011].

[22] "VirtualBox." [Online]. Available: https://www.virtualbox.org/. [Accessed: 15-Nov-2011].

[23] "Xen® hypervisor." [Online]. Available: http://xen.org/. [Accessed: 15-Nov-2011].

[24] "OpenVZ." [Online]. Available: http://wiki.openvz.org/Main_Page. [Accessed: 15-Nov-2011].

[25] "OpenNebula." [Online]. Available: http://opennebula.org/about:about. [Accessed: 16-Nov-2011].

[26] "Eucalyptus." [Online]. Available: http://www.eucalyptus.com/. [Accessed: 16-Nov-2011].

[27] "OpenStack." [Online]. Available: http://www.openstack.org/. [Accessed: 16-Nov-2011].

[28] "Nimbus." [Online]. Available: http://www.nimbusproject.org/. [Accessed: 16-Nov-2011].

[29] "Amazon Elastic Compute Cloud (Amazon EC2)." [Online]. Available: http://aws.amazon.com/ec2/. [Accessed: 15-Nov-2011].

[30] "GoGrid." [Online]. Available: http://www.gogrid.com/. [Accessed: 15-Nov-2011].

[31] "Rackspace." [Online]. Available: http://www.rackspace.com/. [Accessed: 15-Nov-2011].

[32] "Salesforce.com." [Online]. Available: http://www.salesforce.com/. [Accessed: 15-Nov-2011].

[33] "Google App Engine." [Online]. Available: http://code.google.com/appengine/. [Accessed: 15-Nov-2011].

[34] "Bungee." [Online]. Available: http://www.bungeelabs.com/. [Accessed: 15-Nov-2011].

[35] "Heroku." [Online]. Available: http://www.heroku.com/. [Accessed: 15-Nov-2011].

[36] "Penguin Computing." [Online]. Available: http://www.penguincomputing.com/. [Accessed: 15-Nov-2011].

[37] D. Kondo and A. Andrzejak, "Reducing Costs of Spot Instances via Checkpointing in the Amazon Elastic Compute Cloud," in *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, 2010, pp. 236–243.

[38] T. Heath, R. P. Martin, and T. D. Nguyen, "Improving cluster availability using workstation validation," in *Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, New York, NY, USA, 2002, pp. 217–227.

[39] J. Xu, Z. Kalbarczyk, and R. K. Iyer, "Networked Windows NT system field failure data analysis," in *1999 Pacific Rim International Symposium on Dependable Computing, 1999. Proceedings*, 1999, pp. 178–185.

[40] R. K. Sahoo, A. Sivasubramaniam, M. S. Squillante, and Y. Zhang, "Failure Data Analysis of a Large-Scale Heterogeneous Server Environment," in *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, Washington, DC, USA, 2004, p. 772–781.

[41] N. Raju, Y. Liu, C. Leangsuksun, R. Nassar, and S. Scott, "Reliability Analysis in HPC clusters," in *In Proceedings of High Availability and Performance Workshop (HAPCW) 2006, in conjunction with Los Alamos Computer Science Institute (LACSI) Symposium 2006,*, Santa Fe, NM, USA, 2006.

[42] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien, "Checkpointing strategies for parallel jobs," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, 2011, pp. 1–11.

[43] H. Jin, Y. Chen, H. Zhu, and X. H. Sun, "Optimizing HPC Fault-Tolerant Environment: An Analytical Approach," in *2010 39th International Conference on Parallel Processing (ICPP)*, 2010, pp. 525–534.

[44] N. R. Gottumukkala, R. Nassar, M. Paun, C. B. Leangsuksun, and S. L. Scott, "Reliability of a System of k Nodes for High Performance Computing Applications," *IEEE Transactions on Reliability*, vol. 59, no. 1, pp. 162–169, Mar. 2010.

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

[46] N. R. Gottumukkala, "Failure analysis and reliability-aware resource allocation of parallel applications in high performance computing systems," Louisiana Tech University, Ruston, LA, USA, 2008.

[47] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," in *International Conference on Dependable Systems and Networks, 2006. DSN 2006*, 2006, pp. 249–258.

[48] K. V. Vishwanath and N. Nagappan, "Characterizing cloud computing hardware reliability," in *Proceedings of the 1st ACM symposium on Cloud computing*, New York, NY, USA, 2010, pp. 193–204.

[49] Y. S. Dai, B. Yang, J. Dongarra, and G. Zhang, "Cloud Service Reliability: Modeling and Analysis," presented at the the 15th IEEE Pacific Rim International Symposium on Dependable Computing, 2009.

[50] S. A. Ahson and M. Ilyas, *Cloud Computing and Software Services: Theory and Techniques*, 1st ed. CRC Press, 2010.

[51] T. J. Hacker and Z. Meglicki, "Using queue structures to improve job reliability," in *Proceedings of the 16th international symposium on High performance distributed computing*, New York, NY, USA, 2007, pp. 43–54.

[52] A. Wood, "Software reliability growth models: assumptions vs. reality," in *PROCEEDINGS The Eighth International Symposium On Software Reliability Engineering*, Cupertino, CA, USA, 1997, pp. 136–141.

[53] J. D. Musa, *Software Reliability Engineering.* Osborne/McGraw-Hill, 1998.

[54] T. Shanmugam and D. R. P. Williams, "Analysis of Testing and Operational Software Reliability in SRGM based on NHPP," *International Journal of Computer and Information Engineering*, vol. 1, pp. 284–289, 2007.

[55] B. Yang and M. Xie, "A study of operational and testing reliability in software reliability analysis," *Reliability Engineering & System Safety*, vol. 70, no. 3, pp. 323–329, Dec. 2000.

[56] D. Hamlet, "Are we testing for true reliability?," *IEEE Software*, vol. 9, no. 4, pp. 21–27, Jul. 1992.

[57] K. Goseva-Popstojanova and K. S. Trivedi, "Failure correlation in software reliability models," *IEEE Transactions on Reliability*, vol. 49, pp. 37–48, Mar. 2000.

[58] K. Abdelkader, J. Broeckhove, and K. Vanmechelen, "Commodity resource pricing in dynamic computational grids," in *IEEE/ACS International Conference on Computer Systems and Applications, 2008. AICCSA 2008*, 2008, pp. 422–429.

[59] D. Dash, V. Kantere, and A. Ailamaki, "An Economic Model for Self-Tuned Cloud Caching," in *Proceedings of the 2009 IEEE International Conference on Data Engineering*, Washington, DC, USA, 2009, pp. 1687–1693.

[60] M. Mihailescu and Y. M. Teo, "On Economic and Computational-Efficient Resource Pricing in Large Distributed Systems," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, Washington, DC, USA, 2010, pp. 838–843.

[61] M. Woitaszek and H. M. Tufo, "Developing a Cloud Computing Charging Model for High-Performance Computing Resources," in *2010 IEEE 10th International Conference on Computer and Information Technology (CIT)*, 2010, pp. 210–217.

[62] X. Li, Y. Li, T. Liu, J. Qiu, and F. Wang, "The Method and Tool of Cost Analysis for Cloud Computing," 2009, pp. 93–100.

[63] E. Walker, "The Real Cost of a CPU Hour," *Computer*, vol. 42, no. 4, pp. 35–41, Apr. 2009.

[64] D. Niyato, S. Chaisiri, and Bu-Sung Lee, "Economic analysis of resource market in cloud computing environment," in *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, 2009, pp. 156–162.

[65] D. D. Hanagal, "A multivariate Weibull distribution," *Economic Quality Control*, vol. 11, pp. 193–200, 1996.

[66] A. W. Marshall and I. Olkin, "A Multivariate Exponential Distribution," *Journal of the American Statistical Association*, vol. 62, no. 317, pp. 30–44, Mar. 1967.

[67] R. V. Hogg, A. Craig, and J. W. McKean, *Introduction to Mathematical Statistics*, 6th ed. Prentice Hall, 2004.

[68] F. Proschan and P. Sullo, "Estimating the Parameters of a Multivariate Exponential Distribution," *Journal of the American Statistical Association*, vol. 71, no. 354, pp. 465–472, Jun. 1976.

[69] D. Kundu and A. K. Dey, "Estimating the parameters of the Marshall-Olkin bivariate Weibull distribution by EM algorithm," *Comput. Stat. Data Anal.*, vol. 53, no. 4, pp. 956–965, Feb. 2009.

[70] F. Parsons and P. Wirsching, "A Kolmogorov - Smirnov goodness-of-fit test for the two-parameter weibull distribution when the parameters are estimated from the data," *Microelectronics Reliability*, vol. 22, no. 2, pp. 163–167, 1982.

[71] B. C. Aguado Sánchez, J. Blomer, A. Harutyunyan, and M. Mudrinic, "A practical approach to virtualization in HEP," *The European Physical Journal Plus*, vol. 126, no. 1, pp. 1–8, 2011.

[72] "Amazon AWS Case Studies: High Performance Computing." [Online]. Available: http://aws.amazon.com/solutions/case-studies/#hpc. [Accessed: 18-Nov-2011].

[73] "High-Performance Computing as a Service (HPCaaS)." [Online]. Available: http://www.mellanox.com/blog/2009/04/high-performance-computing-as-a-service-hpcaas/. [Accessed: 18-Nov-2011].

[74] R. Masud and M. J. Sottile, "High Performance Computing with Clouds," University of Oregon, Technical Report CIS-TR-2010-06, Jan. 2010.

[75] M. Snir, J. Dongarra, J. S. Kowalik, S. Huss-Lederman, S. W. Otto, and D. W. Walker, *MPI: The Complete Reference*, 2nd ed. The MIT Press, 1998.

[76] Y. Peng and F. Wang, "Cloud computing model based on MPI and OpenMP," in *2010 2nd International Conference on Computer Engineering and Technology (ICCET)*, 2010, vol. 7, pp. V7–85–V7–87.

[77] N. Bonvin, T. G. Papaioannou, and K. Aberer, "Cost-efficient and differentiated data availability guarantees in data clouds," in *2010 IEEE 26th International Conference on Data Engineering (ICDE)*, 2010, pp. 980–983.

[78] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu, "Performance and availability aware regeneration for cloud based multitier applications," in *2010 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2010, pp. 497–506.

[79] A. Andrzejak, D. Kondo, and S. Yi, "Decision Model for Cloud Computing under SLA Constraints," in *2010 IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2010, pp. 257–266.

[80] K. H. Prasad, T. A. Faruquie, L. V. Subramaniam, M. Mohania, and G. Venkatachaliah, "Resource Allocation and SLA Determination for Large Data Processing Services over Cloud," in *2010 IEEE International Conference on Services Computing (SCC)*, 2010, pp. 522–529.

[81] T. Thein and J. S. Park, "Availability analysis of application servers using software rejuvenation and virtualization," *J. Comput. Sci. Technol.*, vol. 24, no. 2, pp. 339–346, Mar. 2009.

[82] M. Grottke, Lei Li, K. Vaidyanathan, and K. S. Trivedi, "Analysis of Software Aging in a Web Server," *IEEE Transactions on Reliability*, vol. 55, no. 3, pp. 411–420, Sep. 2006.

[83] I. Goiri, F. Julià, J. Guitart, and J. Torres, "Checkpoint-based fault-tolerant infrastructure for virtualized service providers," in *2010 IEEE Network Operations and Management Symposium (NOMS)*, 2010, pp. 455–462.

[84] M. Slawinska, J. Slawinski, and V. Sunderam, "Unibus: Aspects of heterogeneity and fault tolerance in cloud computing," in *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, 2010, pp. 1–10.

[85] S. E. Landsburg, *Price Theory and Applications*, 6th ed. South-Western College Pub, 2004.

[86] T. T. Nagle and J. E. Hogan, *The Strategy and Tactics of Pricing: A Guide to Growing More Profitably*, 4th ed. Pearson Prentice Hall, 2005.

[87] R. Phillips, *Pricing and Revenue Optimization*, 1st ed. Stanford Business Books, 2005.

[88] P. Dube, Y. Hayel, and L. Wynter, "Yield management for IT resources on demand: Analysis and validation of a new paradigm for managing computing centres," *Journal of Revenue and Pricing Management*, vol. 4, no. 1, pp. 24–38, 2005.

[89] D. Scott, "The High Cost of Achieving Higher Levels of Availability," Gartner Research, Research Note, Strategic Planning SPA-13-9852, Jun. 2001.

[90] H. J. Moon, Yun Chi, and H. Hacigu□mu□s□, “SLA-Aware Profit Optimization in Cloud Services via Resource Scheduling,” in *2010 6th World Congress on Services (SERVICES-1)*, 2010, pp. 152–153.

[91] S. Arunagiri, J. T. Daly, and P. J. Teller, “Propitious Checkpoint Intervals to Improve System Performance by Sarala Arunagiri, John T. Daly et al.,” University of Texas at El Paso, Technical Report UTEP-CS-09-09, Mar. 2009.

[92] U. Hoelzle and L. A. Barroso, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, 1st ed. Morgan and Claypool Publishers, 2009.