Spring 2012

Near-optimal scheduling and decision-making models for reactive and proactive fault tolerance mechanisms

Nichamon Naksinehaboon

NEAR-OPTIMAL SCHEDULING AND DECISION-MAKING MODELS FOR REACTIVE AND PROACTIVE FAULT TOLERANCE MECHANISMS

by

Nichamon Naksinehaboon, B.S.

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

COLLEGE OF ENGINEERING AND SCIENCE
LOUISIANA TECH UNIVERSITY

May 2012

LOUISIANA TECH UNIVERSITY

THE GRADUATE SCHOOL

2/16/2012
Date

We hereby recommend that the dissertation prepared under our supervision by NICHAMON NAKSINEHABOON

entitled NEAR-OPTIMAL SCHEDULING AND DECISION-MAKING MODELS FOR REACTIVE AND PROACTIVE FAULT TOLERANCE MECHANISMS

be accepted in partial fulfillment of the requirements for the Degree of DOCTOR OF PHILOSOPHY

[illegible]
Supervisor of Dissertation Research

[illegible]
Head of Department

Computational Analysis and Modeling
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

As High Performance Computing (HPC) systems increase in size to fulfill computational power demand, the chance of failure occurrences dramatically increases, resulting in potentially large amounts of lost computing time. Fault Tolerance (FT) mechanisms aim to mitigate the impact of failure occurrences to the running applications. However, the overhead of FT mechanisms increases proportionally to the HPC systems' size. Therefore, challenges arise in handling the expensive overhead of FT mechanisms while minimizing the large amount of lost computing time due to failure occurrences.

In this dissertation, a near-optimal scheduling model is built to determine when to invoke a hybrid checkpoint mechanism, by means of stochastic processes and calculus of variations. The obtained schedule minimizes the waste time caused by checkpoint mechanism and failure occurrences. Generally, the checkpoint/restart mechanisms periodically save application states and load the saved state, upon failure occurrences. Furthermore, to handle various FT mechanisms, an adaptive decision-making model has been developed to determine the best FT strategy to invoke at each decision point. The best mechanism at each decision point is selected among considered FT mechanisms to globally minimize the total waste time for an application execution by means of a dynamic programming approach. In addition, the model is adaptive to deal with changes in failure rate over time.

APPROVAL FOR SCHOLARLY DISSEMINATION

Author N. NAKSINEHABOON [illegible]

Date 02/16/2012

DEDICATION

To my late mother, Tanya Naksinehaboon, I dedicate this work.

TABLE OF CONTENTS

ABSTRACT iii

DEDICATION v

LIST OF TABLES viii

LIST OF FIGURES ix

ACKNOWLEDGMENTS xi

CHAPTER 1 INTRODUCTION 1

1.1 Failure Prone Environment in HPC Systems 1

1.2 Fault Tolerance Mechanisms 2

1.2.1 Reactive FT Mechanisms 3

1.2.2 Proactive FT Mechanisms 4

CHAPTER 2 BACKGROUND AND RELATED WORK 8

2.1 Checkpoint Scheduling Models 8

2.2 Rejuvenation Scheduling Models 13

2.3 Scheduling and Decision-making Models for More than One FT Mechanism 15

CHAPTER 3 A HYBRID CHECKPOINT SCHEDULING MODEL FOR ARBITRARY FAILURE DISTRIBUTIONS 18

3.1 An Optimal Checkpoint Frequency Function 21

3.2 Estimation of the Re-computing Time Coefficient k 27

3.3 The Optimal Number of Incremental Checkpoints between Two Consecutive Full Checkpoints m for Arbitrary Failure Distributions 30

CHAPTER 4 A HYBRID CHECKPOINT SCHEDULING MODEL FOR WEIBULL DISTRIBUTIONS 34

4.1 Near-optimal Checkpoint Times for Weibull Distributions 34

4.2 Comparisons between the Hybrid and the Full Checkpoint Mechanisms for Weibull Distributions 38

4.2.1 Discussion on the Number of Incremental Checkpoints m 40

4.2.2 Discussion on Waste Time 41

4.2.3 Discussion on Re-computing Time, Checkpoint Overhead, and Recovery Cost 44

CHAPTER 5 A DECISION-MAKING MODEL FOR REACTIVE AND PROACTIVE FT MECHANISMS 48

5.1 An Adaptive Decision-making Model 49

5.2 A Decision-making Model for Full Checkpoints, Incremental Checkpoints, and Rejuvenation 52

5.3 Simulations 60

5.4 Simulation Results 62

5.4.1 The Best Policy 63

5.4.2 Conditions when there is No Need for any FT Mechanisms 64

5.4.3 Conditions when the Best Decision Interval is Less than 10% of the Completion Time 69

CHAPTER 6 CONCLUSIONS 72

APPENDIX A MINIMUM OF A SET OF RECURSIVE, LINEAR FUNCTIONS 75

BIBLIOGRAPHY 78

LIST OF TABLES

Table 3.1: Notations in the hybrid checkpoint scheduling model 21

Table 4.1: Parameter values in simulations 40

Table 5.1: Notations in the decision-making model 54

Table 5.2: Parameter values in simulations 61

Table 5.3: Cases that the best decision interval is equal to the completion time ($\Delta^* = d$) (No need for any FT mechanisms) 65

Table 5.4: Cases that the best decision interval is less than or equal to 10% of the completion time, where d is the application completion time... 68

LIST OF FIGURES

Figure 3.1: Hybrid checkpoint/restart mechanism scheme 19

Figure 3.2: Relationship between the re-computing time and the checkpoint interval 25

Figure 4.1: Averages of number of incremental checkpoints between two consecutive full checkpoints (m) when MTTF is 3 hours, and the shape parameters are 0.5 (left), 1 (middle), and 1.5 (right) 41

Figure 4.2: Averages of the number of incremental checkpoints between two consecutive full checkpoints (m) when MTTF is 1 day, and the shape parameters are 0.5 (left), 1 (middle), and 1.5 (right) 42

Figure 4.3: Percentages of the waste time when the system running time is 1 day, MTTF is 1 day, and the shape parameters are 0.5 (left), 1 (middle), and 1.5 (right) 42

Figure 4.4: Conditional probability of a failure occurrence before time $a + \Delta$, given that the system survives until time a, where Δ is the checkpoint interval when the MTTF is 1 day, the shape parameter is 1, the full checkpoint overhead O_F is 5 minutes, and the incremental checkpoint overhead O_I is 10% of O_F 43

Figure 4.5: Percentages of the waste time when the system running time is 7 days, MTTF is 1 day, and the shape parameters are 0.5 (left), 1 (middle), and 1.5 (right) 44

Figure 4.6: Percentages of the re-computing time when the system running time is 1 day, MTTF is 1 day, and the shape parameters are 0.5 (left), 1 (middle), and 1.5 (right) 45

Figure 4.7: Averages of the number of checkpoints when the system running time is 1 day, MTTF is 1 day, and the shape parameters are 0.5 (left), 1 (middle), and 1.5 (right) 46

Figure 4.8: Percentages of the checkpoint overheads when the system running time is 1 day, MTTF is 1 day, and the shape parameters are 0.5 (left), 1 (middle), and 1.5 (right) 46

Figure 4.9: Percentages of the recovery cost when the system running time is 1 day, MTTF is 1 day, and the shape parameters are 0.5 (left), 1 (middle), and 1.5 (right) 47

Figure 5.1: Three scenarios of the decision-making model of an $N-$ stage process with the decision interval of length Δ: a) there is no failure occurrence, and no FT mechanism is performed during the application execution; b) an FT mechanism is performed at each decision point, but there is no failure occurrence; and c) an FT mechanism is performed at each decision point and a failure occurs during the application execution........ 50

Figure 5.2: Example graphs of the expected total waste times when $\Delta^* = d$ (Left), $\frac{d}{10} < \Delta^* < d$ (Middle), and $\Delta^* \leq 10\%$ of d (Right), where Δ^* is the decision interval that gives the smallest expected total waste time and d is the application completion time........ 63

Figure 5.3: Conditional Probability $(P(\Omega < a + \Delta | \Omega > a))$ of the Weibull distributions with the shape parameters (β) of 0.5, 1, and 1.5, completion time (d) of 2 days, the decision interval (Δ) of 2 days, and MTTF of 16 days. The x-axis is the age of the software state (a) in minutes, ranged from 0 to 4 days........ 66

Figure 5.4: Pie Chart of the MTTFs as multiples of the completion time (d) (Left), the shape parameter (β)(Middle), and the incremental checkpoint overheads (O_I) as percentages of the completion times (Right) for the cases that the best decision interval is less than or equal to 10% of the completion time 69

Figure 5.5: Conditional Probability $(P(\Omega < a + \Delta | \Omega > a))$ of the Weibull distributions with the shape parameters of 0.5, 1, and 1.5, the completion time (d) of 2 days, and the MTTF of 12 hours. The x-axis is the age of the software state (a) in minutes, ranged from 0 to 2 days 70

ACKNOWLEDGMENTS

This dissertation would not have been possible without the wise guidance and support of my advisor, Dr. Chokchai (Box) Leangsuksun. I will be forever grateful to him for giving me opportunities and challenges to drive this dissertation to its fullest.

My eternal gratefulness shall be extended to Dr. Raja Nassar for his wisdom in probability and statistics and his patience in editing the technical papers. I would like to express my gratitude to my research committee; Dr. Galen Turner and Dr. Bernd Schroeder for suggesting and validating my research in mathematical analysis; Dr. Weizhong Dai for guiding me through the CAM program; and Dr. Dexter Cahoy for providing insightful statistical discussion. Besides, I would like to express my gratitude to Dr. Richard Greechie for giving me opportunities to explore the beautiful world of mathematics; Dr. Mihaela Paun for suggestions in my initial research; and Dr. Jinko Kanno for philosophy in life.

Personally, I would like to gratefully thank my husband, Narate Taerat, for his neverending professional and personal support. This journey would have never begun without my family; my mother, Tanya Naksinehaboon, who had seeded in me a thought of pursuing a doctoral degree; my father, Wikorn-ake Naksinehaboon, who has always supported me and believed in me; my uncle, Somkit Naksinehapon, who has been a pillar of support throughout. Finally, I would like to thank my friends who have added fun colors in this journey.

CHAPTER 1

INTRODUCTION

High Performance Computing (HPC) systems such as supercomputers play an important role in solving advanced computational problems because of their high computational power. The demand for high computational power has increased and has driven the HPC systems' physical sizes to a level where reliability is a major concern. Thus, HPC systems with a large number of compute components are prone to failure occurrences [14, 26, 60].

1.1 Failure Prone Environment in HPC Systems

In 2006, Schroeder and Gibson [59] analyzed the failure datasets, collected over 9 years, of 22 HPC systems hosted at Los Alamos National Laboratory (LANL). They reported that the major root cause of failures is hardware failures. Also, the "failure rate," which is the number of failures observed in a given period of time, is high during the configuration period and drops when the systems are in production time. During the production time, the failure rate on different systems ranges from 10 to 700 failures per year. Gibson et al. [27] showed that it is possible that the number of failures reaches 2,000 failures per year. Further, Schroeder and Gibson [60] predicted the failure rate of exascale systems to be once every 3-26 minutes. Glosli et al. [28] observed that the BlueGene/L system at Lawrence Livermore National Laboratory

(LLNL), which is comprised of more than 100,000 compute nodes, has experienced a failure every 7-10 days.

Failure occurrences interrupt the running applications, resulting in lost computed work. Upon a failure occurrence, the systems need to re-compute the portion of the applications that has been computed before the failure. We call the amount of time that the system spends to re-compute the application the "re-computing time." Due to the high failure rate in HPC systems, the re-computing time is significant. Thus, the HPC community has been trying to find a solution that reduces the re-computing time.

1.2 Fault Tolerance Mechanisms

Fault tolerance (FT) is "the ability of a system or a component to continue normal operation in spite of the presence of hardware or software faults" [1]. FT mechanisms are hardware and software techniques that alleviate the effects of failure occurrences on HPC systems. They can be categorized into reactive and proactive FT mechanisms. Reactive FT mechanisms, such as checkpoint/restart and process redundancy, are mechanisms that aim to reduce the failure impact on the running applications. On the other hand, proactive FT mechanisms, such as process migration and rejuvenation, are mechanisms that aim to prevent running applications from failure occurrences. Consequently, reactive and proactive FT mechanisms complement each other.

1.2.1 Reactive FT Mechanisms

Reactive FT mechanisms aim to reduce the amount of lost computed work. upon a failure occurrence. Examples of reactive mechanisms are the checkpoint/restart mechanism and process redundancy. In this work. the only reactive mechanism that we focus on is the checkpoint/restart mechanism because it is the most widely used FT mechanism in HPC environments [56].

The checkpoint/restart mechanism periodically saves the application states to a stable storage. A "checkpoint" is a point at which the system saves the application state [35. 75]. Upon a failure occurrence, the system loads the last saved state [34, 35] and computes the application from the loaded state.

Since saving an application state also interrupts the computation and requires additional execution time to complete the applications, the "checkpoint overhead" is the additional execution time due to a checkpoint [63]. Besides the checkpoint overhead. "checkpoint latency" is the time that the system spends to store the saved state to a stable storage [63]. If the system pauses the computation of the running application until completely storing the saved state, the checkpoint overhead is equal to the checkpoint latency [34]. Besides the cost for saving the application states. the time duration that the system spends to load the saved state or recover the application is called the "recovery cost."

A *full checkpoint mechanism* is a conventional checkpoint technique that saves a whole application state at each checkpoint. Unfortunately. saving a whole application state causes an expensive checkpoint overhead. Due to its high overhead. the full checkpoint technique becomes limited in large-scale HPC systems [2]. Thus, there are alternative checkpoint techniques that aim to reduce the full checkpoint overhead.

An *incremental checkpoint mechanism* is an alternative checkpoint mechanism that has been introduced by Bauer [5, 6]. Instead of saving a whole application state, the incremental checkpoint mechanism saves only the changes made by the application from the previous saved state [52]. Thus, the incremental checkpoint overhead is potentially smaller than the full checkpoint overhead. Furthermore, because the application state is incrementally saved, upon a failure, the system must load all incremental checkpoints to recover the last saved state of the application [52]. Consequently, the overall cost of the incremental checkpoint technique could be larger than that of the full checkpoint technique.

1.2.2 Proactive FT Mechanisms

Proactive FT mechanisms aim to prevent the running applications from failure occurrences [22]. To protect the running applications, the proactive FT mechanisms perform some actions according to failure prediction. Two well-known proactive FT mechanisms are process migration and software rejuvenation.

Process migration is a mechanism that migrates or moves the application processes from the unreliable resources to more reliable resources in order to continue application execution. Similarly to the checkpoint/restart mechanisms, migrating the processes interrupts the running applications and causes additional execution time. Furthermore, process migration is heavily based on failure prediction to determine a point in time to migrate the processes with the least impact [67].

Haung et al. [33] have proposed *software rejuvenation* as a complementary approach to the checkpoint/restart mechanism. The concept is to intentionally terminate an application and restart it at a clean internal state to avoid transient software failures caused by the software aging phenomenon. The *software aging phenomenon* refers to the aggregation of errors during an application execution, resulting in performance degradation or software failures [24]. Software rejuvenation complements the checkpoint/restart mechanism because it reduces the number of failures that the running applications might encounter. If a failure occurs, the systems are still able to recover the applications from the checkpoints.

Utilizing FT mechanisms does not completely eradicate the re-computing time. In addition, it introduces overhead and lengthens the application execution time. Therefore, the frequency of performing FT mechanisms is crucial to efficiently utilize them. For example, running applications are periodically saved because the failure time is random. If the applications are saved very often, the re-computing time is potentially small. However, the total checkpoint overhead is large. In contrast, if the system rarely saves the applications, it then spends a large amount of time to re-compute the lost work after a failure, but a small amount of time to save the application states. Moreover, as HPC systems grow in size, the checkpoint/restart mechanism limits application scalability. Particularly, large-scale HPC systems may spend large amount of their time saving application states instead of executing useful work [21, 42, 68]. Therefore, the means to utilize the checkpoint/restart mechanism should be intelligent [50].

To improve the utilization of the checkpoint/restart mechanism. we have derived a checkpoint scheduling model to determine a sequence of checkpoint times that minimizes the waste time. The "waste time" is the accumulation of the checkpoint overhead. the recovery cost and the re-computing time upon failures. A "checkpoint time" is a point in time at which the system initiates a checkpoint mechanism. The scheduling model is derived for the hybrid checkpoint mechanism that combines both the full and incremental checkpoint techniques. We consider the combination of the full and incremental checkpoint techniques because the combination balances the total checkpoint overhead and the recovery time. Moreover, the proposed scheduling model is the first scheduling model for the hybrid checkpoint mechanism. The details of the hybrid checkpoint mechanism are discussed in Chapter 3. Also, in Chapter 3 we derive the optimal checkpoint frequency function for the hybrid checkpoint mechanism for arbitrary failure distributions. To give a concrete example of the sequence of near-optimal checkpoint times, the failure process is assumed to follow a Weibull distribution, which is the best fitted failure distribution [29, 59]. In Chapter 4, the formula for checkpoint times of the hybrid checkpoint mechanism is presented for failures that follow a Weibull distribution.

The checkpoint frequency depends on the failure rate of the system. With an increasing failure rate, the system should save application states more often over time to cope with the higher chance of failures. Rejuvenating the applications. when the failure rate is high. will keep the software in a fresh state. so the checkpoint frequency will not increase over time. resulting in lower total checkpoint overhead. For both checkpoint and rejuvenation. the critical question is when to save and to rejuvenate the applications. This is because it might cause unnecessary overhead if the application is rejuvenated saved too often. Therefore, in Chapter 5, based on a dynamic programming approach, we have derived an adaptive decision-making model to determine the best mechanism at each decision point to minimize the waste time during an application execution. The concept of the decision-making model is general enough to be applied for any FT mechanisms. Moreover, the proposed decision-making model is the first decision model that globally minimizes the application execution time for a given decision interval. Most existing models make a decision based on the local minimization at each decision point. Finally, the conclusion is given in Chapter 6. In the next chapter, we discuss background and related work of this research.

CHAPTER 2

BACKGROUND AND RELATED WORK

Fault Tolerance (FT) mechanisms play an important role in dealing with failures in HPC systems. However, the overhead of FT mechanisms is increasing because of the increase in the physical size of HPC systems and the complexity of computational problems. This leads to the question when or how often to invoke these FT mechanisms so that it is worth to pay the cost of the FT mechanisms for maintaining low re-computing time.

The research in answering the questions of when or how often to perform FT mechanisms has a rich history. Specifically, the problems of when or how often to perform FT mechanisms are called "FT-mechanism scheduling problems." Moreover, a mathematical model to determine the optimal frequency or times is called a "scheduling model."

2.1 Checkpoint Scheduling Models

The checkpoint/restart mechanisms can be invoked in two approaches [12]. Firstly, the programmers analyze their program and insert checkpoint calls in the codes intuitively or based on a scheduling model. Secondly, the checkpoint/restart mechanism is invoked periodically by the systems, regardless of the running applications. Most early studies in checkpoint scheduling models have the assumption that the failures follows an exponential distribution which is not the case in practice.

The first study that aims to assist programmers to determine the insertion points for checkpoints is by Chandy and Ramamoorthy [12]. They developed a decision-making algorithm to aid programmers in determining the optimum points to save the application states, based on the checkpoint overhead after each instruction in the codes. In [11], Chandy proposed another checkpoint scheduling model that minimizes the checkpoint overhead and the re-computing time per unit time. Most scheduling models concentrate on minimizing the application execution time. The "application execution time" is the time from the start of the computation until the end of the computation, including the overhead and the recovery cost of the FT mechanism as well as the re-computing time in case of failure occurrences. Some early works are discussed as follows.

Young [75] proposed a first order approximation of the optimal checkpoint interval that takes into account only the checkpoint overhead. Leung and Choo [38] determined a checkpoint interval which would minimize the application execution time and allowed failures during saving the application states. Instead of minimizing the application execution time, Geist et al. [25] determined an optimal checkpoint strategy that maximizes the probability of application completion. In 1998, Plank and Elwasif [54] performed experimental research to corroborate the applicability of some existing checkpoint scheduling models at the time (Duda's model [20], Vaidya's model [64] and Young's model [75]). Plank and Elwasif's work [54] serves as a good list of significant work, before 1998, on the checkpoint scheduling models that minimize the application execution time by employing the checkpoint overhead and the re-computing time.

Later, Daly [15] improved Young's work [75] by considering the re-computing time in the model. To obtain a more accurate model, Daly [17] approximated the waste time function by using a higher order approximation, and he further refined his series of works again in [16]. Young [75] and Daly [15] have established a strong basis in developing checkpoint scheduling models that minimizes the application execution time. Hong et al. [31] developed a scheduling scheme to make a decision whether or not to save the application state based on the predicted checkpoint overhead. They predicted the checkpoint overhead by utilizing the memory usage profile and time series analysis. Oliner et al. [49] proposed a so-called cooperative checkpointing, which is a checkpoint scheme based on Young's periodic checkpoint model [75]. The scheme considers the sequence of checkpoint times obtained from Young's model and decides whether to skip or to perform the checkpoints based on the expected cost.

Most checkpoint scheduling models were developed based on the assumption of constant failure rates, equivalently that the failures on the systems follow an exponential distribution. However, there is evidence that failures in HPC systems are not always exponentially distributed [30, 41, 55, 58, 59, 73]. Recent studies have attempted to relax the assumption of exponentially distributed failures. Ling et al. [39] presented optimal checkpoint scheduling models for an infinite horizon time by using a calculus of variations technique. They theoretically concluded that the fixed checkpoint interval is optimal if and only if the system failures follow a Poisson process with constant failure rate. Ozaki et al. [51] extended the concept of the variational calculus in [39] to a finite horizon time and incomplete system failure information. However, both of these papers considered the re-computing time as a linear function for demonstrating model applicability which is the restriction of the model.

Liu et al. [40, 41] enhanced the model in [51] by relaxing the assumption of linearity of the re-computing time. Liu's model is strongly related to the hybrid checkpoint scheduling model in Chapter 3 because the hybrid checkpoint scheduling model is an extension of Liu's model.

Kwak and Yang [36] determined the optimal checkpoint intervals in multiple real-time applications. The width of the optimal checkpoint interval is constant through each application execution, but the widths are different with respect to each kind of applications. Moody et al. [42] designed a scalable multi-level checkpointing system that saves the whole application state to different storages according to the severity of the predicted failures. Furthermore, they developed a Markov model to schedule the checkpoints to different storages. The strong point of their work is that the designed multi-level checkpoint mechanism has been deployed on the clusters at LLNL. Bougeret et al. [9] developed a dynamic programming model to determine the optimal checkpoint interval that maximizes the amount of work completed before the next failure. Also, they considered various models of job parallelism. This is a strong point of this work because there are a few existing studies that addressed the checkpoint overhead for parallel applications. To determine each checkpoint interval, they iteratively compared the expected execution time of the remaining work over different lengths of checkpoint intervals. They also proposed algorithms to solve the model for exponential and Weibull distributions.

Up to this point, we have mentioned only the scheduling models for the full checkpoint technique. The studies to optimally schedule other checkpoint techniques are limited because few techniques have been implemented, and none of them have been deployed on HPC systems. Besides the full checkpoint technique, only the incremental checkpoint technique gains attention from researchers to study the optimal strategy to utilize it.

Yi et al. [74] have proposed an adaptive decision-making model for incremental checkpoints. However, the objective is to determine the optimal checkpoint interval. The model is a decision-making model because, at each decision point, the expected recovery costs for the cases with and without incremental checkpoints are compared. If the expected recovery cost without the incremental checkpoint is less than the expected recovery cost with the incremental checkpoint, then the model decides to skip the checkpoint at that particular decision point.

The first prototype of the hybrid checkpoint mechanism was developed by Wang et al. [69]. They conducted experiments on the benefits of the hybrid checkpoint mechanism against the full checkpoint mechanism, and the experimental results showed that the hybrid checkpoint mechanism outperforms the full checkpoint mechanism with the assumption of exponentially distributed failures.

2.2 Rejuvenation Scheduling Models

Software rejuvenation is tightly related to software degradation. so there are many studies in software degradation or aging phenomena that serve as evidence of the benefits of software rejuvenation. Garg et al. [24] detected and estimated trends and time to exhaustion of operating system resources due to software aging. Vaidyanathan and Trividi [66] took system workload into account and constructed a model for estimating resource depletion times. Bobbio et al. [7] applied Fluid Stochastic Petri Nets [32] to capture the behavior of aging software systems with the checkpoint/restart mechanism and software rejuvenation enabled.

There are several studies on software rejuvenation. Most existing works have focused on maximizing the availability of the system or minimizing the overhead of software rejuvenation. Dohi et al. [18, 19] formulated a stochastic model via a semi-Markov process to schedule software rejuvenation that maximizes the system availability. To numerically determine the optimal schedule, they also developed non-parametric statistical algorithms. Bobbio et al. [8] presented a fine grained degradation level based on the observation of a system parameter. They then presented an optimal rejuvenation policy based on a risk criterion and an alert threshold. Cassidy et al. [10] employed pattern recognition methods on large on-line transaction processing server datasets. using their model to identify sufficient warning to initiate rejuvenation. Xin-yuan et al. [72] proposed a prediction-based software rejuvenation mechanism into a clustering architecture and analyzed the availability of the systems enabling the proposed rejuvenation scheduling. The numerical results suggested that the availability of the system increased by rejuvenating software. Xie et al. [71]

proposed two rejuvenation policies by considering the cluster workload (peak hour and off-peak hour). A Markov process was modeled and numerically analyzed by the Stochastic Petri Net Package (SPNP) [13]. The first policy is Fixed Rejuvenation. When a node is in a failure-prone environment, it is rejuvenated within a deterministic duration after entering the failure-prone state, regardless of whether it is a peak hour or an off-peak hour. The second policy is Delayed Rejuvenation. During peak hours, a node is not rejuvenated until the off-peak hours start, although this may mean that the node enters the failure-prone state.

In 2005, Vaidyanatham and Trividi [65] extended the workload-based approach by performing transient analysis and formulating the estimated time to resource exhaustion as the mean time to accumulated reward in a semi-Markov reward model. Then, they developed an upper-level availability model that accounts for failure and rejuvenation. Lastly, they used this model to derive optimal rejuvenation schedules. Okamura et al. [48] derived a rejuvenation scheduling model, based on a dynamic programming approach, that minimizes the expected execution time. Zhao and Song [76] applied continuous time Markov chains and the concept of common software aging-related faults to study the effectiveness of software rejuvenation. According to their numerical example, the steady state availability of the system was improved by avoiding common faults. Avritzer et al. [3, 4] determined the best time to trigger the software rejuvenation by tracking the progress of the application computation. They found that utilizing software rejuvenation improves the chance of completing applications significantly. Tian and Meng [62] presented a coordinated selective rejuvenation scheme. The scheme uses Bayesian network to identify the problematic component that is responsible for the system violation. To make a decision for rejuvenation, they used a rejuvenation gain/cost model and dynamic multi-threshold algorithm, based on the assumption that the longer the rejuvenation interval, the more gain from the rejuvenation.

2.3 Scheduling and Decision-making Models for More than One FT Mechanism

There are very few existing studies that consider more than one FT mechanism. Such studies can be categorized into two types: scheduling models and decision-making models. Moreover, they often consider a combination of reactive and proactive FT mechanisms. The differences between the scheduling models and the decision-making models are the following. The scheduling models determine the time to invoke each mechanism corresponding to some predefined factors such as the number of checkpoints between rejuvenation. On the other hand, the decision-making models select the best mechanism based on some criteria with the predefined interval between two decision points.

Garg et al. [23] dealt with the analysis of three scenarios; 1) without checkpoints and rejuvenation, 2) with only checkpoints, and 3) with both checkpoints and rejuvenation. They claimed that their work is the first paper that proposed the combination of checkpoints and rejuvenation. They minimized the application execution time by means of a dynamic programming approach. In the model, the checkpoint interval is constant, and rejuvenation is triggered right after every k^{th}

checkpoint. A significant result of this paper is that if the failure process is exponentially distributed, employing rejuvenation does not shorten the execution time because of the memoryless property of the exponential distribution. The exponential distribution renews the failure process every instant, so the renewal by rejuvenation does not help the failure process at all. Instead, it introduces additional overhead. The adaptive model in Chapter 5 is more flexible than Garg et al.'s model because the proposed model in Chapter 5 does not fix the number of checkpoints between rejuvenations.

Lan and Li [37] proposed an adaptive decision-making model. At each decision point, three actions are considered: 1) performing a checkpoint, 2) migrating the processes, and 3) doing nothing. The expected cost of each action is derived, and the best action at each decision point is the action that gives the minimum expected cost. However, the derived expected cost of each action is not the expected total cost until the completion of the execution. Instead, they are the expected costs for the interval between the current decision point and the next decision point. Therefore, the execution time based on their model might not be the minimum one. In contrast, the proposed decision-making model in Chapter 5 globally minimizes the expected total waste time, resulting in achieving the minimal expected execution time.

Okamura and Dohi [46] developed a stochastic model with three reactive/proactive FT mechanisms (rejuvenation, restoration and checkpoint). To determine a joint optimal maintenance schedule, they proposed a dynamic programming algorithm to maximize the steady-state system availability. Later in [47], Okamura and Dohi considered a concept of full maintenance in which both checkpoint and rejuvenation are performed. They called the duration between two full maintenances a "cycle." Two schemes were considered in the work. Checkpoint prior to rejuvenation (CPTR) is a process in which only checkpoints are performed in a cycle. Rejuvenation prior to checkpoint (RPTC), in contrast, is a process that only performs rejuvenations. Moreover, they proposed a dynamic programming algorithm to solve for optimal intervals in which the optimal maintenance schedule must be satisfied. However, the number of checkpoints or rejuvenations in a cycle are heuristically determined. In contrast, the decision-making model in this dissertation selects the best FT mechanism based on the least expected waste time among all considered FT mechanisms.

The adaptive decision-making model in Chapter 5 is the first decision-making model that globally minimizes the expected application execution time. Moreover, it can be applied for arbitrary failure distributions and FT mechanisms, but the complexity of the model varies corresponding to the mechanisms and the failure distributions.

CHAPTER 3

A HYBRID CHECKPOINT SCHEDULING MODEL FOR ARBITRARY FAILURE DISTRIBUTIONS

The hybrid checkpoint mechanism is a checkpoint/restart mechanism that utilizes both full and incremental checkpoints. We combine both checkpoint techniques because of the expensive overhead of full checkpoints and the large recovery cost of incremental checkpoints. In this chapter, the details of the hybrid checkpoint and the scheduling model for the hybrid checkpoint mechanism will be discussed. The work in this chapter is partially embedded in [43, 44, 53].

In the hybrid checkpoint mechanism, the first checkpoint since the starting or restarting point must be a full checkpoint and is followed by incremental checkpoints. After a certain number of incremental checkpoints, a full checkpoint will be performed again in order to balance between the recovery cost and the full checkpoint overhead. Thus, there are many sets of a full checkpoint followed by a certain number of incremental checkpoints. Upon a failure occurrence, the system loads the last full checkpoint and all the following incremental checkpoints to recover the application. Next, the system re-computes the lost computed work and then continues to compute the remaining work. Therefore, the recovery cost of the hybrid checkpoint mechanism is proportional to the number of incremental checkpoints between two consecutive full checkpoints. Thus, finding the optimal number of incremental checkpoints between two consecutive full checkpoints that balances the recovery cost and the total checkpoint overhead is crucial. This is because a disproportionate number of incremental checkpoints will lead to unnecessary recovery cost. Figure 3.1 illustrates the hybrid checkpoint mechanism.

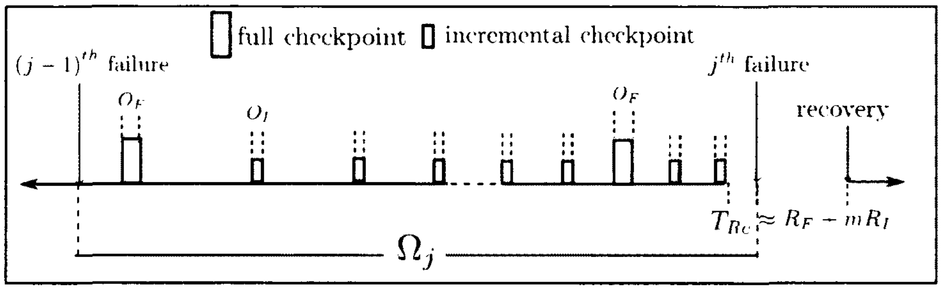

Figure 3.1: Hybrid checkpoint/restart mechanism scheme

Although the checkpoint mechanism is deployed on the systems, the systems still need to spend an amount of time to re-compute the application after a failure occurrence. We call this amount of time the "re-computing time." The amount of time the system spends to load the checkpoints is called "recovery cost." To efficiently perform the hybrid checkpoint mechanism, we derive a stochastic model to determine an optimal checkpoint frequency that minimizes the waste time. Since an application might fail more than one time, the "total waste time" is the sum of the total checkpoint overhead, the total recovery cost, and the total re-computing time. Hence, the optimality of the checkpoint frequency is contingent on the equilibrium of the three costs. The following assumptions are made to derive the checkpoint frequency function.

1. A running application may be interrupted by a series of random failures where the time-to-failure (TTF) has a certain probability density function (PDF) $f(t)$.
2. The full and incremental checkpoint overheads are constant. In practice, the full checkpoint overhead is quite stable during an application runtime because each full checkpoint is a whole application state. Moreover, programmers are likely to allocate the memory size expected for usage during the computation. Thus, the assumption is valid for the full checkpoint overhead. However, there are both small and large incremental checkpoint overheads for an application execution. The assumption of constant for incremental checkpoint overheads is still reasonable because we aim to globally minimize the total waste time. Hence, the constant incremental checkpoint overhead in the model can be the average of the actual incremental checkpoint overheads.
3. The recovery costs of full and incremental checkpoints are also constant.
4. The full checkpoint overhead is larger than the incremental checkpoint overhead. Similarly, the recovery cost of a full checkpoint is larger than that of an incremental checkpoint. This assumption is valid because a full checkpoint is a saved state of an entire application which is likely larger than an incremental checkpoint. If an incremental checkpoint is as large as a full checkpoint, the incremental checkpoint should not be performed.
5. The number of incremental checkpoint between two consecutive full checkpoints (m) is a constant.

In Section 3.1, to obtain the optimal checkpoint frequency, we first derive the waste time of the hybrid checkpoint mechanism. Then by means of calculus of variations, we derive a checkpoint frequency function that globally optimizes the expected waste time with a general failure distribution. Table 3.1 gives the notations in the derivation.

Table 3.1: Notations in the hybrid checkpoint scheduling model

Notations	Descriptions
O_F	Overhead of a full checkpoint
O_I	Overhead of an incremental checkpoint
T_{Re}	Re-computing Time
R_F	Recovery cost of a full checkpoint
R_I	Recovery cost of an incremental checkpoint
m	Number of incremental checkpoints between two consecutive full checkpoints
k	Re-computing time coefficient
Ω	Random variable of TTFs
$n(t)$	Checkpoint frequency function

3.1 An Optimal Checkpoint Frequency Function

Let Ω_j denote the random variable of TTF for the j^{th} failure, where $\Omega_0 \equiv 0$. Equivalently, Ω_j is the elapsed time between the $(j-1)^{th}$ and j^{th} failures, called the "j^{th} cycle," shown in Figure 3.1. Because the system is restarted after each failure occurrence, the Ω_j are positive independent identically distributed random variables, denoted by Ω, and $0 < E[\Omega_j] = E[\Omega] < \infty$, for all $j \in \{1, 2, 3, \ldots\}$, where $E[\cdot]$ is the expected value.

According to the first assumption, multiple failure occurrences are allowed during an application execution, so the total waste time is the aggregation of the waste time in each cycle. Let W_j be the waste time of the j^{th} cycle. The total waste time until time t, denoted by W_t, can be expressed in Eq. 3.1, where $J_t := \max\{n \in \{1,2,3,...\} | \sum_0^n \Omega_j \leq t\}$.

$$W_t = \sum_{j=0}^{J_t} W_j. \tag{3.1}$$

According to [57], J_t is a renewal process, and W_t is a renewal-reward process. The elementary renewal theorem states the following.

$$\lim_{t \to \infty} \frac{\sum_{j=1}^{J_t} W_j}{t} = \frac{E[W_1]}{E[\Omega_1]}. \tag{3.2}$$

Equation 3.2 suggests that minimizing the overall expected total waste time is equivalent to minimizing the waste time of the first cycle. It is sufficient to derive only the waste time of the first cycle. Henceforth, the waste time refers to the waste time of the first cycle. For simplicity, it is denoted by W, and it can be derived from the checkpoint frequency function $n(t)$.

The idea of the checkpoint frequency function is to formulate a function that describes the checkpoint frequency at a time instance. Then, the waste time during an elapsed time between two consecutive failures (W) is derived from the checkpoint frequency function. The checkpoint frequency function is formally defined in Definition 3.1.

Definition 3.1. *Let the sequence of discrete checkpoint times in a cycle be* $0 = t_0 < t_1 < t_2 < t_3 < ....$ *The checkpoint frequency function for the hybrid checkpoint mechanism denoted by* $n(t)$ *is a continuous function on* $[0, \infty)$, *defined by:* $\int_a^b n(t)dt :=$ *the number of full and incremental checkpoints from time* a *to time* b. *Note that* t *is the time from the starting point of a cycle.*

Because the number of checkpoints, whether full or incremental, between the i^{th} and the $(i+1)^{th}$ checkpoints is equal to 1, Proposition 3.2 follows.

Proposition 3.2. *Let t_i be the i^{th} checkpoint time, where $i \in \{0, 1, 2, ...\}$ and $t_0 = 0$. Then, we have that*

$$\int_{t_i}^{t_{i+1}} n(t)\,dt = 1. \tag{3.3}$$

According to Figure 3.1, the total number of checkpoints in the first cycle is the sum of the number of the full checkpoints and the number of the incremental checkpoints. Hence, it can be expressed in Eq. 3.4, where N_F and N_I are the numbers of full and incremental checkpoints in the first cycle, respectively.

$$\int_0^{\Omega} n(t)\,dt = N_F + N_I. \tag{3.4}$$

Because a full checkpoint is followed by m incremental checkpoints, except perhaps the last full checkpoint, we have that $N_I \approx m\,N_F$. Also, the approximation of the number of full checkpoints in the first cycle is expressed in Eq. 3.5, where m is the number of incremental checkpoints between two consecutive full checkpoints.

$$N_F \approx \frac{1}{m+1}\int_0^{\Omega} n(t)\,dt. \tag{3.5}$$

Note that the exact value of the number of full checkpoints in the first cycle can be obtained as in Eq. 3.6. However, using the exact value of N_F probably leads to unnecessarily complicated formulas that do not give better results than the approximation for large N_F.

$$N_F = \left\lceil \frac{\int_0^{\Omega} n(t)dt}{m+1} \right\rceil. \tag{3.6}$$

We recall that the waste time is the sum of the checkpoint overhead, the recovery cost, and the re-computing time. The total checkpoint overhead in the first cycle is equal to $O_F N_F + O_I N_I$. Thus, we have that

$$\begin{aligned}
\text{total checkpoint overhead in the } 1^{st} \text{ cycle} &\approx O_F\left(\frac{1}{m+1}\int_0^{\Omega} n(t)\,dt\right) \\
&+ O_I\left(\frac{m}{m+1}\int_0^{\Omega} n(t)\,dt\right). \\
\text{Total checkpoint overhead in the } 1^{st} \text{ cycle} &\approx \frac{O_F + mO_I}{m+1}\int_0^{\Omega} n(t)\,dt. \qquad (3.7)
\end{aligned}$$

Next, the re-computing time can be estimated from the checkpoint frequency function. Their relationship is illustrated in Figure 3.2. Note that Ω has a value between the two checkpoint times that a failure occurs in-between, say t_{i_0} and t_{i_0+1}. So, by the Mean Value Theorem for Integrals, we can estimate the checkpoint frequency of this interval by $n(\Omega)$ as Eq. 3.8.

$$\frac{1}{t_{i_0+1} - t_{i_0}} = \frac{\int_{t_{i_0}}^{t_{i_0+1}} n(t)dt}{t_{i_0+1} - t_{i_0}} \approx n(\Omega). \qquad (3.8)$$

The value of the re-computing time T_{Re} is in the interval $(0, t_{i_0+1} - t_{i_0})$. Therefore, from Eq. 3.8, T_{Re} can be approximated by Eq. 3.9, where k is the re-computing time coefficient between $(0, 1)$, of which the estimation will be given in Section 3.2.

$$T_{Re} = k(t_{i_0+1} - t_{i_0}) \approx \frac{k}{n(\Omega)}, \quad \textit{where} \ k \in (0, 1). \qquad (3.9)$$

The recovery cost upon a failure occurrence is estimated by $R_F + m\,R_I$, which is the upper bound of the recovery cost. Thus, the three costs that contribute to the waste time are obtained. Now, the waste time of the first cycle can be expressed as in Theorem 3.3.

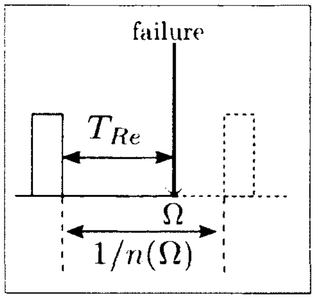

Figure 3.2: Relationship between the re-computing time and the checkpoint interval

Theorem 3.3. *Let W be the random variable of the waste time of the first cycle. Then*

$$W(\Omega) \approx \frac{O_F + mO_I}{m+1} \int_0^{\Omega} n(\tau)d\tau + \frac{k}{n(\Omega)} + (R_F + mR_I). \tag{3.10}$$

Recall that $f(t)$ is the probability density function of the TTF random variable Ω. The expected waste time of the first cycle can be expressed as follows.

$$E[W] = \int_0^{\infty} \left[\frac{O_F + mO_I}{m+1} \int_0^{t} n(\tau)d\tau + \frac{k}{n(t)} + (R_F + mR_I) \right] f(t)dt. \tag{3.11}$$

By applying the calculus of variations theory, we obtain the optimal checkpoint frequency function $n(t)$ as in Theorem 3.4.

Theorem 3.4. *Let $f(t)$ and $F(t)$ denote the probability density function (PDF) and the cumulative density function (CDF) of the TTF random variable Ω. The optimal checkpoint frequency function that minimizes the expected waste time*

$$n(t) = \sqrt{\frac{(m+1)k}{O_F + mO_I}} \sqrt{\frac{f(t)}{1 - F(t)}}. \tag{3.12}$$

Proof. First we denote $y(t) := \int_0^t n(\tau)d\tau$. Then, $y'(t) = n(t)$. Thus, Eq. 3.11 becomes;

$$E[W] = \int_0^\infty \left[\frac{O_F + mO_I}{m+1} y(t) + \frac{k}{y'(t)} + (R_F + mR_I)\right] f(t)dt. \tag{3.13}$$

Next we denote $h(y, y', t) := \left[\frac{O_F + mO_I}{m+1} y(t) + \frac{k}{y'(t)} + (R_F + mR_I)\right] f(t)$.

Thus, Eq. 3.13 becomes;

$$E[W] = \int_0^\infty h(y, y', t)dt. \tag{3.14}$$

The extremum of Eq. 3.14 must satisfy the Euler-Lagrange's equation (Eq. 3.15).

$$\frac{\partial h}{\partial y} - \frac{d}{dt}\left(\frac{\partial h}{\partial y'}\right) = 0. \tag{3.15}$$

Taking the partial derivative of h with respect to y and y', respectively, we obtain

$$\frac{\partial h}{\partial y} = \frac{O_F + mO_I}{m+1} f(t). \tag{3.16}$$

$$\frac{\partial h}{\partial y'} = -\frac{k}{(y'(t))^2} f(t). \tag{3.17}$$

By substituting Eqs. 3.16 and 3.17 into Eq. 3.15, we obtain

$$\frac{O_F + mO_I}{m+1} f(t) + \frac{d}{dt}\left(\frac{k}{(y'(t))^2} f(t)\right) = 0. \tag{3.18}$$

Integrating from 0 to t on both sides of Eq.3.18 and keeping in mind that f is a PDf, we obtain

$$\frac{O_F + mO_I}{m+1} F(t) + \frac{k}{(y'(t))^2} f(t) = C. \tag{3.19}$$

where C is a constant.

We want to obtain the unique solution of the differential equation in Eq. 3.19. Because the right-hand endpoint (∞) of the integral in Eq. 3.14 is undetermined, the function y needs to satisfy the conditions in Eqs. 3.20 and 3.21 [70].

$$y(0) = 0. \tag{3.20}$$

$$\lim_{t\to\infty} \frac{\partial h}{\partial y'} = 0. \tag{3.21}$$

It is easy to see that the function y satisfies the first condition by its definition. In addition, the function y satisfies the second condition because $\lim\limits_{t\to\infty} f(t) = 0$ and $\lim\limits_{t\to\infty} y'(t) \neq 0$. Recall that $y'(t) = n(t)$. The checkpoint frequency function does not approach 0 because eventually the systems will fail.

$$\lim_{t\to\infty} \frac{\partial h}{\partial y'} = -\lim_{t\to\infty} \frac{k}{(y'(t))^2} f(t) = 0.$$

Applying the second condition Eq. 3.21 to Eq.3.19, we obtain that $C = \dfrac{O_F + mO_I}{m+1}$ because $\lim\limits_{t\to\infty} F(t) = 1$. By an algebraic manipulation, the unique solution of the differential equation Eq. 3.19 is Eq. 3.22.

$$n(t) = y'(t) = \sqrt{\frac{(m+1)k}{O_F + mO_I}} \sqrt{\frac{f(t)}{1 - F(t)}}. \tag{3.22}$$

Hence, the proof is completed. □

3.2 Estimation of the Re-computing Time Coefficient k

According to Figure 3.2, the re-computing time coefficient k can be estimated by the ratio between the re-computing time T_{Re} and the checkpoint interval in which the failure occurs. Additionally, by the definition of the re-computing time, it is the interval between the last checkpoint and the failure. Clearly the re-computing time

T_{Re} is a random variable depending on the TTF random variable Ω. Hence, the re-computing time coefficient k is formally defined as follows.

Definition 3.5. *Let k denote the re-computing time coefficient. It can be expressed as follows.*

$$k = \frac{T_{Re}}{t_{i_0} - t_{i_0-1}} = \frac{\Omega - t_{i_0}}{t_{i_0} - t_{i_0-1}}. \tag{3.23}$$

Because k depends on the re-computing time T_{Re}, we will first find the expected value of T_{Re} for each checkpoint interval. To obtain such expected value, we need the following definition.

Definition 3.6. *Excess life is a random variable $X \geq 0$ which denotes system survival until time $t + X$, given that it survives until time t. We denote the CDF, PDF, and the expected value of X as follows.*

$$\begin{aligned} F(t+x|t) &= P(\Omega < t + x|\Omega > t). \\ f(t+x|t) &= \frac{dF(t+x|t)}{dx}. \\ E[X] &= \int_0^\infty x f(t+x|t)\,dx. \end{aligned}$$

Because each checkpoint time t_i is the time that a failure is expected to occur, the re-computing time during each checkpoint interval (t_{i-1}, t_i), T'_{Re}, is a random variable with values in the interval $(0, t_i - t_{i-1})$. According to the excess life definition, the expected re-computing time of each checkpoint interval T'_{Re} can be calculated as

$$E[T'_{Re}] = \frac{\int_0^{t_i - t_{i-1}} x f(t_{i-1} + x|t_{i-1})\,dx}{\int_0^{t_i - t_{i-1}} f(t_{i-1} + x|t_{i-1})\,dx}. \tag{3.24}$$

Therefore, for the expected k of the i^{th} checkpoint interval (t_{i-1}, t_i) denoted by $\bar{k}_i$, we obtain

$$\bar{k}_i = \frac{E[T^i_{Re}]}{t_i - t_{i-1}}. \tag{3.25}$$

Hence, the expected k denoted by $\bar{k}$ can be expressed as

$$\bar{k} = \frac{\sum_{i=1}^{N} P_i \bar{k}_i}{\sum_{i=1}^{N} P_i}, \tag{3.26}$$

where N is the number of checkpoints, and $P_i = F(t_{i-1} + x|t_{i-1})$, which is the probability that a failure occurs between the $(i-1)^{th}$ and i^{th} checkpoints.

Now the optimal checkpoint frequency function for the hybrid checkpoint mechanism has been derived. Moreover, the re-computing time coefficient is determined as an average. The only parameter in the checkpoint frequency function that has not been determined is the number of incremental checkpoints between two consecutive full checkpoints m, which plays an important role in determining the type of each checkpoint. First, in collaborative works [43, 44], the value of m has been determined based on the assumption that the failures follow an exponential distribution. However, evidently the exponential distributions are not the best fitted distribution for HPC failures [59]. Some HPC failure datasets are well fitted by a Weibull distribution or a gamma distribution. Therefore, the number of incremental checkpoints between two consecutive full checkpoints (m) should be determined for arbitrary failure distributions. Moreover, it should be optimal in the sense that it will yield the least expected waste time. Thus, in the next section, we will show that there is a unique value of m that minimizes the expected waste time for arbitrary failure distributions.

3.3 The Optimal Number of Incremental Checkpoints between Two Consecutive Full Checkpoints m for Arbitrary Failure Distributions

The hybrid checkpoint mechanism aims to reduce the checkpoint overhead from the full checkpoint mechanism. On the other hand, its recovery cost will increase proportionally to the number of subsequent incremental checkpoints because the system requires information from each and every incremental checkpoint since the last full checkpoint. Thus, the number of incremental checkpoints between two consecutive full checkpoints (m) affects the minimum of the expected waste time.

In this section, we focus on obtaining an m value that gives the global minimum of the expected waste time. First, the strict convexity of the expected waste time (Eq. 3.27) as a function of m will be proved in Lemma 3.7. Next, the existence of the minimum point will be shown, and then the uniqueness holds because of the strict convexity, see Theorem 3.8.

Lemma 3.7. *The expected waste time as a function of m is strictly convex if $m \geq 0$.*

Proof. The expected waste time function is strictly convex if its second derivative with respect to m is positive. To show that, first we substitute the optimal checkpoint frequency function (Eq. 3.12) into the expected waste time (Eq. 3.11) to obtain

$$\begin{aligned} E[W](m) =& \sqrt{\frac{(O_F + mO_I)k}{m+1}} \int_0^\infty \left(\int_0^t \sqrt{\frac{f(\tau)}{1-F(\tau)}} d\tau + \sqrt{\frac{1-F(t)}{f(t)}} \right) f(t)dt \\ & + (R_F + mR_I). \end{aligned} \tag{3.27}$$

We denote

$$D := \int_0^\infty \left(\int_0^t \sqrt{\frac{f(\tau)}{1-F(\tau)}} d\tau + \sqrt{\frac{1-F(t)}{f(t)}} \right) f(t)dt. \tag{3.28}$$

Equation 3.27 becomes

$$E[W](m) = \left(\sqrt{\frac{(O_F + mO_I)k}{m+1}}\right)D + (R_F + mR_I). \tag{3.29}$$

where $0 \leq D < \infty$.

The first and second derivatives of the expected waste time with respect to m can be expressed as in Eqs. 3.30 and 3.31.

$$\frac{\partial E[W](m)}{\partial m} = \frac{(O_I - O_F)D}{2}\sqrt{\frac{k}{(O_F + mO_I)(m+1)^3}} + R_I. \tag{3.30}$$

$$\frac{\partial^2 E[W](m)}{\partial m^2} = \frac{(O_F - O_I)(3O_F + O_I + 4mO_I)D}{4}\sqrt{\frac{k}{(O_F + mO_I)^3(m+1)^5}}. \tag{3.31}$$

Because O_F and O_I are the full and incremental checkpoint overheads, respectively, $O_F - O_I > 0$. If $m \geq 0$, then $\frac{\partial^2 E[W](m)}{\partial m^2} > 0$. Hence, the expected waste time is strictly convex. $\square$

Next, Theorem 3.8 shows that there is a unique value of $m \geq 0$ that minimizes the expected waste time.

Theorem 3.8. *The expected waste time as a function of m has a unique minimum point on* $[0, \infty)$.

Proof. We will make an argument with the first derivative of the expected waste time with respect to m. If $\frac{\partial E[W](m)}{\partial m} > 0$ for all $m \in [0, \infty)$, we have that the minimum point is at 0. Assume that 0 is not the minimum point. Then $\frac{\partial E[W](m_1)}{\partial m_1} < 0$.

Because $E[W](m)$ is strictly convex, we suppose for a contradiction that $\frac{\partial E[W](m)}{\partial m} < 0$, for all $m \geq 0$. According to Eq. 3.30, we get

$$\begin{aligned}\frac{(O_I - O_F)D}{2}\sqrt{\frac{k}{(O_F + mO_I)(m+1)^3}} + R_I < 0.&\\ \sqrt{(O_F + mO_I)(m+1)^3} < \frac{(O_F - O_I)D}{2R_I}\sqrt{k}.&\\ (O_F + mO_I)(m+1)^3 < \left(\frac{(O_F - O_I)D}{2R_I}\right)^2 k.& \qquad (3.32)\end{aligned}$$

Because the right-hand side of Eq. 3.32 is a constant, for large enough m_0, the left-hand side will be larger than or equal to the right-hand side, which is a contradiction. Therefore, there is an m_0 so that the expected waste time is decreasing on $[0, m_0)$ and increasing on (m_0, ∞). Hence, m_0 is the unique minimum point. □

We can evaluate the minimum point by finding a point that makes the first derivative disappeared, i.e., by solving the following equation. The roots of Eq. 3.33 can be obtained by using the quartic formula. However, the closed form of the solutions will not be provided here because of the tediousness of the quartic formula.

$$\sqrt{(O_F + mO_I)(m+1)^3} - \frac{(O_F - O_I)D}{2R_I}\sqrt{k} = 0. \qquad (3.33)$$

In this chapter, without any assumptions on failure distributions, the optimal checkpoint frequency function for the hybrid checkpoint mechanism is derived. Moreover, the estimation of the re-computing time coefficient, which is a representation of re-computing time, is proposed. Then, it is proved that there exists a unique number of incremental checkpoints between two consecutive full checkpoints that minimizes the expected waste time. The derivations in this chapter are for arbitrary failure distributions. To give a concrete example of the optimal checkpoint frequency function and checkpoint times. in the next chapter. we derive the checkpoint times for a Weibull distribution as it is well fitted to the failures on HPC systems. Finally. we show simulation results and compare the waste time of the hybrid checkpoint mechanism with the full checkpoint mechanism for various parameter values.

CHAPTER 4

A HYBRID CHECKPOINT SCHEDULING MODEL FOR WEIBULL DISTRIBUTIONS

In this chapter, to illustrate a concrete example of the checkpoint scheduling model for the hybrid checkpoint mechanism, the checkpoint time function for the Weibull distribution will be derived because, evidently, failures in HPC environments follow a Weibull distribution [41, 59]. In addition, for the Weibull distribution, the shape parameter is greater than 1 if the failure rate increases over time, so the number of checkpoints performed in a given time period should be increasing. On the other hand, the shape parameter is less than 1 if the failure rate decreases over time. In this case, the checkpoint frequency should be decreasing. As such, in Section 4.1, we also prove that the checkpoint intervals derived from the checkpoint time function for the Weibull distribution are inversely proportional to failure rates. Moreover, in Section 4.2, the comparisons between the waste time of the hybrid checkpoint mechanism and the full checkpoint mechanism are discussed.

4.1 Near-optimal Checkpoint Times for Weibull Distributions

To derive the checkpoint time function for the Weibull distribution, we first recall the PDF (Eq. 4.1) and CDF (Eq. 4.2)of the Weibull distribution with shape parameter β and scale parameter α, respectively.

$$f_{Weibull}(t) = \frac{\beta}{\alpha}\left(\frac{t}{\alpha}\right)^{\beta-1} e^{-(t/\alpha)^{\beta}}. \tag{4.1}$$

$$F_{Weibull}(t) = 1 - e^{-(t/\alpha)^{\beta}}. \tag{4.2}$$

By substituting the PDF and CDF of the Weibull distribution into Eq. 3.22, the optimal checkpoint frequency function for the Weibull distribution is obtained in Eq. 4.3.

$$n(t) = \sqrt{\frac{(m+1)k}{O_F + mO_I}}\left(\frac{t}{\alpha}\right)^{\frac{\beta-1}{2}}\sqrt{\frac{\beta}{\alpha}}. \tag{4.3}$$

From the optimal checkpoint frequency function in Eq. 4.3, we obtain the optimal checkpoint times that minimize the expected waste time by using Proposition 3.2 in Theorem 4.1.

Theorem 4.1. *Let t_i be the i^{th} checkpoint time of a full or incremental checkpoint for $i = 1, 2, \ldots$. If the failures follow the Weibull distribution with shape parameter β and scale parameter α, then t_i can be expressed as Eq. 4.4.*

$$t_i = \left(i\frac{\beta+1}{2A}\right)^{\frac{2}{\beta+1}}, \tag{4.4}$$

where $A = \sqrt{\frac{(m+1)k}{O_F + mO_I}}\left(\frac{1}{\alpha}\right)^{\frac{\beta-1}{2}}\sqrt{\frac{\beta}{\alpha}}$.

Proof. By Proposition 3.2 of the checkpoint frequency function $n(t)$,

$$\begin{aligned} 1 &= \int_{t_i}^{t_{i+1}} n(t)dt \\ &= \int_{t_i}^{t_{i+1}} A \cdot t^{\frac{\beta-1}{2}} dt \\ &= \frac{2A}{\beta+1}\left(t_i^{\frac{\beta+1}{2}} - t_{i-1}^{\frac{\beta+1}{2}}\right). \end{aligned}$$

Hence,

$$t_i = \left(\frac{\beta+1}{2A} + t_{i-1}^{\frac{\beta+1}{2}}\right)^{\frac{2}{\beta+1}}. \tag{4.5}$$

For $i = 1, 2, ...,$ Eq. 4.4 is obtained by an induction procedure and using the fact that $t_0 = 0$ to prove the base case, where $i = 1$. □

Next we will show that the length of the checkpoint intervals $[t_{i-1}, t_i)$ derived from Eq. 4.4 decreases if the system is aging ($\beta > 1$), and that it increases if the system is becoming more reliable ($\beta < 1$).

Theorem 4.2. *Let $I(i)$ be the width of the interval $[t_{i-1}, t_i)$. $I(i)$ is decreasing if the shape parameter β is greater than 1, and it is increasing if β is less than 1.*

Proof. According to Eq. 4.4, we obtain the following.

$$I(i) = t_i - t_{i-1}.$$

$$I(i) = \left(i\frac{\beta+1}{2A}\right)^{\frac{2}{\beta+1}} - \left((i-1)\frac{\beta+1}{2A}\right)^{\frac{2}{\beta+1}}$$

$$= \left(i^{\frac{2}{\beta+1}} - (i-1)^{\frac{2}{\beta+1}}\right)\left(\frac{\beta+1}{2A}\right)^{\frac{2}{\beta+1}}. \tag{4.6}$$

$$\text{where } A = \sqrt{\frac{(m+1)k}{O_F + mO_I}}\left(\frac{1}{\alpha}\right)^{\frac{\beta-1}{2}}\sqrt{\frac{\beta}{\alpha}} > 0. \tag{4.7}$$

The first derivative of $I(i)$ can be expressed as follows.

$$\frac{d}{di}I(i) = \left(\frac{\beta+1}{2A}\right)^{\frac{2}{\beta+1}}\frac{d}{di}\left(i^{\frac{2}{\beta+1}} - (i-1)^{\frac{2}{\beta+1}}\right)$$

$$= \left(\frac{\beta+1}{2A}\right)^{\frac{2}{\beta+1}}\frac{2}{\beta+1}\left[i^{\frac{2}{\beta+1}-1} - (i-1)^{\frac{2}{\beta+1}-1}\right].$$

Hence,

$$I'(i) = \left(\frac{\beta+1}{2A}\right)^{\frac{2}{\beta+1}}\frac{2}{\beta+1}\left[i^{\frac{1-\beta}{\beta+1}} - (i-1)^{\frac{1-\beta}{\beta+1}}\right]. \tag{4.8}$$

We can see that if $\beta > 1$, $I'(i) < 0$, and if $\beta < 1$, $I'(i) > 0$. Also, for $\beta = 1$, i.e., failures follow the exponential distribution with rate $\frac{1}{\alpha}$, the length of the checkpoint intervals is constant. $\square$

We note that the parameters k and m in Eq. 4.4 can be obtained from Eq. 3.26 and Eq. 3.33, respectively. Because k and m are related, in practice, we have to calculate both values at the same time. Algorithm 4.1 is an algorithm based on the fixed point approach to estimate the values of k and m.

Algorithm 4.1 Algorithm to calculate the re-computing time coefficient k and the number of incremental checkpoints m

```
Require: $O_F$, $O_I$, $R_I$, and $Threshold$
Ensure: $k$ and $m$
 1: $\hat{k} \leftarrow 0.5$
 2: //Find $\hat{m}$ corresponding to $\hat{k}$
 3: Calculate $\hat{m}$ by solving Eq. 3.33
 4: //Finish finding $\hat{m}$ corresponding to $\hat{k}$
 5: Calculate the checkpoint time sequence $t_1, t_2, ..., t_N$ corresponding to $\hat{k}$ and $\hat{m}$
    by Eq. 4.4
 6: Calculate $\bar{k}$ from Eqs. 3.24-3.26
 7: if $|\bar{k} - \hat{k}| \leq Threshold$ then
 8:    $k \leftarrow \bar{k}$
 9:    Done
10: else
11:    $\hat{k} \leftarrow \bar{k}$
12:    repeat line 3
13: end if
```

4.2 Comparisons between the Hybrid and the Full Checkpoint Mechanisms for Weibull Distributions

We have conducted experiments to compare the performances of the hybrid checkpoint and full checkpoint mechanisms on systems with Weibull distributed failures. However, the experiments in this section are not based on the failure information of actual HPC systems. We want to study the performance of the hybrid checkpoint model in different failure behaviors. so in the experiments we take various values of mean-time-to-failure (MTTF) and shape parameters of Weibull distributions. By varying values of the parameters in the models, we can study the models in various perspectives instead of narrowing the study scope to particular systems.

On a fully occupied system. the time duration from the system start to the present time is called “the system running time.” In the experiments, we want to study that how much time the system spends to perform checkpoints, to recover applications, and to re-compute applications for a certain system running time. In the experiments, the shape parameters of Weibull distributions are 0.5, 1, and 1.5 which represent decreasing. constant, and increasing failure rates, respectively. We take values of MTTFs to be 3 hours. 1. 3. 5. and 7 days. Also, we vary the values of the system running times to be 5 hours. 1. 3. and 7 days. In the experiments. the full checkpoint overheads are 5. 10. 30. and 60 minutes. and the incremental checkpoint overheads are 10%. 30%. 50%. and 70% of the full checkpoint overheads. We assume that the recovery cost of a full checkpoint is equal to the full checkpoint overhead. Similarly. the recovery cost of an incremental checkpoint is equal to the incremental checkpoint overhead. Therefore, the recovery costs of a full checkpoint or an incremental checkpoint are varied with the checkpoint overhead.

The equality between the full checkpoint overhead and the recovery cost of a full checkpoint is assumed because the recovery cost of a full checkpoint does not affect to the checkpoint frequency function, according to Eq. 3.22. The recovery cost contributes to the waste time exclusively when a failure occurs. Upon a failure, the system always load only one full checkpoint, excluding the following incremental checkpoints. Therefore, the recovery cost of a full checkpoint is a price to pay regardless of the checkpoint frequency. The assumption is also made for the incremental checkpoint because the recovery cost of an incremental checkpoint does not directly affect the checkpoint frequency function, similar to the recovery cost of a full checkpoint. However, the number of incremental checkpoints between two consecutive full checkpoints (m) is inversely proportional to the recovery cost of an incremental checkpoint. The larger the recovery cost of an incremental checkpoint is, the less incremental checkpoints should be scheduled between two consecutive full checkpoints. Otherwise, upon a failure, the system would spend a large amount of time to load a number of incremental checkpoints. The sensitivity of the number of incremental checkpoints between two consecutive full checkpoints (m) is discussed in Section 4.2.1. Moreover, Table 4.1 lists all parameter values used in the simulations.

Therefore, there are 960 cases of distinct combinations of parameter values. To have a normally distributed population, we have generated 50 sets of 200 TTFs for each pair of MTTF and shape parameter. Hence, in total, we have 48000 simulations for the hybrid checkpoint mechanism and 12000 simulations for the full checkpoint mechanism. Note that the number of simulations for the full checkpoint model is less than that of the hybrid checkpoint model because there are no incremental checkpoints in the full checkpoint mechanism, i.e., $m = 0$. For each simulation, we simulate the waste time that the system spends to perform checkpoints, to recover applications, and to re-compute applications.

Table 4.1: Parameter values in simulations

Parameter	Values
system running time	5 hours, 1, 3 and 7 days
$MTTF$	3 hours, 1, 3, 5 and 7 days
Shape parameter (β)	0.5, 1 and 1.5
Full checkpoint overhead (O_F)	5, 10, 30 and 60 minutes
Incremental checkpoint overhead (O_I)	10%, 30%, 50% and 70% of O_F

4.2.1 Discussion on the Number of Incremental Checkpoints m

This section focuses on the sensitivity study of the number of incremental checkpoints between two consecutive full checkpoints m. Figure 4.1 illustrates the averages of the number of incremental checkpoints m when MTTF is 3 hours, and the failure rate is decreasing (left), constant (middle), and increasing (right). According to Figure 4.1, obviously the number of incremental checkpoints m is proportional to the values of the checkpoint overheads. When the incremental checkpoint overheads are 50% or 70% of the full checkpoint overheads, the hybrid checkpoint model often does not schedule any incremental checkpoint, i.e. $m = 0$. We can conclude that the hybrid checkpoint mechanism should be considered over the full checkpoint mechanism if the incremental checkpoint overhead is less than 50% of the full checkpoint overhead.

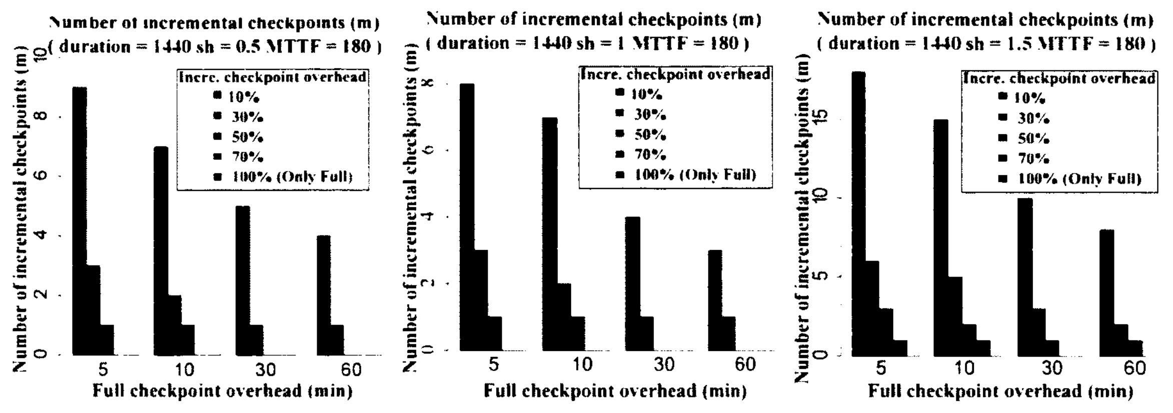

Figure 4.1: Averages of number of incremental checkpoints between two consecutive full checkpoints (m) when MTTF is 3 hours, and the shape parameters are 0.5 (left), 1 (middle), and 1.5 (right)

Moreover, Figure 4.2 illustrates the averages of the number of incremental checkpoints m when the MTTF is 1 day, which is longer than the MTTF in Figure 4.1. According to Figure 4.2, the number of incremental checkpoints m increases with the MTTF. MTTF is inversely proportional to the failure rate, so a large MTTF indicates that the chance of failure occurrences is small. This means that the chance that the system needs to load the checkpoints is small. Consequently, the number of incremental checkpoints m is larger because the total recovery cost is proportional to the value of m. In contrast, m should be small if the chance of failures is high.

4.2.2 Discussion on Waste Time

This section discusses the simulation results regarding the waste times. Figure 4.3 illustrates the graphs of the percentages of waste time of both hybrid checkpoint and full checkpoint mechanisms for decreasing (left), constant (middle), and increasing (right) failure rates when the system running time and the MTTF are 1 day.

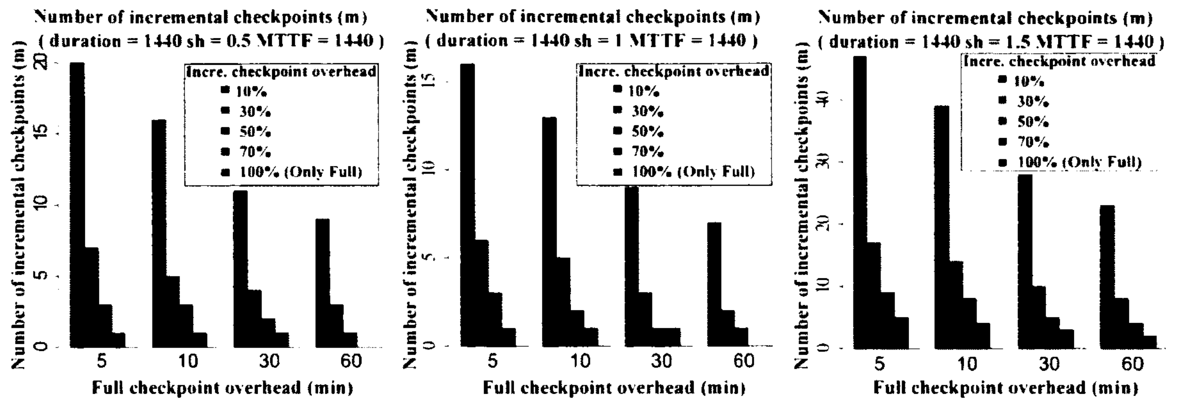

Figure 4.2: Averages of the number of incremental checkpoints between two consecutive full checkpoints (m) when MTTF is 1 day, and the shape parameters are 0.5 (left), 1 (middle), and 1.5 (right)

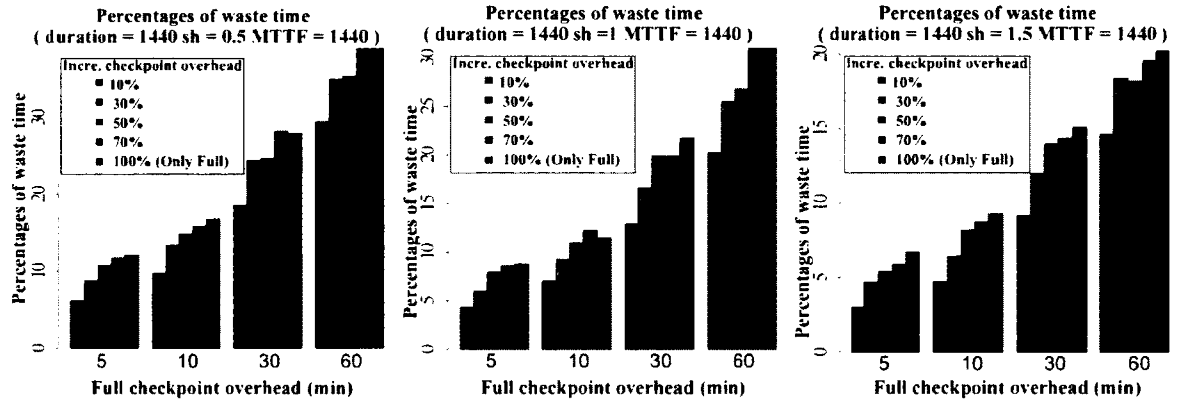

Figure 4.3: Percentages of the waste time when the system running time is 1 day, MTTF is 1 day, and the shape parameters are 0.5 (left), 1 (middle), and 1.5 (right)

According to Figures 4.3, the waste times of the hybrid checkpoint mechanism are mostly smaller than or equal to those of the full checkpoint mechanism for all full and incremental checkpoint overheads. Specifically, if the incremental checkpoint overhead is 70% of the full checkpoint overhead of 1 hour, the hybrid checkpoint model does not schedule any incremental checkpoints, showed in Figure 4.2, except for the increasing failure rate. Consequently, the waste times of the hybrid checkpoint mechanism when the incremental checkpoint overhead is 70% of the full checkpoint overhead is equal to that of the full checkpoint mechanism.

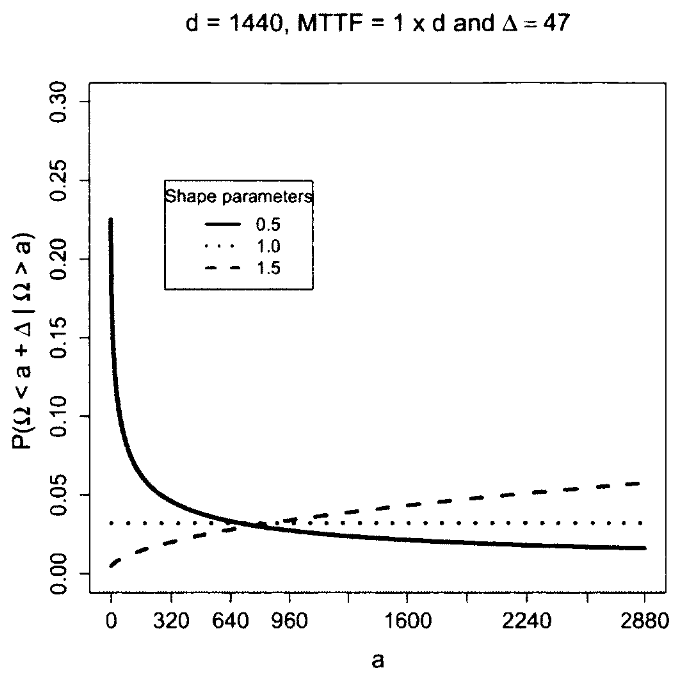

Figure 4.4: Conditional probability of a failure occurrence before time $a + \Delta$, given that the system survives until time a, where Δ is the checkpoint interval when the MTTF is 1 day, the shape parameter is 1, the full checkpoint overhead O_F is 5 minutes, and the incremental checkpoint overhead O_I is 10% of O_F

Furthermore, the percentage of waste time is highest when the failure rate is decreasing and it is lowest when the failure rate is increasing. This is because, for a decreasing failure rate, the probability of a failure occurrence in the beginning of the execution is very high, comparing to the constant and increasing failure rates with the same MTTF, shown in Figure 4.4. Therefore, the re-computing time is the major cause of the large amount of waste times in the decreasing failure rate cases. The more details for the re-computing time will be discussed in Section 4.2.3. Moreover, the waste times are closed to each other among the three values of the shape parameter when observing the waste time of a long system running time, shown in Figure 4.5.

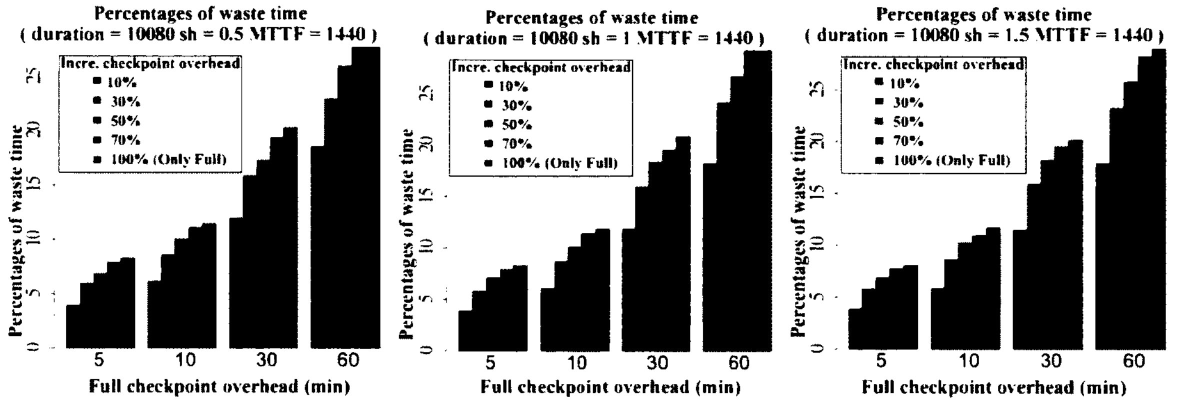

Figure 4.5: Percentages of the waste time when the system running time is 7 days. MTTF is 1 day, and the shape parameters are 0.5 (left), 1 (middle), and 1.5 (right)

Considering a long system running time, Figure 4.5 illustrates the graphs of the percentages of the waste time of both hybrid checkpoint and full checkpoint mechanisms for system running time of 7 days. The percentages of waste times in Figure 4.5 are very close for different failure rates because the hybrid checkpoint model aims to globally optimize the waste time. Hence, the longer the system running time is, the smaller the differences of waste time among failure rates are. Next, we will discuss the checkpoint overhead, the recovery cost, and the re-computing time that are aggregated into the waste time.

4.2.3 Discussion on Re-computing Time, Checkpoint Overhead, and Recovery Cost

In this section, the behavior of the re-computing time, the total checkpoint overhead, and the recovery cost when the system running time and the MTTF are

1 day will be discussed. Figure 4.6 illustrates the graphs of the percentages of the re-computing time of both hybrid checkpoint and full checkpoint mechanisms for decreasing (left), constant (middle), and increasing (right) failure rates. Similar to the waste time, the re-computing time is small if the checkpoint overhead is small. This is because, according to Figure 4.7, more checkpoints whether full or incremental checkpoints are performed if the checkpoint overhead is small, comparing to the number of checkpoints when the checkpoint overhead is large.

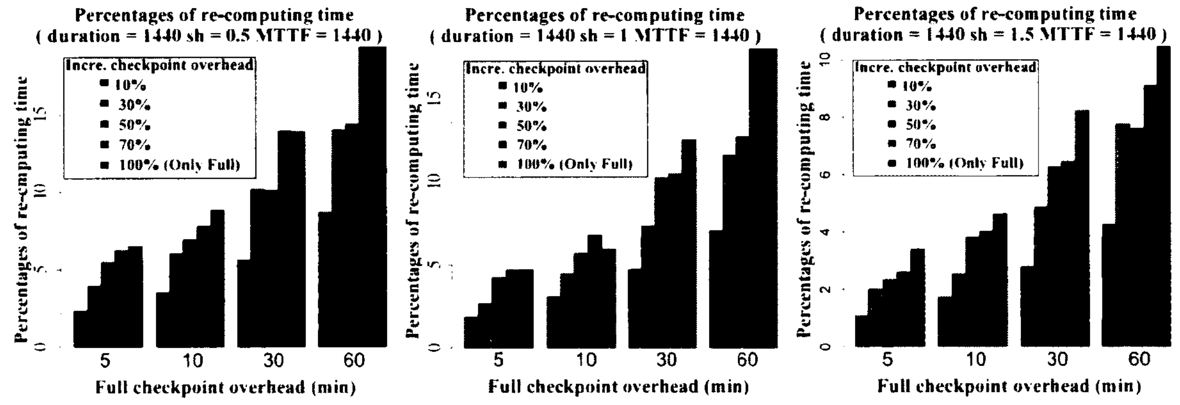

Figure 4.6: Percentages of the re-computing time when the system running time is 1 day, MTTF is 1 day, and the shape parameters are 0.5 (left), 1 (middle), and 1.5 (right)

Figure 4.8 illustrates the graphs of the percentages of the total checkpoint overheads of both hybrid checkpoint and full checkpoint mechanisms for decreasing (left), constant (middle), and increasing (right) failure rates. Obviously, the total checkpoint overhead is large if each checkpoint overhead is large. However, there are some cases that the total overheads of small incremental checkpoint overheads are larger than those of larger incremental checkpoint overhead, such as the cases of the increasing failure rate (right) and the full checkpoint overhead of 60 minutes. This is because more checkpoints are performed within the same duration if the overhead is relatively small, resulting in a possibility that the total overhead of the relatively small incremental overhead is larger than that of the relatively large incremental overhead.

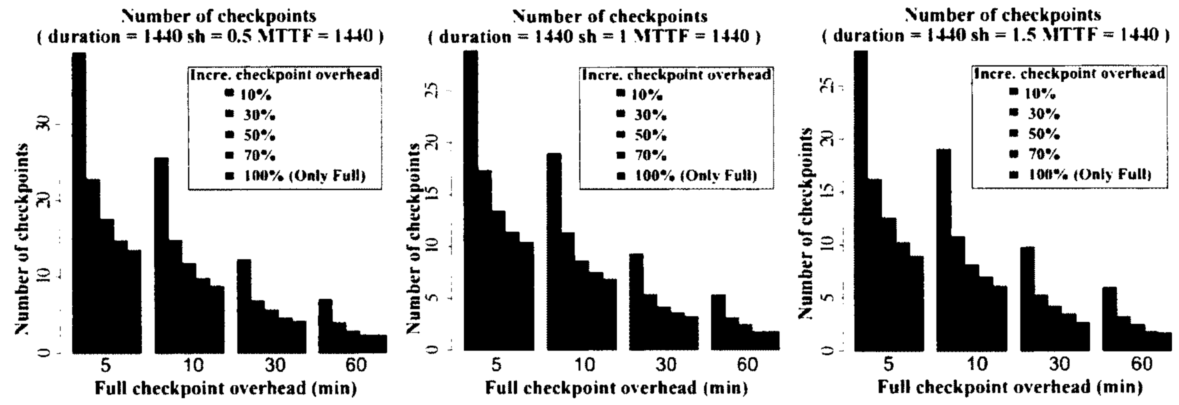

Figure 4.7: Averages of the number of checkpoints when the system running time is 1 day, MTTF is 1 day, and the shape parameters are 0.5 (left), 1 (middle), and 1.5 (right)

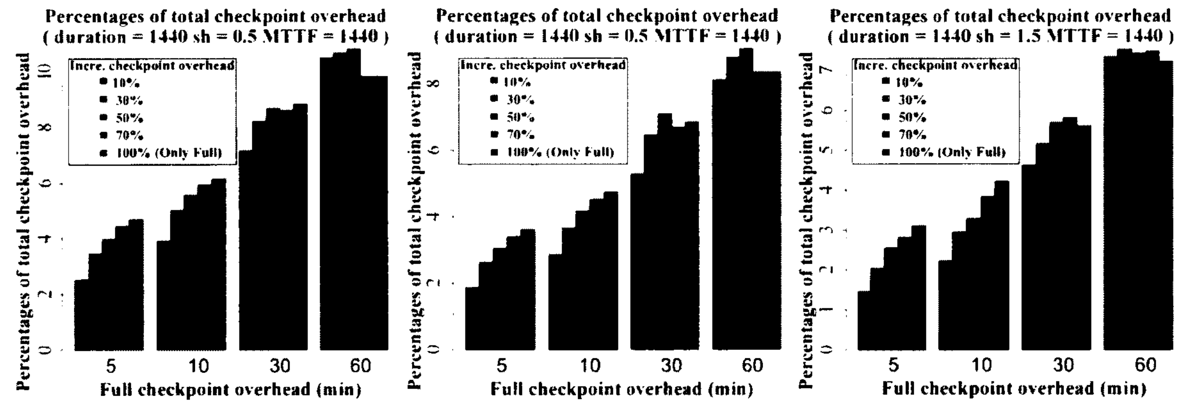

Figure 4.8: Percentages of the checkpoint overheads when the system running time is 1 day, MTTF is 1 day, and the shape parameters are 0.5 (left), 1 (middle), and 1.5 (right)

Figure 4.9 illustrates the graphs of the percentages of the recovery cost of both hybrid checkpoint and full checkpoint mechanisms for decreasing (left), constant (middle), and increasing (right) failure rates. According to all graphs in Figure 4.9, the recovery costs of the hybrid checkpoint mechanism are slightly larger than those of the full checkpoint mechanism because, upon a failure occurrence, the system must load all incremental checkpoints that follow the last full checkpoint.

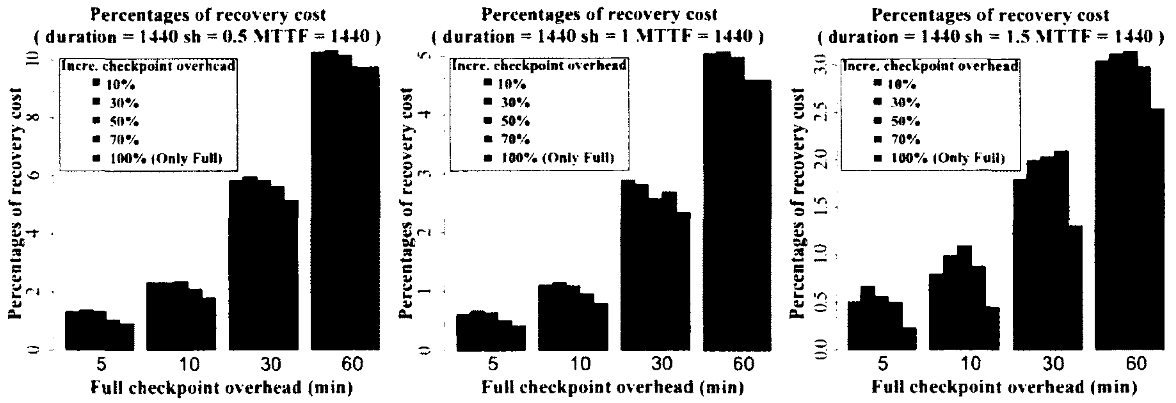

Figure 4.9: Percentages of the recovery cost when the system running time is 1 day, MTTF is 1 day, and the shape parameters are 0.5 (left), 1 (middle), and 1.5 (right)

We conclude that the hybrid checkpoint mechanism does not always give less waste time than the full checkpoint mechanism. However, if the incremental checkpoint overhead is less than 50% of the full checkpoint overhead, the hybrid checkpoint mechanism is preferable. Also, the recovery cost of the hybrid checkpoint mechanism is larger than that of the full checkpoint mechanism, but both the re-computing time and the total checkpoint overhead of the hybrid checkpoint mechanism are likely to smaller than those of the full checkpoint mechanism, resulting in smaller waste time.

CHAPTER 5

A DECISION-MAKING MODEL FOR REACTIVE AND PROACTIVE FT MECHANISMS

In this chapter, instead of focusing on only the checkpoint/restart mechanism, the combination of reactive and proactive FT mechanisms will be considered. Deploying the checkpoint/restart mechanism introduces additional overhead to the execution time. Especially in a high failure rate environment, the checkpoint mechanism needs to be invoked more frequently to cope with the failure occurrences. Therefore, proactive FT mechanisms should be considered to prevent the application from failure. Moreover, we expect that deploying a proactive FT mechanism together with the checkpoint/restart mechanism will reduce the total waste time.

Therefore, in this chapter, we will derive a novel decision-making model that determines the best choice among all considered FT mechanisms at each decision point to obtain the least application execution time. The "application execution time" is the duration of time from the start of the execution until the end, at which we obtain the results of the application. The "application completion time" is the duration of time that the system spends to compute the application in a failure-free environment. Also, the "waste time" is the duration of time that the system spends to perform FT mechanisms, to recover the application, and to re-compute the application upon a failure occurrence. Therefore, the execution time is the sum of the completion time and the waste time. Furthermore, we study the benefits of deploying both reactive and proactive FT mechanisms in running applications based on the proposed decision-making model. To provide a concrete decision-making model, we choose the checkpoint/restart mechanisms and rejuvenation as examples of reactive and proactive FT mechanisms, respectively. Specifically, two checkpoint techniques are considered in this work, namely full checkpoints and incremental checkpoints.

5.1 An Adaptive Decision-making Model

For each application and a given constant interval length, the proposed decision-making model determines the best strategy to perform an FT mechanism at the end of each interval. Each interval of constant length is called a "decision interval," and the end of each decision interval is called a "decision point." For a decision interval of size Δ, we chunk the application completion time into N intervals of size Δ, except that the last interval can be shorter than Δ. In addition, we call these N decision intervals "an N-stage process" in Figure 5.1 a).

Figure 5.1 a) illustrates the scenario that there is no failure occurrence, and no FT mechanism is performed during the application execution. Therefore, the application completion time and the application execution time are equal. Next, Figure 5.1 b) illustrates the scenario that an FT mechanism is performed at each decision point, but there is no failure occurrence. In this scenario, the execution time is the accumulation of the completion time and the total overhead of FT mechanisms. Lastly, Figure 5.1 c) illustrates the scenario that an FT mechanism is performed at each decision point and a failure occurs during the application execution. Consequently, the execution time of this scenario is the aggregation of the completion time and the total waste time, consisting of the total FT mechanism overhead, the recovery cost, and the re-computing time. "The recovery cost" is the time duration in which the system recovers the application, and "the re-computing time" is the time duration in which the system re-computes the lost work due to failure occurrences.

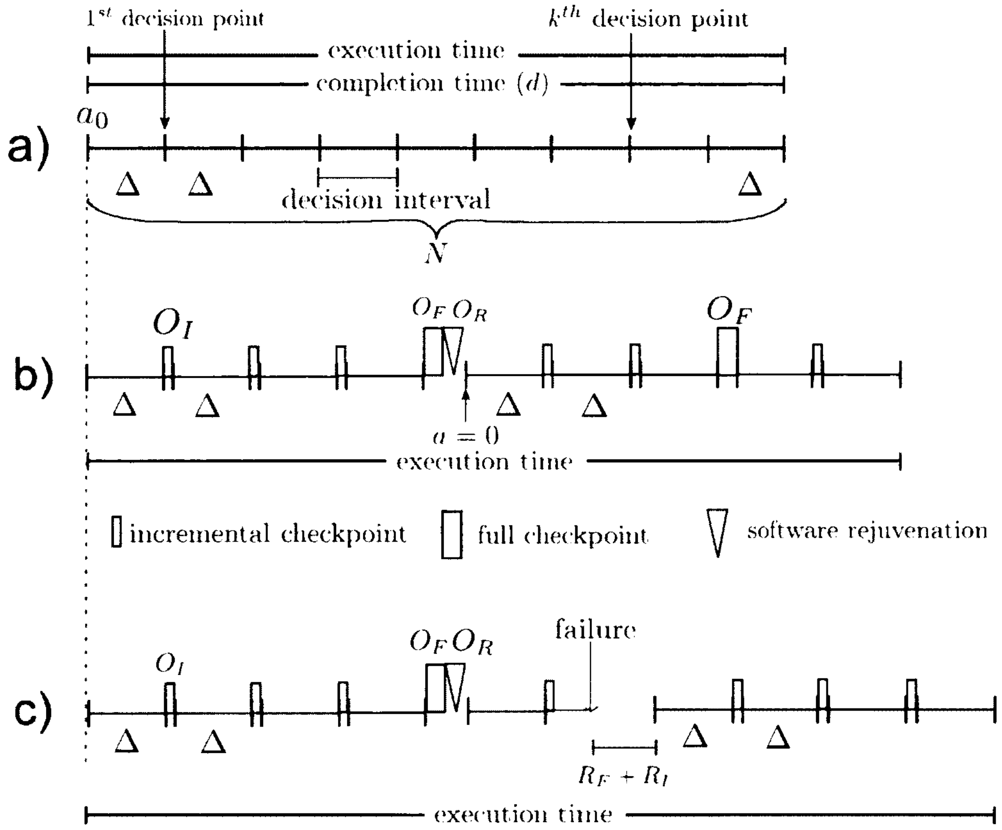

Figure 5.1: Three scenarios of the decision-making model of an $N-$ stage process with the decision interval of length Δ: a) there is no failure occurrence, and no FT mechanism is performed during the application execution; b) an FT mechanism is performed at each decision point, but there is no failure occurrence; and c) an FT mechanism is performed at each decision point and a failure occurs during the application execution

At the k^{th} decision point, the model compares the expected waste time (from the k^{th} decision point until the completion of the application) of each considered FT mechanism. Then, it chooses the FT mechanism that gives the least expected waste time. Therefore, the FT mechanisms can be different at each decision point. In general, at each decision point, there are three possible cases. First, there is a failure before the FT mechanism completes, so a portion of the computed work is lost. Thus, the system needs to recover the application and then resumes the execution. Second, the FT mechanism completes, but a failure occurs before the next decision point. This case is similar to the first case, but the amount of the lost work may be different, and the system has spent an amount of time to perform the FT mechanism before the failure occurrence. Third, the FT mechanism completes, and no failures occur before the next decision point. The last case differs from the second case in that there is no lost work and so no recovery cost.

Given that there is no failure before the k^{th} decision point, P_{FT}^{k} is the probability that a failure occurs before the FT mechanism completes. Similarly, P_k is the probability that a failure occurs before the $(k+1)^{th}$ decision point, given that the FT mechanism at the k^{th} decision point is complete. Therefore, for each FT mechanism, the expected waste time from the k^{th} decision point on of an $N-$stage process $(EW[FT]_k^N)$ can be expressed in Eq. 5.1.

$$\begin{aligned}
EW[FT]_k^N = P_{FT}^{k} & \left[\text{recovery cost} + \text{re-computing time} + EW[FT]_0^{N-k+1}\right] \\
& + (1 - P_{FT}^{k})\Big[\text{overhead of FT} + (1 - P_k) EW[FT]_{k+1}^{N} \\
& \quad + P_k \left(\text{recovery cost} + \text{re-computing time} + EW[FT]_0^{N-k}\right)\Big] \qquad (5.1)
\end{aligned}$$

The expected waste time of each FT mechanism varies based on its nature. For example, the recovery cost of the incremental checkpoint mechanism is the time duration in which the system loads all incremental checkpoints. In contrast, the recovery cost of the full checkpoint mechanism is the time duration in which the system loads only the last full checkpoint. The least expected waste time and the best action at the k^{th} decision point are formally defined in Definition 5.1.

Definition 5.1. *Let f_k^N be the least expected waste time of an N-stage process from the decision point k on, where $k = 1, 2, 3, \ldots$. The function f_k^N is expressed in Eq. 5.2. The best choice at the k^{th} decision point is the FT mechanism that gives the least expected waste time, expressed in Eq. 5.3, where $\underset{x}{\operatorname{argmin}}\{f(x)\}$ are values of x for which $f(x)$ attains the smallest value.*

$$f_k^N := \min\{EW_k^N[FT] \mid FT \textit{ is an } FT \textit{ mechanism}\}. \tag{5.2}$$

$$\textit{The best choice} := \underset{FT}{\operatorname{argmin}}\{EW_k^N[FT]\}. \tag{5.3}$$

To show an example of a concrete decision-making model, we derive the decision-making model for checkpoint/restart mechanisms and rejuvenation as examples of reactive and proactive FT mechanisms, respectively. We discuss the scheme and give the analytical decision-making model in Section 5.2.

5.2 A Decision-making Model for Full Checkpoints, Incremental Checkpoints, and Rejuvenation

At each decision point, the model selects an FT mechanism (full checkpoint, incremental checkpoint, or rejuvenation) that results in the least expected waste time of the application, given its completion time. Upon a failure occurrence, the application is recovered from the last checkpoint. If the last checkpoint is an incremental checkpoint (Figure 5.1 c)), the last full checkpoint and all incremental checkpoints following it are loaded. Then, the system re-computes the lost work and continues to compute the rest of the work. If a failure occurs during the re-computing period, the same process is performed.

During an application execution, one full checkpoint should be performed right before each rejuvenation to protect the computed work. Otherwise, the computed work will be lost and unrecoverable. Henceforth, the rejuvenation action refers to a full checkpoint, followed by a rejuvenation. After a failure occurrence or a rejuvenation, we assume that the software has a fresh state. To derive the decision-making model, we assume the following conditions. Moreover, all notations in the model are listed in Table 5.1.

1. The overhead of each FT mechanism is constant throughout the application execution. In practice, we expect that the full checkpoint overhead insignificantly changes for a particular application. This expectation might not be true for the incremental checkpoint overhead, but the average of the incremental checkpoint overhead could be used in the model. Lastly, the time taken to rejuvenate the software state depends on the system architecture and the number of nodes that the application is running on, so the assumption well represents the practical rejuvenation overhead.
2. The recovery time of each full and incremental checkpoint is also constant.
3. The system is able to recover the application from the last complete checkpoint.

4. A failure might occur during performing an FT mechanism. However, if it occurs during a rejuvenation, we ignore the additional overhead in the model because the system can recover the application from the full checkpoint completed before the failed rejuvenation.

Table 5.1: Notations in the decision-making model

Notation	Descriptions
O_F	Overhead of a full checkpoint
O_I	Overhead of an incremental checkpoint
O_R	Overhead of a rejuvenation
R_F	Recovery cost of a full checkpoint
R_I	Recovery cost of an incremental checkpoint
d	Application completion time in failure-free environment
Δ	Length of a decision interval
m_F	Number of full checkpoints loaded upon a failure
m_I	Number of incremental checkpoints loaded upon a failure
a_0	Age of the software state at the beginning of the execution
a	Age of the software state
P_{FT}^k	The probability that the FT mechanism (FT) fails at the k^{th} decision point, given that the system survives until the k^{th} decision point
P_k	The probability of a failure occurrence during the k^{th} decision interval, given that the system survives and completes the FT mechanism at the k^{th} decision point

The expected total waste time is the expected waste time at decision point 0. At the starting point of the execution, no FT mechanism will be performed, so we denote the expected total waste time of an N-stage process by EW^N. In an $N-$stage process, the expected total waste time is the expected waste time from the decision point 0 on, denoted by $EW^N(0, 0, a_0)$. The first parameter is the number of full checkpoints that has to be loaded upon a failure occurrence (m_F); the second parameter is the number of incremental checkpoints that has to be loaded upon a failure occurrence (m_I); and the third parameter is the age of the software state at the decision point 0 (a_0). Because there is no FT mechanism performed at the decision point 0, we have that $m_F = m_I = 0$ at that point.

Moreover, there are two possible cases at the beginning of the execution (decision point 0). First, a failure occurs before the 1^{st} decision point, so the work that is computed from the beginning is lost, and the system needs to re-compute the work from scratch. Second, there is no failure before the 1^{st} decision point. Hence, the expected total waste time of an $N-$stage process ($EW^N(0,0,a_0)$) can be expressed in Eq. 5.4, where P_0 is the probability that a failure occurs before the 1^{st} decision point, given that the system software survives until time a_0, and $EW_1^N(0,0,a_0+\Delta)$ is the expected waste time from the 1^{st} decision point on. Note that $\mathbf{0} = (0,0,0)$.

$$EW^N(0,0,a_0) = P_0(\Delta + EW^N(\mathbf{0})) + (1-P_0)EW_1^N(0,0,a_0+\Delta). \tag{5.4}$$

We denote the expected waste times from the k^{th} decision point on of the full checkpoints, the incremental checkpoints, and the rejuvenation by $EW[full]_k^N(m_F,m_I,a)$, $EW[incre]_k^N(m_F,m_I,a)$, and $EW[rej]_k^N(m_F,m_I,a)$, respectively. Specifically, the expected waste time is the expected waste time from the k^{th} decision point in an $N-$stage process until the application completes. Assuming that at least one full checkpoint or a rejuvenation has been performed before the k^{th} decision point, upon a failure occurrence, only the last full checkpoint needs to be loaded to recover the application, then $m_F = 1$. In contrast, if there is no full checkpoint or no rejuvenation performed before the k^{th} decision point, $m_F = 0$. m_I is equal to the number of the incremental checkpoints following the last full checkpoint.

Moreover, after a rejuvenation, the age of the software state is reset to 0 because rejuvenating leads to a fresh software state.

According to the assumptions and Eq. 5.1, at the k^{th} decision point, we can express the expected waste times of the full checkpoint ($EW[full]_k^N$), the incremental checkpoint ($EW[incre]_k^N$), and the rejuvenation ($EW[rej]_k^N$) as Eqs. 5.5, 5.6, and 5.7, respectively. Additionally, we estimate the re-computing time by the length of the decision interval Δ, so the expected waste time of each FT mechanism is an upper bound for the actual waste time.

$$\begin{aligned} EW[full]_k^N(m_F, m_I, a) =& P_{full}^k(a)\Big[m_F R_F + m_I R_I + \Delta + EW^{N-k+1}(\mathbf{0})\Big] \\ &+ (1 - P_{full}^k(a))\Big[O_F + P_k(a)\left(R_F + \Delta + EW^{N-k}(\mathbf{0})\right) \\ &+ (1 - P_k(a))\ EW[FT]_{k+1}^N(1, 0, a + O_F + \Delta)\Big]. \end{aligned} \quad (5.5)$$

$$\begin{aligned} EW[incre]_k^N(m_F, m_I, a) =& P_{incre}^k(a)\Big[m_F R_F + m_I R_I + \Delta + EW^{N-k+1}(\mathbf{0})\Big] \\ &+ (1 - P_{incre}^k(a))\Big[O_I \\ &+ P_k(a)\big(m_F R_F + (m_I + 1)R_I + \Delta + EW^{N-k}(\mathbf{0})\big) \\ &+ (1 - P_k(a))\ EW[FT]_{k+1}^N(m_F, m_I + 1, a + O_I + \Delta)\Big]. \end{aligned} \quad (5.6)$$

$$\begin{aligned} EW[rej]_k^N(m_F, m_I, a) =& P_{rej}^k(a)\Big[m_F R_F + m_I R_I + \Delta + EW^{N-k+1}(\mathbf{0})\Big] \\ &+ (1 - P_{rej}^k(a))\Big[O_F + O_R + P_k(0)\left(R_F + \Delta + EW^{N-k}(\mathbf{0})\right) \\ &+ (1 - P_k(0))EW[FT]_{k+1}^N(1, 0, \Delta)\Big]. \end{aligned} \quad (5.7)$$

Moreover, $EW^{N-k+1}(\mathbf{0})$ is the expected total waste time of the remaining work if a failure occurs before the checkpoint completes; $EW^{N-k}(\mathbf{0})$ is the expected waste time of the remaining work if a failure occurs before the $(k+1)^{th}$ decision point; and $EW[FT]_{k+1}^{N}$ is the expected waste time in the future, i.e., the expected waste time from the $(k+1)^{th}$ decision point on. Furthermore, the expected waste time at the N^{th} decision point is equal to zero because the application has completed. To obtain the optimal policy, we minimize $EW^{N}(0, 0, a_0)$ by means of dynamic programming.

For an $N-$ stage process, the idea is as follows. At the $(N-1)^{th}$ decision point, suppose that all decisions before the $(N-1)^{th}$ decision point are known. Then the expected waste time of each FT mechanism $(EW[FT]_{N-1}^{N})$ can be calculated because the expected waste time at the N^{th} decision point is equal to zero. Thus, the optimal expected waste time at the $(N-1)^{th}$ decision point is the least expected waste time among the full checkpoint, the incremental checkpoint, and the rejuvenation. Next, for each FT mechanism, the expected waste time at the $(N-2)^{th}$ decision point is calculated by using the optimal expected waste time at the $(N-1)^{th}$ decision point. Again, the optimal expected waste time at $(N-2)^{th}$ decision point is the smallest expected waste time among considered FT mechanisms. We repeat this process until the optimal expected waste time at the 1^{st} decision point is obtained. Therefore, the optimal expected total waste time can be calculated by using the optimal expected waste time at the 1^{st} decision point. The global minimum of the expected total waste time is formulated based on the multistage stochastic programming approach [61] as in Formulation 5.2.

Formulation 5.2. *Let $f^N(0,0,a_0)$ be the optimal expected total waste time of an N-stage process that starts with the software state of age a_0. It can be expressed in Eq. 5.8, where $f_1^N(0,0,a_0)$ is the optimal expected waste time at the 1st decision point. The first two zeros as the parameters of f^N and f_1^N are the values of m_F and m_I.*

$$f^N(0,0,a_0) = \begin{cases} P_0(a_0)(\Delta + f^N(0,0,0)) & \\ +(1 - P_0(a_0))f_1^N(0,0,a_0+\Delta), & a_0 > 0 \\ \dfrac{P_0(0)\Delta}{1-P_0(0)} + f_1^N(0,0,\Delta), & a_0 = 0 \end{cases} \tag{5.8}$$

In Eq. 5.8, the first relation can be obtained by substituting $EW[FT]_1^N$ in Eq. 5.4 by f_1^N, and the second relation can be obtained by solving the recursive equation for $f^N(0)$. At the k^{th} decision point, for $k = 1, 2, 3, \ldots, N-1$, the optimal expected waste time f_k^N is the minimum among the expected waste times of the checkpoints, incremental checkpoints, and rejuvenation. By means of multistage stochastic programming, we formulate the optimal expected waste time from the k^{th} decision point on in Formulation 5.3.

Formulation 5.3. *Let $f_k^N(m_F, m_I, a)$ be the optimal expected waste time of an N-stage process from the decision point k on, assuming that the software age is a; and m_F full checkpoints and m_I incremental checkpoints must be loaded upon a failure occurrence. It can be expressed in Eq. 5.9, where $f_1^N(\mathbf{0}) = f_1^N(0,0,0)$ and $k \in \{1, 2, 3, \cdots, N\}$. F, I, and R are notations for the choices of full checkpoint, incremental checkpoint, and rejuvenation, respectively.*

$$f_k^N(m_F, m_I, a) = \min \begin{cases} F: & P_{full}^k(a)\Big(m_F R_F + m_I R_I + \Delta + f^{N-k+1}(\mathbf{0})\Big) \\ & +(1 - P_{full}^k(a))\Big(O_F \\ & \quad +P_k(a)\left(R_F + \Delta + f^{N-k}(\mathbf{0})\right) \\ & \quad +(1 - P_k(a)) f_{k+1}^N(1, 0, a + O_F + \Delta)\Big) \\ I: & P_{incre}^k(a)\Big(m_F R_F + m_I R_I + \Delta + f^{N-k+1}(\mathbf{0})\Big) \\ & +(1 - P_{incre}^k(a))\Big(O_I \\ & \quad +P_k(a)\left(m_F R_F + (m_I + 1) R_I + \Delta + f^{N-k}(\mathbf{0})\right) \\ & \quad +(1 - P_k(a)) f_{k+1}^N(m_F, m_I + 1, a + O_I + \Delta)\Big) \\ R: & P_{rej}^k(a)\Big(m_F R_F + m_I R_I + \Delta + f^{N-k+1}(\mathbf{0})\Big) \\ & +(1 - P_{rej}^k(a))\Big(O_F + O_R \\ & \quad +P_k(0)\left(R_F + \Delta + f^{N-k}(\mathbf{0})\right) \\ & \quad +(1 - P_k(0)) f_{k+1}^N(1, 0, \Delta)\Big) \end{cases} \tag{5.9}$$

subject to $m_F \in \{0, 1\}$, $0 \le m_I \le k - 1$, $0 \le a \le a_0 + k\Delta + (k-1)O_F$ *and* $f_N^N(m_F, m_I, a) = 0$.

The function f_k^N is obtained by substituting EW^{N-k+1} by f^{N-k+1}, EW^{N-k} by f^{N-k}, and $EW[FT]_{k+1}^N$ by f_{k+1}^N in Eqs. 5.5-5.7. If $a_0 = 0$, Eq. 5.9 becomes a recursive function of $f^N(\mathbf{0})$ when $k = 1$. By an algebraic manipulation, $f_1^N(0, 0, \Delta)$ can be expressed as Eq. 5.10. m_F and m_I are 0 because no checkpoints are performed before the first decision point.

$$f_1^N(0,0,\Delta) = \min \begin{cases} F: & \left(\dfrac{P_{full}^N(\Delta)}{1-P_{full}^N(\Delta)}\right)\left(\dfrac{\Delta}{1-P_0(0)}\right) \\ & +\big(O_F + P_1(\Delta)\left(R_F + \Delta + f^{N-1}(\mathbf{0})\right) \\ & +(1-P_1(\Delta))f_2^N(1,0,a+O_F+\Delta)\big) \\ I: & \left(\dfrac{P_{incre}^N(\Delta)}{1-P_{incre}^N(\Delta)}\right)\left(\dfrac{\Delta}{1-P_0(0)}\right) \\ & +\big(O_I + P_1(\Delta)\left(R_I + \Delta + f^{N-1}(\mathbf{0})\right) \\ & +(1-P_1(\Delta))f_2^N(0,1,a+O_I+\Delta)\big) \\ R: & \left(\dfrac{P_{rej}^N(\Delta)}{1-P_{rej}^N(\Delta)}\right)\left(\dfrac{\Delta}{1-P_0(0)}\right) \\ & +\big(O_F + O_R + P_1(0)\left(R_F + \Delta + f_1^{N-1}(\mathbf{0})\right) \\ & +(1-P_1(0))f_2^N(1,0,\Delta)\big) \end{cases} . \quad (5.10)$$

To obtain the optimal decision interval, we iteratively determine the least expected waste times for various decision intervals by using the decision-making model. Therefore, the optimal decision interval is the interval that results in the least expected waste time among all possible decision intervals. To study the sensitivity of the model for checkpoints and rejuvenation, we have run simulations with various parameter values. In the next section, we discuss the parameter values used in the simulations.

5.3 Simulations

The objective of the simulations is to study the sensitivity of the proposed model and the conditions when the applications do not require any FT mechanisms. The main factors of the decision-making process are the application completion time, the chance of failure occurrences, and the FT overhead. We range the completion times (d) from three hours to eight days to represent various applications.

Schroeder and Gibson [59] have reported that the best fitted failure distribution is a Weibull distribution. Thus, in the simulations, we use the Weibull distribution with various shape (β) and scale (α) parameters. The values 0.5, 1, and 1.5 for the shape parameters represent a decreasing, constant, or increasing failure rate over time. Instead of varying the scale parameters directly, we vary mean-time-to-failure (MTTF) and then calculate the scale parameters. Moreover, we range MTTF from a quarter of the completion time to as large as eight times the completion time. All parameters and values used in the simulations are listed in Table 5.2.

Table 5.2: Parameter values in simulations

Parameter	Values
Completion time (d)	3 hours, 12 hours, 2 days, 8 days
$MTTF$	0.25, 0.5, 1, 2, 4, 8 x d
Shape parameter (β)	0.5, 1, 1.5
Incremental checkpoint overhead (O_I)	0.5%, 1%, 2%, 5% of d
Full checkpoint overhead (O_F)	1, 2, 3 x O_I
Rejuvenation overhead (O_R)	5, 10, 30, 60 minutes

We vary the values of the incremental checkpoint overhead according to percentages of the completion time. Also, the full checkpoint overheads in the simulations are multiples of the incremental checkpoint overheads. In practice, the checkpoint overhead is not proportional to the application completion time. However, we suspect that, if the checkpoint overhead is big enough relative to the completion time, it is not worth to perform the checkpoint mechanisms. In addition, we assume that the recovery costs of both full and incremental checkpoints are equal to the overhead of full and incremental checkpoints, respectively, i.e., $R_F = O_F$ and $R_I = O_I$. In contrast to the checkpoint overheads, we vary the rejuvenation overhead independently from the other parameters because, according to the proposed scheme, a full checkpoint is always performed right before each rejuvenation. Thus, when the model selects the rejuvenation choice, the total overhead is a proportion of the completion time already.

Since we cannot determine the best decision interval directly from the decision-making model, in the simulations, we iteratively give different decision intervals to the decision-making model and determine the best policy for each decision interval. For each combination of the parameters, we vary the decision interval lengths as quotients of the application completion time (d), ranged from 1/10 to 1. If the decision interval is equal to the completion time, no FT mechanism is performed. This case is the control case that suggests whether the FT mechanisms are beneficial or not. The simulation results are shown and discussed in detail in Section 5.4.

5.4 Simulation Results

According to Table 5.2, there are 3456 simulations with different combinations of parameter values. As mentioned, each simulation is run with 10 different decision interval lengths, and we look for the best decision interval which gives the least expected waste time. We categorize all simulations into 3 groups by the decision interval that results in the smallest expected waste time.

The first group consists of the simulations for which the best decision interval (Δ^*) is equal to the completion time ($\Delta^* = d$). We can infer the conditions when the applications do not require any of the FT mechanisms from the simulations in this group. The second group consists of the simulations for which the best decision interval Δ^* is in between the completion time and one-tenth of the completion time ($\frac{d}{10} < \Delta^* < d$). Lastly, the third group consists of the simulations for which the best decision interval is less than or equal to one-tenth or 10% of the completion time ($\Delta^* \leq \frac{d}{10}$). Example graphs of each group are illustrated in Figure 5.2. In the graphs, the expected waste times are plotted against the ten decision intervals.

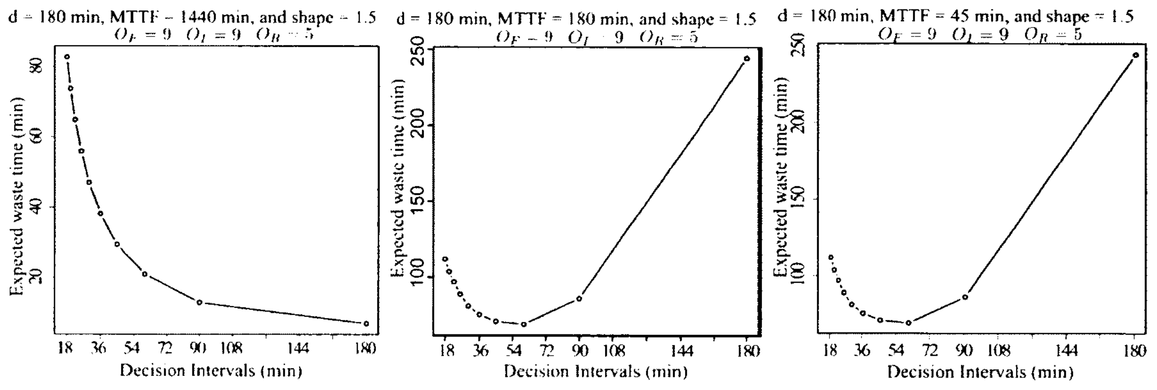

Figure 5.2: Example graphs of the expected total waste times when $\Delta^* = d$ (Left), $\frac{d}{10} < \Delta^* < d$ (Middle), and $\Delta^* \leq 10\%$ of d (Right), where Δ^* is the decision interval that gives the smallest expected total waste time and d is the application completion time

5.4.1 The Best Policy

Among the three FT mechanisms that we consider in this study, the incremental checkpoint has the smallest overhead, so we expect that the model will decide to perform an incremental checkpoint at most of the decision points. From the simulation results, there are 1643 cases out of 3456 cases or 48% of all simulations that the model exclusively selects incremental checkpoints as the best FT policy. Furthermore, we are interested in when a full checkpoint or a rejuvenation is preferable as the FT mechanism of choice.

Evidently, the model will not schedule any incremental checkpoints if its overhead is equal the full checkpoint because the total recovery overhead of the incremental checkpoints is higher than that of the full checkpoint. There are 514 cases that a rejuvenation is scheduled, and, for those 514 cases, the failure rate is increasing. If the failure rate is decreasing or constant, rejuvenation will not protect the running applications from failure occurrences, also corresponding to the results in [23, 45]. This is because rejuvenation helps in refreshing the software state and, with increasing failure rate, we prefer to run applications in the early period of the system software. On the other hand, rejuvenation will aggravate the failure impact to running applications if the failure rate is decreasing. For constant failure rates, rejuvenation does not affect the chance of failure occurrences because of the memoryless property of exponential distribution. Therefore, rejuvenating the software state costs unnecessary waste time.

5.4.2 Conditions when there is No Need for any FT Mechanisms

We observe the conditions when the applications do not require any FT mechanisms from the simulations in the first group ($\Delta^* = d$), illustrated in Figure 5.2 (Left). The equality of the best decision interval and the completion time infers that performing a single FT mechanism leads to a higher execution time than doing nothing. This does not mean that there is no failure during the application execution, but it means that re-computing the work costs less than performing an FT mechanism. We summarize all the simulations that have this property in Table 5.3. "All" in Table 5.3 refers to all values for that particular parameter in Table 5.2.

Table 5.3: Cases that the best decision interval is equal to the completion time ($\Delta^* = d$) (No need for any FT mechanisms)

d	β	$MTTF/d$	O_I% of d	O_F/O_I	O_R(min)
3 hours	1.5	8	2 and 5	All	All
12 hours	1.5	8	2 and 5	All	All
2 days	1.5	8	2	2 and 3	All
				1	30 and 60
			5	All	All
8 days	1.5	8	2	2 and 3	All
				1	60
			5	All	All

In terms of failure characteristics, the simulation results suggest that there is no need for any FT mechanism only when the MTTF is 8 times larger than the completion time. Intuitively we might not need to perform any FT mechanisms if the failure rate is decreasing over time (shape parameter less than 1), but the results in Table 5.3 show the opposite. The reason can be seen from the failure probability used in the model, which is the conditional probability given that the system survives until time a (age of the software state at the k^{th} decision point or at the completion of the FT mechanisms at the k^{th} decision point).

Figure 5.3 illustrates the conditional probabilities of the three cases of failure rates (increasing, constant and decreasing) when the application completion time is 2 days, the MTTF is 16 days, and the decision interval is 2 days. The x-axis is the age (in minutes) of the software state from 0 to 4 days. The graph is an example of the conditional probability P_k of cases that $\Delta^* = d$. From the graph, for the applications with completion time of 2 days, the chance of a failure occurrence during the execution is very high in the early period of the execution if the shape parameter is 0.5 (a decreasing failure rate), resulting in the necessity of the FT mechanisms. Besides, the conditional probability is the least if the shape parameter of 1.5 (an increasing failure rate). Therefore, with the MTTF = $8d$ and $\Delta = d$, the expected waste time is the least if the failure rate is increasing.

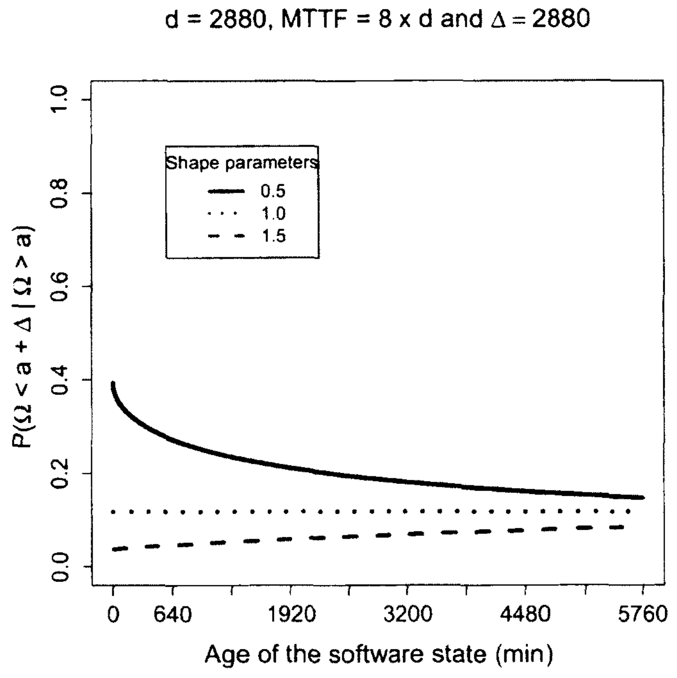

Figure 5.3: Conditional Probability ($P(\Omega < a + \Delta | \Omega > a)$) of the Weibull distributions with the shape parameters (β) of 0.5, 1, and 1.5, completion time (d) of 2 days, the decision interval (Δ) of 2 days, and MTTF of 16 days. The x-axis is the age of the software state (a) in minutes, ranged from 0 to 4 days.

In the overhead perspective, according to Table 5.3, if the incremental checkpoint overhead is 5% of the completion time, there is no need for any FT mechanisms, regardless of the other parameters. In more details, for the cases of the 3- and 12-hour completion times, if the incremental checkpoint overhead is 2% of the completion time, it is not worth to perform any FT mechanisms. In contrast, for the cases with the completion time of 2 and 8 days, if the full checkpoint overhead is 2% of the completion time as the incremental checkpoint overhead ($O_F = O_I$) and the rejuvenation overhead is less than 30 minutes ($O_R <$ 30 min.), it is still worth to perform some FT mechanisms. This is because the rejuvenation overhead is relatively small, compared to the completion time. Besides, a rejuvenation is a preferred choice to perform for the best decision interval for all such cases.

In this section, we will discuss the cases in which the decision interval that gives the least expected total waste time is 10% of the completion time, illustrated in Figure 5.2 (Right). There are 1286 cases out of 3456 cases or around 37% of all simulations in which the decision interval that gives the smallest expected total waste time is 10% of the completion time. Therefore, we can conclude that the best/optimal decision interval is less than or equal to 10% of the completion time.

There are two primary reasons that the best decision interval is relatively small. Firstly, the chance of failure is very high due to occurrences during the execution, and secondly the FT overhead is small enough to perform a number of the FT mechanisms. According to Figure 5.4, the majority of the cases in which the best decision interval is small are when the MTTF is less than the completion time, the incremental checkpoint overhead is 0.5% of the completion time, and the shape parameter is 0.5 which is equivalent to a decreasing failure rate. This means that, with a decreasing failure rate, the system needs to perform full checkpoints, incremental checkpoints, or rejuvenation more often.

Table 5.4: Cases that the best decision interval is less than or equal to 10% of the completion time, where d is the application completion time

MTTF/d	β	O_I% of d	d (min)	O_F/O_I	O_R (min)
8	0.5	0.5	All	All	All
4	0.5	0.5	All	All	All
		1	720	1	All
2	0.5	0.5 and 1	All	All	All
	1	0.5	All	1	All
1	0.5	0.5 and 1	All	All	All
		2	All	1	All
	1	0.5	All	All	All
		1	All	1	All
	1.5	0.5	3 hours	All	All
			12 hours	1 and 2	10, 30, and 60
				3	All
			2 days	1	30 and 60
				2	10, 30 and 60
			8 days	2	5 and 60
			2 and 8 days	3	All
0.5	0.5	0.5, 1, and 2	All	All	All
	1	0.5 and 1	All	All	All
		2	All	1	All
	1.5	0.5	All	All	All
		1	3 hrs and 2 days	All	All
			12 hours and 8 days	1	10, 30 and 60
				2 and 3	All
		2	3 and 12 hours	1	30 and 60
0.25	0.5	All	All	All	All
	1	0.5, 1, and 2	All	All	All
		5	All	1	All
	1.5	0.5, 1, and 2	All	All	All
		5	3 hours	1	10, 30 and 60
		5	12 hours	1	5 and 60
			2 days	1	5 and 10
			8 days	1	All

5.4.3 Conditions when the Best Decision Interval is Less than 10% of the Completion Time

Obviously, from Figure 5.4 (Middle), most cases have an incremental checkpoint overhead of 0.5% of the completion time because the overhead is so small that frequent checkpoints do not significantly lengthen the execution time. The incremental checkpoint overheads in the simulations are impractically large for the large completion times. However, we are able to conclude that, for the applications running on a system with MTTF less than the completion time, the best decision interval is less than 10% of the completion time. All cases in which the best decision interval is less than or equal to 10% of the completion time are listed in Table 5.4.

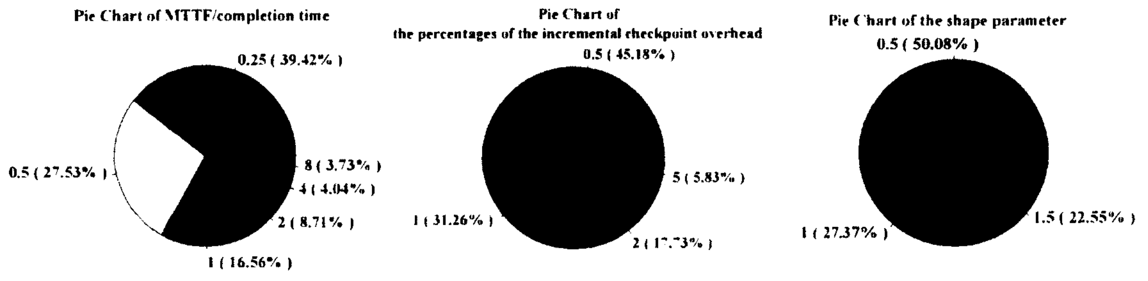

Figure 5.4: Pie Chart of the MTTFs as multiples of the completion time (d) (Left), the shape parameter (β)(Middle), and the incremental checkpoint overheads (O_I) as percentages of the completion times (Right) for the cases that the best decision interval is less than or equal to 10% of the completion time

Figure 5.5 illustrates the conditional probabilities of failures in the simulations in which the application completion time is 2 days and the MTTF is 12 hours (25% of the completion time). The left graph is for a decision interval of 288 minutes which is the smallest decision interval considered in the simulations for the completion time of 2 days. The right graph is for a decision interval of 144 minutes which is not a part of the simulations. It is plotted solely for an illustration that the conditional probability of failures for a smaller decision interval is less than those of decision intervals considered in the simulations. The x-axis is the age (in minutes) of the system software from 0 to 2 days. According to both graphs, when the shape parameter is 0.5, the chance of failure occurrences in the early period of the software state is very high, compared to the shape parameters of 1 and 1.5. Therefore, the FT mechanisms are required to perform more often. Besides, the probability of a failure occurrence for the 144-minute decision interval is less than that of the 288-minute decision interval by 0.1. Consequently, the best decision interval should be less than 288 minutes.

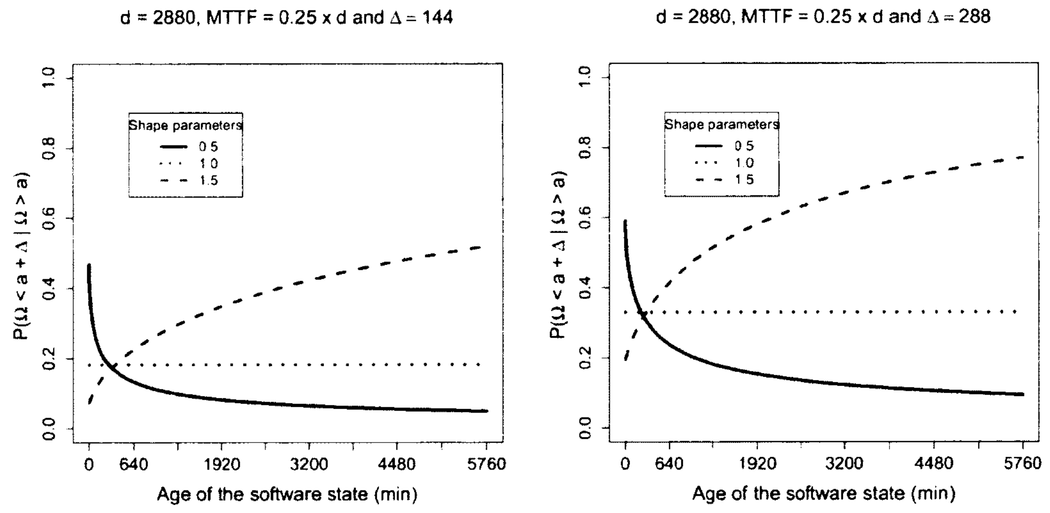

Figure 5.5: Conditional Probability ($P(\Omega < a + \Delta | \Omega > a)$) of the Weibull distributions with the shape parameters of 0.5, 1, and 1.5, the completion time (d) of 2 days, and the MTTF of 12 hours. The x-axis is the age of the software state (a) in minutes, ranged from 0 to 2 days

In conclusion, a novel decision-making model has been derived to determine the best strategy to invoke various FT mechanisms. A concrete example of the decision-making model is given which is for full checkpoints, incremental checkpoints, and software rejuvenation. Finally, simulations to study the sensitivity have been conducted. According to the simulation results, combining reactive and proactive FT mechanisms does not always yield the least expected waste time. Moreover, some short applications do not require any FT mechanisms.

CHAPTER 6

CONCLUSIONS

To attack the problem of expensive overhead of FT mechanisms, we have derived a near-optimal scheduling model for the hybrid checkpoint mechanism that combines the full and incremental checkpoint techniques for arbitrary failure distributions in a HPC environment. To determine a sequence of optimal checkpoint times, we derived the formula for near-optimal checkpoint times for Weibull distributions. Moreover, in contrast to other existing scheduling models for the checkpoints, linearity of the re-computing time has not been assumed to derive the model. Instead, the proposed algorithm numerically estimates the re-computing time coefficient k that is a key contribution. Also, the existence and uniqueness of the number of incremental checkpoints between two consecutive full checkpoints (m) that minimizes the waste time have been proved. From the derived formula for m, the number of incremental checkpoints m depends on the full and incremental checkpoint overheads, the incremental recovery cost, the re-computing time, and the failure rate.

To study the benefits of the proposed scheduling model for the hybrid checkpoint mechanism, the waste times of the full checkpoint mechanism and the hybrid checkpoint mechanism have been studied, simulated, and compared. In most cases, the waste times of the hybrid checkpoint mechanism are less than those of the full checkpoint mechanism, especially when the incremental checkpoint overhead is less than 50% of the full checkpoint overhead. Furthermore, the proportion of the waste times of the hybrid and full checkpoint mechanisms does not relate to the ratio of the incremental and full checkpoint overheads.

To further improve the utilization of the FT mechanisms, we have also developed a novel adaptive decision-making model that determines the best choice among all considered FT mechanisms at each decision point in order to obtain the global minimum of the application execution time. The concept of the model can be applied to any FT mechanisms; however, to give a concrete example, we have derived the model for full and incremental checkpoints as well as software rejuvenation. To study the sensitivity of the decision-making model, we have run simulations with various parameter values in the decision-making model. The simulation results suggest that applications with completion time longer than 2 days definitely require some FT mechanisms to alleviate the failure impacts. Because of the low overhead of the incremental checkpoint, the model preferably selects the incremental checkpoint at each decision point. However, the best policy might be not to perform any FT mechanisms if the overhead of the FT mechanisms and MTTF are relatively large compared to the application completion time.

In the future, the proposed scheduling model should address the fluctuation of the incremental checkpoint overheads. A more accurate estimation of the re-computing time coefficient would enhance the accuracy of the scheduling model. For the decision-making model, a challenge is to solve the proposed decision-making model for a small decision interval because of the time complexity. Therefore, we have planned to develop an algorithm to estimate the optimal policy that reduces the time complexity. Furthermore, because the proposed model is based on constant decision interval which does not handle the increasing or decreasing failure rate cases well, we have planned to add the choice of doing nothing at each decision point to dynamically vary the decision interval based on the FT overhead and the failure characteristics.

APPENDIX A

MINIMUM OF A SET OF RECURSIVE, LINEAR FUNCTIONS

The minimum of a finite set of real numbers is the infimum of that set.

Lemma A.1. *Let A and B be real values.* $f := \min\{A, B\}$ *iff*

$\Big[\big(A < B \Rightarrow f = A\big)$ *and* $\big(B \le A \Rightarrow f = B\big)\Big]$.

Axiom A.2. $f := \min\{A, B(f)\}$ *iff* $\Big[\big(A < B(f) \Rightarrow f = A\big)$ *and* $\big(B(f) \le A \Rightarrow f = B(f)\big)\Big]$

Lemma A.3. *If* $B(f) = af + b$, *where* $0 < a < 1$ *and* $b > 0$, *then* $f = B(f)$ *iff* $f = X$, *where* $X = \frac{b}{1-a}$.

Proof. ($\Rightarrow$) $f = B(f) = af + b$. By solving the equation, we have $f = \dfrac{b}{1-a} = X$.

($\Leftarrow$) $f = X = \frac{b}{1-a}$, then $B(f) = a(\frac{b}{1-a}) + b = \frac{ab+(1-a)b}{1-a} = \frac{b}{1-a} = f$. □

Theorem A.4. *If* $B(f) = af + b$, $\exists a \in (0,1), b > 0$, *then* $f = \min\{A, B(f)\} \Rightarrow$ $\Big[\big(A < X \Rightarrow f = A\big)$ *and* $\big(X \le A \Rightarrow f = X\big)\Big]$, *where* $X = \frac{b}{1-a}$.

Proof. Suppose $f = \min\{A, B(f)\}$. To show that $\big(A < X \Rightarrow f = A\big)$, we assume that $f \ne A$. Then $f = B(f)$. By Lemma A.3, we have that $f = X$. Then, by Axiom A.2, $B(f) \le A$. Therefore, $X = f = B(f) \le A$

To show that $\big(X \le A \Rightarrow f = X\big)$, we assume that $f \ne X$. By Lemma A.3, $f \ne B(f)$, then $f = A$ by Axiom A.2. Therefore, $A < B(f)$ by Axiom A.2. Thus, $A < B(f) = af + b = aA + b$. Hence, $A < \frac{b}{1-a} = X$. □

Axiom A.5. $f := \min\{A(f), B(f)\}$ *iff* $\Big[\big(A(f) < B(f) \Rightarrow f = A(f)\big)$ *and* $\big(B(f) \le A(f) \Rightarrow f = B(f)\big)\Big]$

Theorem A.6. *If $A(f) = a_1 f + a_2$ and $B(f) = b_1 f + b_2$, where $0 < a_1, b_1 < 1$ and $a_2, b_2 > 0$, then $f = min\{A(f), B(f)\} \Rightarrow f = min\{X_A, X_B\}$, where $X_A = \frac{a_2}{1-a_1}$ and $X_B = \frac{b_2}{1-b_1}$ for some $0 < a_1, b_1 < 1$ and $a_2, b_2 > 0$.*

Proof. Suppose $f = \min\{A(f), B(f)\}$. To prove that $f = \min\{X_A, X_B\}$, we will show that $\big(X_A < X_B \Rightarrow f = X_A\big)$ and $\big(X_B \leq X_A \Rightarrow f = X_B\big)$.

To show that $\big(X_A < X_B \Rightarrow f = X_A\big)$, we assume that $f \neq X_A$. By Lemma A.3, $f \neq A(f)$. By Axiom A.5, we know that $f = B(f)$ and $B(f) \leq A(f)$. Thus, by Lemma A.3, we have that $X_B = B(f) \leq A(f) = a_1 f + a_2 = a_1 X_B + a_2$. Then, $X_B \leq \frac{a_2}{1-a_1} = X_A$.

To show that $\big(X_B \leq X_A \Rightarrow f = X_B\big)$, we use a similar argument. $\square$

BIBLIOGRAPHY

[1] IEEE 100 The Authoritative Dictionary of IEEE Standards Terms Seventh Edition, *IEEE Std 100-2000*, 2000.

[2] S. Agarwal, R. Garg, M. S. Gupta, and J. E. Moreira, Adaptive incremental checkpointing for massively parallel systems, in *Proceedings of the 18th annual international conference on Supercomputing*, ICS '04, pp. 277–286, ACM, New York, NY, USA, 2004.

[3] A. Avritzer, R. Cole, and E. Weyuker, Methods and opportunities for rejuvenation in aging distributed software systems, in *Software Reliability Engineering Workshops, 2008. ISSRE Wksp 2008. IEEE International Conference on*, pp. 1–6, 2008.

[4] A. Avritzer, R. G. Cole, and E. J. Weyuker, Methods and opportunities for rejuvenation in aging distributed software systems, *J. Syst. Softw.*, vol. 83, pp. 1568–1578, September 2010.

[5] H. Bauer and C. Sporrer, Reducing Rollback Overhead In Time-warp Based Distributed Simulation With Optimized Incremental State Saving, in *Simulation Symposium, 1993. Proceedings., 26th Annual*, pp. 12–20, 1993.

[6] H. Bauer, C. Sporrer, and T. H. Krodel, On Distributed Logic Simulation Using Time Warp., in *VLSI'91*, pp. 127–136, 1991.

[7] A. Bobbio, S. Garg, M. Gribaudo, A. Horvath, et al., Modeling software systems with rejuvenation, restoration and checkpointing through fluid stochastic Petri nets, in *Petri Nets and Performance Models, 1999. Proceedings. The 8th International Workshop on*, pp. 82–91, 1999.

[8] A. Bobbio, M. Sereno, and C. Anglano, Fine grained software degradation models for optimal rejuvenation policies, *Perform. Eval.*, vol. 46, pp. 45–62, August 2001.

[9] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, et al., Checkpointing strategies for parallel jobs, in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pp. 33:1–33:11, ACM, New York, NY, USA, 2011.

[10] K. Cassidy, K. Gross, and A. Malekpour, Advanced pattern recognition for detection of complex software aging phenomena in online transaction processing servers, in *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pp. 478–482, 2002.

[11] K. M. Chandy, J. C. Browne, C. W. Dissly, and W. R. Uhrig, Analytic Models for Rollback and Recovery Strategies in Data Base Systems, *IEEE Transactions on Software Engineering*, vol. 1(1), pp. 100–110, 1975.

[12] K. M. Chandy and C. V. Ramamoorthy, Rollback and Recovery Strategies for Computer Programs, *Computers, IEEE Transactions on*, vol. C-21(6), pp. 546–556, 1972.

[13] G. Ciardo, J. Muppala, and K. Trivedi, SPNP: stochastic Petri net package, in *Petri Nets and Performance Models, 1989. PNPM89., Proceedings of the Third International Workshop on*, pp. 142–151, December 1989.

[14] J. Coffman. E.G. and E. Gilbert. Optimal strategies for scheduling checkpoints and preventive maintenance. *Reliability. IEEE Transactions on.* vol. 39(1). pp. 9 18. 1990.

[15] J. Daly. A model for predicting the optimum checkpoint interval for restart dumps. in *Proceedings of the 2003 international conference on Computational science.* ICCS'03. pp. 3–12. Springer-Verlag, Berlin. Heidelberg. 2003.

[16] J. Daly, A strategy for running large scale applications based on a model that optimizes the checkpoint interval for restart dumps, *IEE Seminar Digests*, vol. 2004(903), pp. 70–74, 2004.

[17] J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Gener. Comput. Syst.*, vol. 22, pp. 303–312, February 2006.

[18] T. Dohi. K. Goseva-Popstojanova, and K. Trivedi, Analysis of software cost models with rejuvenation, in *High Assurance Systems Engineering, 2000, Fifth IEEE International Symposim on. HASE 2000.* pp. 25 34, 2000.

[19] T. Dohi, K. Goseva-popstojanova, and K. S. Trivedi, Statistical Non-Parametric Algorithms to Estimate the Optimal Software Rejuvenation Schedule. in *Pacific Rim Intl Symp. Dependable Computing (PRDC). IEEE Computer Soc. Press. Los Alamitos. Calif.* pp. 77 84. Press. 2000.

[20] A. Duda. The Effects of Checkpointing on Program Execution Time.. *Inf. Process. Lett..* vol. 16(5). pp. 221 229. 1983.

[21] E. Elnozahy and J. Plank. Checkpointing for peta-scale systems: a look into the future of practical rollback-recovery. *Dependable and Secure Computing. IEEE Transactions on.* vol. 1(2). pp. 97 108. April-June 2004.

[22] C. Engelmann, G. R. Vallee, T. Naughton, and S. L. Scott, Proactive Fault Tolerance Using Preemptive Migration, in *Proceedings of the 2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pp. 252–257, IEEE Computer Society, Washington, DC, USA, 2009.

[23] S. Garg, Y. Huang, C. Kintala, and K. S. Trivedi, Minimizing completion time of a program by checkpointing and rejuvenation, *SIGMETRICS Perform. Eval. Rev.*, vol. 24(1), pp. 252–261, 1996.

[24] S. Garg, A. van Moorsel, K. Vaidyanathan, and K. Trivedi, A methodology for detection and estimation of software aging, in *Software Reliability Engineering, 1998. Proceedings. The Ninth International Symposium on*, pp. 283–292, November 1998.

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

[26] S. Ghemawat, H. Gobioff, and S.-T. Leung, The Google file system, *SIGOPS Oper. Syst. Rev.*, vol. 37, pp. 29–43, 2003.

[27] S. B. D. J. Gibson, Garth A., Failure Tolerance in Petascale computers, *CTWatch Quarterly*, vol. 3(4), November 2007.

[28] J. N. Glosli, D. F. Richards, K. J. Caspersen, R. E. Rudd, et al., Extending stability beyond CPU millennium: a micron-scale atomistic simulation of Kelvin-Helmholtz instability, in *Supercomputing, 2007. SC '07. Proceedings of the 2007 ACM/IEEE Conference on*, pp. 1–11, November 2007.

[29] N. Gottumukkala, R. Nassar, M. Paun, C. Leangsuksun, et al., Reliability of a System of k Nodes for High Performance Computing Applications, *Reliability, IEEE Transactions on*, vol. 59(1), pp. 162–169, 2010.

[30] T. Heath, R. P. Martin, and T. D. Nguyen, Improving cluster availability using workstation validation, *SIGMETRICS Perform. Eval. Rev.*, vol. 30, pp. 217–227, June 2002.

[31] J. Hong, S. Kim, Y. Cho, H. Yeom, et al., On the choice of checkpoint interval using memory usage profile and adaptive time series analysis, in *Dependable Computing, 2001. Proceedings. 2001 Pacific Rim International Symposium on*, pp. 45–48, 2001.

[32] G. Horton, V. G. Kulkarni, D. M. Nicol, and K. S. Trivedi, Fluid stochastic Petri nets: Theory, applications, and solution techniques, *European Journal Of Operational Research*, vol. 105(1), pp. 184–201, 1998.

[33] Y. Huang, C. Kintala, N. Kolettis, and N. Fulton, Software rejuvenation: analysis, module and applications, in *Fault-Tolerant Computing, 1995. Digest of Papers., Twenty-Fifth International Symposium on*, pp. 381–390, June 1995.

[34] J. Hursey, *Coordinated checkpoint/restart process fault tolerance for mpi applications on hpc systems*, Ph.D. thesis, Indianapolis, IN, USA, 2010.

[35] D. P. Jasper, A discussion of checkpoint/restart, *Software Age*, vol. 9, pp. 9–14, October 1969.

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

[37] Z. Lan and Y. Li. Adaptive Fault Management of Parallel Applications for High-Performance Computing. *Computers. IEEE Transactions on.* vol. 57(12), pp. 1647–1660. December 2008.

[38] C. H. C. Leung and Q. H. Choo, On the Execution of Large Batch Programs in Unreliable Computing Systems, *Software Engineering. IEEE Transactions on,* vol. SE-10(4), pp. 444–450. July 1984.

[39] Y. Ling, J. Mi, and X. Lin, A variational calculus approach to optimal checkpoint placement, *Computers. IEEE Transactions on*, vol. 50(7), pp. 699–708, July 2001.

[40] Y. Liu, R. Nassar, C. Leangsuksun, N. Naksinehaboon, et al., A reliability-aware approach for an optimal checkpoint/restart model in HPC environments, in *Proceedings of the 2007 IEEE International Conference on Cluster Computing,* CLUSTER '07. pp. 452–457, IEEE Computer Society, Washington, DC, USA, 2007.

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

[42] A. Moody. G. Bronevetsky. K. Mohror, and B. de Supinski. Design. Modeling, and Evaluation of a Scalable Multi-level Checkpointing System. in *High Performance Computing. Networking. Storage and Analysis (SC). 2010 International Conference for.* pp. 1–11. 2010.

[43] N. Naksinehaboon, Y. Liu, C. Leangsuksun, R. Nassar, et al., Reliability-Aware Approach: An Incremental Checkpoint/Restart Model in HPC Environments, in *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on*, pp. 783–788, May 2008.

[44] N. Naksinehaboon, M. Paun, R. Nassar, C. Leangsuksun, et al., High Performance Computing Systems with Various Checkpointing Schemes, *International Journal of Computers Communications & Control*, vol. 4(4), pp. 386–400, 2009.

[45] N. Naksinehaboon, N. Taerat, C. Leangsuksun, C. Chandler, et al., Benefits of Software Rejuvenation on HPC Systems, in *Parallel and Distributed Processing with Applications (ISPA), 2010 International Symposium on*, pp. 499–506, September 2010.

[46] H. Okamura and T. Dohi, Analysis of a software system with rejuvenation, restoration and checkpointing, in *Proceedings of the 5th international conference on Service availability*, ISAS'08, pp. 110–128, Springer-Verlag, Berlin, Heidelberg, 2008.

[47] H. Okamura and T. Dohi, Comprehensive evaluation of aperiodic checkpointing and rejuvenation schemes in operational software system, *J. Syst. Softw.*, vol. 83, pp. 1591–1604, September 2010.

[48] H. Okamura, K. Iwamoto, and T. Dohi, A Dynamic Programming Algorithm for Software Rejuvenation Scheduling under Distributed Computation Circumstance, in *Parallel and Distributed Systems, 2005. Proceedings. 11th International Conference on*, vol. 2, pp. 493–499, 2005.

[49] A. Oliner, L. Rudolph, and R. Sahoo, Cooperative checkpointing theory, in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, p. 10, April 2006.

[50] A. Oliner, R. Sahoo, J. Moreira, and M. Gupta, Performance implications of periodic checkpointing on large-scale cluster systems, in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, p. 8, April 2005.

[51] T. Ozaki, T. Dohi, H. Okamura, and N. Kaio, Distribution-free checkpoint placement algorithms based on min-max principle, vol. 3, pp. 130–140, April-June 2006.

[52] A. C. Palaniswamy and P. A. Wilsey, An analytical comparison of periodic checkpointing and incremental state saving, *SIGSIM Simul. Dig.*, vol. 23, pp. 127–134, July 1993.

[53] M. Paun, N. Naksinehaboon, R. Nassar, C. Leangsuksun, et al., Incremental Checkpoint Schemes for Weibull Failure Distribution, *International Journal of Foundations of Computer Science*, vol. 21(3), pp. 329–344, 2010.

[54] J. Plank and W. Elwasif, Experimental assessment of workstation failures and their impact on checkpointing systems, in *Fault-Tolerant Computing, 1998. Digest of Papers. Twenty-Eighth Annual International Symposium on*, pp. 48–57, June 1998.

[55] N. Raju. Y. Liu. C. B. Leangsuksun. R. Nassar. et al., Reliability Analysis in HPC clusters. in *High Availability and Performance Workshop (HAPCW). in conjunction with Los Alamos Computer Science Institute (LACSI) Symposium*, October 2006.

[56] E. Roman, A Survey of Checkpoint/Restart Implementations, Tech. rep., Lawrence Berkeley National Laboratory, Tech, 2002.

[57] S. M. Ross, *Stochastic Processes*, Wiley. 1995.

[58] R. Sahoo, M. Squillante, A. Sivasubramaniam, and Y. Zhang, Failure data analysis of a large-scale heterogeneous server environment, in *Dependable Systems and Networks, 2004 International Conference on*, pp. 772–781, June-July 2004.

[59] B. Schroeder and G. A. Gibson, A large-scale study of failures in high-performance computing systems, in *DSN '06: Proceedings of the International Conference on Dependable Systems and Networks*, pp. 249–258, IEEE Computer Society, Washington, DC, USA, 2006.

[60] B. Schroeder and G. A. Gibson. Disk failures in the real world: what does an MTTF of 1,000,000 hours mean to you?. in *FAST '07: Proceedings of the 5th USENIX conference on File and Storage Technologies*. p. 1. USENIX Association, Berkeley, CA. USA, 2007.

[61] A. Shapiro, D. Dentcheva, and A. Ruszczy?ski. *Lectures on Stochastic Programming*. Society for Industrial and Applied Mathematics. Philadelphia, PA. 2009.

[62] G. Tian and D. Meng. Coordinated Selective Rejuvenation for Distributed Services. in *Parallel and Distributed Systems (ICPADS). 2010 IEEE 16th International Conference on*. pp. 389–396. 2010.

[63] N. Vaidya. On Checkpoint Latency. in *In Proceedings of the Pacific Rim International Symposium on Fault-Tolerant Systems*. pp. 60–65, 1995.

[64] N. Vaidya, Impact of checkpoint latency on overhead ratio of a checkpointing scheme, *Computers, IEEE Transactions on*, vol. 46(8). pp. 942–947, August 1997.

[65] K. Vaidyanathan and K. Trivedi, A comprehensive model for software rejuvenation, vol. 2, pp. 124–137, April-June 2005.

[66] K. Vaidyanathan and K. S. Trivedi, A Measurement-Based Model for Estimation of Resource Exhaustion in Operational Software Systems. in *ISSRE '99: Proceedings of the 10th International Symposium on Software Reliability Engineering*, p. 84, IEEE Computer Society, Washington, DC, USA, 1999.

[67] G. Vallee, C. Engelmann, A. Tikotekar, T. Naughton, et al., A Framework for Proactive Fault Tolerance, in *Availability, Reliability and Security. 2008. ARES 08. Third International Conference on*, pp. 659–664. March 2008.

[68] M. Varela, K. Ferreira, and R. Riesen. Fault-tolerance for exascale systems, in *Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS). 2010 IEEE International Conference on*. pp. 1–4. 2010.

[69] C. Wang, F. Mueller, C. Engelmann, and S. Scott, Hybrid Checkpointing for MPI Jobs in HPC Environments. in *Parallel and Distributed Systems (ICPADS). 2010 IEEE 16th International Conference on*. pp. 524–533. 2010.

[70] R. Weinstock, *Calculus of Variations: With Applications to Physics and Engineering*. Dover, 1974.

[71] W. Xie, Y. Hong, and K. Trivedi, Software rejuvenation policies for cluster systems under varying workload, in *Dependable Computing, 2004. Proceedings. 10th IEEE Pacific Rim International Symposium on*, pp. 122–129, March 2004.

[72] F. Xin-yuan, X. Guo-zhi, Y. Ren-dong, Z. Hao, et al., Performance analysis of software rejuvenation, in *Parallel and Distributed Computing, Applications and Technologies, 2003. PDCAT'2003. Proceedings of the Fourth International Conference on*, pp. 562–566, August 2003.

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

[74] S. Yi, J. Heo, Y. Cho, and J. Hong, Taking Point Decision Mechanism for Page-level Incremental Checkpointing based on Cost Analysis of Process Execution Time, *J. Inf. Sci. Eng.*, vol. 23(5), pp. 1325–1337, 2007.

[75] J. W. Young, A first order approximation to the optimum checkpoint interval, *Commun. ACM*, vol. 17, pp. 530–531, September 1974.

[76] L. Zhao and Q. Song, Availability and Cost Analysis of a Fault-Tolerant Software System with Rejuvenation, in *Advanced Computer Theory and Engineering, 2008. ICACTE '08. International Conference on*, pp. 261–265, December 2008.