PREDICTION-BASED FAILURE MANAGEMENT FOR SUPERCOMPUTERS

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

2011

By
Wuxiang Ge
School of Computer Science

Contents

List of Tables

List of Figures

Abstract

The growing requirements of a diversity of applications necessitate the deployment of large and powerful computing systems and failures in these systems may cause severe damage in every aspect from loss of human lives to world economy. However, current fault tolerance techniques cannot meet the increasing requirements for reliability. Thus new solutions are urgently needed and research on proactive schemes is one of the directions that may offer better efficiency. This thesis proposes a novel proactive failure management framework. Its goal is to reduce the failure penalties and improve fault tolerance efficiency in supercomputers when running complex applications.

The proposed proactive scheme builds on two core components: failure prediction and proactive failure recovery. More specifically, the failure prediction component is based on the assessment of system events and employs semi-Markov models to capture the dependencies between failures and other events for the forecasting of forthcoming failures. Furthermore, a two-level failure prediction strategy is described that not only estimates the future failure occurrence but also identifies the specific failure categories. Based on the accurate failure forecasting, a prediction-based coordinated checkpoint mechanism is designed to construct extra checkpoints just before each predicted failure occurrence so that the wasted computational time can be significantly reduced. Moreover, a theoretical model has been developed to assess the proactive scheme that enables calculation of the overall wasted computational time.

The prediction component has been applied to industrial data from the IBM BlueGene/L system. Results of the failure prediction component show a great improvement of the prediction accuracy in comparison with three other well-known prediction approaches, and also demonstrate that the semi-Markov based predictor, which has achieved the precision of 87.41% and the recall of 77.95%, performs better than other predictors.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

I would like to express my sincere gratitude for my supervisor John R. Gurd and joint supervisor John A. Keane for their kindly guidance, invaluable advice and personal support; they have led me to the true essence of academic research and repeatedly help me to review, rethink, and revise my ideas. I am also very grateful to my advisor Steve Furber for his support during my studies.

I would like to acknowledge the effort of Dr. Len Freeman organizing seminars in our group for stimulating scientific exchange, and of Paul Slavin who provided warmhearted help on thesis proof reading.

This work was also greatly improved by fruitful discussions with my colleagues in Software Systems Research Group. In particular, I would like to thank Keping Chen, Qiwei Zhu, Chenchen Xi, Chanwit Kaewkasi, Mohamed Khamiss Hussein, Zixu Song and Fu Chen for their help.

I am very grateful to my family for their dedication, supports and continuous encouragement.

Last but not least, I gratefully acknowledge the China Government and UK Government for their financial supports.

Chapter 1

Introduction

Computer science has been developing rapidly for many years and computer systems have been used to improve productivity in multiple disciplines, such as numerical analysis, physics, chemistry, bioinformatics, etc. Furthermore, the growing requirements of a diversity of applications necessitate large and powerful computational and storage capabilities that not only meet the performance demands but are also available when needed. With the dramatic increase of computer systems in scope, complexity and pervasiveness, dependability,[1] which denotes the ability to deliver service that can justifiably be trusted, is a crucial issue.

However, as the correct functioning of large-scale high performance computing systems is becoming increasingly dependent on physically distributed heterogeneous parallel computing and storage resources, interaction of thousands of multiple software components and hardware processors, different levels of human involvement, etc.; the occurrence of failures is inevitable. More specifically, these failures may occur in hardware, software or the network, caused by faulty hardware components, poor software development, incorrect specification of system requirements, inadequate user training, etc. In spite of the unavoidable occurrence of failures, the time to repair the system must be minimized so that system dependability can be improved from another perspective. However, the system complexity has reached a level that it is impossible for human-beings to know all enough details so as to complete the diagnosis, location and repair of failures in a sufficiently short time. An alternative solution, termed *proactive failure management*, is to perform preventive actions before the actual failures occur. This mechanism starts with prediction of future failures according to an assessment

[1]System dependability is measured through its attributes, such as reliability, availability, confidentiality and integrity.

of the current system status, followed by proactive approaches that try to reduce the effects of future failures.

This thesis develops a proactive failure management framework for large-scale supercomputers, which consists of two parts: proactive failure recovery and failure prediction. The following sections provide a brief description of the background and examine the motivation for, and contributions of, the research.

1.1 Background

The issues presented in this thesis include both proactive failure recovery and failure prediction, belonging to the area known as *fault tolerance*, the history of which dates back to the late 1940's when electronic computers were first developed [95]. The approaches developed at that early stage mainly concerned the unreliable hardware components. As the capability and reliability of hardware have increased, a vast amount of complex software has been developed and deployed in multiple domains, and software failures have been attracting more and more attention so that software fault tolerance techniques, such as recovery blocks [48], retry blocks [5], N-Version programming [18], etc. have experienced a rapid development since the 1970s.

However, these software fault tolerance techniques are *reactive* in that they take recovery actions after a failure occurs, being triggered by failure events. It was not until the 1990s that the ideas of *proactive* fault tolerance techniques were introduced. Huang et al. first established the concept of *software rejuvenation*, which is a technique that makes periodic preemptive rollback of continuously running applications to prevent failures in the future [49]. The technique has been shown to be successful to handle *software aging* [82] problems, such as memory leaks. Anticipating that the software complexity crisis that is itself caused by applications and environments consisting of millions of lines of code might threaten to halt progress in computing, in 2001, an IBM manifesto introduced the concept of *autonomic computing* [55]. The main idea of this is to build up self-management computing systems which contain four aspects: self-configuration, self-optimization, self-healing and self-protection [54]. Thus research efforts have been made in proactive fault tolerance to solve a number of related problems.

Failure prediction is the task of forecasting a future failure occurrence according to the recent history of system states. In order to define the problem, some fundamental concepts, such as faults, errors and failures, must be clearly described. Precise

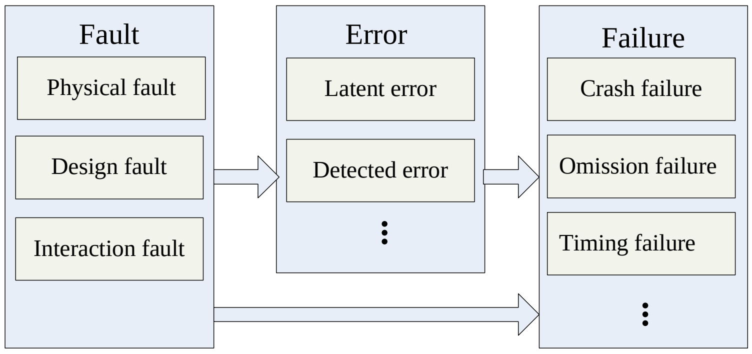


Figure 1.1: Relationships among faults, errors and failures.

definitions of all related terms are provided in Section 1.1.1.

1.1.1 Faults, errors and failures

There exist a wide range of misbehaviours in computer systems; in order to discuss failure prediction issues, the terms of fault, error and failure must be clarified.

Failures in a general sense are common events in computer systems. A *failure* is defined as a delivered service that deviates from the correct service, which has a specified description of expected functions or behaviours. Similar explanations can be found in [7, 6, 58]. A failure occurs because the system does not behave as specified. An important feature is that these misbehaviours must reach the service interface layer so that they can be observed by the final users or monitoring components.

An *error* is a part of a system state that may lead to a subsequent failure; an error turns into a failure once it reaches the service interface.

A *fault* is the root cause of an error; however, there may be undetected faults in the system, and only those that manifest themselves as errors are considered as faults. Likewise, an error that does not actually lead to a failure is denoted a *latent error*. A key point to note here is that faults are unobserved defective states, errors are observed incorrect internal states, and failures are misbehaviours captured at the interface. Figure 1.1 summarizes these relationship.

Faults

Faults have diverse sources and they fall into three categories: physical faults, design faults and interaction faults [6]. Physical faults are induced by adverse hardware phenomena, such as faulty logic, short circuits, temperature, etc. Design faults may come

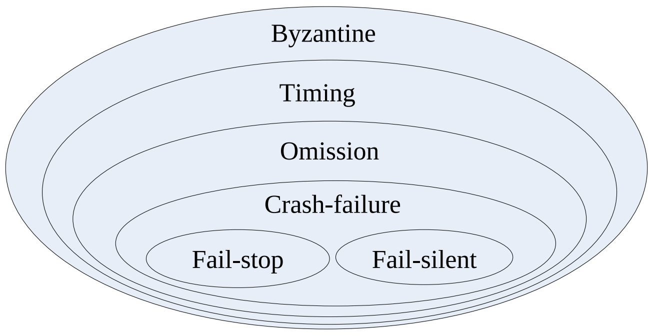


Figure 1.2: Failure classification (taken from [58]).

from either hardware or software, during various processes, such as implementation or maintenance. Interaction faults are mainly due to incorrect engagement by human beings.

Errors

An error is a defective system state that is caused by one or more faults, and each error, which may potentially lead to a failure, can partly be detected by system monitoring mechanisms. Generally, an error will propagate either in a standalone component or among multiple components with interactions. Not all errors will lead to failures, depending on the structure and behaviour of the system, especially whether the system has fault tolerance algorithms or redundancy design.

Failures

Various taxonomies of failures have been published, one of which classifies failures into four categories: crash failures, omission failures, timing failures and byzantine failures [58] (see Figure 1.2).

A *crash failure* occurs when a system stops responding immediately, usually with loss of internal state and information. Two different behaviours in this class of failure include *fail-silent* and *fail-stop*. The main difference between these is that the fail-stop behaviour is easily detectable when it occurs.

An *omission failure* occurs when a system stops responding to a request when it is expected to do so. This has some effect on the resources or services in that the system will only respond to particular requests or input; it either provides correct services or gives no response.

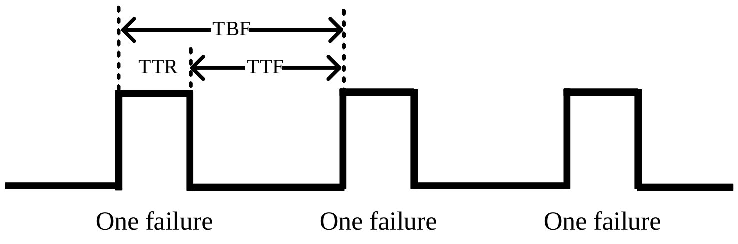


Figure 1.3: Time between failures.

A *timing failure* causes a response at an unexpected time, either too early or too late. Late timing failures are sometimes called *performance* failures.

A *byzantine failure* will cause a system to behave arbitrarily [46].

Fundamental concepts

From a system's perspective, Time Between Failures (TBF), as the name suggests, denotes the time between two failures (see Figure 1.3). Similarly, Time To Failure (TTF) measures the time a piece of software is expected to operate properly before a failure occurs, and Time To Repair (TTR) shows the degree of difficulty involved to repair the software after a failure occurs.

Mean Time Between Failures (MTBF), which is defined as the average time between failures in a system, is an important measure of software reliability. MTBF can be expressed in terms of Mean Time To Failure (MTTF) and Mean Time To Repair (MTTR), as in Equation 1.1.

$$MTBF = MTTF + MTTR \tag{1.1}$$

Analysis

The relationship between faults, errors and failures is not a simple mapping. Some faults will result in errors under certain conditions, whereas others will not; on the other hand, several faults may lead to one single error or one fault may result in several errors. From the viewpoint of severity, a system may still continue execution even with some alert or warning faults; however, sometimes a single design fault may terminate the system. More importantly, some faults will cause failures (errors at the interface layer) directly, but others will remain inactive during the entire system lifetime. A similar relationship holds between errors and failures. Some errors may have no impact on the final result, while some errors may change the output. The purpose of fault diagnosis is to locate the root cause of failures, however, the fact that not all errors may be detected makes fault diagnosis even more difficult. Compared with fault diagnosis, failure prediction tries to estimate the impact of the current state on the expected output in a system.

1.2 Motivation and problem domain

With the increased complexity of computing systems, especially for large-scale supercomputers, new challenges emerge in fault tolerance research. Existing techniques have often been unable to meet future demands. For example, next generation supercomputers may have an SMTTI[2] of 1 hour in 2013-2015 according to the analysis in [15], which means these supercomputers can only have continuous execution for an hour using current fault tolerance techniques. Thus new solutions for fault tolerance are urgently needed and research on proactive fault tolerance techniques is one of the directions that may offer better efficiency. Proactive fault tolerance tries to perform preventive actions before actual failures occur with the help of prediction techniques.

The ultimate goal of this thesis is to design, implement and evaluate a proactive failure management framework for next generation supercomputers. The main objective is to reduce the wasted computational time used for failure penalties and improve fault tolerance efficiency in large-scale supercomputers when running large and complex applications. The methodology used in the thesis to achieve these aims is to use a present generation supercomputer as an experimental platform, and add the capability of failure prediction to the current failure management framework, so that the system is able to take preventive actions before failures occur and the wasted computational time is reduced. Specifically, the IBM BlueGene/L is used for tests because it is capable of yielding the most advanced data available for this generation, and the architecture of this system is presented in Section 1.3.

Two issues are considered in the proposed framework; firstly, what is the functionality of the prediction model; secondly, how can the prediction model be integrated in the failure management framework so that the traditional failure recovery mechanism can take advantage of the prediction results.

[2]SMTTI denotes System Mean Time To Interrupt, sometimes it is replaced by MTTF, which means Mean Time To Failure.

In order to specify the failure prediction model, the new failure management framework needs to be established by consideration of the core issue of proactive failure recovery. Specifically, the coordinated checkpoint mechanism is the normal failure recovery approach used on supercomputers. In order to further improve the recovery performance, a prediction-based failure recovery method has been developed in this thesis based on the following process: problem statement, which analyses the current problems; theoretical model design, which builds a theoretical approach as the solution; then model implementation and evaluation, the target of which is to assess the model.

Having generated a theoretical model for a failure recovery approach which depends on failure prediction accuracy, the search is thus on for good prediction mechanisms. For the issue of failure prediction in supercomputers, several papers have been published and a common method for performing the prediction tasks is that of employing a Hidden Markov Model (HMM)[3] or one of its extended forms [88, 91, 92]. These approaches mainly conduct estimations of future status based on system measurements, such as system log files, etc. Specifically, conditional probabilities are used in the new prediction model to improve failure prediction accuracy.

As identified in the literature analysis, the task of failure prediction has been explored in terms of the following stages: firstly, characteristics of the system have been analysed based on the system logs, so that the key features can be observed and the prediction problem can be specified so that the aim is to predict future failure occurrences based on historical system events. Secondly, a methodology has been developed to model the prediction problem using conditional probabilities. Finally, the methodology has been implemented, tested and evaluated using real data from the IBM BlueGene/L.

1.3 Supercomputers and the IBM BlueGene/L

This section summarizes the main features of the world's top supercomputers in terms of processing speed and socket numbers. More specifically, for the IBM BlueGene/L, its key properties and, of specific relevance, the reliability aspects of its design are introduced.

[3]HMM is a form of *generative model*, which represents probabilistic models that are based on joint probability distributions.

1.3.1 Supercomputer overview

Petascale systems have emerged as the predominant solution for complex and challenging problems like weather forecasting, which can be clearly demonstrated by observing the Top500 (top 500 world's fastest computers) [2]. Much attention has been given to improving overall performance such that the number of sockets in these machines and the maximum performance has nearly doubled every year from 2005 to 2008, as presented in Table 1.1.

Systems	Year	Maximum performance	Number of sockets
RoadRunner	2008	1 PF	20,000
LANL Jaguar	2008	1/4 PF	8,000
Tacc Ranger	2007	1/2 PF	16,000
NMCAC (SGI)	2007	1/8 PF	3,500
NSCA Red Storm	2006	1/10 PF	13,000
ASCI Purple	2006	1/10 PF	12,000
UPC MareNostrum	2006	1/20 PF	5,000
Columbia	2005	1/20 PF	5,000

Table 1.1: Socket numbers and maximum performance of the best computers in Top500 from 2005 to 2008, 1PF means 1Peta FLoating point OPerations per Second (FLOPS).

Fault tolerance remains a challenging issue in supercomputers [15]. It has been estimated that Mean Time To Interrupt (MTTI) is less than 10 hours for systems having 32,000 to 256,000 sockets [97]. If this trend continues, the MTTI of top supercomputers would be approximately 1 hour by 2013-2015 [15]. In contrast, from failure statistics provided by several supercomputer centers including Los Alamos National Laboratory (LANL), National Energy Research Scientific Computing Center (NERSC), Pacific Northwest National Laboratory, Sandia National Laboratory (SNL) and Lawrence Livermore National Labs (LLNL), it can be clearly seen that the number of failures per processor in the different systems has remained stable in the period from 1996 to 2004 [15]. The failure statistics also suggest that even currently these systems may have as many as three failures per day, which means there is little chance for long running applications needing more than 8 hours continuous computing time to finish successfully.

Given their specialised nature, an effective fault tolerant mechanism for these supercomputers must be system-specific so that the particular system features have to be considered during algorithm design. The IBM BlueGene/L, which was at the time one of the world's top 10 fastest computers, is taken as the platform for the research in this thesis. Its architecture and key features are briefly introduced in the following section.

1.3.2 BlueGene/L analysis

The BlueGene/L of specific focus in this thesis is installed at Lawrence Livermore National Labs (LLNL) in Livermore, California. Several challenges have been addressed in its design: the networks were designed with hardware support for extreme scaling and collective operation; another design issue has been reliability, availability, and serviceability (RAS) support. Furthermore, it was designed at the application-specific integrated circuit (ASIC) level, and targeted for applications such as physical phenomena simulation, real-time data processing and offline data analysis. At the programming level, Message Passing Interface (MPI) support was designed for efficient distributed memory usage. Another core issue is low-power design [34].

The supercomputer has 128K PowerPC 440 700MHz processors, and they are deployed into 64 racks, each rack consisting of 2 midplanes. A midplane has 1024 processors, which means that the maximum number of processors assigned to a single parallel job is $2^{16} = 65,536$. System-on-chip technology is used to deliver a target peak processing power of 360 TeraFLOPS (trillion floating-point operations per second). The system is configured as a $64 \times 32 \times 32$ three-dimensional torus to have the highest aggregate bandwidth in this context. Each node can support up to 2GB local memory and contains a second PowerPC 440 processor which is primarily for handling message passing operations. Each processor has a 2 KB L2 cache and a 4MB L3 cache (made from embedded DRAM), with a fast SRAM array for communication between two adjacent processors [105], see Figure 1.4.

Reliability is a critical factor in architecture design. In the case of the BlueGene/L, overall system availability is achieved through both hardware and software level design. In hardware, redundancy techniques are utilized at system-level, node-level and network control components, such as power supplies, cooling fans, cables, network connection, etc. In addition, fault tolerant strategies are built into the system. The RAS scheme is applied to isolate and replace any failing node while restarting the failed job from the latest checkpoint, and a fault isolation scheme is used to locate and isolate failed components. Another important reliability factor is the simplicity in deployment due to the fact that homogeneous components are utilized in the system to a high degree.

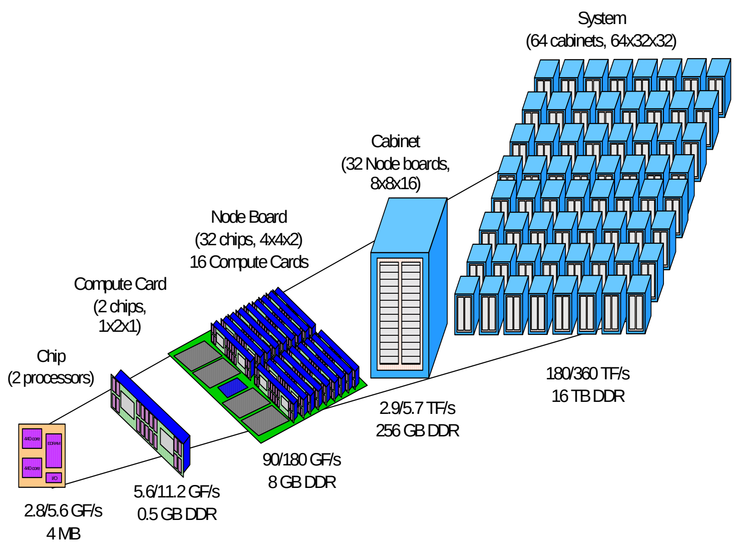


Figure 1.4: BlueGene/L overview (taken from [105]).

1.3.3 Key features

The prediction mechanism proposed in this thesis is built on key features and properties extracted via the analysis of both the IBM BlueGene/L system architecture and its output RAS log records. These features are described below.

- A large number of event records are available. The RAS logs are used for monitoring and prediction purposes, and the interval of the mechanism generating these log items is less than 1 millisecond. According to actual logs from the IBM BlueGene/L, the maximum event numbers each day reaches 381,827, which means there are 15,909 log records per hour on average. In one minute, the average number of events is 265.

- Failures are rare events compared to normal events. The RAS log files suggest that the failure events in total take up less than 10% of the whole data volume.

- Tasks run in parallel. Multiple tasks may have errors at the same time and call the RAS logging action simultaneously, which will cause the items to be recorded in a potential chaotic fashion during a short time scale. Similarly, multiple components may report the same failure event, so that large numbers of redundant items are recorded, which can be seen in the actual logs.

- The complexity of the BlueGene/L makes the prediction task more difficult. Utilization of homogeneous components can only reduce the complexity to a limited degree, and failure prediction is a challenging task even in a single machine. In such a supercomputer, the parallelism style, network connection, memory system, control mechanism and hardware configuration all increase the difficulty of log analysis and failure prediction.

1.4 Main contributions

The main contribution of this thesis is the development of a prediction-based failure management framework including two main parts: failure prediction and failure recovery. In terms of the proactive failure recovery approach, a theoretical model is created to express the expected execution time, analytical results demonstrate better recovery performance in comparison with the normal checkpoint mechanism. For the failure prediction mechanism derived by the proactive failure recovery model, experiments on industrial data from the IBM BlueGene/L show superior prediction accuracy compared with the other three published algorithms conducted on the same data. The following are contributions corresponding to each phase.

1. To the author's knowledge the first theoretical model is demonstrated in the thesis to express the performance of a prediction-based checkpoint mechanism. Theoretical analysis is conducted to investigate the relationship between overhead and prediction accuracy.

2. Following the literature analysis, two taxonomies of failure prediction methods are introduced according to two different aspects: (1) based on prediction methodology and (2) based on prediction algorithm. The first comparison of the most common probability models in terms of their independence assumptions is then provided in this thesis. Furthermore, a survey of the prediction approaches on the target platform: the IBM BlueGene/L is presented.

3. A mechanism for two-level failure prediction using a semi-Markov CRF model is demonstrated that not only predicts whether there is a future failure, but also identifies the specific failure category. To the author's knowledge, both CRF and semi-Markov CRF are for the first time here being applied to solve the problem of failure prediction.

1.5 Thesis outline

The remaining chapters of the thesis are organized as follows.

Chapter 2 discusses the traditional failure management framework and its core components, including failure detection and failure recovery. This chapter then gives an formal definition of the checkpoint mechanism technique and reviews the checkpoint-based failure recovery algorithms.

Chapter 3 introduces the failure prediction problem, followed by a survey of failure prediction methods. In order to have an insight of these methods, the survey is performed in terms of two aspects: methodology and algorithm; furthermore, three published failure prediction approaches on supercomputers are described in detail. It can be seen that the selection of probability model is important for prediction accuracy; the chapter then defines a number of common probability models, and compares these models in terms of their independence assumptions.

Chapter 4 starts by introducing the proactive failure management framework and the core components: prediction-based recovery approach. This chapter then develops a theoretical model to evaluate the performance of the proactive failure recovery approach, which is analysed according to four basic scenarios. Finally, the chapter finds the conditions under which prediction-based approaches perform better than normal approaches.

Chapter 5 describes the prediction process using the selected probability model applied to the key features of next generation supercomputers. Preprocessing is then discussed, including *Categorization, Filtering* and *Sequence extraction*. Next, the two-level failure prediction mechanism using the selected probability model is demonstrated. The next part of the chapter explains how the probability model solves the two basic prediction problems. Finally, the chapter reviews the training algorithms for the prediction model.

Chapter 6 provides the results of each step in the whole process. Specifically, this chapter starts by evaluating the data preprocessing process. Further, characteristics of the target platform are analysed in terms of normal event distribution and alert event distribution. Later, the training process and the parameter space are introduced, and the failure prediction mechanism is evaluated according to different parameters.

Chapter 7 concludes the thesis. It reviews the main contributions, then gives suggestions for future work.

Chapter 2

Failure management framework

Failure management involves a wide range of issues, such as failure detection, failure recovery, fault diagnosis, availability analysis, etc. In the case of real world systems, several of these issues are considered together and combined into a single failure management framework. However, different characteristics and requirements of these systems may diversify the frameworks. This chapter introduces a general framework for failure management and discusses its fundamental components in Section 2.1. Specifically, several failure detection and recovery techniques are also presented in Section 2.1.2 and Section 2.1.3, respectively. Checkpointing is a popular technique that has been successfully applied in various systems. A formal definition of this mechanism is given in Section 2.2.1 and a taxonomy of the main current checkpointing mechanisms is developed in Section 2.2.2. In the case of supercomputers, a coordinated checkpointing mechanism is a common choice for failure recovery, and related work is referenced in Section 2.2.3. The chapter is summarised in Section 2.3.

2.1 Framework introduction

Failure management is a challenging task in large-scale computing systems, and there exist a vast number of successful frameworks that can effectively handle runtime failures. These frameworks are different from each other in terms of functionality, platforms, structure, etc. However, they all have similar fundamental components of failure detection and failure recovery. This section presents an overview of a general failure management framework and describes in detail its basic services: failure detection mechanisms in Section 2.1.2 and recovery approaches (including *Checkpoint, Retry, Replication* and *Alternation*) in Section 2.1.3; it then discusses related issues,

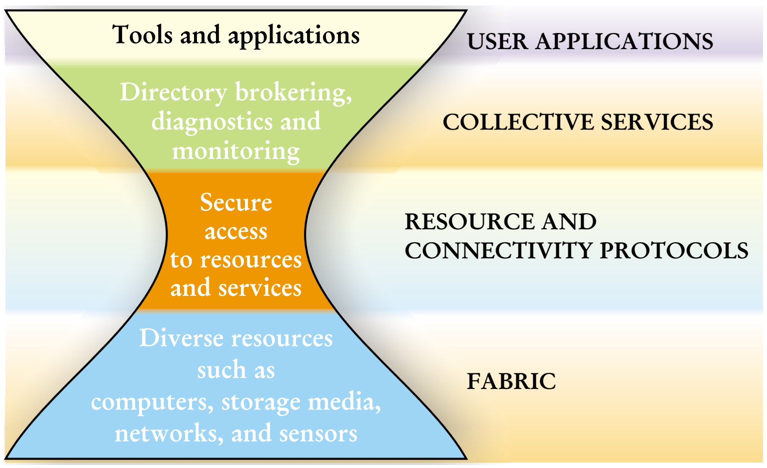

Tools and applications
Directory brokering, diagnostics and monitoring
Secure access to resources and services
Diverse resources such as computers, storage media, networks, and sensors
USER APPLICATIONS
COLLECTIVE SERVICES
RESOURCE AND CONNECTIVITY PROTOCOLS
FABRIC

Figure 2.1: General architecture (taken from [28]).

such as fault diagnosis and availability analysis in Section 2.1.4.

2.1.1 Architecture overview

The growing requirements of a diversity of applications necessitate large and powerful computational and storage capabilities. Parallelism in both computation and storage, involving many thousands of processors, is widely accepted as a method to meet these requirements. With the advancement and development of various technologies, the problems that need to be solved become more complex in computation and larger in size. Development teams focused on technical products, such as semiconductors, are using powerful computing capability to achieve higher throughput. Likewise, the business community is beginning to recognize the importance of distributed systems in applications such as data mining and economic modelling. Nowadays more and more domains try to use grid technology to solve their specific problems, such as the areas of genomics, bioinformatics, chemical engineering, geophysics, astronomy, etc.

These powerful systems must provide mechanisms for sharing and coordinating the use of diverse resources and thus enable the creation of different virtual computing systems that are sufficiently integrated to deliver desired qualities of service [27]. There are several such kinds of architectures in both industry and academia according to different focuses and applications, but they all have common system components and functions of resource sharing, authentication and resources access. As an example, a general architecture in grid computing [28] is presented in Figure 2.1 and these modules can be partitioned into four basic layers: fabric, resource and connectivity

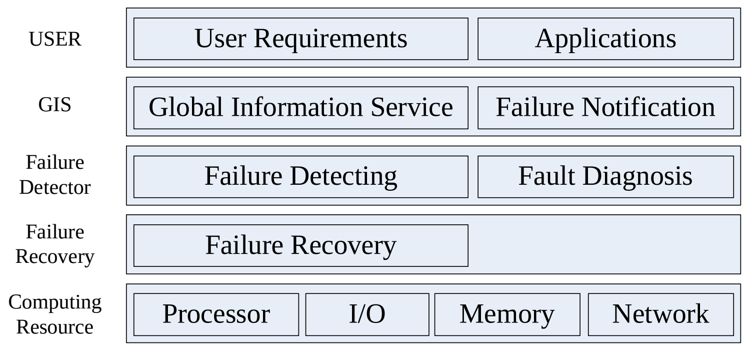


Figure 2.2: General failure management framework.

protocols, collective services and user applications. The fabric layer implements local resource operations that occur on specific resources. It needs coordination between operating systems, hardware and software. The resource and connectivity layer contains basic components, standard communication and authentication protocols. The collective service layer provides advanced functionalities such as resource management, performance steering, monitoring and so on. The application layer consists of computationally intensive, data intensive or communication intensive applications in specific areas; applications are constructed in terms of, and by calling on, services defined by the other layers.

There are lots of fault tolerance challenges in such large-scale systems. Even in a single desktop machine, failures and errors are common events. Large-scale computing systems increase the complexity by using thousands of cores, multiple storage devices, complex communication and cooperation control management, parallelism, and so on. Potential failures may occur in hardware or software. In some situations, it is impossible to trace the root cause of a failure because of the high-level of complexity. Such failures strongly impact system resource usage and overall performance, especially for long running and complex applications.

Failure management is a key issue in such systems and much work has been published on this topic. Figure 2.2 shows the components contained in a general failure management framework. There are three steps included in a failure management process: failure detection, fault diagnosis and failure recovery. Failure detection detects abnormal system behaviour and notifies the controller; the focus of diagnosis is to find the root cause of the failure; and a failure recovery approach is necessary to recover a system from failure status.

Taxonomy for architectures

Previous work on failure management can be classified into two groups: **reactive** approaches that take recovery actions *after* a failure occurs, and **proactive** approaches that take preventive actions *before* a failure occurs. Reactive approaches do not have any preventive cost but incur significant failure penalty. Many existing mechanisms follow this approach, such as Retry [31, 38, 16] and Replication [17, 14]. Proactive approaches offer better fault tolerance and integrate a failure prediction module to forecast future failure occurrence, based on which suitable actions can be taken before a true failure happens, such as turning on adaptive checkpointing.

2.1.2 Failure detection

Failure detection is a basic component in fault management that tries to locate failures during execution. The detection mechanisms are closely related to applications and platforms. Furthermore, large-scale computing platforms can be used for diverse applications and, based on the taxonomy given in [30, 29], these applications can be divided into five categories: distributed supercomputing applications, high-throughput applications, on-demand applications, data intensive applications and collaborative applications. Failure characteristics of different applications may be distinct; for example, it is easier for high-throughput applications to have input and output failures, whereas data intensive applications may have more disk and memory failures. These failures have many possible causes, such as memory leaks, network failures, etc. According to the classification stated in Chapter 1, there are four types of failure: crash failures, omission failures, timing failures and byzantine failures. In order to make applications behave robustly in the presence of failures at runtime, these failures must be detected and handled automatically. However, the detection mechanism varies from failure to failure. A special case is crash failure, during which a component enters a permanent erroneous state that can be detected by other components. More complex failure cases are also possible, such as Byzantine failures, where a component fails by not functioning correctly and may operate in a potentially malicious fashion. The problem of detecting crash failures is considered in this thesis, and many solutions have been published [4, 10, 31, 45, 51, 57, 106, 111].

Pinpoint [57] tries to aggregate low-level behaviours over a large collection of requests to establish a baseline for "normal" operation and then detect anomalies with respect to that baseline. Kiciman et al. [57] model a set of path shapes as a probabilistic context-free grammar (PCFG), which is a structure used in natural language processing to calculate the probabilities of different parses of a sentence. Combined with decision tree learning, the paper analyses component interactions to yield information about transient faults in an application. There are some limitations in the mechanism: the approach is only suitable for request/reply online applications and it only works for components with Multi-Modal behaviours (a component exhibits multiple modes of behaviour depending on its physical location); furthermore, it can only detect transient errors which change a component's interactions.

Soonwook et al. propose a generic failure detection service in [51]. The detection architecture comprises two components: a set of notification generators located on each node, generating notification messages submitted to the listener nodes, and a notification listener receiving generic events, "heartbeat" messages and task specific event notification messages. The framework mainly focuses on task-level failures and detects errors through continuous sending of "heartbeat" messages between nodes.

Condor-G [31] adopts ad hoc failure detection mechanisms because the underlying protocol ignores fault tolerance issues. It uses "periodic polling of the generic server" to detect certain types of failure such as host and network crash failures. However, it cannot detect task crash failures.

Alter [111] is another failure detection service organized in a hierarchical structure, which incorporates the techniques of unreliable failure detection service and the idea of relational grid monitoring architecture (R-GMA). The architecture works through periodically sending messages to failure detectors.

Youhei et al. propose a failure detection and recovery mechanism using a transactional agent model [104, 103]. A transactional agent can move to another operational computer if the target computer is faulty. The service focuses on system level failures in the client/server model; however, it cannot deal with real-time failures.

The Globus Heartbeat Monitor (HBM) [100] provides a generic failure detection service designed to be incorporated into distributed systems, tools or applications. It enables applications to detect both host/network failures, by recognition of missing heartbeat messages and task crash failures following receipt of notification messages from the HBM local monitor. However, in the context of [100], it is impossible for grid applications to detect failures other than fail-stop failures (shown in Figure 1.2).

2.1.3 Recovery approaches

Another key component in failure management is failure recovery, which recovers a system from an erroneous state to the normal state. Different types of failures should be able to be recovered from in a variety of ways depending on both the running application and the platform. For example, in the case of a short task, if not completed within an expected execution time, terminate the task, retry it on the same machine or restart it on a new machine; for a long running task, an appropriate recovery strategy is to generate checkpoints periodically and recover the task from the last good state when failure occurs. Several mechanisms have been published and the resulting approaches can be divided into four categories: retry, alternation, checkpoint and replication [114], which are described below.

Retry

Retry is a common way to deal with failures, especially in environments with an unstable nature, such as high-throughput computing or desktop grid platform. It restarts the task on the same resource once the task fails to finish; however, in some hardware failure cases, this approach cannot effectively recover the failure. Several systems apply this approach as it is relatively simply and has high efficiency [31, 38, 16]. For example, in the case of independent and short-running tasks, this approach can achieve good efficiency as it needs neither additional stable dedicated storage devices nor extra network bandwidth.

Alternation

The alternation approach is like Retry, but selects another candidate node[1] to which the failed tasks are submitted; this incurs the increased overhead of choosing an alternative node and transferring instructions, data and resource, etc. In terms of hardware failures, this mechanism should have good recovery performance [56].

Checkpoint

Checkpoint approach is a widely accepted way of recovering failures. It restarts the failed tasks transparently from the latest checkpoint, so that the task can continue its execution from the last known position before the point of failure.

[1]Node means a component or a computational resource here.

Replication

Replication is another common approach which will make several copies of the task and run these on different nodes simultaneously to try to ensure that at least one of the replicas completes successfully. The advantage of this approach is that it tries to ensure successful completion of the task, because if one of the task replicas fails during execution, there are other identical ones doing the same processing. However, in order to improve the success ratio, it will incur computing and network resources wastage. This approach has been studied in [17, 14].

2.1.4 Related issues in fault tolerance

Apart from failure detection and failure recovery, there are many other topics in fault tolerance, such as availability analysis, fault diagnosis, failure prediction (discussed in Chapter 3), etc. Availability analysis is concerned with long term system assessment based on its architectural properties or the number of bugs that have been fixed. Fault diagnosis is to identify the root cause of a problem [75]. It plays an important role in quality assessment and system maintenance. However, fault diagnosis is a challenging task in a complex environment. In order to analyse the problem, one would have to know both what is happening and what should be happening. This information can partly be obtained from system logs and monitoring schemes. However, two other problems arise: the first is that not all the errors and intermediate results can be captured; secondly, some results are a composite effect of several other events, which makes the diagnosis more difficult. Several mechanisms have been explored for fault diagnosis. Automated tests, which are similar to functional tests,[2] are used for fault diagnosis and recovery in [24, 75]. However, test units must be implemented for every software component in the system. Another issue is that this mechanism cannot detect certain errors, such as transient errors and Heisenbugs. [3]

[2]Functional tests are programs or scripts configured to test that packages (groups of clusters of classes) meet external requirements and achieve goals, such as performance. They include screen-driving programs that test GUIs from without.

[3]A bug named after the Heisenberg Uncertainty Principle that disappears or alters its behaviour when one attempts to probe or isolate it.

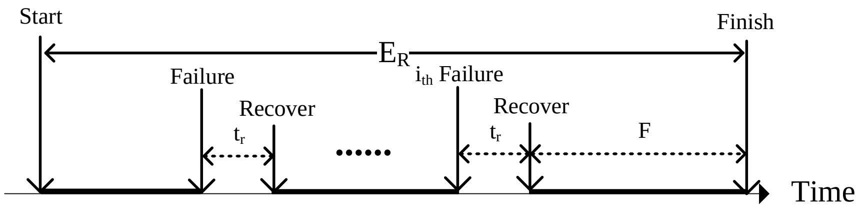


Figure 2.3: Execution without checkpoint.

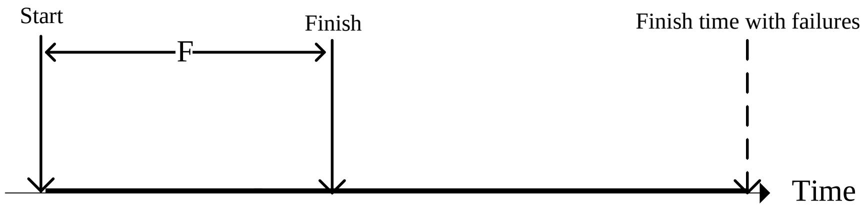


Figure 2.4: Execution without failures.

2.2 Checkpoint based failure recovery

Checkpointing is the most common failure recovery technique that can be applied in different systems. A large amount of work has been published on this topic and there are lots of diverse checkpoint mechanisms that suit different environments. This section gives a formal definition of a checkpoint mechanism, introduces its main parameters and shows equations to calculate the expected application running time under a traditional checkpoint mechanism. Two different taxonomies of checkpoint are developed, in terms of creator and coordinator,[4] respectively. In the case of supercomputers, a coordinated checkpoint mechanism is the most widely accepted technique and related work is presented in Section 2.2.3.

2.2.1 Checkpoint definition

Providing a checkpoint mechanism is an important step towards fault tolerance. It provides applications the capability to take a snapshot of the current executing state periodically or on command. The checkpoint data can be used to restore the application to its previous running state after a failure. However, constructing a checkpoint also increases system overhead in the form of additional network and storage load, and execution overhead in terms of periodically generating a checkpoint.

[4]Creator and coordinator are explained in detail in Section 2.2.2.

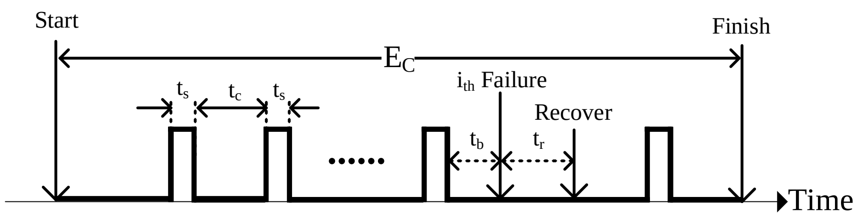


Figure 2.5: Different parameters in a checkpoint model, where each checkpoint overhead t_s and recovery time t_r are assumed to be constant, and t_c is the checkpoint interval, t_b is wasted computing time.

Checkpointing methods have been successfully applied for failure recovery in a wide range of systems. As stated in Section 2.1.2, there are a variety of applications and even members of the same category of application may have diverse requirements for failure recovery. More importantly, the checkpoint mechanisms implemented in various systems are differentiated according to their specific demands. In order to express a checkpoint mechanism precisely, several parameters must be defined. Figure 2.3 shows the execution of a program during which there are no checkpoints, and the program will be restarted from the beginning once a failure occurs; Figure 2.4 presents a scenario where there are no checkpoints or failures during the entire running; Figure 2.5 demonstrates the checkpoint model, where

t_s = Average checkpoint overhead, that is the time required to generate a checkpoint;

t_c = Uninterrupted task execution time between two consecutive checkpoints;

t_b = Wasted computing time when a failure occurs, which denotes the time to rollback to the last checkpoint;

t_r = The combination of down time, during which the normal operation of the system is reestablished, plus the time to reinstate the checkpoint or original task;

E_C = Expected running time in the presence of both checkpoints and failures;

E_R = Expected execution time without checkpoints;

F = Running time without either failures or checkpoints.

According to Duda [25], if failure occurrence during an execution follows a Poisson distribution, and the failure rate is λ, the value of E_R is given by Equation 2.1 where e is the base of natural logarithm. The execution scenario is illustrated in Figure 2.3 and a successful execution requires a time F of uninterrupted operation of the system. The following assumptions are adopted.

1. The occurrences of failures form a Poisson process of parameter λ.
2. Failures do not occur when the system is recovering from a failure.
3. The instants of the detection of the failures are considered as the instants of their occurrence.
4. λ and F are constant during the execution.
5. t_r is ignored in this case because no checkpoints are restored and t_r denotes the time to restart the program.

$$E_R = \frac{e^{\lambda F} - 1}{\lambda} \tag{2.1}$$

The value of E_C is given in Equation 2.2 (also from [25]) and the execution scenario is illustrated in Figure 2.5. The first four assumptions of Equation 2.1 and the following ones will be adopted.

1. Failures do not occur during the construction of a checkpoint.
2. t_c is constant during the execution.

$$E_C = \frac{F}{t_c}(t_s + (t_r + \frac{1}{\lambda})(e^{\lambda t_c} - 1)) \tag{2.2}$$

It can be concluded that the checkpoint interval t_c significantly affects the overall running time E_C. If the value of t_c is assumed to be constant, an approximation of the optimal value t_{c-opt} of t_c is given in Young [113] by Equation 2.3. Following assumptions are adopted.

1. The occurrences of failures form a Poisson process of parameter λ.
2. Failures can be observed at the occurrences.

3. λ and F are constant during the execution.

$$t_{c-opt} \approx \sqrt{2t_s/\lambda} \tag{2.3}$$

2.2.2 Taxonomy of checkpointing algorithms

There are a set of parameters in a checkpointing algorithm which can be adjusted to suit different environments; a large number of algorithms have been disscussed. In order to demonstrate the correlations and differences, two taxonomies are next presented in terms of two distinct rules: creator and coordination.

Creator based taxonomy

Based on different creators of checkpoints, checkpoints schemes can be divided into three categories: system-level checkpoints, application-level checkpoints and cooperative checkpoints [81, 98, 99].

System-level checkpoint

A system-level checkpoint is produced automatically and transparently at operating system or middleware level, and the currently running application can be seen as a black-box from which the complete process image of the applications is captured.

Application-level checkpoint

An application-level checkpoint is generated by the application and extra implementations need to be embedded to get real-time process information. In this case, checkpoint requests are inserted in the source code by programmers at appropriate places.

Cooperative checkpoint

In a cooperative checkpoint mechanism, whether to make a checkpoint or not depends on the application programmer, the compiler and the running system. The programmer inserts checkpoint requests in the code at places where the application state size is minimal; the compiler then evaluates the current state, removes incorrect and extra states and optimizes the intermediate outputs; at the runtime level, the system receives the checkpoint request and makes the final decision of whether to construct checkpoints by checking the current status of memory, hard disk, network traffic, I/O (input/output), etc.

The above three checkpoint mechanisms have different characteristics and are suitable for various environments. Table 2.1 compares them based on features, such as semantics, state size, portable, compiler, runtime, kernel state and transparency. *Semantics* means whether the checkpoint scheme skips useless data and stores only necessary state information; *state size* denotes whether a checkpoint is produced at places where the application state is minimal; *portable* expresses whether the checkpoint is platform independent; *compiler* means whether the application is optimised when doing checkpoints; *runtime* states whether the checkpoint decision is made at runtime; *kernel state* denotes whether kernel-level information can be accessed; finally, *transparent* describes whether user intervention is required.

Features	System-level	Application-level	Cooperative
Semantics		$\surd$	$\surd$
State-size		$\surd$	$\surd$
Portable		$\surd$	$\surd$
Compiler		$\surd$	$\surd$
Runtime	$\surd$		$\surd$
Kernel state	$\surd$		$\surd$
Transparent	$\surd$		

Table 2.1: Comparison of system-level, application-level and cooperative checkpoint mechanisms based on several key features (taken from [81]).

Coordination based taxonomy

Based on the required coordination where an application comprises multiple constituent processes, a further factor comes into play. Checkpoints can be broadly classified into four sub-categories: uncoordinated checkpoints, coordinated checkpoints, communication-induced checkpoints and diskless checkpoints [36, 98].

Uncoordinated checkpoint

In an uncoordinated checkpoint mechanism, each process saves the state independently; when failures occur, these processes will search the saved set of checkpoints for a consistent state copy, from which execution can be resumed. The main advantage is that each process can save its status when necessary; in order to reduce the space usage and checkpoint overhead, checkpoint action can be performed when the execution snapshot is small in size. However, in terms of recovery efficiency, an uncoordinated checkpoint may cause the problem of rollback propagation — the domino effect — which will make all processes restart from the beginning, and take the risk of losing previous useful work. To solve this problem, each processor may need to save multiple checkpoints.

Coordinated checkpoint

A coordinated checkpoint requires processes to coordinate in order to form a consistent global state that will avoid the rollback propagation problem, and only one checkpoint is needed during recovery, which reduces both storage space usage and network overhead. However, coordinated checkpoint schemes suffer from the large latency in saving checkpoints as a consistent checkpoint needs to be determined before it can be written to the stable hardware store. For some iterative applications, coordinated checkpoints can be generated between iterations, which can effectively reduce the time to identify the previous checkpoint and to transport the data.

Communication-induced checkpoint

In a communication-induced checkpoint mechanism, processes can generate parts of their checkpoints independently, and the other checkpoints are taken based on application messages received from other processors to avoid the domino effect. The communication-induced checkpoints must be produced before the application messages are processed, which will increase the system's latency.

Diskless checkpoint

The diskless checkpoint technique requires no stable store when saving checkpoints, especially in a distributed environment; the checkpoint data is saved in free memory space to improve overall system performance. However, the recovery of a failed processor needs checkpoints from all other application processors and some extra communication processors that are used for global consistency.

2.2.3 Related work on coordinated checkpoint

The most widely accepted checkpoint mechanism for supercomputers is coordinated checkpoint because of its simplicity; it requires that all processes coordinate to form a global state before saving individual checkpoints, and the recovery scheme just restores the last global checkpoint when a failure occur. Many coordinated mechanisms have been published for supercomputers, especially the IBM BlueGene/L [13, 71, 83].

Wang et al. [83] present a coordinated checkpoint mechanism for large-scale supercomputers considering failures during checkpoint generation and failure recovery. Two kinds of failures are taken into account in the mechanism modelled by Möbius [9]: error propagation problems and general crash failures. With respect to failure distribution, a hyper-exponential distribution [63] is assumed. Because all processes need to coordinate for a global state, a timeout threshold is set to prevent indefinite waiting.

An analytical checkpoint model for large-scale supercomputers is given by Slim et al. in [13]. The model can express expected application finish time using various distribution laws, and the optimal checkpoint intervals using different configurations are given with consideration of both checkpoint cost and the wasted computing time. Similar work is presented by Liu et al. in [71].

In [81], a checkpoint mechanism is developed for the IBM BlueGene/L using real-world trace results, from which the failure distribution model is derived. And the checkpoint decision is made depending on the likelihood of a failure occurrence at a future point.

2.3 Summary

This chapter has introduced the concept of a failure management framework for large-scale computing systems, and has analyzed its fundamental components: failure detection and failure recovery. Diverse failures tend to have different root causes and thus are handled in various ways. A brief survey of failure detection mechanisms has been discussed. In terms of failure recovery, there are four basic approaches: *Retry, Checkpoint, Replication* and *Alternation*. Apart from failure detection and failure recovery, there are several other issues, such as availability analysis, fault diagnosis, failure prediction, etc., that can assist in reducing adverse effects on runtime of failures.

The second goal of this chapter was to present related work on the most common failure recovery technique — the checkpoint. A definition and a comprehensive survey of checkpoint algorithms have been given. Coordinated checkpoint, which is widely accepted as the failure recovery approach in supercomputers, have been referenced and discussed.

Contributions of this chapter

This chapter has provided an review of the general framework for failure management in large-scale computing systems. Secondly, this chapter presents comprehensive taxonomies in terms of two distinct rules: creator and coordination.

Relation to other chapters

This chapter has presented background related to failure management, then introduced checkpoint-based failure recovery approaches, based on which the proposed proactive recovery mechanism is developed in Chapter 4.

Chapter 3

Failure prediction methods review

The challenges of real-time failure prediction are addressed in this chapter, and the problem definitions are presented in Section 3.1. A vast amount of work has been published in the area of failure prediction, for example, for the specific problem of real-time failure prediction on the IBM BlueGene/L, there exist a variety of approaches. In Section 3.2, a survey of general failure prediction methods and prediction algorithms specifically designed for the IBM BlueGene/L are presented. For the problem of failure prediction, CRF models are considered later in this thesis. Section 3.3 introduces the definitions of both the standard CRF model and the semi-Markov CRF model, presents some of their applications, and then compares them with other models.

3.1 Failure prediction statement

Failure prediction is a common term in the field of dependable computing either to assess the future reliability of a system according to its specification or to make maintenance adjustments based on historical events analysis [110, 73]. The term is used in a wide range of domains from software to hardware. In this thesis, the term is used specifically to denote real-time forecasting of failure occurrence at a future point using a specific range of historical system states or events.

Diverse failures have various root causes, and the same failure may have different sources. More importantly, failure prediction methods may differ among multiple systems according to their unique architectural design, data flows and component dependencies. This means that an effective failure prediction method is system-oriented and must take into account the specific characteristics of the system. This is particular

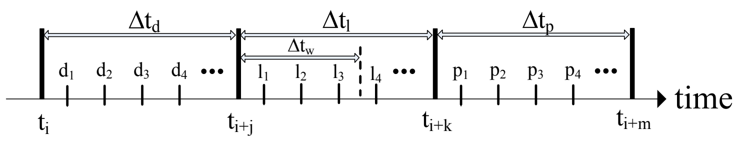


Figure 3.1: Temporal concepts in the failure prediction process ordered by time t, where $m > k > j$, Δt_d is data window, Δt_l is lead-time window, Δt_w is warning time window, Δt_p is prediction period, and d_i, l_i, and p_i are events that occurred in the various windows.

true with high performance systems which usually make use of customised and system-specific components. In dependable computing, failure prediction can be grouped into two classes from the perspective of time scale: real-time failure prediction (online failure prediction) and long-term failure prediction (reliability analysis). Real-time failure prediction forecasts for a short time period, such as ten minutes ahead. Four different time parameters are defined for the purpose of analysis, as shown in Figure 3.1.

Δt_d defines the time length of the data window used for failure prediction. Not all prediction algorithms use the same method: some approaches, such as Markov Process-based mechanisms, use only the current system status, while others take into account the system status during a short time period just before the current time. However, some totally different measurement might be applied, for example, some algorithms do not use a time window but rather use a fixed number of messages or events. Δt_d therefore has a different meaning according to the various mechanisms.

Δt_l defines the time span from the current time to the future point at which a failure may occur. The value of Δt_l must be carefully selected in online failure prediction mechanisms: if it is too short, administrators may not have enough time to recover or solve the potential problems; whereas, a long lead-time may affect the prediction accuracy.

Δt_w is termed the warning time window, which defines a minimum value for Δt_l.

Δt_p defines the time window during which a predicted failure is expected to occur. The shorter the length of the prediction window, the lower the accuracy of the result is likely to be. A longer Δt_p may increase the precision of forecasting where it is unclear exactly when a future failure will occur, but it is unlikely to be of use in real-time.

Formalization

Each event has an associated occurrence time — this association is described as an attribute-value pair. Given E as a set of event types, an event can be depicted as (e, t), where $e \in E$ is an event type and t is the occurrence time. More generally, the event type e can contain multiple attributes or features. However, sometimes we need to consider a sequence of events, termed an event sequence, that are ordered by time occurring in a fixed period. There are three aspects needed in order to specify an event sequence: event type e, start time t_b and end time t_e, so that an event sequence s, as shown in Equation 3.1, can be represented as a triple (e_i, t_b, t_e), where $e_i \in E$.

$$s = \langle (e_1, t_1), (e_2, t_2), \cdots, (e_n, t_n) \rangle \text{ for } e_i \in E, t_i \leq t_{i+1}, t_b \leq t_i \leq t_e \tag{3.1}$$

For example, in Figure 3.1 event sequence s_d in data window Δt_d can be written as:

$$s_d = \langle (d_1, t_{i+1}), (d_2, t_{i+2}), (d_3, t_{i+3}), \cdots \rangle$$

and event sequence s_l in lead-time window Δt_l can be written as:

$$s_l = \langle (l_1, t_{i+j+1}), (l_2, t_{i+j+2}), (l_3, t_{i+j+3}), \cdots \rangle$$

3.2 Survey of failure prediction methods

This section discusses previous research on failure prediction. In order to demonstrate the differences between various methods, two types of taxonomy are provided: a taxonomy of general prediction methods and a taxonomy of specific prediction approaches designed for RAS data from the IBM BlueGene/L. Section 3.2.1 explains the general prediction methods in terms of two aspects: algorithm and technique. Three prediction approaches for the IBM BlueGene/L data are analyzed in detail in Section 3.2.2.

3.2.1 Failure prediction methods taxonomy

Much work has been done in the area of real-time failure prediction. As mentioned in Section 3.1, the efficiency of failure prediction methods depends on several aspects, such as system infrastructure, application characteristics, component dependencies, etc. Various systems, such as telecommunication systems, computer clusters, electronic systems, etc., have their specific characteristics; even the same system running diverse applications may have different requirements for failure prediction approaches. These approaches differ from each other in two aspects: methodology and algorithm. From the viewpoint of algorithms, taxonomies are based mainly on theoretical models. Different taxonomies have been published based on different criteria [112, 93, 107]. This section presents taxonomies for both methodology and algorithms, as shown in Table 3.1.

Criteria	Taxonomy
Methodology	Observation based approaches
	Symptom measurement oriented approahces
	Error detection approaches
Algorithms	Time series
	Markov models
	Bayesian classifiers
	Rule-based models
	Instance-based methods
	Others

Table 3.1: Taxonomies of failure prediction.

Methodology taxonomy

Depending on whether the current system state is evaluated and whether system event logs are assessed, failure prediction approaches can be divided into three main categories: *observation based approaches, symptom measurement oriented approaches,* and *error detection approaches.*

Observation based approaches

Failure prediction approaches that fall into this category normally have two process steps. The first step is to find some kinds of regularities in failures, such as failure distribution. The distribution function can then be fitted using historical data. There is risk that it is easy to reach a false conclusion because of the limited data volume. Empirical studies in [96, 78, 90] show that $MTTR$ and *Failure Rate* vary across systems, and that $MTTF$ is modelled better with a Weibull distribution than an exponential distribution, which has previously been widely accepted as the distribution function. Another feature needing consideration in parallel systems and clusters is concurrency. It can be seen from the raw system reports that failures are either repetitive in nature or occur close together in time or in space, which can be used to exploit dependencies among failures for future occurrence prediction [64].

Secondly, the current system state will be measured to predict the next failure occurrence time using the established distribution model.

In order to improve scheduling performance, Rood and Lewis [89] present a multi-state available model to predict host failures in Condor [20], a large-scale grid environment. The model monitors real-time CPU usage and divides host states into five categories: *user present*, *available* (to grid), *CPU threshold exceeded*, *job shutdown*, and *unavailable*. The prediction algorithm takes as input a resource's historical availability data and combines two approaches of analysing the resource's behaviour to determine the most probable next state. The first approach makes predictions based on the percentage of time a resource spent in each state during the analysis period; the second approach calculates the probabilities of the next state by counting the number of transitions from the *available* state to each of the other states.

Symptom measurement oriented approaches

Due to the cooperation of multiple components in a single structure, some faults will affect the system gradually before a failure is detected. An infinite loop will cause CPU usage to become especially high, and may even make the system unresponsive; a protocol problem in telecommunication systems is likely to increase network traffic, and may flood the network if the problem continues; other problems may gradually increase memory usage, and a memory exception will be caught later when there remains no free memory. All these symptoms are caused by faults in the system, which lead to failures becoming apparent some time later, but not immediately. Prediction approaches in this category attempt to make use of these side effects before they become a failure, and monitoring any such unusual changes may help forecast future failure occurrence.

Jiman et al. [47] describe an approach to choose the optimal checkpoint interval, based on the memory usage profile predicted by an adaptive time series model. The traditional recursive algorithm applied in their model is presented in Equation 3.2, where $\hat{\Theta}(t)$ is the parameter value at time t, $y(t)$ is the output value, $\hat{y}(t)$ is the expected output value, and $K(t)$ is a coefficient.

$$\hat{\Theta}(t) = \hat{\Theta}(t-1) + K(t)(y(t) - \hat{y}(t)) \tag{3.2}$$

Hamerly and Elkan [43] describe two Bayesian-based methods to forecast disk drive failures using SMART (Self-Monitoring and Reporting Technology) values from Quantum Inc. disk drives. The first method, named NBEM, is a mixture of naive Bayesian models trained by the expectation-maximization (EM) algorithm. For a data point X with the assumption of independency in each model, its probability is estimated by Equation 3.3, where every $P(n)$ is a naive Bayesian submodel and the calculation takes into account all submodels. Secondly, a standard naive Bayes classifier is explored using the same data set. Based on experimental results, they conclude that both models perform better than the failure prediction algorithm proposed in [50].[1]

$$P(X) = \sum_n P(X|n)P(n) \tag{3.3}$$

Error detection approaches

These kinds of approach make use of error events recorded in log files, which is the output of a system monitor mechanism. These error events are discrete items with well defined fields that can be classified into certain categories. In contrast, symptom measurement is normally represented by continuous values or observations, and they do not necessarily use log files. The key notion of log-based failure prediction has been described in Section 3.1. Event logs can reflect system internal actions to a certain degree and are sufficient for building up the prediction model. Furthermore, root failure causes may be diagnosed by analysis of failure items.

Several analysis methods can be used on event logs for the purpose of failure prediction. A straightforward analysis is to calculate the failure occurrence frequency. A widely accepted assumption in failure prediction methods is that there are more possibilities to have errors before a failure occurs. Prediction approaches can rely on frequency changes to perform forecasting. Several methods have been published based on this assumption [64, 61]. Another purpose of analysis is to identify rules that lead to failures, and these rules are represented

[1]The publication date of [50] was later than that of [43], but the former paper was originally submitted in 2000.

as events that occurred ahead of failures; more specifically, in the case of standalone failures, this method does not work. In order to improve the accuracy and efficiency, enough rules must be generated from the training data set to capture true failures. The approach presented in [108] falls into this category. Decision tree learning is another way of building failure prediction approaches. It is a common method used for decision support and the goal is to predict target events preceded by several conditions. Through decision tree analysis on event logs, a failure tree can be generated in which leaf nodes are failures and parent nodes are conditions under which failures will occur. Gu et al. [40] describe a decision tree, generated by the C4.5 algorithm, for failure prediction in distributed stream processing systems. They focus on latent software failures by monitoring both the component and host metrics, such as component input, output, average processing time, etc.

Stochastic models, such as Markov and Bayesian models, etc. can also be applied to event logs for failure prediction. These models differ from each other with regard to their dependency assumptions over observations, and the details are discussed in the next part of this section.

Algorithm taxonomy

Failure prediction can be conducted in various ways using different approaches. Several models have been applied in failure prediction, and they can be broadly classified into five categories: *Time series models*, *Markov models*, *Bayesian classifier*, *Rule-based models* and *Statistical methods*.

Time series models

Time series data is measured at successive times spaced at uniform intervals. These data have a natural feature of time ordering, which makes time series analysis distinct from other techniques. For example, continuous time series models are generally based on the fact that close observations in time will be more related than those further apart. Specifically, there are two types: linear time series models and non-linear time series models, of which linear models have been successfully used in the area of failure prediction. Several models have been explored in [90], such as Autoregressive Model (AR), Moving Average Model (MA) and Autoregressive Moving Average Model (ARMA), shown in Equations 3.4, 3.5 and 3.6, respectively, where c is a constant value, φ_i and

θ_i are model parameters, $\hat{X}_t$ is the expected value of X_t, and ε_t denotes white noise.

$$X_t = c + \sum_{i=1}^{n} \varphi_i X_{t-i} + \varepsilon_t \tag{3.4}$$

$$X_t = \hat{X}_t + \sum_{i=1}^{m} \theta_i \varepsilon_{t-i} + \varepsilon_t \tag{3.5}$$

$$X_t = \varepsilon_t + c + \sum_{i=1}^{n} \varphi_i X_{t-i} + \sum_{i=1}^{m} \theta_i \varepsilon_{t-i} \tag{3.6}$$

Markov models

A Markov model is a random process with the property that future states of the process depend only on the present state and the past does not affect the future (the Markov property). This property is an important assumption for stochastic processes and has been successfully used for forecasting in finance, economics, and computer science. In the area of failure prediction, three extended forms of the Markov model have been used successfully: the Discrete Time Markov Chain (DTMC), described in [92]; the Hidden Markov Model (HMM), presented in [91]; and the Semi-Markov Model (SMM), introduced in [88].

In a DTMC for failure prediction, each state represents an event category, and the transition possibility from one state to another state is calculated for future failure occurrence forecasting. DTMC is defined in a discrete data space $\Omega = \{v_i, i = 1, \cdots, n\}$, where each v_i represents a state and $\Psi(v_j|v_i)$ represents the transition probability from state v_i to v_j, denoted by Equation 3.8. Matrix Ψ is stochastic in that, for each v_i, every row $\{p_{ij}, v_j \in \Omega\}$ is a probability distribution and sums to one, as shown in Equation 3.7.

$$\forall i : \sum_{j=1}^{n} p_{ij} = 1 \tag{3.7}$$

$$\Psi(v_j|v_i) = \{p_{ij}, v_i, v_j \in \Omega\} \tag{3.8}$$

A DTMC process starts from an initial state $S_0 = v_0$, then transits from the current state $S_i = v_i$ to state $S_{i+1} = v_{i+1}$ based on the transition matrix Ψ; the

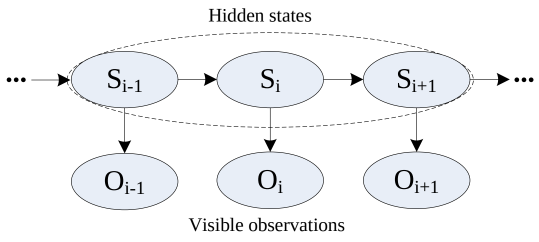


Figure 3.2: A Hidden Markov Model.

process (Ω, Ψ) can be expressed by Equation 3.9.

$$\Psi(S_{i+1}|S_0, \cdots, S_i) = \Psi(S_{i+1}|S_i) = \Psi(v_{i+1}|v_i) = p_{i,i+1} \tag{3.9}$$

In a HMM, every state S_i which is not directly visible has a corresponding observation O_i, and the observation is reachable. Both states and observations are drawn from finite spaces, Ω and O respectively, and each state has a probability distribution over its observations. Figure 3.2 shows a HMM. The sequence of observations reflect the sequence of states under two independency assumptions. Firstly, it is assumed that the state sequence follows the Markov property, which means that the current state only depends on its previous state. Secondly, it is assumed that each observation O_i depends only on the current state S_i. In order to formally express the model, the initial state of the instance needs to be defined. More specifically, two extra distributions are required: the first is an initial distribution $\Psi(S_i) = p(S_i) = p_i$ over states Ω, and the second is an observation distribution over its state $\Psi(O_i|S_i)$, where $S_i \in \Omega$ and $O_i \in O$. Then the joint probability of a sequence of states $S_{1,\cdots,n} = \{S_0, \cdots, S_n\}$ over an observation sequence $O_{1,\cdots,n} = \{O_0, \cdots, O_n\}$ is defined by Equation 3.10.

$$\Psi(S_{1,\cdots,n}|O_{1,\cdots,n}) = \prod_{i=1}^{n} \Psi(S_i|S_{i-1})\Psi(O_i|S_i) \tag{3.10}$$

A semi-Markov Model extends DTMC models to time-dependent stochastic behaviours. In addition to the notion in DTMC, $d_{ij}(t)$ is defined in Equation 3.11 to denote a probability distribution for the duration of a transition from state v_i to state v_j. Then the transition probability $g_{ij}(t)$ in a SMM can be defined by p_{ij} and $d_{ij}(t)$ in Equation 3.12. A SMM process starts from an initial state $S_0 = v_0$, then transits from the current state $S_i = v_i$ to state $S_{i+1} = v_{i+1}$ based on the transition matrix Ψ' and time duration matrix T; the process (Ω, Ψ', T) can be expressed by Equation 3.13.

$$d_{ij}(t) = P(T < t | S_i = v_i, S_{i+1} = v_j) \tag{3.11}$$

$$g_{ij}(t) = p_{ij} d_{ij}(t) \tag{3.12}$$

$$\Psi'(S_{i+1}|S_0, \cdots, S_i) = \Psi'(S_{i+1}|S_i) = \Psi'(v_{i+1}|v_i) = g_{i,i+1} \tag{3.13}$$

Bayesian classifier

Classifiers are always trained by pairwise data sets represented as $< data, label >$ to identify the category of a target data, and the prediction is a supervised learning process. As an example, failure prediction can be achieved via classifying system variables directly as failure-prone or normal. For learning purposes, the training data set must be labeled with expert knowledge or experience. Bayesian classifiers, which are based on a joint probability model, have been exploited for failure prediction in [43].

Rule-based models

Rule-based approaches combine rule mining and classification techniques to capture rules (failure patterns) during which target events (failures) are preceded by conditions [90, 112]. These rules directly express the dependencies between conditions and failures, and the conditions here denote events that frequently precede a failure. More specifically, the condition of a unique failure type is not a single event, and may contain a sequence of events. Formalizing these conditions has a core role in prediction, during which the strategy contains three main steps: the first step is to identify event types that frequently precede target events within a fixed window; the second step is to validate that these event types uniquely characterize target events; and the final step is to combine the validated event types to set up a rule-based system for prediction. Vilalta et al. study a rule-based failure prediction mechanism on the IBM BlueGene/L data in [108].

Instance-based methods

Instance-based methods have been successfully used in large-scale distributed computing platforms to model host or CPU availability [52]; these methods also work for problems with statistical properties, such as failure occurrence frequency, event space features and event time features.

The approach in [50] is an example of using statistical methods for hardware failure prediction. Javadi et al. [52] model the CPU availability in a large-scale distributed system (SETI@home [3]) using statistical methods: CPU traces from around 230,000 computers have been collected and nearly 51% of the traces are found to follow a Gamma distribution. In the case of software failures, a statistics-based method is presented in [35]. In terms of local data collection, a framework based on the Simple Network Management Protocol (SNMP) is built up, and an agent tree module is designed. Robust locally weighted regression [19] and the seasonal Kendall test [37] are used to smooth the observed data for trend detection and estimation.

3.2.2 Failure prediction for the IBM BlueGene/L

The IBM BlueGene/L, which is one of the world's top ten fastest supercomputers, is known for its powerful computing capability, high degree of parallelism, self-healing design and high availability. Furthermore, several aspects have been considered in the infrastructure to improve system reliability, these include hardware redundancy, network fault tolerant control, system monitoring strategy, etc. However, the increasing number of CPU cores has a significant influence on system reliability, and fault tolerance problems are increasingly becoming a bottleneck in such systems. Apart from the existing fault tolerance mechanisms designed in the multiple hardware layers, reliability can also be improved on the software side. Several mechanisms have been applied to RAS logs from the IBM BlueGene/L to predict future failure occurrence. From the experiments and results that have been published, it can be easily concluded that these proactive strategies can significantly reduce system maintenance cost and improve system availability.

Table 3.2 summarises the RAS log based failure prediction approaches conducted for the IBM BlueGene/L during the past 10 years, in which diverse techniques have been applied: rule-based techniques, statistics-oriented prediction and mixed prediction frameworks. The corresponding approaches are described in detail below.

Approach	Technique	Year
Eventset	Rule-based prediction	2002
Nearest neighbour predictor	Instance-based prediction	2007
Meta-learning based predictor	Mixed techniques	2008

Table 3.2: Failure prediction approaches for the IBM BlueGene/L.

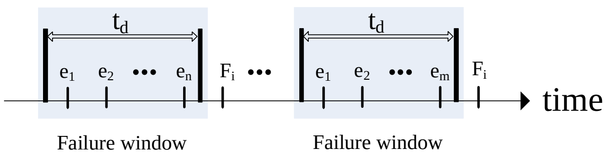


Figure 3.3: The failure window for the Eventset mechanism.

Eventset approach

The Eventset approach (reported in [108]) to forecast failures in the IBM BlueGene/L uses a rule-based method. An event sequence is denoted as $s =< d_1, d_2, \cdots, d_n >$ where $d_i = (e_i, t_i)$; and failures are expressed as target events e_{target}. Figure 3.3 shows a failure window to depict the event sequence s_{di} in a fixed time length t_d preceding failure event F_i. More specifically, the Eventset approach attempts to identify a series of event sequences $\{s_{di}\}$ that can capture as many failures as possible, and misidentify as few normal events as failures as possible.

The approach assumes that both events and failure window event sequences are from a finite space. Three steps are included in the mechanism, described below:

- Frequent event sequence identification: the first step is to find all the frequent failure window sequences s_{di} for each failure F_i. In order to remove rare sequences s_{rare} in the sequence space of S_{F_i} of failure F_i, *support* of each sequence s_{di} has been defined as shown in Equation 3.14, and sequences with *support* below a threshold will be removed from the space.

$$support(s_{di}) = \frac{\text{number of } s_{di} \text{ in } S_{F_i}}{\text{total number of sequences in } S_{F_i}} \tag{3.14}$$

- Confident event sequence identification: once the frequent event sequences are available, there remains a risk that may reduce prediction accuracy. For example, if sequence s_{d0} occurs ten times in the whole sequence space S_F for all failures, one occurrence causes failure F_0, and the other nine sequences lead to failure

F_1, it is easy to conclude that the probability of having failure F_1 is 90% once s_{d0} occurs. In the sequence space S_{F_0} of failure F_0, sequence s_{d0} needs to be removed. For this purpose, the *confidence* associated with sequence s_{di} is as defined in Equation 3.15, and sequences with a *confidence* smaller than a threshold will be deleted.

$$confidence(s_{di}) = \frac{\text{number of } s_{di} \text{ in } S_{F_i}}{\text{total number of } s_{di} \text{ in the whole space}} \tag{3.15}$$

- Rule building: in the remaining sequence space S_{F_i} of failure F_i, after the above processes, there may be redundant sequences. For example, let $S_{F_0} = \{s_{d0}, s_{d1}, s_{d2}, s_{d3}\}$; if $s_{d0} \in s_{d1}$ and $s_{d3} \in s_{d2}$, then s_{d0} and s_{d3} are redundant sequences that need to be removed. This task is termed the *rule building* process.

Nearest neighbour predictor

Liang et al. [65, 68, 115] present a customized nearest neighbour approach, a statistical method, for failure prediction in the IBM BlueGene/L, and achieve higher precision compared to two other classifiers: RIPPER and Support Vector Machine (SVM). Figure 3.4 presents the methodology of the nearest neighbour predictor, which forecasts fatal event occurrence in a future window of duration t_d. The historical events in another two windows are evaluated: the current window, defined as the window with duration t_d preceding the prediction window; and the observation window, which denotes the window with a duration $4 * t_d$ preceding the prediction window.

Several features need to be processed before prediction:

- Identify the event occurrence numbers according to different severity levels in the current window. The events are classified into 6 categories: *INFO, WARNING, SEVERE, FATAL, ERROR* and *FAILURE.*

- Count various event occurrences in an observation window based on these severity levels.

- Generate the distribution of different event categories over the observation window. The whole observation period $(4 * t_d)$ is divided into n uniform time intervals and each short time interval has a duration of $\frac{4*t_d}{n}$. Then count the event occurrences in every time interval.

- Calculate the value of MTBF (see the definition in Section 1.1.1).

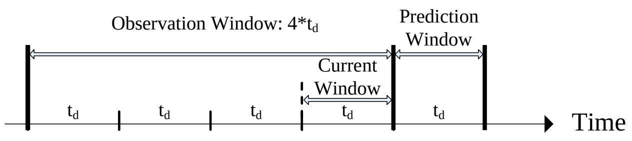


Figure 3.4: Demonstration of nearest neighbour failure predictor.

- Count the occurrence of keywords in the current window.

For the purpose of feature preprocessing, the numerical feature values need to be normalized as shown in Equation 3.16. Another parameter, the "Significance Major" $SIM(V)$, is defined in Equation 3.17.

$$\text{Normalized value } v' = \frac{\text{feature value} - \text{mean value}}{\text{standard deviation of the feature value}} \tag{3.16}$$

$$SIM(V) = \left| \frac{\text{mean of failure feature} - \text{mean of non-failure feature}}{\text{standard deviation of the feature value}} \right| \tag{3.17}$$

The failure prediction is conducted as follows: the whole feature data is divided into three parts, termed *anchor data, training data* and *testing data*. Anchor data is used to calculate the failure event intervals and normalized feature values; the nearest failure feature distances can then be computed between the training data and the anchor data; and the identified failure distances can be applied to forecast failures in the testing data.

Meta-learning based failure prediction

Lan et al. [39, 41, 62] present a dynamic meta-learning framework for failure prediction that combines the benefits of multiple prediction techniques, such as statistics-based methods and association rules. Another key element is that a dynamic approach is embedded in the framework to capture failure patterns while forecasting, and prediction rules are revised and added to the knowledge repository so that the accuracy can be gradually improved.

Figure 3.5 shows the framework with four main components. The first component does the task of data preprocessing. The analysis in [39, 41, 62] shows that, in the raw log files from the IBM BlueGene/L, large numbers of the same events, which are

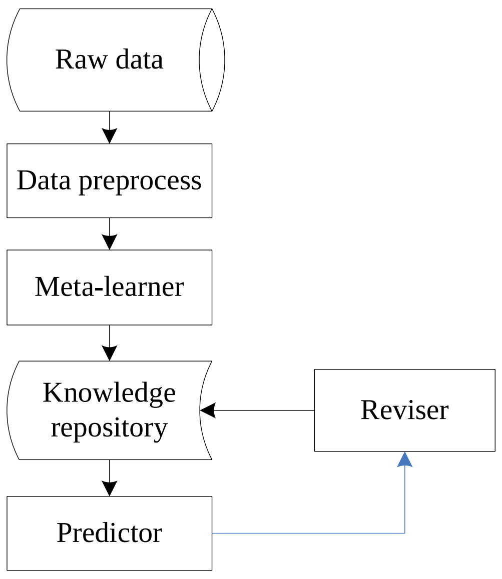


Figure 3.5: The dynamic meta-learning framework for failure prediction.

caused by a single error, are recorded in a short time interval. Much redundant information, such as numbers, hardware addresses, operators, etc., is recorded in each item. This noisy data must be removed before analysis and preprocessing can generate clean logs for meta-learning. Multiple prediction techniques are involved for training, and learned rules are stored in a knowledge repository. The next component, termed "predictor", does real-time failure forecasting according to the rules stored in the knowledge repository. The last key component, termed "reviser", monitors the system actively and does online evaluation of prediction results, so that it can check each rule in the repository, keep the most effective ones and revise or discard less effective ones.

Discussion

The above three published failure prediction approaches were designed for the IBM BlueGene/L platform and they applied mainly statistical or rule-based techniques which means that only straightforward dependencies between normal events and failure events were used for failure prediction; long dependencies and internal dependencies among events were not considered. Conditional Random Field models (CRFs) can express arbitrary relationships in events and these models are used for failure prediction in this thesis. Some related work is introduced in the following section.

3.3 CRF model for failure prediction

Conditional Random Fields (CRFs) were introduced by Lafferty et al. in 2001 [60] and have been used successfully for pattern recognition and hidden relationship mining, such as applications in natural language processing and image processing. Compared to other models, such as the Hidden Markov Model (HMM), the Maximum Entropy Model (ME) and the Bayesian Model, an important aspect about CRFs is that they relax the independence assumptions on observations, and they have the capability to describe arbitrary relationships, which exactly complies with the demands for relationship mining in the RAS logs, in which the records may have various kinds of dependencies. CRFs also overcome the label bias problems in Maxent [70] because they are probabilistic models based on conditional distributions and the state sequence is optimised over the entire observation sequence. Because of these reasons, CRFs have been chosen as the learning model here and they are applied in the area of failure prediction for what is believed to be the first time. Standard CRFs describe a discrete state-space, but failure patterns, represented as event sequences in the RAS logs, are events over continuous time. In order to make the model explicitly express such a scenario, an extension has been made to the standard CRF model to support continuous time, leading to the semi-Markov CRF Model, which is applied in this thesis.

Section 3.3.1 introduces the standard CRF model and the semi-Markov CRF model. Some applications of CRFs in different areas are then provided in Section 3.3.2. In Section 3.3.3, an overview of the probability framework is given, and probability models are classified into two categories: generative models and discriminative models. In order to differentiate various models, a comparison in terms of independence assumptions inside each model is performed.

3.3.1 Model introduction

CRF model

A CRF can be viewed as an undirected graphical model $G = (V, E)$, during which the states S are globally conditioned on O, the observation sequences, where $V = S \cup O$, and each random variable $v_i \in V$ either represents a state S_i or an observation O_i. Define $O_* = \{O_1, O_2, \cdots, O_n\}$ as the whole observation sequence and $S_* = \{S_0, S_1, \cdots, S_n\}$ as the corresponding state sequence in general; for a specific observation sequence and the corresponding state sequence, they are defined as

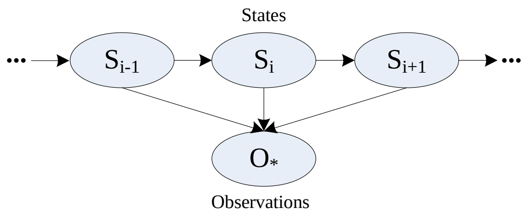


Figure 3.6: Chain-structured CRFs.

$O^{(i)} = \{O_1^{(i)}, O_2^{(i)}, \cdots, O_n^{(i)}\}$ and $S^{(i)} = \{S_0^{(i)}, \cdots, S_n^{(i)}\}$.[2] G is a bipartite graph, whose vertex space V can be divided into two parts S and O. Each state sequence, $S^{(i)}$, is globally conditioned on the observation sequence $O^{(i)}$. If the states S follow the Markov property, G is called a CRF model. Theoretically speaking, the structure of graph G may be arbitrary if it represents the conditional dependencies among states. However, in most real world problems, the states form a chain structure and the corresponding CRF model is called a chain-structured CRF model (shown in Figure 3.6).

According to the introduction in [60], two real-valued sets of functions are necessary to specify dependencies among global observations and correlation among different states:

- $\{H_m(S_i, O_*)\}_{m=1}^{M}$, real-valued state feature functions to demonstrate dependencies between the entire observation O_* and state S_i.
- $\{G_n(S_i, S_j, O_*)\}_{n=1}^{N}$, transition feature functions of the entire observation sequence O_*, the state S_i at position i and state S_j at position j.

The CRF model can be defined as $\Phi(H, G)$ on variables O and S, then the prediction result $P(S|O;\Phi)$ is shown in Equation 3.18, where $Z(O)$ is a global normalization factor, and β_m and μ_n are parameters that need to be estimated from training data.

$$P(S|O;\Phi) = \frac{1}{Z(O)} \exp\left(\sum_i \sum_{m=1}^{M} \beta_m H_m(S_i, O) + \sum_{i,j} \sum_{n=1}^{N} \mu_n G_n(S_i, S_j, O) \right) \tag{3.18}$$

Equations 3.19 and 3.20 show the Markov property of chain-structured CRFs. If feature functions H and G are combined into a single set of feature functions

[2]In this thesis, O and S are used to represent the whole sequence of observation and state, respectively, for simplification.

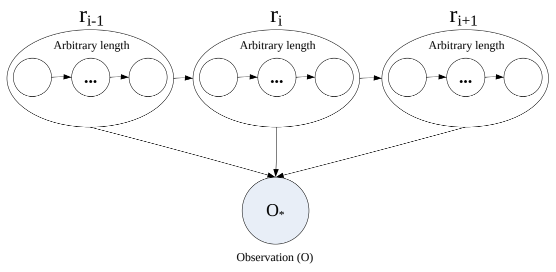


Figure 3.7: Semi-Markov CRF model.

$\{F_k(S_i, S_{i-1}, O_*)\}_{k=1}^{K}$, then F_k is either a state function or a transition function. The probability of the prediction result $P(S|O;\Phi)$ over the whole observation sequence can be written as Equation 3.21, Equation 3.22 shows the global normalization factor.

$$H_m(S_i, O) = H_m(S_i, S_{i-1}, O) \tag{3.19}$$

$$G_n(S_i, S_j, O) = G_n(S_i, S_{i-1}, O) \tag{3.20}$$

$$P(S|O;\Phi) = \frac{1}{Z(O)} \exp\left(\sum_i \sum_{k=1}^{K} \lambda_k F_k(S_i, S_{i-1}, O)\right) \tag{3.21}$$

$$Z(O) = \sum_S \exp\left(\sum_i \sum_{k=1}^{K} \lambda_k F_k(S_i, S_{i-1}, O)\right) \tag{3.22}$$

Semi-Markov CRFs

Traditional CRFs can only model discrete events or messages; they do not work well for continuous situations, which means they cannot analyze the time duration for which each state persists. Given that most real-world problems are best represented in terms of continuous features, this representation issue is a fundamental problem with CRFs. Clearly, a model that supports this feature is necessary and it must have a corresponding data representation. One way to satisfy the continuous time demand is to use a continuous segmentation $r_j = (s_j, e_j, S_j)$ over an arbitrary length of input sequences from start position s_j to end position e_j, rather than a normal segmentation over a unit length of input.

Semi Conditional Random Fields (Semi-CRFs) are a formalism to extend traditional CRFs with a continuous nature introduced recently by Sarawagi and Cohen [94], as shown in Figure 3.7. In a semi-CRF, $R = \{r_1, r_2, \cdots, r_n\}$ denotes all the prediction results, where each r_i represents a single result over an arbitrary length of observation $O_{s_i}^{e_i}$. The real-valued feature functions $F = \{F_1, F_2, \cdots, F_K\}$ map the triple (i, R, O) to a measurement. According to the new segmentation representation, the feature functions can be written as Equation 3.23.

$$F_k(i, R, O) = F_k^{'}(S_i, O, s_i, e_i) \tag{3.23}$$

In real world problems, most failures have life cycles and a system will not be affected by a problem that occurred a long time ago, which means that long-range dependency is not appropriate for this scenario and the current state is only dependent on the previous state. With consideration of the *Markov property*, the feature function F_k can be rewritten as Equation 3.24.

$$F_k(i, R, O) = F_k^{'}(S_i, S_{i-1}, O, s_i, e_i) \tag{3.24}$$

The resulting Semi-Markov CRF is shown in Equation 3.25, and this is the model that is applied later in the thesis.

$$\begin{aligned} P(R|O;\Phi) &= \frac{1}{Z(O)} \exp\left(\sum_i \sum_{k=1}^{K} \lambda_k F_k(r_i, O)\right) \\ &= \frac{1}{Z(O)} \exp\left(\sum_i \sum_{k=1}^{K} \lambda_k F_k(S_i, O, s_i, e_i)\right) \\ &= \frac{1}{Z(O)} \exp\left(\sum_i \sum_{k=1}^{K} \lambda_k F_k^{'}(S_i, S_{i-1}, O, s_i, e_i)\right) \end{aligned} \tag{3.25}$$

3.3.2 Applications of CRFs

Various forms of CRFs, such as general CRF, semi-Markov CRF and Hidden CRF, have been successfully applied in a variety of domains, including gene prediction, natural language processing, image processing, bioinformatics, global ranking, and others. Different applications are briefly presented in this section.

Gene prediction

In gene prediction, CRFs have been shown to outperform HMMs in most cases. Matthew et al. use a semi-Markov CRF for gene prediction and compare the results with a generalized HMM on the same data set; the semi-Markov CRF is found to outperform the HMM in cross-validation of two different gene structures [23]. Similar work has been done to identify genes and proteins in biological texts using a general CRF because of its capability of naturally incorporating arbitrary, non-independent features of the input without making extra assumptions [22, 53].

Natural language processing

A traditional CRF used for Chinese word segmentation achieved the highest F measure in four tracks of an international competition [116]. For the problem of Chinese lexical analysis, a Hidden semi-CRF has been applied to further improve prediction accuracy [101]. A similar experiment has been conducted using a semi-CRF in [94, 79]. In another framework, designed for sentence boundary detection, a CRF achieves a lower error rate when compared with a HMM and Maxent [70].

Image processing

CRFs have also been widely applied in image processing. A successful case has been to match laser scans generated by car navigating equipment, by considering arbitrary shape and appearance features [86]. In [109], Hidden CRFs are used for gesture sequence recognition, as they relax the limitations of conditional independency among observations in HMMs.

Others

There are many other applications using CRFs, such as the global ranking problem in information retrieval [84], phone classification [42], and object recognition [85].

3.3.3 Analysis and comparison

Section 3.2 presented the main techniques used for failure prediction in various domains. The techniques can be classified into two categories: *generative models*, including naive Bayes and the HMM model, and *discriminative models*, including logistic regression and the CRF model.

Generative models are based on a joint distribution $P(S, O)$ while discriminative models are based on a conditional distribution $P(S|O)$, which can be used to predict S from O, where O and S are both random variables. In order to define a $P(S, O)$, the distribution $P(O)$ must be identified by enumerating all possible observations on O. However, it is difficult to model $P(O)$ for most real-world problems. The first difficulty comes from the fact that it is impossible to observe all possible O or values of O, and only an approximate distribution $P(O)$ can be established. Secondly, the observations often contain highly dependent features which mean that the observation value in any given instance may only directly depend on state S at that time. More precisely, most real-world problems are best represented in terms of complex dependencies.

CRFs are a probabilistic framework for predicting states over a series of observations. Apart from the differences described above, the comparison with other models can be conducted in terms of various assumptions on dependencies (presented in Table 3.3). The Naive Bayes classifier is a basic generative model, in which both states S and observations O are assumed to be independent. One step further than the Bayes classifier, logistic regression assumes that the logarithm function of each state over the whole observation $\log P(S_i|O)$ is a linear function of O, which will maximize the distribution of each state S_i over the whole sequence O. In the case of the HMM model, as stated in Equation 3.10, two dependency assumptions can be derived that the states S follow the Markov property, and the observations O depend only on the corresponding states. However, there is a mismatch problem in a HMM between the learning objective function and the prediction objective function; a HMM learns a joint distribution $P(S, O)$, whereas a conditioned distribution $P(S|O)$ is used in the prediction task. Furthermore, the training of a HMM requires more computation due to the mismatch between learning and prediction, and the target conditional distribution is not maximized. The Maximum Entropy Markov Model (MEMM) [74], an enhancement to HMM, explicitly models dependency between each state and the full observation and saves the effort of modelling $P(x)$. However, a label bias problem will arise in some instances, which can be addressed by the CRF model because of the application of a global normaliser $Z(O)$; thus CRFs should model data like RAS logs from the

IBM BlueGene/L more accurately and improve prediction precision.

Category	Model	Assumptions
Generative	Naive Bayes	S is independent O is independent
	HMM	O is independent S depends only on neighbour states O depends only on corresponding S
Discriminative	Logistic regression	S is independent S depends on entire O Global normaliser Z is used
	MEMM	S depends only on neighbour states S depends on entire O
	Chain-structured CRF	S depends only on neighbour states S depends on entire O Global normaliser Z is used

Table 3.3: Comparison between different probability models.

3.4 Summary

This chapter has defined the problem of real-time failure prediction in terms of different time windows, provided a survey of failure prediction methods for large-scale computing systems, and introduced taxonomies based on two different aspects: methodology and algorithm. Three different existing failure prediction approaches conducted on the IBM BlueGene/L have been described in detail: eventset approach, nearest neighbour predictor and meta-learning based predictor.

The chapter has also introduced the CRF model that is used for prediction in this thesis. Based on the overview of probability frameworks, which can be grouped into generative and discriminative models, the main advantages of CRFs over other models, such as Bayesian classifier, logistic regression, HMM, MEMM, have been outlined: the independency assumption among observations is relaxed and the model is based on a conditional probability distribution. In order to support continuous time, the extended form — Semi-Markov CRFs — are used later in this thesis.

Contributions of this chapter

This chapter introduces the problem of failure prediction and a comparison among different probability models is performed in terms of various independency assumptions for the first time. Secondly, the CRF model is considered for handling the failure prediction problem, and an extension, Semi-Markov CRF, is used to improve prediction accuracy. This appears to be the first application of both CRF and Semi-Markov CRF in the area of real-time failure prediction.

Relation to other chapters

Chapter 2 covers related theoretical work in proactive failure management. An overview of the prediction approach is also presented in this chapter, and the detailed prediction mechanism, including preprocessing and prediction algorithm design, is described in Chapter 5. More specifically, the introduction of related work and the CRF model in this chapter is the theoretical foundation of the prediction mechanism presented in Chapter 5.

Chapter 4

Proactive failure recovery mechanism

The main idea of a proactive failure recovery mechanism is that a real-time prediction of failure occurrence is used to help construct checkpoints at appropriate times, so that wasted computational time can be significantly reduced. This chapter systematically addresses the area of failure management and identifies several components within the process, including failure detection and failure recovery. In Section 4.1, the overall framework is presented in order to demonstrate how the failure detection module interacts with the prediction-based failure recovery module, which is the core component of the framework. Section 4.2 further discusses the prediction-based checkpoint model and identifies four different scenarios in which failure prediction can influence checkpoint placement. Section 4.3 summarises the chapter.

4.1 Proactive failure recovery

The two fundamental issues in a failure management framework are *failure detection* and *failure recovery*. Failure detection monitors and identifies system misbehaviours and failure recovery adopts appropriate approaches to handle these misbehaviours. Following an introduction in Section 4.1.1, this section introduces basic failure recovery approaches and presents a comparison in Section 4.1.2. Following this, the architecture and the main components, including the prediction-based checkpoint module and the failure detection module, are described in Section 4.1.3. Finally, the overall failure recovery mechanism is discussed in Section 4.1.4.

4.1.1 Introduction

Fault tolerant system behaviour is increasingly important in supercomputing. As described in Chapter 2, there exist many fault tolerant approaches. A traditional way of achieving fault tolerance is checkpointing; a dump of the system state is taken at fixed intervals and, if a failure is detected during execution, the latest dump is reinstated and the computation restarted from that point. Under certain assumptions, the overall resulting run-time of each job is given by Equation 2.2, and the overhead compared with an error-free job running is given by Equation 4.1.

$$Overhead = \frac{F}{t_c}(t_s + (t_r + \frac{1}{\lambda})(e^{\lambda t_c} - 1)) - F \tag{4.1}$$

Minimising the associated overhead is as important as achieving reliable behaviour. A possible means of decreasing the overhead is to add a failure prediction mechanism that activates the dump "just-in-time" before a predicted failure. Such prediction usually entails some form of machine learning being applied to data produced by the system during run-time. This can be done combined with the periodic dump. The overall resulting run-time (and overhead) for each job now depends on the accuracy (precision and recall) of the prediction mechanism (details are given in Section 4.2).

4.1.2 Traditional failure recovery approach

Failure recovery is a core issue in fault tolerance, especially in a supercomputer environment which runs large-scale complex computational tasks of strategic importance. As discussed in Chapter 2, there are four fundamental categories of failure recovery approaches: *Retry, Alternation, Checkpoint* and *Replication,* which are appropriate in various situations. Thus for our target platform—the IBM BlueGene/L, a straightforward question arises:

Which failure recovery approach (or approaches) is the most appropriate?

In order to answer this question, for each recovery approach, two dimensions are considered: computational resource usage and application execution time. The optimum approach should complete the application in the shortest time using the least computational resource. However, each recovery approach has its advantages and disadvantages. For example, *alternation* can easily overcome hardware failures. Furthermore, various applications have different requirements, such as time synchronization systems that require some modules to complete their tasks simultaneously. Due to

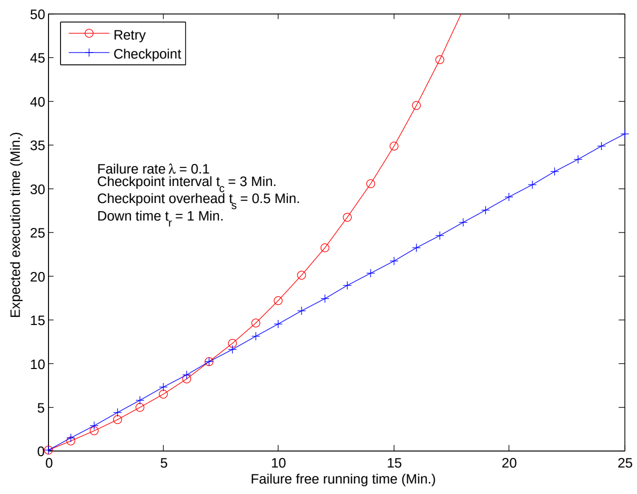


Figure 4.1: Result of expected execution time from Equations 2.2 and 2.1.

the characteristics of different recovery approaches and the varying requirements of applications, it is impossible to conduct a comparison that adequately addresses all cases. For the IBM BlueGene/L, in order to assess the performance of different recovery approaches, the following assumptions are made with regard to the most common scenarios:

1. Synchronized applications are not included in the comparison.
2. Computational resource usage is considered prior to application execution time because the resources are costly for supercomputers.
3. Permanent hardware failures are not taken into account.

By definition, *replication* makes several runs of a single task which implies significant waste of computational resources, while *alternation* requires more available resources but yields approximately the same completion time as *retry* assuming homogeneity. For *checkpoint* and *retry*, analytic expressions for the expected application execution time are shown in Equation 2.2 (page 37) and Equation 2.1 (page 37), respectively. Figure 4.1 illustrates the effect of these formulae on expected execution times using selected parameters under the retry and checkpoint failure recovery approaches for tasks of different size. It can be seen that short-running tasks favour use of retry, while long-running tasks benefit from use of checkpoint. In the middle ground, there is a crossover point, at which the balance of favour shifts from one scheme to the other. The value of execution time at the crossover is less than 10 minutes, given the

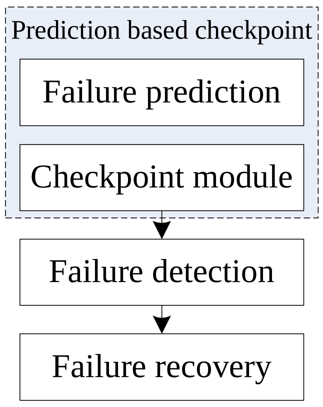


Figure 4.2: The architecture of the proposed failure recovery mechanism.

prevailing failure rate and parameters for the recovery schemes. For the IBM Blue-Gene/L, which is designed for large-scale applications, most executions last for days. Evidently, checkpoint is the most appropriate failure recovery approach here.

4.1.3 Proactive failure recovery mechanism

Figure 2.2 on page 30 shows the general failure management framework and itemises several fundamental components, some of which are essential, such as failure detection and failure recovery. In the proactive failure recovery mechanism for the IBM Blue-Gene/L, a failure prediction module is integrated into the framework to offer better fault tolerance, as shown in Figure 4.2. Three modules are involved in the framework: a prediction-based checkpoint module, a failure detection module and a failure recovery module.

The most common reactive checkpoint approach for supercomputers is described in [13, 71, 83] and the main idea can be summarized as follows:

> For a particular platform, analyse the output logs and identify the failure distribution that gives the best overall fit among the data, from which an optimal checkpoint interval value t_{c-opt} can be calculated. Then the established checkpoint model with this static interval value will be applied in the real-world platform.

Figure 4.3 demonstrates the above checkpoint approach. When a failure occurs, the last good checkpoint will be restored, and the computation between the restore point and the failure occurrence will essentially be discarded. In the proactive mechanism, a further improvement is made to the approach in that a failure prediction module is

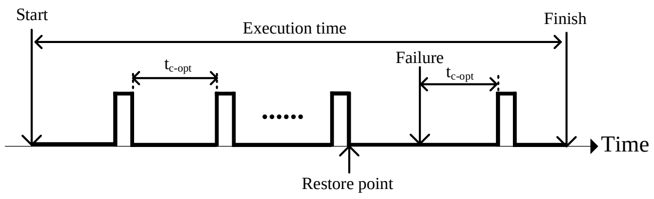


Figure 4.3: The most common checkpoint approach without failure prediction.

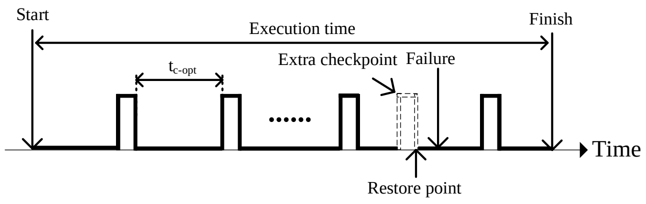


Figure 4.4: The proposed checkpoint approach with failure prediction.

integrated to reduce wasted computing, as presented in Figure 4.4. Once a failure is predicted, an extra checkpoint will be generated just before the predicted failure occurrence thus, ideally, significantly reducing wasted computation. The details of this approach will be discussed in Section 4.2.

Failure detection is a fundamental service that checks the system status periodically and locates failure occurrences. According to the way that detectors monitor and interact with components, there are two failure detection models, as shown in Figure 4.5, namely the *push* and the *pull* models [26].

In the push model, the component periodically sends heartbeat messages to the detector hence evidencing its healthy status; on the detector side, once the message is received, a timeout threshold is set that triggers a suspicion of failure if the next receipt of a live message from the same component occurs after this threshold. However, if there are a large number of components in the system sending heartbeat messages simultaneously, the potential for network congestion needs to be handled. In contrast, in the pull model, the detectors are active while the components are passive. The detector sends periodic live requests to the component and, in response, the component should send a reply to the detector. If the detector does not receive a reply before the end of a timeout threshold period, the component is suspected of having failed. The pull model can reduce the network load to a certain degree, depending on the number of request messages sent by the detector. The pull model is adopted in the framework for failure detection.

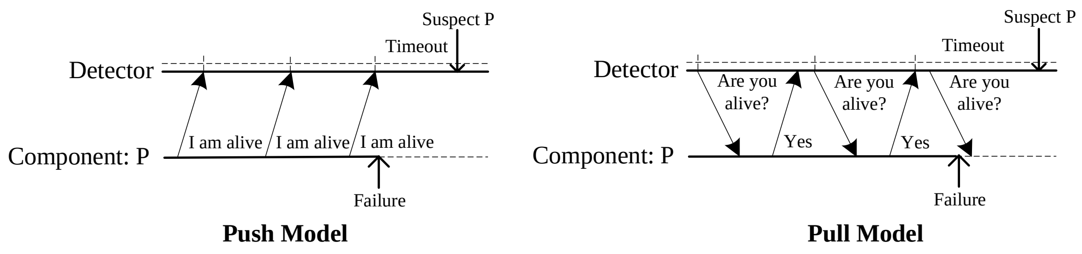


Figure 4.5: Two failure detection models: push and pull.

4.1.4 Summary

The proactive failure recovery mechanism implemented for supercomputers like the IBM BlueGene/L adapts a checkpoint approach by integrating a failure prediction module, because a checkpoint approach can efficiently offer fault tolerance capability while retaining past useful work.

The choice of an appropriate checkpoint interval is a core issue in the design of a checkpoint mechanism: too short an interval for generating checkpoints may cost the system much computational time; in contrast, too large an interval may result in a significant waste of useful computation. More specifically, the interval selection is closely related to the failure distribution; the lower the failure rate, the longer the interval, and vice versa. In terms of supercomputers, especially for a particular platform, a common way of choosing the optimum checkpoint interval is to fit the system historical failure data using a popular distribution model. However, the commonly used distributions can only capture and characterise the dynamism within the failure data to a certain degree due to the uncertainty of the failure occurrence. Furthermore, platforms differ one from another, and they have distinct failure distributions. Evidently, two problems may arise for the traditional coordinated checkpoint mechanisms:

1. The chosen distribution model can only approximately express the failure occurrence in real-world systems.

2. The optimum checkpoint interval varies from system to system which would require a pre-analysis of failure characteristics before the checkpoint mechanism is established.

Thus a prediction-based checkpoint mechanism is derived to offer better fault tolerance. The optimum checkpoint interval produced from the pre-analysis is adopted to generate checkpoints periodically. For improvement, a real-time prediction of future failure occurrence is conducted, and extra checkpoints will be made once failures are predicted. The system can then restore from the latest valid checkpoint once failures occur. This way, wasted computation can be further reduced. However, to what degree the prediction model will help improve the efficiency of the checkpoint mechanism depends mostly on the accuracy of forecasting. Relatively low prediction accuracy will lead to a consequent generation of a relatively large number of extra checkpoints, and thus incur even more computational waste. As a consequence, determination of what is an acceptable level of accuracy becomes a core issue, as will be discussed in the next section.

4.2 Prediction-based checkpoint model

The checkpoint mechanism proposed here is different to others in that a failure prediction module is integrated with the original coordinated checkpoint approach and the checkpoint placement takes advantage of the prediction results. Section 4.2.1 introduces the original checkpoint mechanism and demonstrates how a failure prediction module can influence checkpoint placement in terms of four distinct scenarios. An analysis of the mechanism in terms of time penalties is given in Section 4.2.2.

4.2.1 The coordinated checkpoint model

The growing requirements of complicated applications in industry and research demand powerful computational capability, which in turn has driven the development of highly parallel supercomputers, such as the IBM BlueGene/L. The resulting parallelism in execution, which may require cooperation between thousands of processors, is widely accepted as an important method to meet these requirements. For fault tolerance purposes, coordinated checkpoint methods, which require that all processes coordinate with each other during checkpointing to capture the global state in a consistent manner, are adopted in such systems. When failures occur, each processor will be restored from the last checkpoint. As there are no time differences for each individual processor when generating checkpoints in a coordinated checkpoint mechanism, all processes can be regarded as a single process for simplicity.

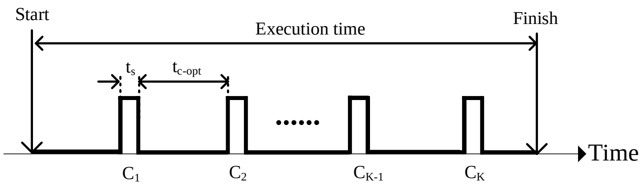


Figure 4.6: An execution with K checkpoints in the absence of failures.

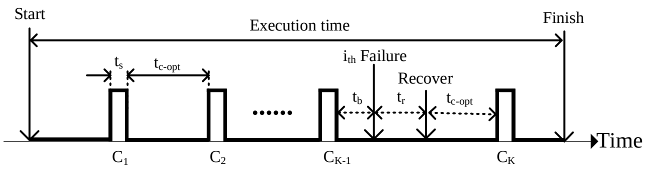


Figure 4.7: An execution with K checkpoints in the presence of failures.

Execution without failures

Given availability of the historical failure logs of a system, the optimum checkpoint interval t_{c-opt} can be worked out via a series of analyses, and an execution with K checkpoints in the absence of failures can be expressed by Figure 4.6, where checkpoints are represented as C_i $(1 \leq i \leq K)$.

If the time taken to generate each checkpoint is assumed to be constant, then the overall computational time wasted due to the construction of K checkpoints I_K can be calculated by Equation 4.2.

$$I_K = K * t_s \tag{4.2}$$

Execution with failures

If there are N failures during execution, the occurrence of each failure will lead to a further two segments of wasted time: t_b, which denotes the wasted computational time from the current point back to the last checkpoint, from which repeated computation is necessary, and t_r, the time used to restore from the last checkpoint, as shown in Figure 4.7. It is assumed that the time to restore from a checkpoint is constant and, if t_{bi} $(1 \leq i \leq N)$ represents the wasted computational time during the ith failure, then the overall wasted time W_N can be expressed as shown in Equation 4.3. If t_b is assumed to be the mean value of t_{bi} $(1 \leq i \leq N)$, then Equation 4.3 translates to Equation 4.4.

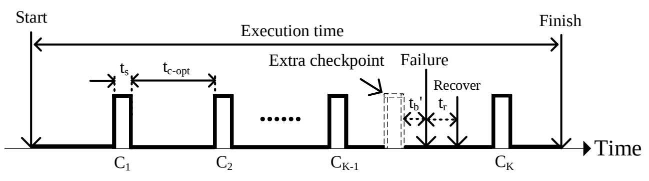


Figure 4.8: An execution using prediction-based checkpoint.

$$W_N = I_K + \sum_{i=1}^{N} t_{bi} + N * t_r \tag{4.3}$$

$$W_N = I_K + N * t_b + N * t_r \tag{4.4}$$

Checkpoint with prediction

Figure 4.8 demonstrates the proposed checkpoint mechanism integrating a failure prediction module by which a future failure occurrence can be forecast and an extra checkpoint will be constructed just before a correctly predicted failure occurrence. Similarly to the standard checkpoint mechanism, there are also two time penalties when a correct prediction is made: $t_b^{'}$ describes the time cost between the failure occurrence point and the extra checkpoint, and t_r denotes the cost for recovery from the checkpoint.

Compared with the checkpoint mechanism presented in Figure 4.7, the proposed mechanism will reduce the wasted computational time from t_b to $t_b^{'}$ when a prediction proves to be accurate. However, extra time penalties will be incurred in the case of an inaccurate prediction. More specifically, there are four prediction scenarios, as shown in Figure 4.9:

- *True positive*, denotes an accurate prediction of failure. There are three components of wasted time: $t_b^{'}$, t_r and t_s (the time used for the construction of a checkpoint).
- *False negative*, denotes an inaccurate prediction of failure. Two extra costs have been considered in this case: t_b and t_r.
- *True negative*, denotes an accurate prediction of normal behaviour, where no extra cost has been added to the overall execution time (other than the overhead of monitoring and prediction but this can be ameliorated by accurate prediction and recovery).

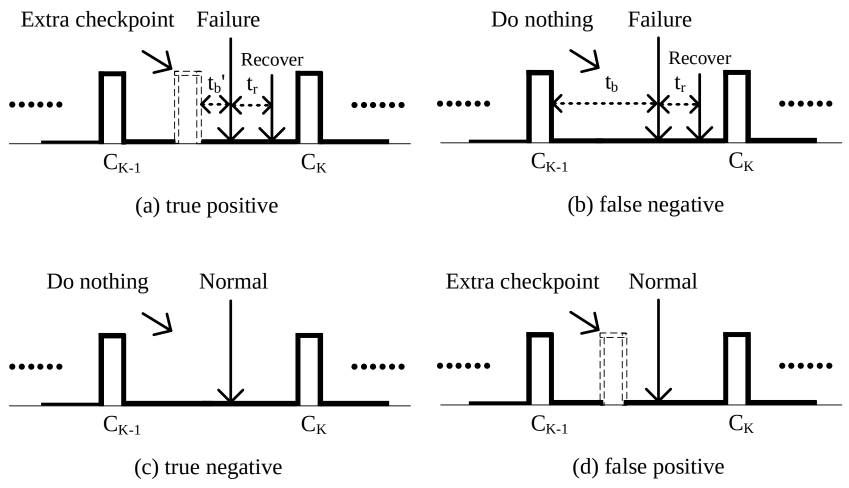


Figure 4.9: Four scenarios in a prediction-based checkpoint mechanism. Figure 4.9(a) describes an accurate prediction when one failure occurs, and an extra checkpoint will be made; Figure 4.9(b) shows an inaccurate prediction when a real failure occurs, and the system does nothing; Figure 4.9(c) presents an accurate prediction of normal behaviour; Figure 4.9(d) expresses the situation of normal behaviour when a failure is forecast and hence an extra checkpoint is constructed.

- *False positive*, denotes an inaccurate prediction of normal behaviour, where an extra checkpoint will be constructed leading to an extra time cost of t_s.

As presented in Table 4.1, each prediction is assigned to one of the four cases and the prediction accuracy depends on the number of occurrences of each case.

	Prediction results	
	Failure	Non-failure
True failure	True Positive (TP)	False Negative (FN)
True non-failure	False Positive (FP)	True Negative (TN)

Table 4.1: Confusion matrix.

If the *precision*[1] P and the *recall*[2] R of the failure prediction module, the total number of failures N are represented by Equations 4.5, 4.6 and 4.7, respectively, where N_{TP} represents the total number of true positives, N_{FP} denotes the total number of false positives, while N_{FN} is the total number of false negatives. Thus N_{TP}, N_{FP} and N_{FN} can be represented by P, R and N in Equations 4.8, 4.9 and 4.10, respectively.

[1]Precision gives the ratio of correctly identified failures compared with the total number of failure predictions

[2]Recall defines the ratio of correctly predicted failures compared with the total number of failures. The detailed content is further discussed in Chapter 6

$$P = \frac{N_{TP}}{N_{TP} + N_{FP}} \tag{4.5}$$

$$R = \frac{N_{TP}}{N_{TP} + N_{FN}} \tag{4.6}$$

$$N = N_{TP} + N_{FN} \tag{4.7}$$

$$N_{TP} = N * R \tag{4.8}$$

$$N_{FP} = (\frac{1}{P} - 1) * N * R \tag{4.9}$$

$$N_{FN} = N * (1 - R) \tag{4.10}$$

For each scenario, the occurrence numbers and the corresponding time penalty can be expressed analytically as shown in Table 4.2, where N_{TN} denotes an accurate prediction of normal behaviour. Thus the overall wasted computational time $W_N^{'}$ can be expressed as shown in Equation 4.11.

$$\begin{aligned} W_N^{'} &= I_K + (t_s + t_b^{'} + t_r) * N_{TP} + (t_b + t_r) * N_{FN} + 0 * N_{TN} \\ &\quad + t_s * N_{FP} \\ &= W_N + (t_s/P + t_b^{'} - t_b) * N * R \end{aligned} \tag{4.11}$$

Scenarios	Case numbers	Time penalty per case	Overall wasted time
(a)	N_{TP}	$t_s + t_b^{'} + t_r$	$(t_s + t_b^{'} + t_r) * N_{TP}$
(b)	N_{FN}	$t_b + t_r$	$(t_b + t_r) * N_{FN}$
(c)	N_{TN}	0	0
(d)	N_{FP}	t_s	$t_s * N_{FP}$

Table 4.2: Case number and time penalty of different prediction scenarios, where (a), (b), (c) and (d) represent the scenario of an accurate prediction of failure, an inaccurate prediction of failure, an accurate prediction of normal behaviour, and an inaccurate prediction of normal behaviour, respectively.

4.2.2 Overhead analysis

The proposed checkpoint mechanism differs from the original checkpoint mechanism in terms of when to construct checkpoints according to the four scenarios given in Figure 4.9. Evidently, an accurate prediction algorithm can help the system to establish checkpoints at appropriate times, however, inaccurate prediction leads to time wasted on constructing extra checkpoints. Compared with the original checkpoint mechanism, the time penalties of the prediction-based mechanism are listed in Table 4.3 in terms of the four scenarios. It can be concluded that the prediction module may reduce the time cost for scenario (a) while it results in extra wasted time in scenario (d).

Scenarios	Original time waste	New time penalties	Difference
(a)	$t_b + t_r$	$t_s + t_b^{'} + t_r$	$t_b - t_s - t_b^{'}$
(b)	$t_b + t_r$	$t_b + t_r$	0
(c)	0	0	0
(d)	0	t_s	$-t_s$

Table 4.3: Comparison between original checkpoint mechanism and prediction-based mechanism in terms of wasted time for each of the four scenarios introduced in Figure 4.9.

$$W_N^{'} - W_N = (t_s/P + t_b^{'} - t_b) * R * N \tag{4.12}$$

$$\begin{aligned} & W_N^{'} - W_N < 0 \\ \Rightarrow\ & (t_s/P + t_b^{'} - t_b) * R * N < 0 \\ \Rightarrow\ & t_s/P + t_b^{'} - t_b < 0 \\ \Rightarrow\ & P > t_s/(t_b - t_b^{'}) \end{aligned} \tag{4.13}$$

In order to analyse the improvement made by the prediction-based checkpoint mechanism, Equation 4.12 is derived from Equations 4.4 and 4.11. Obviously, the prediction module must meet the condition that $W_N^{'} - W_N < 0$, and the derivation process is shown in Equation 4.13. This leads to the conclusion that the prediction module must meet the necessary condition that $P > t_s/(t_b - t_b^{'})$ if it is to offer improved fault tolerance; in terms of the recall R, the more precise the prediction made, the more computational time the system saves. In terms of t_b and $t_b^{'}$, they have different values in various systems and several analytical solutions have been published in [13, 71]. In this thesis, t_b is estimated as $t_c/2$ approximately, and $t_b^{'}$ is ignored for simplicity purposes.

Comparison with traditional checkpoint model

The expected running time of applications when using traditional checkpoint mechanism is given in Equation 2.2. As analysed above, the overhead difference between traditional checkpoint mechanism and the prediction-based checkpoint mechanism is $W_N^{'} - W_N$, hence the expected execution time of applications when using the prediction-based checkpoint mechanism ($E_C^{'}$) can be expressed as Equation 4.14. It can be seen that the overhead of the proactive failure recovery approach now depends on the failure prediction accuracy (precision and recall). In order to investigate how execution time varies with relevant independent variables, especially in terms of prediction accuracy, Figure 4.10 shows the results of the formulae with three different sets of parameter settings. It can be concluded that the proactive checkpoint mechanism with better prediction accuracy will decrease the overhead compared with the traditional mechanism, however, a lower accuracy will add on an extra overhead.

$$\begin{aligned} E_C^{'} &= E_C + (W_N^{'} - W_N) \\ &= \frac{F}{t_c}(t_s + (t_r + \frac{1}{\lambda})(e^{\lambda t_c} - 1)) + (t_s/P + t_b^{'} - t_b) * R * N \end{aligned} \tag{4.14}$$

Dependence on precision

Figure 4.10 shows the effects of accuracy (precision and recall) on the prediction-based checkpoint mechanism. In order to investigate how the execution time varies with precision, recall is set at 100% for comparison. Figure 4.11 demonstrates the effects of the formulae with precision varying from 0% to 100%. It can be seen that when precision equals 28%, the prediction-based checkpoint mechanism has the same overhead as the traditional checkpoint mechanism, which means $P = t_s/(t_b - t_b^{'})$. When precision accuracy increases ($P > t_s/(t_b - t_b^{'})$), the curve declines and the prediction-based checkpoint mechanism performs better. Similarly, the traditional checkpoint mechanism is better when precision is lower than 28%.

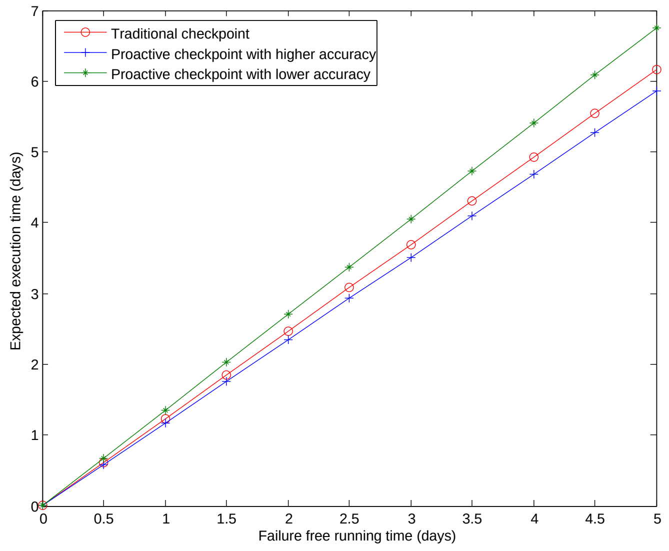


Figure 4.10: Comparison between traditional checkpoint mechanism and prediction-based checkpoint mechanism according to various application lengths when recall is 100%. The curve of the proactive checkpoint with higher accuracy (P = 100%) shows the effects when precision meets the condition $P > t_s/(t_b - t_b^{'})$, whereas the curve of the proactive checkpoint with lower accuracy (P = 10%) shows the results of the formulae when $P < t_s/(t_b - t_b^{'})$. Other parameters are configured that λ = 0.1, t_s = 0.2, t_c = 1.6, t_b = 0.8, and t_r = 0.2

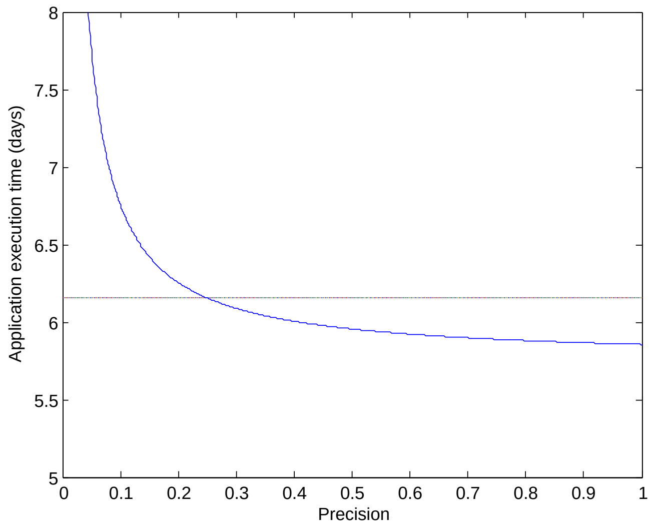


Figure 4.11: The effects of Equation 4.14 with varying prediction *precision* — the horizontal line in the chart denotes the application execution time of the traditional checkpoint mechanism. Other parameters are configured that R = 100%, λ = 0.1, t_s = 0.2, t_c = 1.6, t_b = 0.8, and t_r = 0.2

Dependence on recall

Equation 4.13 shows that the accuracy of precision determines whether the prediction-based checkpoint mechanism performs better, and Equation 4.14 shows that the amount of saved overhead is determined by the accuracy of recall. In order to show the effects of Equation 4.14 with various recall values, parameters are set to ensure that condition 4.13 is met. Figure 4.12 shows the results. It can be seen that the total amount of saved overhead has a linear relationship with recall, the higher the accuracy of recall, the greater the saved overhead from prediction-based checkpoint mechanism will be.

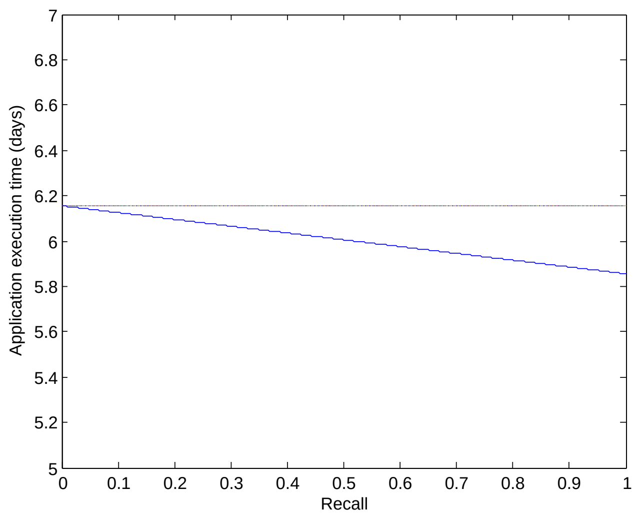


Figure 4.12: The effects of Equation 4.14 according to varying prediction *recall*; the running time using the traditional coordinated checkpoint approach has also been plotted. Other parameters are configured that P = 100%, λ = 0.1, t_s = 0.2, t_c = 1.6, t_b = 0.8, and t_r = 0.2

Dependence on precision and recall

It can be seen that the proactive checkpoint mechanism performs better when the precision accuracy meets the condition shown in Equation 4.13, while Equation 4.14 shows that both precision and recall can affect the execution time. Figure 4.13 shows the execution of an application with varying precision and recall. It can be seen that increasing precision and recall can both reduce the overall execution time (except at low precision).

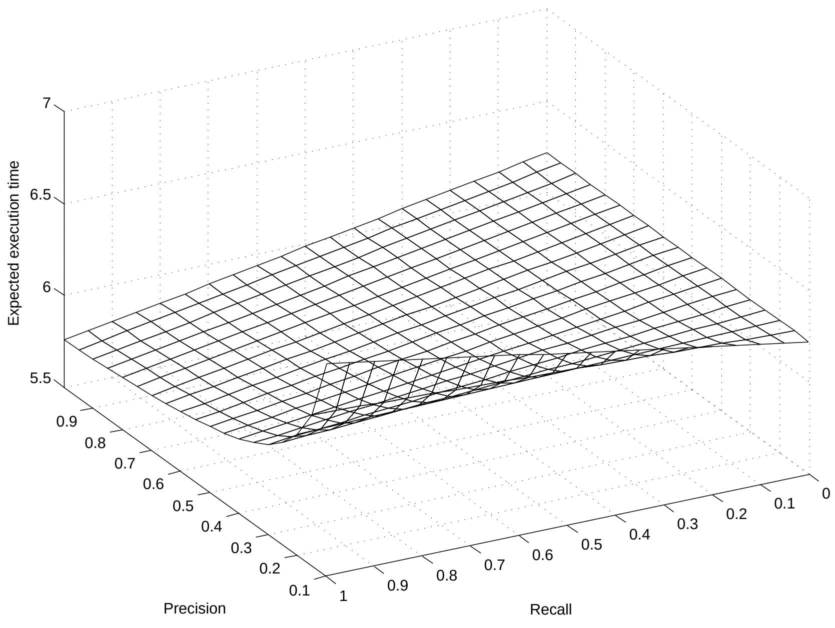


Figure 4.13: The relationship between expected execution time, precision accuracy and recall accuracy of the proactive failure recovery mechanism. 5-days failure free running time is chosen and other parameters are configured that λ = 0.1, F = 5 days, t_s = 0.2, and t_r = 0.2

4.2.3 Effects on checkpoint interval

The proactive checkpoint mechanism can improve the traditional coordinated checkpoint mechanism by using an extra failure prediction model when the prediction accuracy meets the condition shown in Equation 4.13, and a fixed value of t_{c-opt} has been chosen in the above analysis. However, a predictor with specific accuracy (precision and recall) will influence the choice of the optimum checkpoint interval t_{c-opt}, and the proactive checkpoint mechanism can be further improved by choosing a new checkpoint interval $t^{'}_{c-opt}$.

Figure 4.11 shows that when precision is lower than the threshold derived from Equation 4.13, it is better not to use the failure prediction scheme; otherwise progressive gains can be obtained from both precision and recall (see Figure 4.12), which lead to better choices for checkpoint interval. In the proactive checkpoint mechanism, a predictor with specific precision P and recall R will correctly predict $R * N$ failures, where N denotes the total failure numbers; thus there remain $(1 - R) * N$ failures that will incur actual performance penalties. The system can then be approximately modelled with a new failure rate $\lambda^{'}$, shown in Equation 4.15. Equation 4.16, which is derived from Equation 2.3, gives the new optimum checkpoint interval $t^{'}_{c-opt}$.

$$\lambda' \approx (1 - R) * \lambda \tag{4.15}$$

$$\begin{aligned} t'_{c-opt} &\approx \sqrt{2t_s/\lambda'} \\ &= \frac{t_{c-opt}}{\sqrt{1-R}} \end{aligned} \tag{4.16}$$

Figure 4.14 shows how the optimum checkpoint interval varies according to prediction accuracy. When $P < t_s/(t_b - t'_b)$, the traditional checkpoint mechanism performs better and the choice of optimum checkpoint interval is a fixed value t_{c-opt}, which can be seen from the first part of the curve. When $P > t_s/(t_b - t'_b)$, the proactive checkpoint mechanism can have a better choice of checkpoint interval t'_{c-opt}, which is shown in the second part of the curve. The value of t'_{c-opt} increases with the improvement of recall and when $R = 100\%$, t'_{c-opt} tends to infinity.

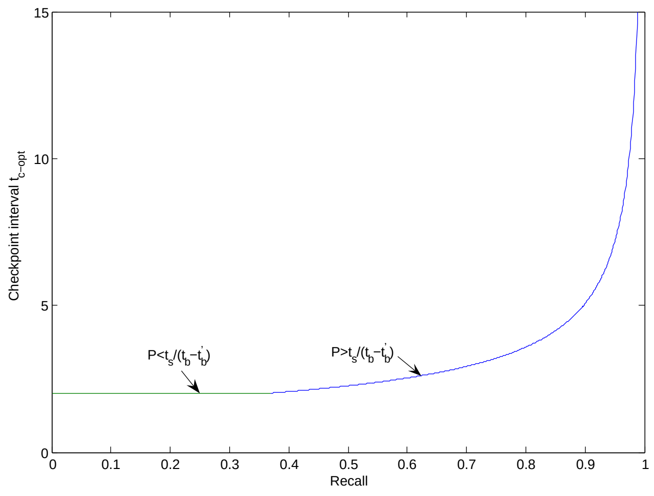


Figure 4.14: The choice of optimum checkpoint interval t_{c-opt} according to various prediction accuracy: precision P and recall R.

The above analytical results demonstrate the future trends of checkpoint interval when the accuracy of recall increases. In order to obtain the optimum checkpoint interval under specific parameter settings, Equation 4.17 that is developed in [13] is used to estimate the optimum checkpoint number, where L denotes the Lambert function [21]. The optimum checkpoint interval can then be obtained by Equation 4.18. Figure 4.15 shows the relationship between checkpoint number and recall accuracy. It can be seen that when recall approaches 100%, the optimum checkpoint number is close to 0, which demonstrates the equivalent results in Figure 4.14.

$$\hat{k} = \frac{F * \lambda}{1 + L(-e^{-1-\lambda(t_s+t_r)})} \tag{4.17}$$

$$t'_{c-opt} = F/\hat{k} \tag{4.18}$$

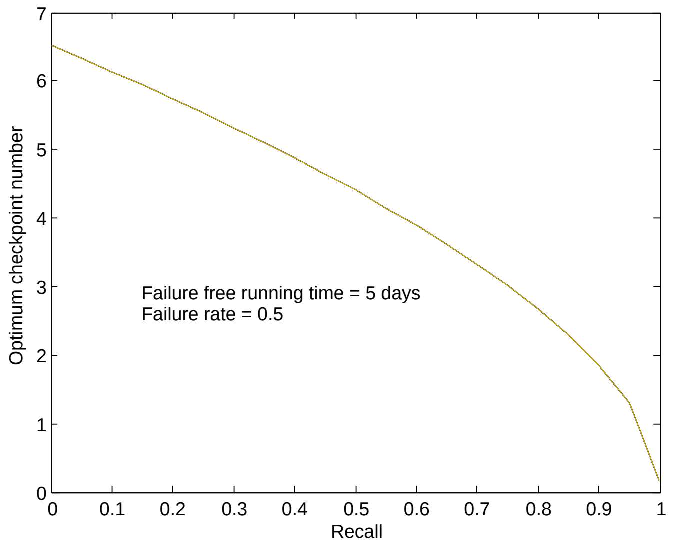


Figure 4.15: The choice of optimum checkpoint number for an application with failure free running time of 5 days in terms of different recall accuracy.

4.2.4 Analysis on further performance improvement

The above analysis has shown that different prediction accuracy will influence the choice of optimum checkpoint interval that is used by the proactive checkpoint mechanism, however, the mechanism described in Section 4.2.1 has chosen a fixed checkpoint interval for all instances. Accordingly, the overall execution time can be further reduced by choosing an optimum checkpoint interval in terms of the prediction accuracy.

It can be seen that the proactive checkpoint mechanism performs better when the precision accuracy meets the condition shown in Equation 4.13, and the checkpoint interval needs to be adaptively changed to achieve better performance (see Figure 4.14). Specifically, the execution time in a system using the proactive checkpoint mechanism can be divided into two parts for analytical purposes. Firstly, with consideration of correct predictions, the new failure rate λ' can be approximately derived by Equation

4.15 and the new execution time can be calculated by Equation 4.19, which is derived by Equation 2.2. Secondly, the system would generate extra checkpoints taking into account incorrect predictions, and the additional computational time that is added to the execution can be calculated by Equation 4.20, where N denotes the total number of failures.. Thus, the new overall execution time can be expressed by Equation 4.21.

$$E_{C1} = \frac{F}{t'_{c-opt}}(t_s + (t_r + \frac{1}{\lambda'})(e^{\lambda' t'_{c-opt}} - 1)) \tag{4.19}$$

$$E_{C2} = R * N * t_s / P \tag{4.20}$$

$$\begin{aligned} E'_C &= E_{C1} + E_{C2} \\ &= \frac{F}{t'_{c-opt}}(t_s + (t_r + \frac{1}{\lambda'})(e^{\lambda' t'_{c-opt}} - 1)) + R * N * t_s / P \\ &= \hat{k} * (t_s + (t_r + \frac{1}{\lambda'})(e^{F\lambda'/\hat{k}} - 1)) + R * N * t_s / P \end{aligned} \tag{4.21}$$

Figure 4.16 shows the execution of an application with varying precision and recall. It can be seen a similar trend as Figure 4.13, however, a fixed change in recall will reduce the execution time faster than the same fixed change of precision. When the accuracy of both precision and recall approach 100%, there will be no penalty cost and the execution time will be close to 5, which is equivalent to the failure free running time.

4.2.5 Assumptions of the model

This section has demonstrated the prediction-based checkpoint model (shown in Equations 4.13 and 4.14), which adds prediction facility to the traditional checkpoint model to reduce performance penalties; and the improved model (shown in Equation 4.21), which chooses an appropriate checkpoint interval according to the prediction accuracy.

Different equations have various assumptions. Specifically, Equations 4.13 and 4.14 are derived from Equation 2.2 and the figures of Equation 4.14 shown in this section are generated based on the assumptions of 2.2 (described in Section 2.2.1) and the following assumptions.

1. The optimum checkpoint interval t_c is used in 4.14.

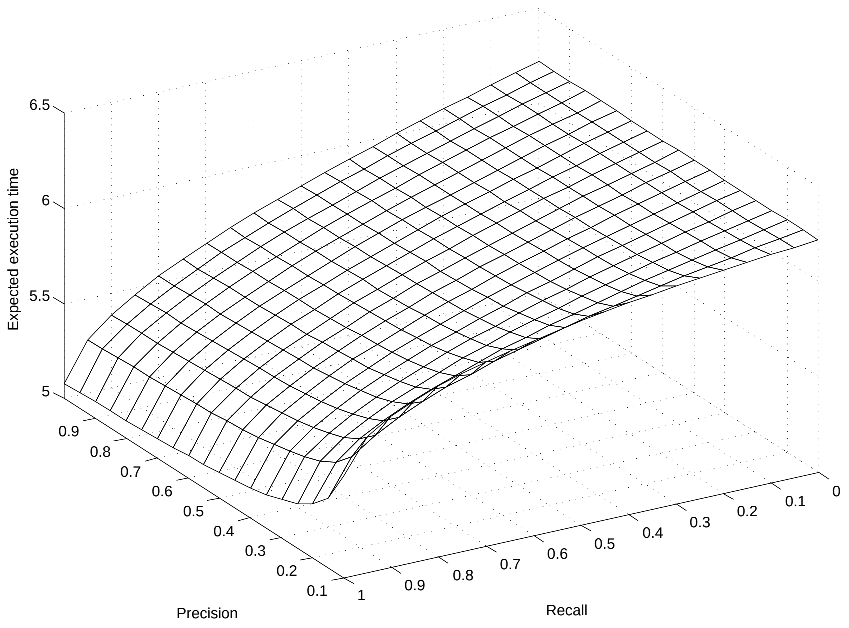


Figure 4.16: The relationship between expected execution time, precision and recall of the improved proactive failure recovery mechanism using revised checkpoint interval. 5-days failure free running time is chosen and other parameters are configured that λ = 0.1, F = 5 days, t_s = 0.2, and t_r = 0.2

2. The value of t_b is considered as $t_c/2$.

3. The instants of true positive predictions are considered as the instants just before the failure occurrence.

4. $t_b^{'}$ is ignored becase the previous assumption.

In terms of Equation 4.21, it uses Equation 4.17 to calculate the new optimum checkpoint interval and the related figures are produced based on the above assumptions and the following assumptions.

1. The occurrences of failures follow Poisson process.

2. t_s and t_r are constant during the execution.

4.3 Summary

This chapter has described the proactive failure management framework that includes two fundamental components: a prediction-based coordinated checkpoint module and a failure detection module. In the case of failure recovery, a checkpoint based approach is adopted for the IBM BlueGene/L due to its high efficiency in reducing wasted computational time whilst using no extra hardware resources. For failure detection, the pull model is utilised in the framework as it is likely to reduce network traffic.

Current coordinated checkpoint mechanisms for supercomputers tend to use static distribution models to express failure distributions in these systems. However, these models cannot describe exactly the dynamic failure characteristics, which motivates the design of a prediction-based checkpoint mechanism in which failure prediction is used to choose an appropriate time for taking checkpoints. However, inaccurate predictions will result in wasted time in the construction of extra checkpoints. More specifically, prediction will reflect the checkpoint placement in four scenarios: (a) correct prediction of failure, (b) incorrect prediction of failure, (c) correct prediction of normal behaviour and (d) incorrect prediction of normal behaviour. The prediction module can reduce time cost in scenario (a), while incurring extra wasted time in scenario (d). Obviously, a prediction module will be considered favourably for integration into a failure recovery approach if and only if it satisfies the condition that the wasted time $W_N^{'}$ is less than the amount that occurs in the original checkpoint mechanism W_N. Thus an analytical model has been derived to evaluate the efficiency of the prediction module, for which two aspects are considered: *Precision* and *Recall*. According to the analytical model, a prediction module must satisfy the condition $P > t_s/(t_b - t_b^{'})$ in order to improve failure recovery efficiency.

In order to investigate the efficiency of the prediction-based checkpoint approach, a formula to describe the expected execution time of the new model is given in Equation 4.14 based on overhead analysis, and the formula shows that the overhead of the new model depends on prediction accuracy. Different accuracy settings have been configured to demonstrate the effect of prediction accuracy. Specifically, there are five scenarios simulated by the theoretical model: comparison with traditional checkpoint model using various prediction accuracy, dependence on precision, dependence on recall, dependence on both precision and recall, and effect of prediction on optimum checkpoint interval.

Contributions of this chapter

This chapter has presented the proactive failure recovery mechanism which can take advantage of the prediction of failure occurrence by placing checkpoints just before the occurrence of predicted failure so that wasted time can be significantly reduced. A theoretical model has been derived to investigate the effects of failure prediction on the efficiency of failure recovery, and several analytical results have been given, the model is believed to be innovative. Next, further improvement of the model has been obtained by having better choices of checkpoint interval, and an equation has been developed to express the relationship between prediction accuracy and optimum checkpoint interval. This appears to be the first work to address this problem.

Relation to other chapters

This chapter has described a proactive failure recovery approach, and introduced an analytical model to investigate the relationship between prediction accuracy and overhead, which derives the search for good prediction mechanisms. The design of new failure prediction mechanisms on the target platform — the IBM BlueGene/L — is considered in the next chapter.

Chapter 5

Failure prediction model for the IBM BlueGene/L

The performance and efficiency of a prediction algorithm are closely related to the infrastructure and key features of target systems. The platform which the research is conducted is the IBM BlueGene/L, which has system log data available for experimentation. Section 5.1 presents the prediction process on the target system. The overall approach to real-time failure prediction consists of two main parts: data preprocessing and prediction. Some preknowledge of the IBM BlueGene/L logs is required before the prediction approach is introduced. In Section 5.2, key properties about the collected logs, including their format and the overall log volume, are described. The three steps of data preprocessing — *categorization*, *filtering* and *sequence extraction*, are then introduced, in Section 5.3. Section 5.4 describes the essence of the proposed failure prediction approach, the semi-Markov CRF model, including how the model is applied in the framework, the inference process, and how to estimate model parameters. Section 5.5 summarises the chapter.

5.1 Prediction process

From statistical studies of the IBM BlueGene/L RAS logs [80] given in Section 5.2, the data volume is too large to be handled by human beings. Further, it is clear that there are certain relationships and correlations among the events: for example, approximately 80% percent of failure events in the RAS logs have preceding non-failure events. Due to the huge amount of data and the complexity of the system structure, it is impossible to explicitly describe and encode all relationships manually. From the

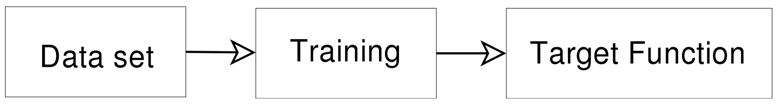


Figure 5.1: Training process.

viewpoint of autonomic computing [55, 54], the system environments and hardware configurations change over time, so an automated approach, that can adapt to these changes and reduce redesign cost, needs to be established. Due to these reasons, a machine learning method has been chosen to meet the demands.

There are two steps in the approach: training and testing. During the training process, a target function (the model) will be produced from a data set so that the parameters of the function are adjusted to fit the training data (Figure 5.1). There are many training algorithms based on different principles, and Maximum Likelihood Estimation (MLE) [76] is applied in our approach. Secondly, the model needs to be tested. Real-time testing on the production system cannot be conducted because we do not direct access to the supercomputer. Instead a static data set representing input/output (I/O), months of RAS log between the field *dates* that is publicly available is used [80]. The normal approach in machine learning in such circumstances is to divide the data set into two parts, one for testing, the other for training.

The basic idea of the approach is to make use of dependencies among the records. Except for stand-alone failures, that are caused by unexpected conditions or human beings, the other failures are preceded by non-failure events. These kinds of relationships are called *failure patterns*. This scenario can be seen in current computing systems where multiple components in a single infrastructure cooperate with each other and finish one or several tasks together; this kind of internal dependency will result in certain relationships in the output. In the case of the IBM BlueGene/L, the output is an RAS log and the main target of the learning approach is to find the failure patterns and utilise these during failure forecasting.

Machine learning is a complicated process and it has several aspects apart from the learning model itself, such as data volume, data quality, training algorithm, testing data set, etc. Each of these factors may impact the final result. For failure prediction in the IBM BlueGene/L, these problems are briefly discussed in the following sections: Training and Prediction.

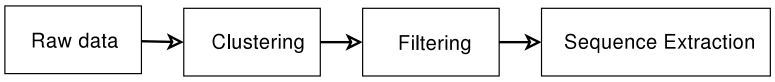


Figure 5.2: Preprocess steps.

Training

The main purpose of the training process is to analyse correlations among events in the RAS logs and adjust the parameters of the Semi-Markov CRF to best represent these correlations. Failure patterns, represented as failure sequences, are the final data set used for machine learning. The model is trained and evaluated with these sequences. It can be seen that there exist multiple categories of failures, and each category corresponds to a failure pattern. Categorising failure patterns is a key problem in data pre-processing (see details in Section 5.3). Each item in the raw RAS log files contains diverse information about a single event, and this redundant data will increase the difficulty of analysis. As was mentioned in Section 1.3, failure items are rare events in this large data space, taking up approximately 10% of the entire data volume. However, there are multiple failure patterns, and for each pattern, the quantity is even smaller. Predicting a specific failure pattern is a more challenging task than the forecasting of general failures. Another aspect to be noted is that the parallel supercomputer may generate multiple items for a single event so that the resulting noisy records in RAS logs may lead us to incorrect conclusions. Removing noisy records is another problem that needs to be considered. All these problems are tackled in the preprocess stage, shown in Figure 5.2, which includes three steps: Clustering, Filtering and Sequence Extraction. The detail of these processes is discussed in Section 5.3.

Prediction

After the model has been trained and optimised, it is used for forecasting future failure occurrences. Two things can be forecast by the model: firstly, whether a failure will occur after a period of lead-time Δt_l (see detail in Chapter 3); secondly, if a failure is predicted, the failure category can be determined. The prediction can be viewed as a mapping process that matches the two sequences with the highest similarity. Different models have distinct approaches to determine similarities, with Semi-Markov CRF, both the event occurrences and their internal dependencies are considered. Further, the model needs to be validated and tested before being used for real prediction. The quality of both the training and testing data set (prediction data set) needs to be assessed,

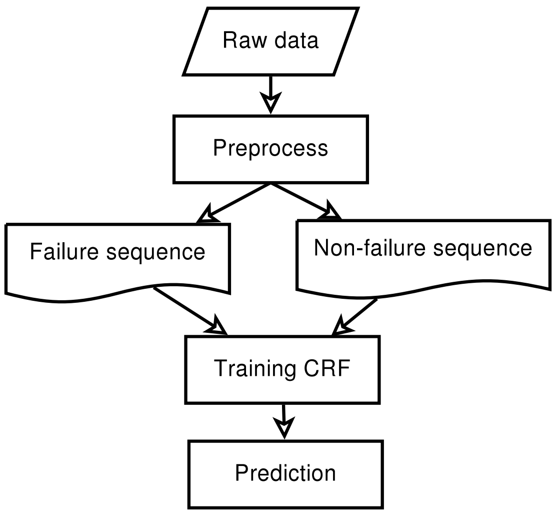


Figure 5.3: An overview of the process flow.

and cross-validation techniques have been used for this purpose. An overview of the procedure for failure prediction is depicted in Figure 5.3.

5.2 The IBM BlueGene/L Log

RAS log format

The RAS event logs are generated through the Machine Monitoring and Control System (MMCS) and are stored in a database, with a logging granularity of less than 1 millisecond. MMCS records target *events* when the machine is running and these events include information from processors, network components, memory, software, operating system, etc. However, events about scheduled maintenance, reboot and repair are not recorded. Events are the key items that are found in the RAS logs in the form of messages. Table 5.1 presents an event example from the logs. Each event has a fixed format, which contains seven well defined fields, as follows:

Event ID	Event Time	Event Type	Facility	Severity	Event Location	Description
1123346974	2005-08-06-09.49.34.522591	RAS	KERNEL	FATAL Error	R00-M1-NC-I:J18-U01	unable to mount file system

Table 5.1: An event example from the IBM BlueGene/L RAS logs.

- **EventID** denotes the sequence number, which is a unique ID for the event and has automatic increment for each new log entry.
- **EventTime** is the timestamp of the event, which includes date and time to a precision level of 1 microsecond.
- **EventType** denotes the logging mechanisms that generate these log events. In the IBM BlueGene/L most of the events are created by the RAS.
- **Facility** shows the component that has generated the event. In the logs, the main components include Application, Kernel, Network Link-Card, Hardware, Monitor, Control Unit, Networks, and Unknown components.
- **Severity** is the field that describes the basic nature of the event: INFO, WARNING, SEVERE, ERROR, FATAL or FAILURE. INFO events show the intermediate output or status from applications and the operating system; the main purpose is to enable understanding of internal actions and processes. WARNING events report unusual actions in nodes, node cards and link cards; these are often transient events and may cause SEVERE or ERROR events. SEVERE events may cause problems on processors or network control cards. ERROR events denote common problems or report problems from non-critical components. Both SEVERE and ERROR events may terminate the current running applications and lead to FATAL or FAILURE events. FATAL and FAILURE events indicate severe problems that usually lead to system or application crashes, which must be fixed by the administrator.
- **EventLocation** gives detailed addresses where the event occurred.
- **Description** gives a short description of the event, and sometimes provides simple analysis of its root cause.

Severity is effectively the most important field, based on which the system status at a certain point can be decided. FATAL and FAILURE events, referred to as *alert events* in the following, are likely to have significant impact on system performance and resource utility, and hence are the focus in this thesis. In contrast, the other events can be collectively termed *general events*, in that they are common events providing information or reporting problems with little impact on the system, and which may be resolved by the system itself.

Log collection

RAS logs have been collected from LLNL covering 215 days from June 3, 2005. They are managed by MMCS and collected via a polling frequency of around one millisecond; they are then stored in the local DB2 database. Nearly all possible errors from software and hardware are included in the RAS logs [80]. The collected RAS logs are summarized in Table 5.2.

Owner	LLNL
Start date	2005-06-03
End date	2006-01-04
Total no. of events	4,747,963
No. of alert events	348,460
Size	1.207 GB

Table 5.2: Summary of the collected IBM BlueGene/L RAS logs.

5.3 Log pre-processing

The information in the RAS logs enables understanding of the internal execution of the IBM BlueGene/L. However, for several reasons, they need to be appropriately preprocessed before being used for prediction [117]. Firstly, event items are not recorded in a uniform format, as some are human readable events whereas others are intermediate outputs. Secondly, redundant events, which may greatly affect the analysis, occupy a significant part of the original log files. For example, the same event may be recorded multiple times in a single log file and many general events are saved that do not lead to failure. Finally, event sequences, derived from raw log items in the pre-processing stage, are used for the purpose of training and testing of the CRF model, and the raw log cannot be processed by the model. For the IBM BlueGene/L logs, preprocessing involves three steps: *Categorisation*, which categorises events, *Filtering*, which removes redundant events and *Sequence extraction*, which identifies significant sequences of events.

5.3.1 Categorization

Generally, event logs are recorded to a certain degree in natural language and they are commonly analysed by operators to quickly identify problems. A difficulty in categorising events (grouping them into coherent sub-sets) is that event logs are not designed for automatic processing.

For the IBM BlueGene/L logs, consideration of the events fields *severity* or *facility* can give us high-level categories. For example, based on *severity*, there are six categories: info, warning, sever, error, fatal and failure. However, these are not enough for the purpose of failure prediction and further categorisation needs to be performed; for example, with a fatal event, it needs to be further analysed to determine whether it is a memory or a network event. Work has been published categorising the IBM BlueGene/L logs; for example, in [66], alert events are categorised according to the *facility* field: memory failures, node card failures, network failures, service card failures and midplane switch failures. However, in order to establish the dependency between different events, several fields need to be considered together in this process of categorisation to form sub-categories, such as those identified in [41].

The main task here is to analyse the differences between *alert* events and *general* events, and to then distinguish them into various identified categories. More specifically, alert events that result in application crashes are the real causes of performance penalty, and hence they need to be further sub-categorised. However, in order to identify the correlation between alert events and other events, and diagnose their root causes, general events also need to be considered. In the collected log files, there are 348,460 alert events and 4,399,503 general events. In our categorisation approach, the fields—*facility, severity, eventlocation* — combined with the *description* recorded in natural language, are taken into account to include as much information as possible. In order to simplify the analysis of such a large volume of events, the key terms in the *description* field have been counted. Based on these key terms, a semi-supervised hierarchical mechanism has been designed for categorization and 62 sub-categories have been generated. Detailed process steps are described below:

- Redundant information removal

  The *Description* field contains human readable language or intermediate machine results, and there is much redundant information as shown in Table 5.3. This information must be removed to simplify the classification.

- Keyword detection

  This step identifies the most frequent words in the logs. For this purpose, LogHound [1] is applied and the 10 most active terms are shown in Table 5.4.

Punctuation	= , ’ , ”
Articles	a, an, the
Verbs	be, being, been, is , are, do, $\cdots$
Prepositions	of, at, in, on, $\cdots$
Hardware address	8DigitHex or 2DigitHex
Operators	+, -, *, ÷
Numbers	0, $\cdots$, 9
Directory path	such as ’/home/bin’
Date	such as 2005-08-06

Table 5.3: Redundant information in the *description* field.

Term	Count
RAS	507,103
FATAL	507,103
KERNEL	342,412
program	168,508
APP	164,691
ciod	120,437
ERROR	168,508
loading	115,120
invalid	113,951
missing	112,584

Table 5.4: Number of keyword terms detected.

- Pattern mining

  The main target is to mine frequent event patterns (event types) from raw event logs. The patterns can be derived from event descriptions, for example, the three messages:

  Connection to 192.168.1.50 down
  Connection to 10.2.8.96 down
  Connection to 138.1.15.98 down

  represent the event type *Connection Down* and the corresponding pattern can be written as:

  Connection to * down

- Pattern regularization

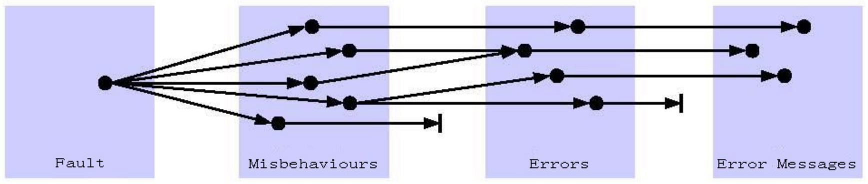


Figure 5.4: Event logging mechanism (taken from [44]). A fault can result in various misbehaviours, some of which can be detected once or several times as errors, while others are not detected. Similarly, some errors will be recorded as messages and others not.

The pattern mining process generates a detailed list of all possible patterns, from which two problems may arise. Firstly, there may be overlap between different patterns. Secondly, for some extremely rare events, the corresponding patterns have few matches. To solve the first problem, these overlapped patterns must be combined into a single pattern; and for the second problem, a threshold value has to be set so that only the patterns with more matched events than the threshold value are accepted.

5.3.2 Filtering

In a computer system, it is clear that a failure may affect various components across different levels; in turn, many devices may detect and report alerts in the same category. This phenomenon can clearly be seen throughout the logs where the alert messages are found in bursts over a very short interval in the same location or across multiple components. This ensures that every message can be captured; however, the corresponding logging mechanism, shown in Figure 5.4, may lead to two types of redundant messages: temporal redundancy and spatial redundancy. To improve efficiency, these redundant messages should be reduced as much as possible while retaining any useful information. Several algorithms have been published that filter the IBM BlueGene/L logs with consideration of both temporal and spatial redundancy [32, 66, 67, 117].

Temporal filtering

Temporal filtering is used to reduce failure instances in the time domain, because multiple failures may occur in a short time period. After investigating the RAS logs, this temporal feature is found to be due to two possible reasons:

1. Some faults may result in several error or failure instances.

2. A single failure event may appear several times referring to the same location.

According to the above two causes, there are two approaches to filter logs in the temporal domain:

1. Different kinds of errors that have a time interval less than $\epsilon^{'}$ are grouped as a single event.
2. All errors of the same type having an inter-arrival time less than a threshold ϵ are reduced to a single error.

Two main problems may arise from filtering:

1. *Collision*, where log messages referring to several categories may be combined.
2. *Truncation*, where the entire span of a single category of event is large and multiple records of the same category remain after filtering.

To simplify, $\epsilon^{'}$ is set equal to ϵ. A formula (Equation 5.1) for the probability of collision is given in [44] to analyse this problem. The formula shows that *collision* and *truncation* are closely related to ϵ; if ϵ is large, the possibility of collision is greater than truncation, and vice versa.

$$P\_Collision(\epsilon) = 1 - e^{-\lambda_F \epsilon}(\sum p_j e^{\lambda_F l_j}) \tag{5.1}$$

where

$P\_Collision \equiv$ the probability of collision;

$\epsilon \equiv$ the time window threshold for filtering;

$e \equiv$ the mathematical constant that denotes the base of the natural logarithm;

$\lambda_F \equiv$ the fault rate;

$l_j \equiv$ the time span of a message estimated from the logs;

$p_j \equiv$ the probability that the length is l_j.

The mechanism presented here derives from the observation that messages of the same category in a short time window ϵ can be represented as a single message. As can be seen from the formula, a different time window length would impact significantly on both collision and truncation.

A method to select the most appropriate ϵ is as follows: plot the number of events over different ϵ. When ϵ is close to 0, the event number after filtering is equal to the original event number. As ϵ is increased, the number of events should decrease sharply at the beginning and then gradually to the lowest point. This decrease is an L-shaped curve and the work in [44] suggests that the value of ϵ chosen should be slightly greater than the value at the vertex of the L-shaped curve. For reasons of simplicity in calculation, the value at the turning point is used in this thesis. For the IBM BlueGene/L logs, the total number of events after filtering with different ϵ values is plotted in Figure 5.5, from which the optimal value of ϵ is approximately set to 430 milliseconds in our mechanism. The final results after filtering can be seen in Table 6.1.

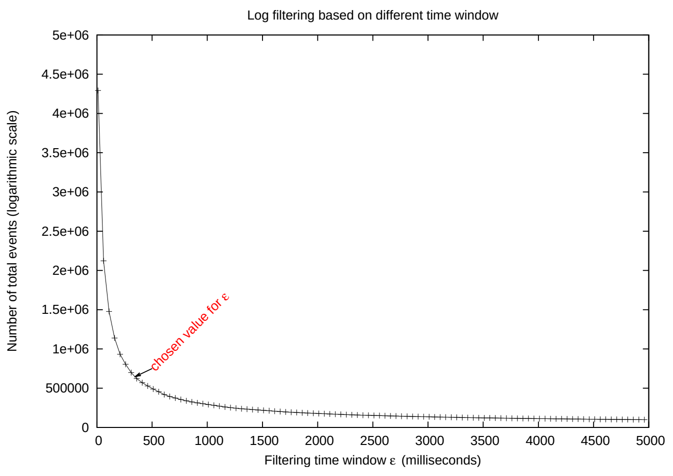


Figure 5.5: Plotting event numbers using different time length ϵ.

Spatial filtering

Spatial filtering is used to remove similar events occurring at different locations because failure events are correlated in the spatial domain [33]. Spatial redundancy is caused by several reasons, such as component dependency and parallel running applications, and there are two common cases described below:

1. A failure or error may happen at multiple locations more-or-less simultaneously.

2. A single failure may cause other failures in different components.

A corresponding approach can be derived to remove redundant events so that all errors across multiple locations that have an inter-arrival time less than ϵ are grouped,

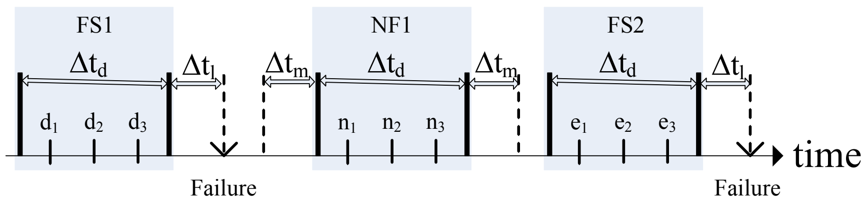


Figure 5.6: Sequence extraction from event logs: failure sequences, which lead to failures, are extracted from a fixed time window Δt_d before the lead-window Δt_l, such as FS1 and FS2; non-failure sequences are event sequences between failures with a marginal time window Δt_m either side, such as NF1.

and the value of ϵ can be identified using the same method described in temporal filtering.

5.3.3 Sequence extraction

CRF models are trained using both failure sequences and non-failure sequences. A failure sequence is an event sequence (defined in Section 3.1) preceding a failure, and the time duration of a failure sequence depends on the time window Δt_d (defined in Section 3.1). A non-failure sequence is the event sequence between two failures. A marginal time window Δt_m is defined to separate system normal status from failure status, and a valid non-failure sequence is one with no failures occurring in time window Δt_m before and after the time window Δt_d. Figure 5.6 shows examples of both failure sequences and non-failure sequences. Δt_l denotes the time length from the current time to the future failure occurrence point. The resulting sequences are summarized in Table 5.5.

Parameters	Values
The data window Δt_d	5Min.
The lead-window Δt_l	5Min.
The margin time window Δt_m	5Min.
Total no. of failure sequences	770
Total no. of non-failure sequences	82,118

Table 5.5: Summary of extracted sequences.

5.4 The prediction model

In a machine learning process, the target model is first trained, using a training data set to adjust the model parameters, then the model is applied to testing data. For the problem of failure prediction in the IBM BlueGene/L, this procedure involves two steps: estimating model parameters, as described in Section 5.4.3, and system future behaviour prediction. More specifically, there are two aspects included in the forecasting: *the prediction strategy*, presented in Section 5.4.1, which introduces the process flow and various models from a high-level perspective, and *the inference process*, presented in Section 5.4.2, which describes the background algorithms and mathematical proof, including the *forward*, the *backward* and the *Viterbi* algorithms, respectively.

5.4.1 Prediction strategy

Sequence prediction, the second step in real-time failure prediction, processes event sequences extracted from pre-processed data. The main target is to estimate the future behaviour of the system from a temporal sequence. More precisely, two different prediction models are combined in a single mechanism, and the overall procedure is presented in Figure 5.7.

Prediction procedure

For failure prediction on the IBM BlueGene/L, two prediction models are applied for different estimation purposes. Given a sequence, as much further extrapolation as possible needs to be assessed so that appropriate actions can be launched to prevent future misbehaviour. More precisely, two different prediction models are required to answer the following questions:

1. Whether failure will occur within some time duration?
2. If a failure is forecast, which type of failure is it most likely to be?

As shown in Figure 5.7, an event sequence is initially processed by model 1 to forecast whether there will be a future failure, the same sequence is then passed to model 2 for further analysis if a failure is estimated by model 1. It can be concluded that, in order to answer the first question, model 1 conducts high-level prediction and determines whether the system is failure-prone. For the solution to the next question, model 2 is established to further forecast the particular failure type once a failure has

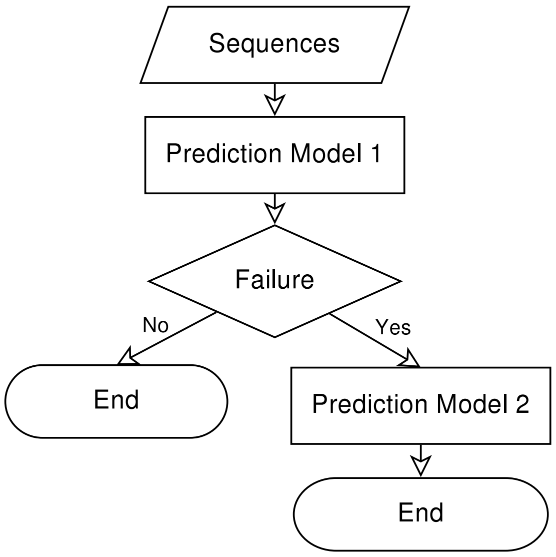


Figure 5.7: Sequence prediction using two different models: Model 1 forecasts whether a failure will occur and Model 2 anticipates the specific failure type if a failure is predicted by Model 1.

been estimated. Although the output from model 2 is not used by the failure recovery mechanism described in Chapter 4, it may be pertinent to some future mechanism which is capable of taking advantage of knowing the type of failure that is about to occur. More specifically, the two models can be distinguished in terms of the following three aspects:

1. Event patterns: Model 1 divides the entire event sequences into two patterns: failure sequences and non-failure sequences, while model 2 further categorizes failure sequences into various sub-patterns, each of which indicates a particular failure category.

2. Training: model 1 is trained using both failure sequences and non-failure sequences. In contrast, model 2 is trained using only failure sequences.

3. Output: the output of model 1 is a binary classification, while the result of model 2 is classified into multiple values.

5.4.2 Inference

Given a chain-structured CRF model[1] and an event sequence, different predictions can be made. Furthermore, two inference problems can be solved:

1. Given any observation sequence $O = \{O_1, O_2, \cdots, O_N\}$ and a corresponding state sequence $S = \{S_0, S_1, \cdots, S_N\}$, in terms of the model Φ, the issue is how to efficiently compute $P(S|O;\Phi)$.
2. Given an observation sequence $O = \{O_1, O_2, \cdots, O_N\}$ and the model Φ, the issue is how to predict the optimal state sequence $S = \{S_0, S_1, \cdots, S_N\}$, which has the maximum likelihood.

For both tasks, all possible state sequences S need to be taken into account and efficient algorithms are required for calculation without enumerating all possible state sequences. In a chain-structured CRF, algorithm design can take advantage of the fact that a state depends only on its adjacent states, and both inference tasks can be performed efficiently by variants of the HMM algorithms.

In this section, the *forward* and *backward* algorithms that are used in HMM are applied to solve the first problem; similarly, the *Viterbi* algorithm is applied to address problem 2.

Inference for problem 1

The chain-structured CRF is shown in Equation 3.21 and 3.22, and the difficulty of computing $P(S|O;\Phi)$ transfers to the task of calculating $Z(O)$, which needs to enumerate all possible state sequences S. In order to formalize the solutions, the states are assumed to be in a space of size M and the state transition function is defined as $\Psi_i(S_i, S_j, O)$, shown in Equation 5.2.

$$\begin{aligned}\Psi_i(S_i, S_j, O) &\overset{def}{=\!=} \Psi_i(S_i, S_{i-1}, O)\\ &= \exp\left(\sum_i \sum_{k=1}^{K} \lambda_k F_k(S_i, S_{i-1}, O)\right)\end{aligned} \tag{5.2}$$

Given an observation sequence O and the corresponding state sequence S, the global normaliser $Z(O)$ can be written as Equation 5.3, from which it can be seen

[1]A chain-structured CRF is used to make inference for simplicity and generality, because the same algorithms can be applied similarly for other CRF models.

that the intermediate results are reused many times during the calculation and considerable time can be saved by caching these results. This leads to two different caching algorithms: the forward algorithm and the backward algorithm, which reduce computational complexity. Table 5.6 shows a comparison of the different algorithms in terms of computational complexity.

$$\begin{aligned} Z(O) &= \sum_S \exp\left(\sum_i \sum_{k=1}^{K} \lambda_k F_k(S_i, S_{i-1}, O)\right) \\ &= \sum_S \prod_{i=1}^{N} \Psi_i(S_i, S_{i-1}, O) \\ &= \sum_S \sum_{S_{N-1}} \Psi_N(S_N, S_{N-1}, O) \sum_{S_{N-2}} \Psi_{N-1}(S_{N-1}, S_{N-2}, O) \sum_{S_{N-3}} \cdots \end{aligned} \tag{5.3}$$

Algorithms	Complexity
Equation 5.3	$O(N * M^N)$
Forward algorithm	$O(N * M^2)$
Backward algorithm	$O(N * M^2)$

Table 5.6: Comparison of different algorithms in terms of computational complexity.

Forward algorithm

As the name suggests, the *forward* algorithm is based on a set of forward variables $\alpha_t(j)$, each of which stores one of the intermediate results denoting the probability of sub-sequence $S_{<0,\cdots,t>} = \{S_0 \cdots S_t\}$ under the assumption that the stochastic process is in state j at time t. The sub-sequence $S_{<0,\cdots,t-1>} = \{S_0 \cdots S_{t-1}\}$ ranges over all possible assignments to each random variable S_i, where $1 \leq i \leq t-1$. Then $\alpha_t(j)$ can be recursively computed using Equation 5.4.

$$\begin{aligned}
\alpha_t(j) &\stackrel{def}{=\!=} P(S_t = j|O) \\
&= \sum_{S_{<1,\cdots,t-1>}} \Psi_i(j, S_{t-1}, O) \prod_{i=1}^{t-1} \Psi_i(S_i, S_{i-1}, O) \\
&= \sum \Psi_i(j, S_{t-1}, O)\alpha_{t-1}(S_{t-1}) \qquad (5.4)
\end{aligned}$$

Let $S_{t-1} = i, i \in S$, then Equation 5.4 translates to Equation 5.5:

$$\alpha_t(j) = \sum_{i \in S} \Psi_i(j, i, O)\alpha_{t-1}(i) \qquad (5.5)$$

As $\alpha_N(i)$ is the probability of the whole sequence that ends in state i, $Z(O)$ can be represented by forward vectors α_t using Equation 5.6.

$$Z(O) = \sum_{S_N} \alpha_N(S_N) \qquad (5.6)$$

Backward algorithm

Similarly, a *backward* variable $\beta_t(j)$ is defined as the probability of the sequence $S_{<t,\cdots,N>} = \{j, S_{t+1} \cdots S_N\}$ with state j at time t. Equation 5.7 gives the recursive definition of the backward variable.

$$\begin{aligned}
\beta_t(j) &\stackrel{def}{=\!=} P(S_t = j|O) \\
&= \sum_{S_{<t+1,\cdots,N>}} \prod_{i=t+1}^{N} \Psi_i(S_i, S_{i-1}, O) \\
&= \sum_{i \in S} \Psi_{t+1}(S_{t+1}, j, O)\beta_{t+1}(S_{t+1}) \qquad (5.7)
\end{aligned}$$

Let $S_{t+1} = i, i \in S$, then $\beta_t(j)$ can be written as Equation 5.8. Then $Z(O)$ can be represented by the backward vectors β_t in Equation 5.9.

$$\beta_t(j) = \sum_{i \in S} \Psi_{t+1}(i, j, O)\beta_{t+1}(i) \tag{5.8}$$

$$\begin{aligned} Z(O) &= \beta_0(S_0) \\ &= \sum_{S_1} \Psi_1(S_1, S_0, O)\beta_1(S_1) \end{aligned} \tag{5.9}$$

Inference for problem 2

The second problem attempts to find the most probable sequence of states given an observation sequence. A straightforward solution to obtain the optimal sequence S_{opt} is shown in Equation 5.10, where the computational complexity is $O(N * M^N)$. In order to improve efficiency, the *Viterbi* algorithm is introduced.

$$S_{opt} = \arg\max_S P(S|O, \Phi) \tag{5.10}$$

The Viterbi algorithm

Similarly to the way the forward variable $\alpha_t(j)$ was defined, $\delta_t(i)$ is defined in Equation 5.11 to denote the probability of the most likely sub-sequence $S_{<1,\cdots,t-1>}$ $= \{S_1, \cdots, S_{t-1}\}$ under the assumption that the process is in state i at time t. It can also be recursively calculated, using Equation 5.12.

$$\delta_t(i) \overset{def}{=\!=} \max_{S_{<1,\cdots,t-1>}} P(S_t = i|O, \Phi) \tag{5.11}$$

$$\delta_{t+1}(j) = \max_i[\Psi_j(j, i, O)\delta_t(i))] \tag{5.12}$$

As $\delta_N(i)$ represents the probability of the most probable sequence that ends in state i, then the probability can be calculated by Equation 5.13. In order to identify the states that contributed the most likely sequence, each state can be established working backwards from the sequence using Equation 5.14.

$$P_{opt}(S_N = i|O, \Phi) = \max_{i \in S} \delta_N(i) \tag{5.13}$$

$$S_{opt} = \arg\max_{i \in S} \delta_N(i) \tag{5.14}$$

5.4.3 Model training

As can be seen in Equation 3.21, the form of CRF is motivated by Maxent [12], which is a model to estimate probability distributions from a training data set. The training data set can be seen as a finite series of features or constraints, and the problem turns into a constrained optimization problem. The principle of Maxent can help to find an analytical solution. Log-likelihood, which is an approach to Maximum Likelihood Estimation (MLE) [76], is calculated when estimating the parameters $\lambda^* = \{\lambda_k\}$ of CRF models.

In a linear-chain CRF, the training data set is defined as $D = \{O^{(i)}, S^{(i)}\}_{i=1}^{N}$, where each $O^{(i)} = \{O_1, O_2, \cdots, O_M\}$ is an observation sequence and each $S^{(i)} = \{S_0, S_1, \cdots, S_M\}$ is a sequence of prediction states. It is assumed that during each sequence there may exist arbitrary dependencies, whereas distinct sequences are independent of each other. The CRF model (Equation 3.21) is a probability function and the purpose of the training process is to maximise the logarithm of the likelihood function (log-likelihood). Equation 5.15 gives the log-likelihood function and the estimation principle is shown in Equation 5.16.

$$\begin{aligned} l(\lambda) &= \sum_{1}^{N} \log P(S^{(i)}|O^{(i)}) \\ &= \sum_{i=1}^{N} \sum_{m=1}^{M} \sum_{k}^{K} \lambda_k F_k(S_m^{(i)}, S_{m-1}^{(i)}, O^{(i)}) - \sum_{i=1}^{N} \log Z(O^{(i)}) \end{aligned} \tag{5.15}$$

$$\lambda^* = \arg\max_{\lambda_k} l(\lambda) \tag{5.16}$$

In order to optimize the function $l(\lambda)$, numerical optimization methods are applied and the partial derivative function with respect to λ is given in Equation 5.17. The second term is the expected value of F_k and a normal method of calculating λ is to make the derivative function equal to zero, which will give a general constraint on the model: the expectation for each feature under the model distribution is equal to the expected value of each feature under the empirical distribution. This is also the principle of the maximum entropy method.

$$\begin{aligned}\frac{\partial l}{\partial \lambda_k} &= \sum_i \sum_m F_k(S_m^{(i)}, S_{m-1}^{(i)}, O^{(i)}) - \sum_i \sum_m \sum_{S,S'} F_k(S, S', O^{(i)}) P(S, S' | O^{(i)}) \\ &= \sum_i \sum_m F_k(S_m^{(i)}, S_{m-1}^{(i)}, O^{(i)}) - \sum_i \sum_m F_k(S, S', O^{(i)}) \tilde{P}(S, S' | O^{(i)})\end{aligned} \tag{5.17}$$

Like logistic regression, estimation using the common log-likelihood function suffers from the problem of overfitting, where the trained model may be more accurate in fitting known data but less accurate in predicting new data. Thus a penalty is imposed on large parameter values (known as *regularization*). The most commonly used penalty is based on the Euclidean norm and a regularization parameter $\frac{1}{2\sigma}$ is assigned to determine the strength of the penalty. The resulting penalized log-likelihood function and its partial derivative function are shown in Equations 5.18 and 5.19, respectively.

$$l(\lambda) = \sum_{i=1}^{N} \sum_{m=1}^{M} \sum_{k}^{K} \lambda_k F_k(S_m^{(i)}, S_{m-1}^{(i)}, O^{(i)}) - \sum_{i=1}^{N} \log Z(O^{(i)}) - \sum_{k=1}^{K} \frac{\lambda_k^2}{2\sigma} \tag{5.18}$$

$$\frac{\partial l}{\partial \lambda_k} = \sum_i \sum_m F_k(S_m^{(i)}, S_{m-1}^{(i)}, O^{(i)}) - \sum_i \sum_m F_k(S, S', O^{(i)}) \tilde{P}(S, S' | O^{(i)}) - \sum_{k=1}^{K} \frac{\lambda_k}{\sigma} \tag{5.19}$$

In order to optimize $l(\lambda)$, various methods have been considered, such as iterative scaling [12, 11] and gradient-based [102] optimization. Newton series methods are considered here for the purpose of parameter training because the curvature of the likelihood is taken into account to make it converge faster. There are many expansion forms of the Newton method. More specifically, L-BFGS (Limited Storage Quasi-Newton Method) is applied.

Specifically, the Quasi-Newton method, an extended form of the Newton method, is used to obtain the Hessian matrix from the first derivative of the objective function, thus avoiding direct calculation. There are various algorithms that approximately generate the Hessian matrix, such as DFP (Davidon-Fletcher-Powell) [77] and BFGS (Broyden-Fletcher-Goldfarb-Shanno) [8], where BFGS is an improved form of DFP. A new problem arises in the Quasi-Newton method in that the intermediate results of each iteration need to be stored. For example, assume the parameter vector has a size of N, then at least $N * (N + 1)/2$ storage space is required, which is costly for a large computation problem. Thus L-BFGS has been derived to decrease the intermediate caching size.

Newton method

Given a real-valued function $f(\mathbf{x})$, where $\mathbf{x} = \{x_1, x_2, \cdots, x_N\}$, based on the *Taylor* expansion, function $f(\mathbf{x} + \mathbf{\Delta x})$ can be approximately written as Equation 5.20, where $\mathbf{\Delta x}$ is the increment of x, $\nabla f(\mathbf{x})$, defined in Equation 5.21, is the *gradient*[2] of function $f(\mathbf{x})$, and $\nabla^2 f^2(\mathbf{x})$, the *Hessian* matrix, denotes the second partial derivative function of $f(\mathbf{x})$.

$$f(\mathbf{x} + \mathbf{\Delta x}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})\mathbf{\Delta x} + \frac{1}{2}\mathbf{\Delta x}^{\mathrm{T}}\nabla^2 f^2(\mathbf{x})\mathbf{\Delta x} \tag{5.20}$$

$$\nabla f(\mathbf{x}) = (\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_N}) \tag{5.21}$$

Based on Equation 5.20, the first derivative of function $f(\mathbf{x} + \mathbf{\Delta x})$ is given by Equation 5.22. If $\mathbf{\Delta x}$ approaches zero, let $f^{'}(\mathbf{x} + \mathbf{\Delta x}) = 0$ and $\mathbf{H} = \nabla^2 f(x)$, the Newton update rule can be obtained as shown in Equation 5.23.

$$f^{'}(\mathbf{x} + \mathbf{\Delta x}) \approx \nabla f(\mathbf{x}) + \nabla^2 f(\mathbf{x})\mathbf{\Delta x} \tag{5.22}$$

$$\begin{aligned} \mathbf{\Delta x} &= \mathbf{x}_{k+1} - \mathbf{x}_k \\ &= -[\nabla^2 f(\mathbf{x})]^{-1}\nabla f(\mathbf{x}) \\ &= -\mathbf{H}^{-1}\nabla f(\mathbf{x}) \end{aligned} \tag{5.23}$$

[2]The gradient of function f is defined to be the vector field whose components are the partial derivatives of f.

Quasi-Newton method

In order to avoid constructing the inverse of the Hessian matrix $\mathbf{H}^{-1}$ directly, the Quasi-Newton method tries to generate an approximate matrix $\mathbf{H}^{-1}$ from the gradient. Equation 5.24 can be derived from Equation 5.22.

$$\mathbf{\Delta x} \approx [\nabla^2 f(\mathbf{x})]^{-1}[\nabla f(\mathbf{x} + \mathbf{\Delta x}) - \nabla f(\mathbf{x})] \tag{5.24}$$

Let $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$, $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$, then Equation 5.24 translates to Equation 5.25.

$$\mathbf{s}_k \approx [\nabla^2 f(\mathbf{x_k})]^{-1}\mathbf{y}_k \tag{5.25}$$

Denoting $[\nabla^2 f(\mathbf{x}_k)]^{-1}$ by $\mathbf{H}_k^{-1}$, then $\mathbf{s}_k$ can be defined using Equation 5.26.

$$\mathbf{s}_k \approx \mathbf{H}_k^{-1}\mathbf{y}_k \tag{5.26}$$

Thus $\mathbf{H}_k$ can be updated by iteration as shown in Equation 5.27.

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \Delta\mathbf{H}_k \tag{5.27}$$

The DFP algorithm [77] gives an approximation for the iterative step in Equation 5.28, $\mathbf{H}_{k+1}$ can then be defined in the form of Equation 5.29.

$$\Delta\mathbf{H}_k = \frac{\mathbf{s}_k\mathbf{s}_k^{\mathrm{T}}}{\mathbf{s}_k^{\mathrm{T}}\mathbf{y}_k} - \frac{\mathbf{H}_k\mathbf{y}_k\mathbf{y}_k^{\mathrm{T}}\mathbf{H}_k}{\mathbf{y}_k^{\mathrm{T}}\mathbf{H}_k\mathbf{y}_k} \tag{5.28}$$

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \frac{\mathbf{s}_k\mathbf{s}_k^{\mathrm{T}}}{\mathbf{s}_k^{\mathrm{T}}\mathbf{y}_k} - \frac{\mathbf{H}_k\mathbf{y}_k\mathbf{y}_k^{\mathrm{T}}\mathbf{H}_k}{\mathbf{y}_k^{\mathrm{T}}\mathbf{H}_k\mathbf{y}_k} \tag{5.29}$$

Similarly to the DFP algorithm, if $\mathbf{H}_k^{-1}$ is substituted by $\mathbf{B}_{k+1}$, then the BFGS algorithm [8] is obtained as shown in Equation 5.30.

$$\mathbf{B}_{k+1} = \mathbf{B}_k + [1 + \frac{\mathbf{y}_k^{\mathrm{T}}\mathbf{B}_k\mathbf{y}_k}{\mathbf{s}_k^{\mathrm{T}}\mathbf{y}_k}]\frac{\mathbf{s}_k\mathbf{s}_k^{\mathrm{T}}}{\mathbf{s}_k^{\mathrm{T}}\mathbf{y}_k} - \frac{\mathbf{s}_k\mathbf{y}_k^{\mathrm{T}}\mathbf{B}_k}{\mathbf{s}_k^{\mathrm{T}}\mathbf{y}_k} \tag{5.30}$$

Limited storage Quasi-Newton method

The L-BFGS algorithm [69] can significantly reduce the storage space by only saving a limited number (such as m) of the intermediate sums. The basic idea is as follows:

for the parameter vector with a dimension of N, in the $(m+1)th$ iteration when calculating the $\mathbf{B}_{k+1}$, only the intermediate results of the previous m iterations are stored, as $\Delta\mathbf{B}_{k-m+1}, \cdots, \Delta\mathbf{B}_k$, and the results of earlier iterations, like $\Delta\mathbf{B}_0, \cdots, \Delta\mathbf{B}_{k-m}$, are thrown away. L-BFGS has been experimentally shown to be a practical optimization algorithm for real world problems. If $m = N$, then L-BFGS is equivalent to the standard BFGS algorithm. The inference process of L-BFGS is described below.

Let $\rho_k = \frac{1}{\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k}$, then $\mathbf{v}_k$ is a matrix of size $N * N$, and Equation 5.30 can be rewritten as Equation 5.31, where $\mathbf{I}$ is the identity matrix.

$$\mathbf{B}_{k+1} = (\mathbf{I} - \rho_k \mathbf{s}_k \mathbf{y}_k^{\mathrm{T}})\mathbf{B}_k(\mathbf{I} - \rho_k \mathbf{y}_k \mathbf{s}_k^{\mathrm{T}}) + \rho_k \mathbf{s}_k \mathbf{s}_k^{\mathrm{T}} = \mathbf{v}_k^{\mathrm{T}}\mathbf{B}_k\mathbf{v}_k + \rho_k \mathbf{s}_k \mathbf{s}_k^{\mathrm{T}} \tag{5.31}$$

Given m, if $\mathbf{v}_k \neq \mathbf{I}$ and $\rho_k \neq 0$, then the calculation of $\mathbf{B}_{k+1}$ can be partitioned into two parts depending on whether $k+1$ is greater than m.

If $k+1 \leq m$, $\mathbf{B}_{k+1}$ can be calculated by Equation 5.32.

$$\begin{aligned} \mathbf{B}_{k+1} = \; & \mathbf{v}_k^{\mathrm{T}}\mathbf{v}_{k-1}^{\mathrm{T}} \cdots \mathbf{B}_0\mathbf{v}_0 \cdots \mathbf{v}_{k-1}\mathbf{v}_k + \mathbf{v}_k^{\mathrm{T}} \cdots \mathbf{v}_1^{\mathrm{T}}\rho_0\mathbf{s}_0\mathbf{s}_0^{\mathrm{T}}\mathbf{v}_1 \cdots \mathbf{v}_k \\ & + \mathbf{v}_k^{\mathrm{T}} \cdots \mathbf{v}_2^{\mathrm{T}}\rho_1\mathbf{s}_1\mathbf{s}_1^{\mathrm{T}}\mathbf{v}_2 \cdots \mathbf{v}_k \\ & \cdots \\ & + \mathbf{v}_k^{\mathrm{T}}\rho_{k-1}\mathbf{s}_{k-1}\mathbf{s}_{k-1}^{\mathrm{T}}\mathbf{v}_k \\ & + \rho_k\mathbf{s}_k\mathbf{s}_k^{\mathrm{T}} \end{aligned} \tag{5.32}$$

If $k+1 > m$, the calculation of $\mathbf{B}_{k+1}$ follows Equation 5.33.

$$\begin{aligned} \mathbf{B}_{k+1} = \; & \mathbf{v}_k^{\mathrm{T}}\mathbf{v}_{k-1}^{\mathrm{T}} \cdots \mathbf{v}_{k-m+1}^{\mathrm{T}}\mathbf{B}_0\mathbf{v}_{k-m+1} \cdots \mathbf{v}_{k-1}\mathbf{v}_k \\ & + \mathbf{v}_k^{\mathrm{T}} \cdots \mathbf{v}_{k-m+2}^{\mathrm{T}}\rho_{k-m+1}\mathbf{s}_{k-m+1}\mathbf{s}_{k-m+1}^{\mathrm{T}}\mathbf{v}_{k-m+2} \cdots \mathbf{v}_k \\ & \cdots \\ & + \mathbf{v}_k^{\mathrm{T}}\rho_{k-1}\mathbf{s}_{k-1}\mathbf{s}_{k-1}^{\mathrm{T}}\mathbf{v}_k \\ & + \rho_k\mathbf{s}_k\mathbf{s}_k^{\mathrm{T}} \end{aligned} \tag{5.33}$$

From Equation 5.32 and Equation 5.33, it can be clearly seen that the calculation of $\mathbf{B}_{k+1}$ only needs $(2*m+1)$ vectors: $\mathbf{s}_{k-m+1}, \cdots, \mathbf{s}_k, \mathbf{y}_{k-m+1}, \cdots, \mathbf{y}_k$ and $\mathbf{B}_0$. The computational complexity is $O(2*m+1) = O(2*N)$.

5.5 Summary

In this chapter, the key features of the target platform — the IBM BlueGene/L — have been identified, and the proposed solution — machine learning — has been outlined. The process of the prediction approach has been described as follows:

- Internal system dependencies will lead to failure sequences.
- Not every error will result in a failure, and a complex relationship exists between errors and failures.
- Failure patterns need to be identified for prediction.
- RAS log files contain sufficient system trace information to reflect the dependencies.
- Because of the redundant information contained in log items, pre-processing is applied.
- Arbitrary dependencies may hold in a failure sequence; CRFs are used to model failure patterns.
- The extended form of CRF with continuous time support has been used to represent the forecasting problem.
- The semi-Markov CRF model is trained with both failure sequences and non-failure sequences.
- The failure category could be further specified.
- Evaluation and testing using a different data set is conducted on the established model.
- The optimised model is used for failure prediction.

This chapter also covers the detailed content of data pre-processing, which describes the main steps that generate a set of training sequences from the raw logs. More specifically, this process involves:

- **Categorization** maps natural language event logs to various category IDs in terms of several event properties, such as severity, location, etc.

- **Filtering** removes redundant event logs of the same cause according to both temporal and spatial dimensions.
- **Sequence extraction** generates the sequences that lead to failures from the filtered logs.

For the prediction model, semi-Markov CRF is applied as a combination of a semi-Markov process and the standard CRF model: standard CRF employs discrete time for the stochastic process of state traversals, which is replaced by a continuous time semi-Markov process. In terms of the prediction strategy, two different semi-Markov models are used to achieve precise forecasting on what type of failure, if any, may occur at a future point.

For the inference, two basic prediction problems have been solved: first, it might be of interest what the probability of a state sequence would be given the observation sequence under a specific model, and second, how to generate the optimal state sequence with maximum likelihood given an observation sequence. The forward and backward algorithms are introduced to solve the first problem and greatly reduce computational complexity from $O(N * M^N)$ to $O(N * M^2)$, and the Viterbi algorithm is used to work out the most probable state sequence.

To train the target CRF model and estimate the parameter vector, a penalized log-likelihood method is used. However, the function of the parameter vector cannot be maximized in closed form and there are no analytic solutions, so numerical optimization methods are used. More specifically, L-BFGS is employed. Firstly, it does not directly calculate the Hessian matrix, which is infeasible for some complex problems, but is required by the original Newton method. Secondly, compared to the quasi-Newton method, the storage space required can then be reduced to $2 * N$ from $N * (N + 1)/2$, which significantly improves computational efficiency.

Contributions of this chapter

This chapter has introduced a novel approach for categorization of raw log events in terms of its keywords, which is also helpful for failure analysis, and a novel approach to remove redundant events in logs with respect to both spatial and temporal characteristics. Secondly, this chapter has described the prediction strategy using semi-Markov CRF, where the prediction is achieved on various levels that not only can tell whether there is a future failure, but also distinguish failure categories. The detailed prediction result could be very useful for failure recovery and system maintenance. To the best of the author's knowledge, this work is the first to use semi-Markov CRF for the failure prediction problem. Finally, the semi-Markov CRF model, extended from the standard CRF to support continuous time events, has been discussed. The model provides great flexibility in terms of dependencies and is more appropriate for real-world problems with dependencies than other models, such as Bayesian models, Markov models, etc.

Relation to other chapters

The previous chapter has described a proactive failure management framework, which derives the search for good prediction mechanisms. This chapter has investigated the failure prediction process: analyse event sequences that lead to severe failures, then use the result model for future failure forecasting, finally the resulting output is sent to the failure recovery module to help the system make the right decision in choosing an appropriate recovery action. The next chapter introduces experimental scenarios and presents results.

Chapter 6

Experimental results

The prediction-based failure management framework presented in this thesis contains two main components: failure prediction and failure recovery. The failure prediction approach has been applied to the real system-log data from the IBM BlueGene/L, then a proactive failure recovery mechanism based on the prediction model has been set up to reduce wasted computational cost. This chapter discusses the experimental approach and analyses the associated results. This chapter is organized based on the experimental process flow: starting with the raw data preprocessing in Section 6.1; characteristics of the data are presented in Section 6.2; the training of the semi-Markov CRFs is discussed in Section 6.3; analysis of the final failure predictor and its prediction quality is given in Section 6.4. Section 6.5 summarises the chapter.

6.1 Data preprocessing

As described in Chapter 5, there are three steps in preprocessing: categorization, filtering and sequence extraction. This section introduces the detailed process and presents the results for each step.

6.1.1 Categorization

Understanding the raw data is a key issue and some analysis is conducted at this stage. Log statistics based on the field *facility*, which is defined in the RAS log format in Chapter 5, are shown in Figure 6.1. It can be seen that events from *KERNEL* are the most frequent, with *MMCS* and *APP* as the second most active components. Figure 6.2 presents the event number statistics according to different severity levels. *INFO*

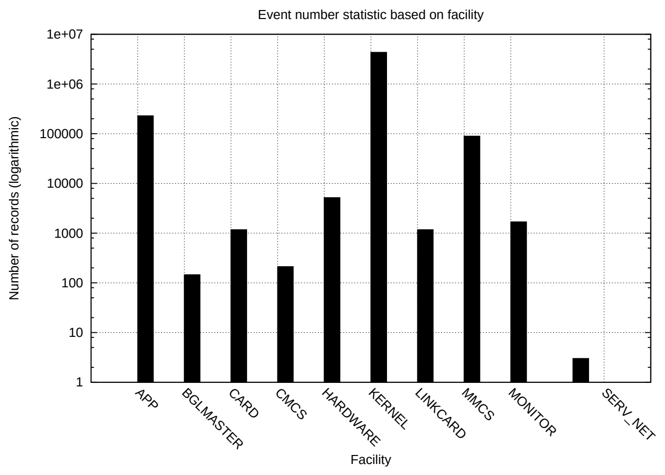


Figure 6.1: Facility based log statistics.

events make up around 80% of the total. From Figure 6.1 we can conclude that most *KERNEL* events are at the *INFO* level. Note that Figures 6.1 and 6.2 both have their y-axes on logarithmic scales.

The main task of categorization is to assign a category ID which reflects the key characteristics of the event, such as severity, location and impact on the system, to each event in the RAS logs. However, as described in Chapter 5, the RAS logs consist of events that are recorded using various fields and they are not machine-processable. In order to analyse these events, several steps have to be performed, including *redundant information removal, keyword detection, pattern mining and pattern regularization* (the details are given in Chapter 5).

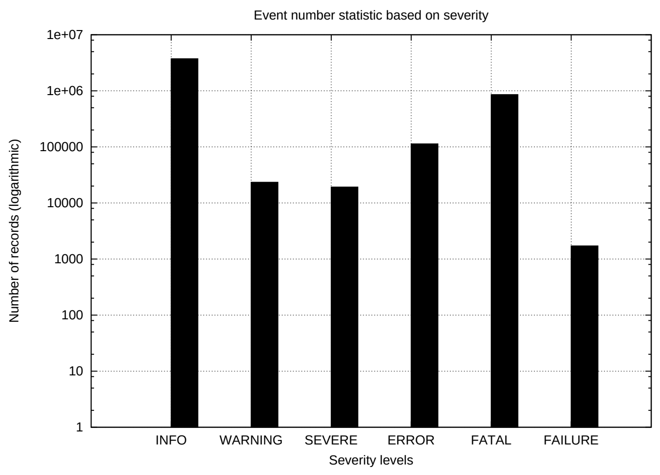


Figure 6.2: Severity based log statistics.

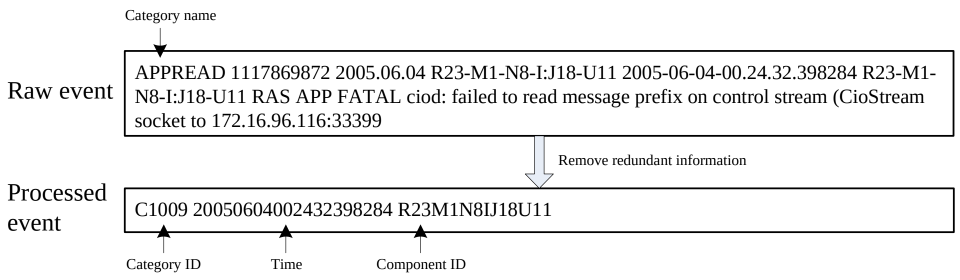

Category name
Raw event
APPREAD 1117869872 2005.06.04 R23-M1-N8-I:J18-U11 2005-06-04-00.24.32.398284 R23-M1-N8-I:J18-U11 RAS APP FATAL ciod: failed to read message prefix on control stream (CioStream socket to 172.16.96.116:33399
Remove redundant information
Processed event
C1009 20050604002432398284 R23M1N8IJ18U11
Category ID
Time
Component ID

Figure 6.3: An example of a raw event and the corresponding processed event.

The first step is to remove redundant information (see Table 5.3) contained in the events. Figure 6.3 presents an example of both a raw event and the corresponding processed event. It can be seen that the raw event has been tagged with a category name by latter steps and there are multiple fields, part of which are recorded in natural language. The processed event includes only three fields: category ID, time and component ID. More specifically, two issues need to be noted here. Firstly, the timestamp in the raw event is recorded in the form of "2005-06-04-00.24.32.398284" denoting that the event occurred at 00 am, 24 minutes and 32.398284 seconds on June 4th in 2005. In order to calculate the time interval between two successive events, the timestamp has to be transformed to a standard format so that it can be processed automatically. A common solution for this problem is to adopt an Coordinated Universal Time (UTC) integer, which counts time since January 1st, 1972. Secondly, category names are represented as category IDs in the processed events for the purpose of automatic processing. In order to be able to match a category ID with a unique category name, a dictionary with entries in the form of $< ID, category\ name >$ is established.

```
...
C1011 APP FATAL ciod.*file.*directory
C1012 APP FATAL ciod.*loading
C1013 APP FATAL ciod.*creating
C0601 RAS KERNEL INFO.*soft failures
C0602 RAS KERNEL INFO.*input interrupt
...
```

Figure 6.4: An example of event patterns.

After removing the redundant information, the next step is to find the most frequent keywords, some of which are presented in Table 5.4. The target patterns in the form of regular expressions, an example of which is shown in Figure 6.4, are then generated using these keywords. As can be seen, the first field in the patterns indicates the category ID and the remaining part is used to match the log events.

Figure 6.5 demonstrates the hierarchical structure of the final categories. Three fields in the events have been considered in the categorization: *Facility*, *Severity* and *Description*; finally, 62 subcategories have been identified.

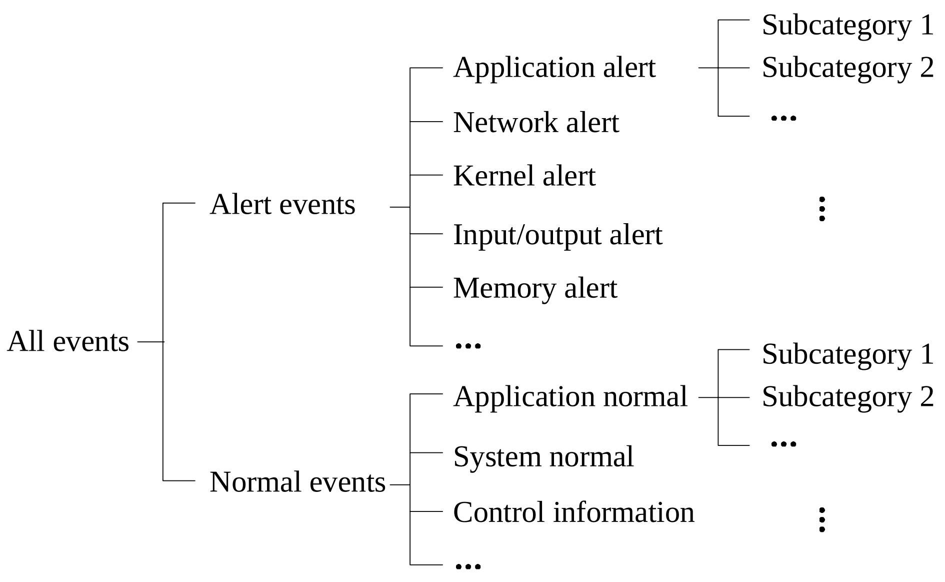


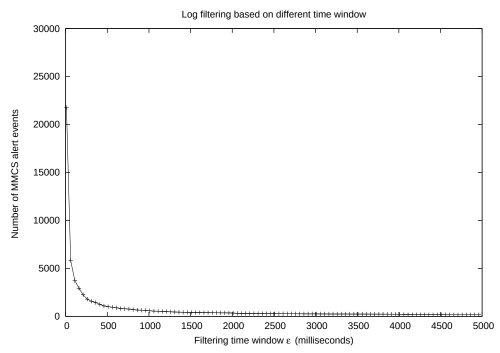
Figure 6.5: Hierarchical structure of the categories.

6.1.2 Filtering

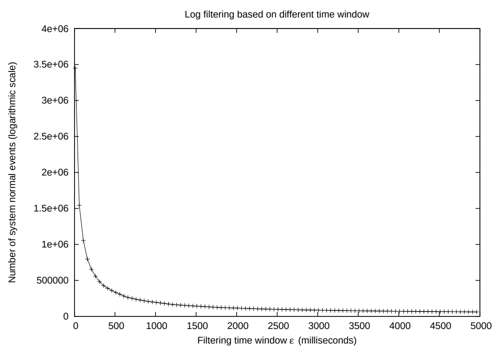
As described in Section 5.3.2, filtering combines several occurrences of the same event to prevent multiple reporting of the same problem. More specifically, there are two kinds of filtering techniques: temporal filtering, which removes temporally repeated events (redundancy in time), and spatial filtering, which removes spatially repeated events (redundancy in space).

To achieve temporal filtering, the optimal filtering time window ϵ, which is a core issue during the process, is determined by the method shown in Figure 5.5. The optimal value is identified graphically by plotting the number of resulting event numbers over various ϵ values. It can be seen that the optimal value of ϵ may differ for each category in filtering. The heuristic approach presented in Section 5.3.2 hopes to approximate the optimal ϵ. Figure 6.6 shows the plots for four different categories of events: Figure 6.6(a) presents MMCS alert events and the optimal ϵ is set to be approximately 250 milliseconds according to the heuristic approach; similarly, the optimal ϵ for filtering both system normal events and kernel alert events shown in Figure 6.6(b) and Figure 6.6(c) is close to 430 milliseconds; for the plot of application alert events in Figure 6.6(d), the optimal value is around 500 milliseconds. In contrast, Figure 5.5 shows the curve of total event numbers over various values for ϵ and the optimal value is set to 430 milliseconds.

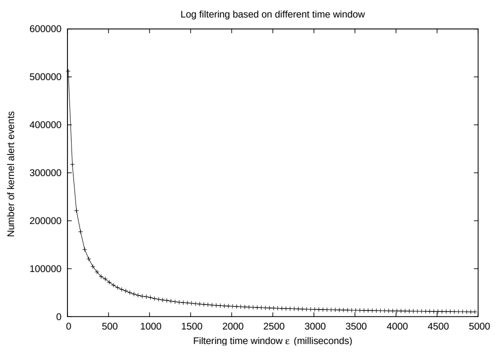
Evidently, a unique value for ϵ in the filtering is not optimum for some categories, for example, as shown in Figure 6.6, the usage of 430 milliseconds for the filtering time window will remove useful MMCS alert events, while retaining redundant application alert events. In our filtering mechanism, the optimal ϵ for each category is identified for the purpose of filtering so that all 62 categories may use different values for ϵ to remove redundancy to the greatest extent possible. Table 6.1 summarizes the results after temporal filtering.



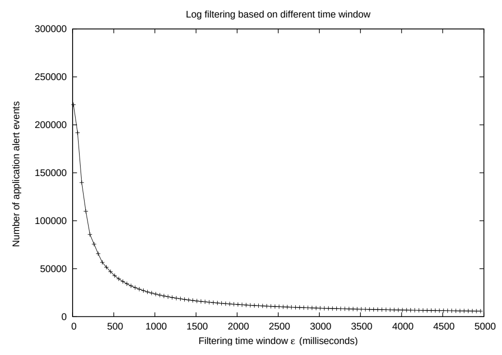
(a) MMCS alert events



(b) System normal events



(c) Kernel alert events



(d) Application alert events

Figure 6.6: Event filtering on various categories using the temporal filtering mechanism. 6.6(a) plots the curve of MMCS alert events, while 6.6(b), 6.6(c) and 6.6(d) present the curves of system normal events, kernel alert events and application alert events respectively.

Spatial filtering is not considered directly in this thesis for the following. Firstly, temporal filtering can remove spatial redundancy to a certain degree, because the occurrences of most spatially redundant events are close in time. Secondly, the redundancy in events has been significantly reduced after temporal filtering as shown in Table 6.1. Finally, as described in Section 5.3.2, before removal of spatial redundancy occurs, analysis of the dependencies between events should be carried out, which is a fundamental part of the prediction model. Thus, retaining such redundancy may improve prediction accuracy.

BlueGene/L	All categories
Original volume	4,747,963
After filtering	334,056
Compress ratio	14.2

Table 6.1: Summary of log filtering.

6.1.3 Sequence extraction

Event sequences, defined in Section 3.1, are extracted from the filtered event logs. As explained in Section 5.4.1, sequence extraction depends on three time intervals: lead-window Δt_l, data window Δt_d and margin time window Δt_m. For lead-window Δt_l, a value of five minutes has been used for the default configuration under the assumption that failure management actions, such as checkpoint construction, job restart, etc. can be performed in five minutes for most computer systems. Furthermore, failure prediction using various values for Δt_l is studied in later sections. In terms of data window Δt_d, a default value of five minutes is adopted. As demonstrated in Figure 5.6, margin time window Δt_m is used when extracting non-failure sequences. The value of Δt_m determines time intervals between healthy system states. As the time length a failure lasts cannot be measured, a value of five minutes for Δt_m is chosen in the experiments based on the assumption that the effects of a failure cannot last for more than five minutes.

As presented in Figure 5.7, two ordered prediction models are established for different forecasting purposes. The first prediction model tells whether the system is failure-prone, whilst the second prediction model further forecasts the particular failure type once a failure has been predicted. Obviously, the tags identified by multiple models over the same sequence are different. More specifically, model 1 marks a sequence as being either a failure sequence or a normal sequence; in contrast, for the same sequence, model 2 further analyses the failure categories if it is marked *failure* by model 1.

6.2 Failure characteristics

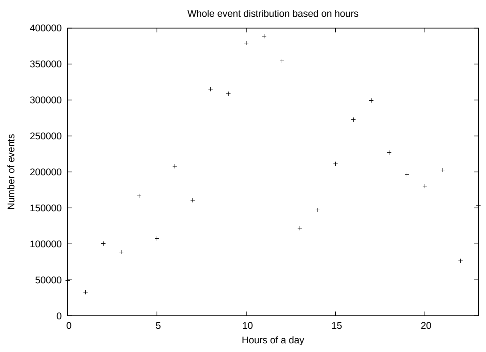
Key properties of the preprocessed data are analysed, such as distributions and delays of both normal events and alert events (described in Section 6.2.1 and Section 6.2.2, respectively), the correlations between normal events and alert events are presented in Section 6.2.3. Furthermore, data analysis helps to better understand the platform under investigation and may help other researchers to determine whether the methods and models presented in this thesis can be transferred to other systems.

6.2.1 Distribution of normal events

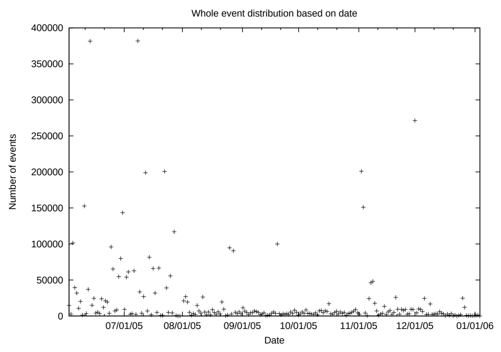
The prediction model builds on the timing of failure occurrence combined with the internal dependencies among events, then uses the trained patterns derived from the historical logs to estimate future failures. Therefore, normal events, including common errors, play an important role in the model.

Figure 6.7 shows two aspects of normal events: distribution and inter-arrival time (delays). More specifically, Figure 6.7(a) shows the normal event distribution broken down by hours in each day and demonstrates that midday is the system's busiest time, with night-time as its least active time. Figure 6.7(b) shows the event numbers in terms of days and that the dataset used for analysis is comprised of 215 days; it can be seen from the figure that the system usage declined towards the end of 2005. For the intervals between events, Figure 6.7(c) presents the histogram of frequency over normal events with interval ranging from 0 to 30 seconds, we can see the majority of the intervals fall into 0[1] and 1 second. The dataset contains 118,111 samples, 58,683 intervals fall into category 0 and 57,917 intervals fall into category 1, however, only 1,511 intervals range from 2 to 30 seconds. Figure 6.7(d) shows that delay 1 and delay 0 have the top two highest density values.

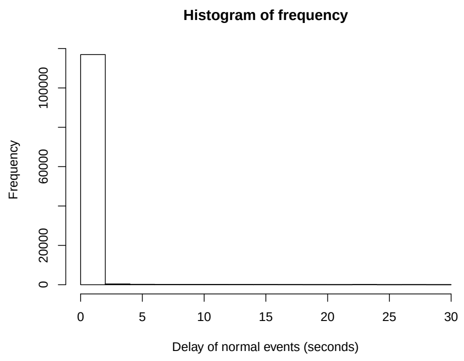
[1]A delay of 0 means two events have a time difference less than 1 second in the log. Specifically, the granularity of the IBM BlueGene/L RAS logging mechanism is less than 1 millisecond and multiple events may well have occurred in one second.



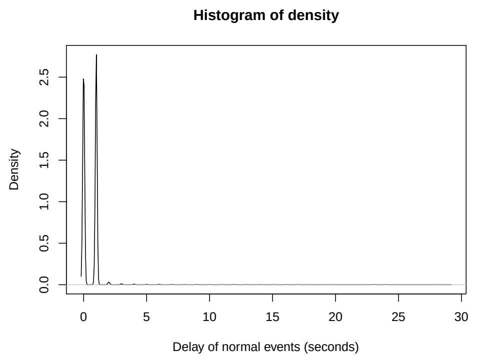
(a) Hourly normal event distribution



(b) Daily normal event distribution



(c) Histogram of frequency



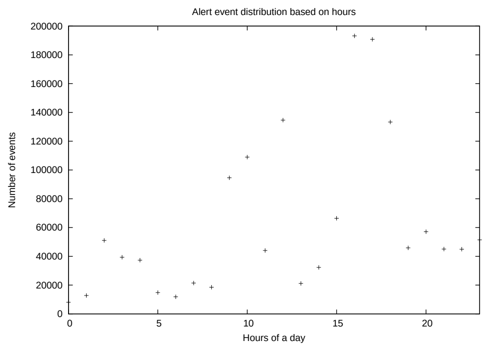
(d) Density distribution

Figure 6.7: Normal event distributions and delays (interval arrival time). Figure 6.7(a) shows the number of normal events broken down into 24 hour slots each day, Figure 6.7(b) presents the daily normal event distribution, Figure 6.7(c) demonstrates a histogram of normal event frequency, whilst Figure 6.7(d) shows the corresponding event density.

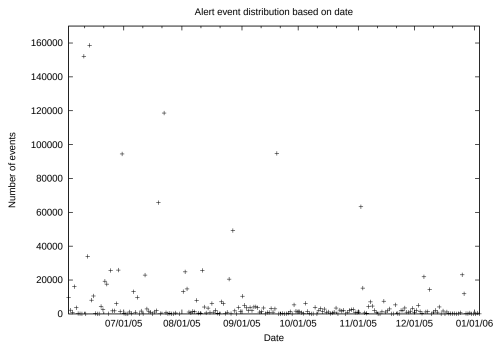
6.2.2 Distribution of alert events

Analysis of alert events is a major focus in this thesis and it has been studied in several other papers investigating the problem of failure prediction, as mentioned in Chapter 3. For example, the nearest neighbour predictor described in [65] makes use of event frequency, especially alert events, for future failure occurrence prediction. In contrast, other papers perform an analysis of the distribution of Time Between Failures (TBF) and try to fit the failure occurrence using an existing distribution law [13, 71, 83].

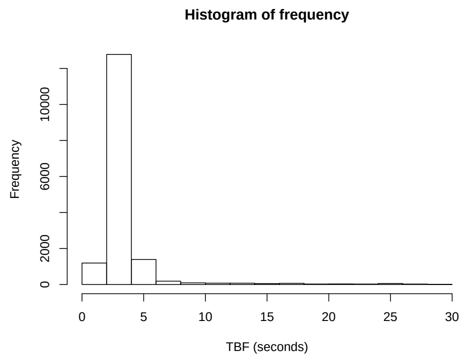
As the distribution of failures varies from system to system, and failures are not as common as normal events, the whole dataset of 215 days has been analysed. Similar to the analysis of normal events, Figure 6.8 provides the distributions and TBF of



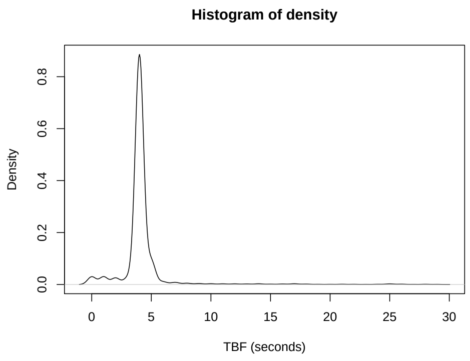
(a) Hourly alert event distribution



(b) Daily alert event distribution



(c) Histogram of frequency



(d) Density distribution

Figure 6.8: Alert event distributions and Time Between Failures (TBF). Figure 6.8(a) and Figure 6.8(b) presents the daily and hourly alert event distribution, Figure 6.8(c) shows the histogram of TBF frequency and Figure 6.8(d) shows the distribution of TBF density.

alert events. More specifically, Figure 6.8(a) presents the hourly alert event distribution, which has the same tendency as with normal events that midday has the largest number of alert events. In contrast, Figure 6.8(b) shows the daily number of alerts event numbers covering 215 days. It can be seen from the figure that the frequency of alert event occurrence changes from day to day and failure prediction-based on failure frequency is likely to be inappropriate in this circumstance. For TBF analysis, 17,573 values have been considered. Figure 6.8(c) presents a histogram of the frequency on TBF spanning from 0 to 30 seconds; correspondingly, Figure 6.8(d) presents the histogram of the density.

In order to obtain an overview of the TBF distributions, quantile-quantile plots

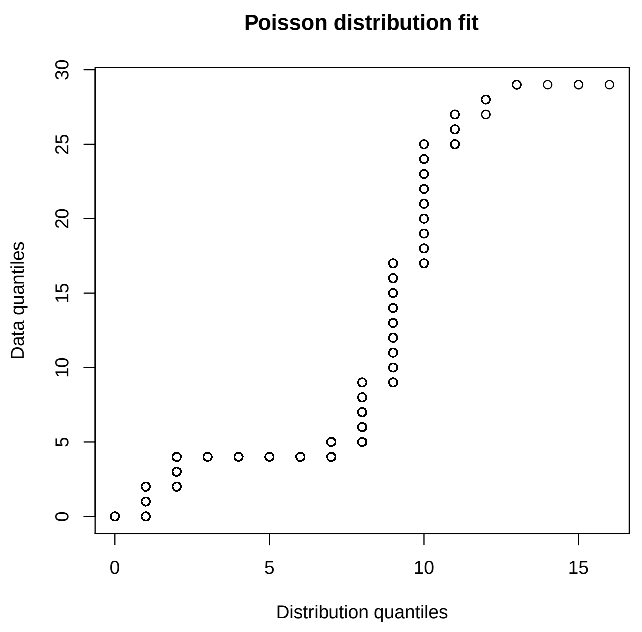
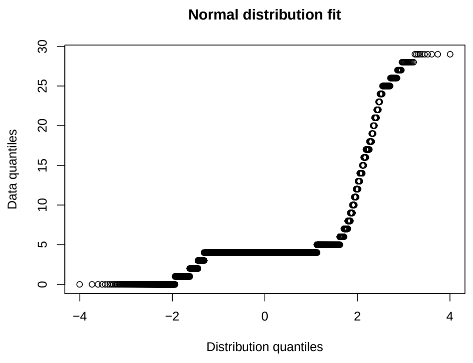


Figure 6.9: QQ-diagram of TBF using Poisson distribution.

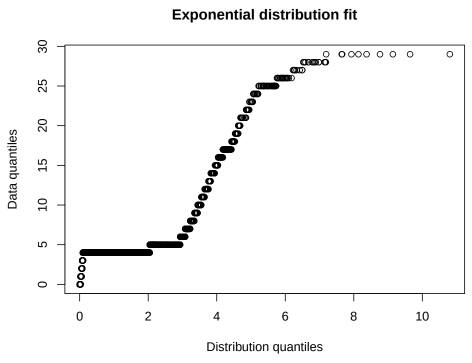
(QQ-plots), which plot quantiles of the observed data distribution against the parametric family of distributions, are provided in Figures 6.9 and 6.10 in order to investigate which parametric family of distributions best fit the data. More specifically, the fit with a Poisson distribution is present in Figure 6.9, and normal, exponential, lognormal and Weibull distributions are considered in Figures 6.10(a), 6.10(b), 6.10(c) and 6.10(d), respectively. Theoretically, a perfect match of quantiles is a straight line; thus it can be easily seen from the figures that exponential and lognormal distributions fit the data better than do the other three distributions. None have a perfect fit in this instance; however, these distributions can be used to give approximate answers to pertinent questions as discussed in later sections.

6.2.3 Correlation between normal events and alert events

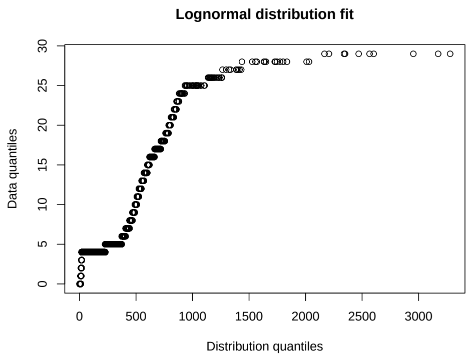
Correlation between normal events and alert events reflects the internal dependencies in components and this correlation in RAS logs has been explored for failure prediction, for example, in rule based mechanisms like Eventset [108]. However, these rules can only describe simple and straightforward correlations in events, rather than express complex dependencies. Thus the CRF model presented in this thesis is adopted to address this issue. In order to correctly understand the dependencies in this context, this



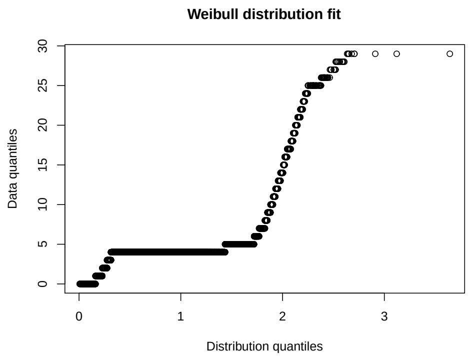
(a) Normal distribution fit



(b) Exponential distribution fit



(c) Lognormal distribution fit



(d) Weibull distribution fit

Figure 6.10: QQ-diagram of TBF. QQ-plots plots the distribution of TBF observed in the sample dataset versus several parametric distributions: normal, exponential, log-normal and weibull, which are shown in Figures 6.10(a), 6.10(b), 6.10(c) and 6.10(d) respectively.

section explores direct correlations among events.

It can be easily seen from the logs that some events[2] have the same event ID, which means there may be multiple events describing the execution progress of a single job. Motivated by this observation, for each alert event with a valid event ID, a search for normal events with the same event ID can help to examine whether the same job has reported ahead of normal events. For example, among the 168 memory alert events in the logs under investigation, there are 98 reported normal events from the corresponding job. Evidently, it can be concluded that there are normal events prior to most alert events and these correlations are captured for failure prediction.

[2]Events here means messages in RAS log files.

6.3 Training semi-Markov CRF

Data preprocessing, which deals with the raw RAS logs, is an important prerequisite for the failure prediction models presented in this thesis. This section introduces the steps involved in training semi-Markov CRF for failure prediction in the IBM Blue-Gene/L. In order to yield a more stable and reliable model, the dataset used for training covers a large number of events.

As described in Chapter 5, the training dataset, which is composed of event sequences, is used to learn the parameters of the final semi-Markov models. More specifically, the L-BFGS algorithm, which is presented in Section 5.4.3, is adopted in the experiments to improve training efficiency while reducing memory usage. The training task attempts to capture the following features in the training dataset.

> Each event sequence in the training dataset is the basic data instance and it is assigned a category label, which expresses the future events to which this sequence will lead. The sequence is extracted in a particular time window and the sequence category label depends on each event and its position in the sequence. These long range dependencies among events are captured in the training process by the target model.

As shown in Section 5.4.1, there are two prediction models contained in the mechanism. They both adopt the semi-Markov CRF model and use the L-BFGS algorithm for model training, however, the two models are used for different prediction purposes, thus experiments using different datasets and parameter spaces have been performed.

Parameter space

The prediction model depends on several aspects, such as dataset volume, category numbers, internal graphical model (semi-Markov CRF here), etc. Thus many parameters are involved in the training process. Although some of these have already been explained in Chapter 5, a brief introduction is provided here. Some parameters are closely related to performance issues (referred to as *performance-based parameters*, such as memory space, number of threads, etc.), while others have close links with the prediction model itself (referred to as *model-based parameters*). Due to the emphasis on model-based parameters, their possible values are explored in detail; in contrast, reasonable values are assumed for performance-based parameters.

More specifically, the following parameters have been configured during the training of the semi-Markov CRF model.

- *Number of sequence category labels.* As described in Chapter 5, each event sequence in the training dataset is assigned a category label. For different prediction models, the total number of category labels may be different. More specifically, for prediction model 1, according to its particular prediction target that determines whether there is a future failure, there are only two sequence category labels: failure or normal. In contrast, for prediction model 2, which further forecasts the failure category, the label number depends on the failure patterns that have been identified.

- *Number of event categories.* Each event sequence is comprised of multiple events, therefore the event category number determines the complexity of an event sequence. Furthermore, event categories in the dataset are processed in the *Categorization* stage (shown in Section 5.3.1). There are 62 categories in total. For both prediction model 1 and prediction model 2, they adopt the same event categories.

- *Number of optimization attempts.* As mentioned in Chapter 5, the L-BFGS algorithm, a step-based numerical optimization method, is used during the training process. The problem is determining how many attempts should be conducted to reach the global optimum. Evidently, this depends on various aspects such as the volume of the dataset, the complexity of each training instance, etc. In the experiment, a maximum value of 100 is configured for both models. If an optimum value has been reached before the algorithm has been performed 100 times, then the later tries will be ignored.

As shown in Figure 5.7, there are two prediction models and the detailed configuration of prediction model 1 are listed in Table 6.2. In terms of prediction model 2, it is used to further identify the specific failure type if a failure has been estimated, and it is not considered in the failure recovery scheme described in Chapter 4. However, it is important for fault diagnosis and system maintenance, and may be of value in a future fault recovery system. The detailed parameter configuration and experimental results are therefore presented in Appendix A.

6.4 Failure prediction

Failure prediction modelling is one of the main aspects of the work in this thesis, with details presented in Chapter 5. This section first introduces the evaluation metrics in

Parameters	Model 1
Sequence label number	2
Event category number	62
Maximum optimization tries	100
Underlying graphical model	semi-Markov CRF

Table 6.2: Configuration for prediction model 1.

terms of prediction quality and prediction stability, in Section 6.4.1. Section 6.4.2 demonstrates the results of prediction accuracy and analyses its relationships with several parameters. Finally, the results comparing this technique with four other prediction models are given in Section 6.4.3.

6.4.1 Evaluation metrics

This section introduces the metrics used for evaluation of the failure predictor in terms of prediction quality and prediction stability. Prediction quality depends on both the model and the dataset, while prediction stability reflects the generalization of the model over the target dataset.

Metrics for prediction quality

Widely accepted metrics for evaluation of such scenarios are *Precision*, *Recall* and *F-measure*, which are defined by the confusion matrix. *Precision* gives the ratio of correctly identified failures compared to the total number of failure predictions (Equation 6.1); *Recall* defines the ratio of correctly predicted failures compared with the total number of failures (Equation 6.2). The definitions of True Positive (TP), False Negative (FN), False Positive (FP) and True Negative (TN) are presented in Chapter 4 on page 74.

$$P_{precision} = \frac{TP}{TP + FP} \tag{6.1}$$

$$R_{recall} = \frac{TP}{TP + FN} \tag{6.2}$$

In order to integrate precision and recall into a single metric while considering the trade-off, *F-measure* is used [72]. *F-measure* is the weighted harmonic mean of precision and recall, defined by Equation 6.3, where $\alpha \in [0, 1]$. A common case is $F_{0.5}$ where precision and recall are equally weighted (shown in Equation 6.4).

$$F_{\alpha} = \frac{1}{\frac{\alpha}{P} + \frac{1-\alpha}{R}} = \frac{P * R}{(1-\alpha) * P + \alpha * R} \tag{6.3}$$

$$F_{0.5} = \frac{2 * P * R}{P + R} \tag{6.4}$$

Evaluation of prediction stability

Prediction accuracy mainly depends on two aspects: the internal graphical model and the target dataset. Semi-Markov CRF is adopted as the graphical model here; in terms of dataset, various factors may produce an effect on the final accuracy, such as data process, dataset volume, etc. Data process is explained in Section 5.3; this section mainly considers the problem of data volume.

For some real-world problems the available dataset is limited, hence in order to assess data quality and conduct experiments in an effective way, a technique named cross-validation is applied. Cross-validation assesses how well a prediction model will generalize to an independent data set [59]. It is an important technique to estimate the accuracy of a prediction model in a practical case. For the problem of failure prediction in the IBM BlueGene/L, although the log volume is very large, valuable events remaining after preprocessing are rare and all available data should be utilized as appropriate.

A common solution is to use m-fold cross validation, which processes the limited data set cyclically. More precisely, in m-fold cross validation, the original dataset is randomly partitioned into m disjoint subsets of equal size N/m, where N is the entire data size. Of the m subsets, select one subset for training each time, this is equivalent to using $1/m$ of the whole dataset as training data with the remaining subsets used as testing data. The experiments are then repeated m times, and each time different subsets are used for testing and training, so that all the data can be validated and the level of fit of a model to a data set can be estimated.

Another solution is to use Monte-Carlo cross-validation [87]. The main idea is that the dataset is randomly divided into a fraction, where α $(0 < \alpha < 1)$ for training and the remainder $(1 - \alpha)$ is for testing; the experiment is repeatedly several times and the results are then averaged over the splits. In this thesis, Monte-Carlo cross-validation is applied and 100 trials are used.

6.4.2 Prediction quality

Precision, recall and F-measure are used here to assess prediction quality. Specifically, 100 repetitions of the experiments have been performed over each model according to the default configurations. The parameters and results are listed in Table 6.3.

		Prediction model 1
Parameters	Data window Δt_d	5 min
	Lead-time Δt_l	5 min
	No. of states	2
Prediction	Precision	87.41%
	Recall	77.95%
	$F_{0.5}$	82.18%
Variance	Precision	0.38%
	Recall	0.69%
	$F_{0.5}$	0.41%

Table 6.3: Prediction accuracy and its variance of prediction model 1 according to the default parameter settings.

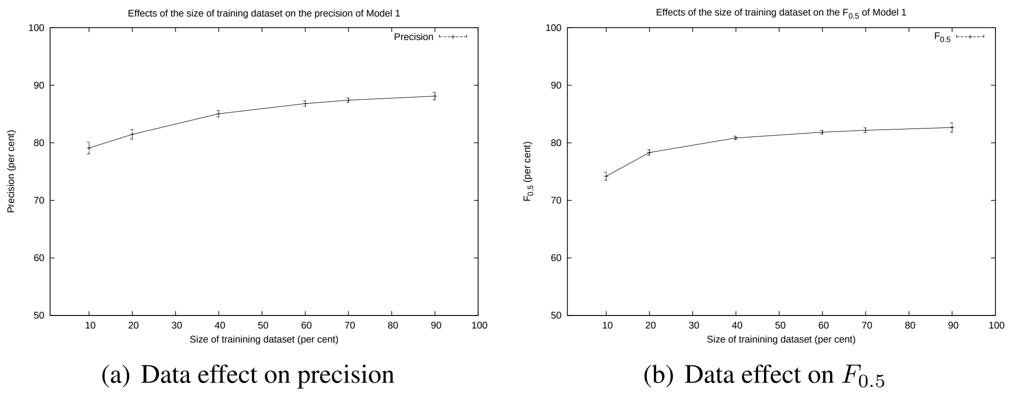
It can be seen from the table that 87.41% of failures can be identified by the first prediction model, with an overall F-measure $F_{0.5}$ of 82.18%. Another issue considered here is the prediction stability. As shown in Table 6.3, the variance of precision over the prediction model 1 is 0.38%, which means it has stable results.

As mentioned in the previous sections, there are multiple steps and parameters involved to set up a failure predictor and they have close relations to the final failure forecasting. Among these aspects, two kinds of issues are investigated here with respect to the prediction quality: *data specific* issues, which refer to data quality and quantity, and *preprocess* issues, which refer to properties imposed by the application domains including data window size Δt_d and lead time size Δt_l.

Dependency on data specific issues

The failure prediction model described in this thesis adopts the semi-Markov CRF model to learn unobservable relationships among the RAS dataset. This is a data-driven machine learning approach, which means the quality and quantity of available data will greatly influence the final prediction quality.

In order to investigate the effects of data quality on the prediction, the techniques of cross-validation, introduced in Section 6.4.1, are used for the evaluation. However, to explore the effects of training data size, two concerns may arise: firstly, in some application domains, the proportion of the entire dataset that is available for training may be limited; secondly, the most appropriate size of data for training is difficult to determine. More specifically, a basic rule in machine learning is to use as much data as possible for training; however, in some specific application domains, the system changes over time and old measured data may be unable to express the current system. Thus experiments have been conducted to show these effects.



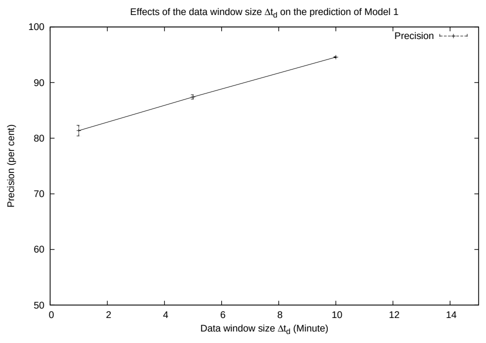
(a) Data effect on precision (b) Data effect on $F_{0.5}$

Figure 6.11: Effects of the size of training dataset on the prediction of model 1. Figure 6.11(a) plots the effects of the training dataset on the precision of model 1 with error bars, similarly, Figure 6.11(b) plots the effects on the $F_{0.5}$ with error bars.

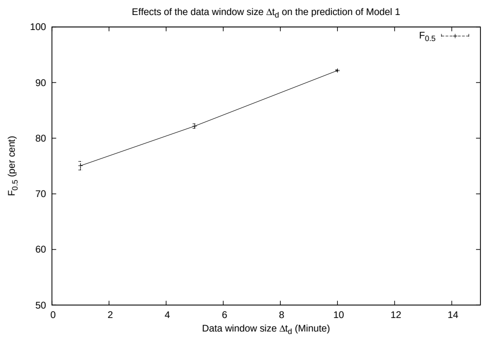
Experiments have been performed with training data sizes of 10%, 20%, 40%, 60%, 70%, 90% of the entire data, correspondingly the remaining 90%, 80%, 60%, 40%, 30%, 10% of the dataset are used for testing. Specifically, 100 iterations are performed for each size. Figure 6.11 shows the effects of the size of the training dataset on the prediction of model 1 and the error bars show variance. It can be seen from Figure 6.11(a) that the precision of model 1 increases gradually as the proportion of the training dataset is increased. In terms of variance, prediction results in the middle part of the curve are most stable. In the case of effects on $F_{0.5}$, a similar trend is found in Figure 6.11(b).

Dependency on data window Δt_d

As described in the previous sections, a general preprocess approach has been introduced and several steps have been conducted on the raw RAS log before they are used for real prediction. More precisely, multiple parameters have been involved with no relationship to specific applications, however, some parameters may significantly affect prediction quality. This section discusses the effects of the data window size Δt_d on prediction accuracy.



(a) Effect on precision



(b) Effect on $F_{0.5}$

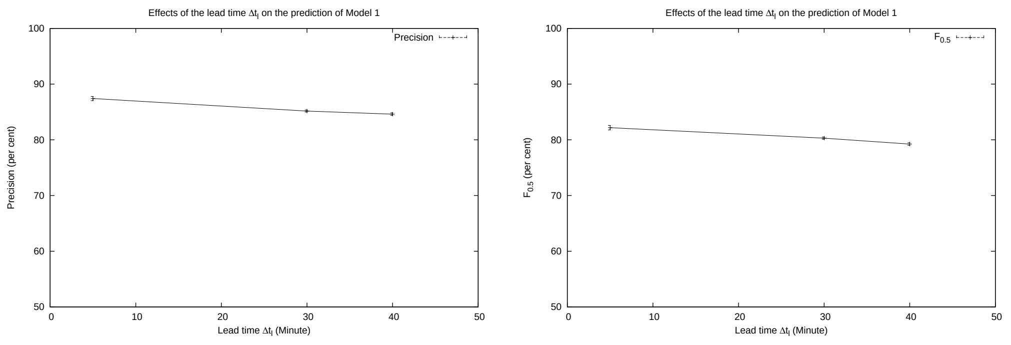
Figure 6.12: Effects of the length of data window Δt_d on the prediction of model 1. Figure 6.12(a) shows the effects of Δt_d on the precision of model 1, in contrast, Figure 6.12(b) presents the effects of Δt_d on the F-measure $F_{0.5}$.

Data window size Δt_d is used in the stage of sequence extraction to determine the length of event sequences used for failure prediction. Generally, longer sequences should produce more precise predictions. Thus experiments have been conducted to demonstrate this rule. For the effects of Δt_d, lengths of 1 minute, 5 minutes and 10 minutes over Δt_d are used in the experiment, and 100 iterations are performed on each length. Figure 6.12 shows the results. In terms of the precision and the $F_{0.5}$ shown in Figures 6.12(a) and 6.12(b), respectively, we can conclude that both have a notable increase as Δt_d increases, however, the opposite holds for the variance as the longer the Δt_d, the more stable is the prediction accuracy.

Another issue to be noted here is that increasing the data window size Δt_d can only improve the prediction accuracy to a certain degree, and the best accuracy can be obtained using a certain value of data window size denoted as $\Delta t_d{}'$, a larger data window will contain more noise, which may cause the problem of overfitting and decrease the prediction accuracy. To obtain the value of $\Delta t_d{}'$, experiments with varying data window size have to be performed, and the value of Δt_d when the turning point occurs is the optimal value. Because the available dataset is limited, particular lengths of data window size have been chosen during the experiment and only several points have been plotted in Figure 6.12.

Dependency on lead time Δt_l

Lead time Δt_l, defined in Chapter 3, denotes the time span from the current time to a future failure occurrence point. It also determines the time duration to perform proactive actions if a future failure is predicted. Lead time Δt_l therefore affects failure prediction from another perspective; if a small value is chosen for Δt_l, there may be not enough time to take actions before actual failure occurs, otherwise, a too large value for Δt_l may fail to capture the essential dependencies among events. In order to assess the effects of lead time, experiments with lead time ranging from 5 minutes to 40 minutes have been performed.



(a) Effect on precision (b) Effect on $F_{0.5}$

Figure 6.13: Effects of the length of lead time Δt_l on the prediction of model 1. Figure 6.13(a) shows the effects of Δt_l on precision and Figure 6.13(b) plots the changes of the $F_{0.5}$ according to Δt_l.

Figure 6.13 plots the effects of the length of Δt_l on precision and $F_{0.5}$. Specifically, lengths of 5 minutes, 30 minutes and 40 minutes are used for Δt_l. It is concluded that both precision and $F_{0.5}$ decline as Δt_l increases.

6.4.3 Comparison

The failure prediction approach has adopted semi-Markov CRFs, which extend standard CRFs in order to support continuous time features. In order to judge the prediction results shown in the previous sections, the new approach has been compared to several other published failure prediction approaches on the same IBM platform, as described in Chapter 3.

The IBM BlueGene/L is a widely used supercomputer listed in the Top500 [2] and much research has been done on failure prediction for this platform. These published approaches utilise a wide range of techniques, such as rule-based learning, statistical-based prediction and meta-learning techniques (discussed in detail in Section 3.2.2). Comparison to these approaches is given in this section, and prediction results using a standard CRF are also provided.

The comparison is conducted in terms of prediction accuracy and prediction scope. All experiments have been carried out using the RAS logs from the IBM BlueGene/L and the mean values of the results are adopted. As discussed in previous sections, the size of data window has great effects on the final prediction accuracy. To make a fair comparison, two aspects have been considered: data window size and variance of prediction accuracy. In terms of data window size, Eventset approach, Standard CRF based predictor and Semi-Markov CRF based predictor used a data window of 5 minutes, Nearest neighbour predictor used a data window of 48 hours, Meta-learning based predictor used a data window of 15 minutes to 1 hour. For variance, the accuracy of Eventset approach did not change much when the data window size was smaller than 10 minutes; the precision of Nearest neighbour predictor changed from 0.2 to 0.6 with data window varying from 4 hours to 48 hours, while recall had a range of 0.3 to 0.75; Meta-learning based predictor can reach the best recall of 0.78 when using a data window of 1 hour, however, the precision decreased to 0.65; the variance of CRF based predictor are shown in Section 6.4.2. The results are listed in Table 6.4.

Approaches	Precision	Recall	$F_{0.5}$
Eventset (IBM) [108]	47.45%	75%	58.06%
Nearest neighbour predictor [65]	48%%	75%	61.5%
Meta-learning based predictor [39]	76.5%	65%	70.75%
Standard CRF based predictor	72.16%	70.68%	71.36%
Semi-Markov CRF based predictor	87.41%	77.95%	82.18%

Table 6.4: Comparison of prediction results for various failure prediction approaches on the IBM BlueGene/L.

Figure 6.14 summarizes the prediction results of the approaches listed in Table 6.4, and also shows the optimum expected running times that are calculated by Equation 4.14. It can be concluded that the meta-learning based predictor has a longer execution time compared to other predictors, and the semi-Markov CRF based predictor achieves the best performance. Figure 6.14 also shows that further potential gains in performance can be achieved by improving the accuracy of precision and recall; moreover, improvement in accuracy of recall is more significant than that of precision.

For comparison of prediction scope, the other approaches are used to forecast

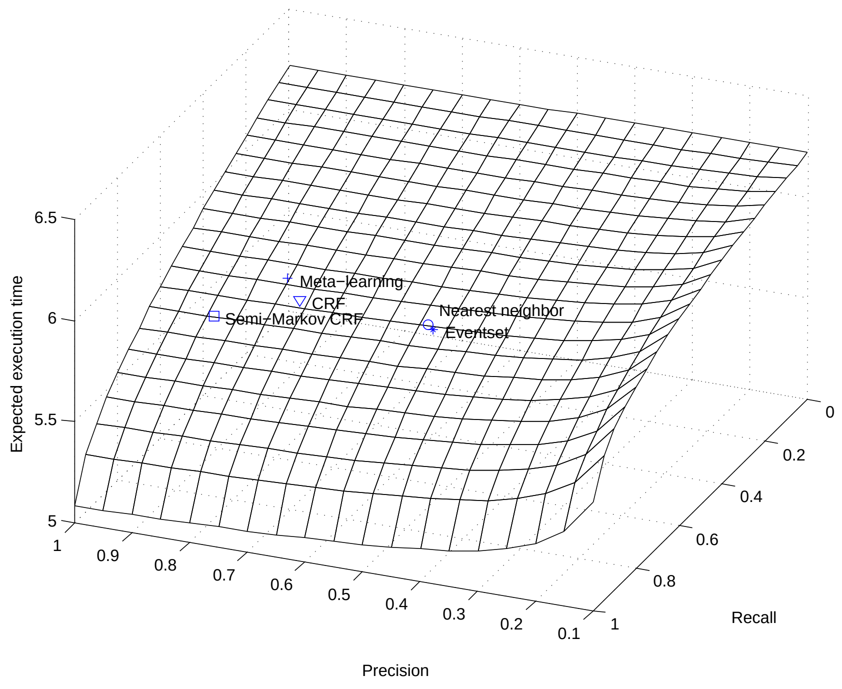


Figure 6.14: Summary of prediction results and the corresponding optimum execution times calculated by Equation 4.21.

whether the system is failure-prone at a future point, while the approach presented in this thesis uses a second semi-Markov CRF to further identify the specific failure type if a failure has been predicted, which is believed to be important for fault diagnosis and system maintenance, and may be exploited by some future fault recovery scheme.

6.5 Summary

In this chapter, the theory developed in previous chapters has been implemented and evaluated using real-world data in order to investigate the improvements that have been made in terms of accuracy and performance. The whole process can be divided into four main stages: data preprocessing, characteristic analysis, failure prediction modeling and proactive failure recovery.

In data preprocessing, real industrial data from the IBM BlueGene/L has been used. In order to prepare the raw data for data analysis and failure prediction model building, they are processed in three main steps: categorization, filtering and sequence extraction. More specifically, *categorization* identifies 62 subcategories, *filtering* reaches a compress ratio of 14.2, while *sequence extraction* generates both failure sequences and non-failure sequences according to various specifications for modelling.

In analysis of the RAS logs, the distributions of normal events and alert events have been investigated. Precisely, distributions of both normal events and alert events using different time scales have been explored to show the time delays between events. For alert events, TBF is an important parameter and the corresponding QQ-diagrams have been provided to show fitting results using four different parametric distributions over the data. Finally, the correlation between normal events and alert events has been explored.

For failure prediction modelling, the training process and the related parameter space are introduced. The prediction accuracy of the semi-Markov CRF used in the prediction approach (prediction model 1) are then provided. Furthermore, the effects of three main aspects on the prediction quality have been investigated. Finally, the failure predictor using semi-Markov CRF model is compared with four other approaches on the same dataset. The results show that the semi-Markov CRF based failure predictor performs better than the others it has been compared with. The main reason for this is believed to be that the semi-Markov CRF can more accurately express the target data.

Contributions of this chapter

There are two main contributions in this chapter. Firstly, the semi-Markov CRF based failure predictor has been evaluated using real industrial data in terms of various parameters to show the accuracy of the new predictor and a comparison with published approaches. Secondly, from the perspective of engineering, this chapter has shown the process to model a real-world system and to investigate the effects of various system specific parameters.

Relation to other chapters

This chapter describes in detail the implementation and results following from previous chapters. The whole process of failure prediction using real dataset from the IBM BlueGene/L has been investigated. Conclusion of the thesis is given in the next chapter.

Chapter 7

Conclusion

This thesis has presented the design and evaluation a prediction-based failure management framework aiming to improve the efficiency of failure recovery in large-scale supercomputers. The basic idea of the process can be described as follows: analyse system behaviours from events recorded in RAS log files, predict system future failures based on correlations derived from these events, then perform proactive failure recovery actions according to the predictions. The results presented in Chapter 6 show that the prediction accuracy achieved by the model described in Chapter 5 exceeds that of the three published approaches to failure recovery on the IBM BlueGene/L, and that prediction-based failure recovery can greatly reduce the wasted computational time of long-running applications compared with the standard coordinated checkpoint approach. This chapter summarizes essential aspects of the work, presents the main contributions and discusses several possible avenues for future work.

7.1 Summary

The ultimate goal of the thesis is to improve system dependability by using a proactive failure management framework. The objective of the framework is to perform proactive actions based on prediction of system failures — prediction is itself based on the system measurements recorded in log files. As a case study, system events stored in the RAS log files from an industrial platform: the IBM BlueGene/L, have been used for prediction. The content of this thesis can be divided into four phases: related work, failure prediction, proactive failure recovery and evaluation.

The first phase: related work discusses background to proactive failure recovery and derives a prediction-based coordinated checkpoint recovery approach (mentioned in Chapter 2) Following this, Chapter 2 starts with a problem statement that specifies the target platform, the IBM BlueGene/L and overviews the proposed solution of failure prediction: system event driven prediction using probabilistic models. Taxonomies of failure prediction methods for large-scale computing systems are given in terms of two aspects: methodology and algorithm. Three published approaches specifically designed for the target platform are discussed. Finally, the definition of the most common failure recovery technique, checkpointing, has been given, and a review of the published work on checkpoint mechanisms designed for supercomputers has been provided.

The proactive failure recovery phase describes the framework and analyses the theoretical motivation for using a checkpoint-based mechanism on supercomputers represented by the IBM BlueGene/L. It presents a theoretical model of the proactive failure recovery approach by deriving a formula to calculate the overall wasted computational time. The analysis is performed in terms of four prediction scenarios and a comparison of the wasted computing time between the proactive approach and the standard approach is provided. It then gives a condition, $P > t_s/(t_b - t_b^{'})$, which must be met by the prediction model to make the proactive approach perform better. Finally, a 3D curve has been demonstrated to show the potential performance gains with varying checkpoint intervals according to precision and recall.

The detail of failure prediction is presented in Chapter 5. The chapter introduces the three steps that make up the raw data pre-processing:

1. *Categorization* identifies 62 sub-categories of events;
2. *Filtering* contains temporal filtering and spatial filtering, and reaches a compress ratio of 14.2;
3. *Sequence extraction* generates both failure sequences and non-failure sequences for the purpose of modelling.

The two-level failure prediction mechanism using a semi-Markov model that can further identify the failure categories is discussed. Two basic inference problems are described given an observation sequence: (a) how to efficiently compute the probability of a specific state sequence; (b) which state sequence has the maximum likelihood. For the first problem, two algorithms, (*forward algorithm* and *backward algorithm*), are introduced to reduce the computation complexity from $O(N * M^N)$ to $O(N * M^2)$; the *Viterbi algorithm* is presented to solve the second problem. The problem of model training is discussed and numerical optimization algorithms including the Newton method, the quasi-Newton method and L-BFGS are reviewed for the training purpose — in this thesis L-BFGS is employed in the experiments. Finally, comparison in terms of independency assumptions is conducted between Conditional Random Field model and other models, such as Naive Bayes, HMM, logistic regression, MEMM, etc. The CRF model is believed to relax the dependency assumptions and hence address the label bias problem.

The evaluation phase reports the experiments that have been performed and the associated results to assess the effectiveness of the failure prediction strategy and the proactive failure recovery mechanism. The data preprocess experiments are described: *Categorization* establishes regular expressions to represent each event category and 40 alert event categories have been identified among the overall 62 categories; *Filtering* discusses the temporal filtering process and the optimal ϵ for each category is identified to remove as much redundancy as possible; for *sequence extraction*, the effects of different time windows are analysed. This phase provides analysis of failure characteristics. For normal events, it can be concluded from the distribution charts and density charts that midday is the busiest time and the majority of intervals between events last between 0 and 1 second. Similarly, the majority of TBF values are approximately 4 seconds. QQ-diagrams are provided to fit the TBF distribution using well-know parametric distributions, it is concluded that the occurrence of failures cannot be exactly expressed by these well-defined distributions.

Several experiments have been performed to evaluate the failure prediction strategy using two semi-Markov CRF models on different levels. The training process is introduced and three model related parameters are discussed for both models. Two aspects of evaluation metrics are presented: a confusion matrix is used to evaluate prediction quality, whilst cross-validation techniques are used to assess prediction stability. Prediction accuracy is discussed: average precision for prediction model 1 is 87.41%, and 41.38% for model 2; in terms of variance, model 1 is more stable and the reasons are analysed. The dependency of prediction quality on several parameters is explored, such as data window size Δt_d, lead time size Δt_l and training data volume. Finally, the comparison between the semi-Markov CRF based predictor and four other prediction approaches is conducted and it is concluded that the semi-Markov CRF based predictor can offer better prediction accuracy than the other tested approaches on the same dataset.

7.2 Contributions

In this thesis, a novel prediction-based failure management approach has been developed that has applied the semi-Markov model to capture the internal dependencies among events. The resulting prediction accuracy outperforms three published approaches when using real industrial RAS logs from the IBM BlueGene/L; the proactive failure recovery is built on the prediction module and the performance has been shown to be better than the standard checkpointing approach. Specifically, several contributions have been made:

- A novel model to theoretically analyse the wasted computational time of predic tion-based coordinated checkpoint mechanisms has been introduced, then analytical results of the model have been provided to demonstrate the efficiency. This is the first approach to integrate the identified prediction scenarios into the model.
- A two-level failure prediction mechanism using semi-Markov CRF has been designed for more detailed identification of the specific failure categories, and this thesis is the first to apply the semi-Markov CRF model and the standard CRF model to solve the problem of failure prediction.
- A comparison of the common probability models in terms of their assumptions is provided, then taxonomies of failure prediction methods have been introduced focusing to two different aspects: prediction methodology and prediction algorithm. A survey of the prediction approaches on the target platform, the IBM BlueGene/L, is also presented.

7.3 Future work

There are several directions for further investigations and improvements from the work described in this thesis. This section will discuss some potential and promising future work in terms of two main issues: failure prediction and proactive failure recovery.

For failure prediction, potential future work is described below.

- Failure prediction using locality

As described in Chapter 2, coordinated checkpoint mechanism is the most common failure recovery technique for supercomputers. The proactive failure recovery approach presented in this thesis has integrated a prediction module to improve the recovery efficiency. However, the prediction is conducted on a global scale. As can be observed, current supercomputers consist of thousands of elements: cores, multiple storage devices, distributed memory system, etc. If the prediction can be performed in local elements, such as memory, hard disk, network card, I/O component, etc, the recovery actions may in turn be able to be conducted locally, so that the overall fault tolerance efficiency can be greatly improved.

The prediction model 2, which further estimates the specific failure types and locates failures on a local device, has been described in Chapter 5, with prediction results shown in Appendix A. This prediction model can be used for failure prediction on a local scale. Furthermore, if hardware support of service monitoring is available on these devices, event logs from each local device can be captured and pre-processed by a similar process demonstrated in Chapter 5, then failure prediction can be conduced on a local device using semi-Markov model, and the prediction results can help design local recovery actions instead of constructing global checkpoints.

- Improvement of recall

  It can be seen from Figure 4.13 and Figure 6.14 that precision accuracy determines whether the prediction-based checkpoint approach performs better than the traditional approach, whereas evident benefit can be obtained by increasing recall accuracy. Thus one important future work would be to improve the accuracy of recall.

  The approach described in this thesis has used a machine learning method, and the dataset that has been collected is limited (for a time duration of 215 days), obviously, a larger dataset may improve recall accuracy. Thus using more data for training would be a future work.

- Failure prediction using varying datasets

  In this thesis, the event-driven failure predictor uses the RAS logs as the input to the semi-Markov Model. This is a machine learning process and the prediction approach is a generic method, thus any event dataset can be used. Two directions can be further investigated according to this principle. Firstly, for systems without event logs, system measurements from monitoring components can be transferred to event-based data using some process techniques, for example, thresholds can be used to transfer electronic signals to events. Secondly, the failure predictor demonstrated in this thesis uses the data from the IBM BlueGene/L as a case study, hence event logs from other supercomputers can be similarly processed with appropriate preprocessing.

In the case of proactive failure recovery, potential further work is considered below.

- Moving to real supercomputers

  A potential future area is evaluation of the proactive failure recovery mechanism in a real supercomputer environment. Evidently, a prerequisite to perform proactive actions is to have support for failure prediction. Thus a prediction module must be built first in the system. Furthermore, system event logs must be available for the purpose of failure prediction.

- Mechanism improvement

  The prediction-based checkpoint mechanism described in the thesis mainly depends on the prediction model and there is scope for improvement. *Fault diagnosis* locates the root cause of failures and accurate location may greatly improve the performance of proactive failure recovery. For example, if a CPU core has been diagnosed faulty, the tasks scheduled on this core can be rescheduled to other resources or the damaged CPU core may be replaced. However, fault diagnosis is a complex problem and future work on this issue is needed.

Bibliography

[1] Loghound - a tool for mining frequent patterns from event logs. http://ristov.users.sourceforge.net/loghound/, last accessed August 1st, 2009.

[2] Top 500 supercomputers. http://www.top500.org/, last accessed August 1st, 2010.

[3] Seti@home. http://setiathome.berkeley.edu, last accessed May 1st, 2010.

[4] J. H. Abawajy. Fault detection service architecture for grid computing systems. In *Computational Science and Its Applications - ICCSA 2004*, pages 107–115. 2004.

[5] Paul E. Ammann and John C. Knight. Data diversity: An approach to software fault tolerance. *IEEE Transactions on Computers*, 37(4):418–425, 1988.

[6] A. Avizienis and J.-C. Laprie. Dependable computing: From concepts to design diversity. *Proceedings of the IEEE*, 74(5):629 – 638, May 1986.

[7] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1:11–33, 2004.

[8] Mordecai Avriel. *Nonlinear Programming: Analysis and Methods*. Dover Publishing, 2003.

[9] Mohammad Abdollahi Azgomi and Ali Movaghar. A modelling tool for hierarchical stochastic activity networks. *Simulation Modelling Practice and Theory*, 13(6):505 – 524, 2005.

[10] Adam Beguelin, Erik Seligman, and Peter Stephan. Application level fault tolerance in heterogeneous networks of workstations. *Journal Parallel Distributed Computing*, 43(2):147–155, 1997. 263044.

[11] Adam Berger. The improved iterative scaling algorithm: A gentle introduction, 1997. Unpublished manuscript, 1997.

[12] Adam L. Berger, Vincent J. Della Pietra, and Stephen A. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.

[13] Mohamed Slim Bouguerra, Denis Trystram, Thierry Gautier, and Jean-Marc Vincent. A new flexible Checkpoint/Restart model. Research Report RR-6751, INRIA, 2008.

[14] James C. Browne, Madulika Yalamanchi, Kevin Kane, and Karthikeyan Sankaralingam. General parallel computations on desktop grid and p2p systems. In *LCR '04: Proceedings of the 7th workshop on Workshop on languages, compilers, and run-time support for scalable systems*, pages 1–8, New York, NY, USA, 2004. ACM.

[15] Franck Cappello. Fault tolerance in petascale/ exascale systems: Current knowledge, challenges and research opportunities. *International Journal of High Performance Computing Applications*, 23(3):212–226, 2009.

[16] Henri Casanova, Jack Dongarra, Chris Johnson, and Michelle Miller. Application-specific tools. In *The Grid: blueprint for a new computing infrastructure*, pages 159–180. Morgan Kaufmann Publishers Inc., 1999. 296098.

[17] A. J. Chakravarti, G. Baumgartner, and M. Lauria. The organic grid: self-organizing computation on a peer-to-peer network. *IEEE Transactions on Systems, Man and Cybernetics, Part A,*, 35(3):373–384, 2005. 1083-4427.

[18] Liming Chen and Algirdas Avizienis. N-version programming: A fault-tolerance approach to reliability of software operation. *Proceedings of the 8th International Symposium on Fault-Tolerant Computing Systems.*, pages 3–9, 1974.

[19] William S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, 74(368):829–836, 1979.

[20] The Condor project. http://www.cs.wisc.edu/condor/. last accessed May 1st, 2010.

[21] Robert M. Corless, David J. Jeffrey, and Donald E. Knuth. A sequence of series for the lambert w function. In *Proceedings of the 1997 international symposium on Symbolic and algebraic computation*, ISSAC '97, pages 197–204, New York, NY, USA, 1997. ACM.

[22] Aron Culotta, David Kulp, and Andrew McCallum. Gene prediction with conditional random fields. Technical report, 2005.

[23] David Decaprio, Jade P. Vinson, Matthew D. Pearson, Philip Montgomery, Matthew Doherty, and James E. Galagan. Conrad: Gene prediction using conditional random fields. *Genome Research*, 1:1389–1398, 2007.

[24] A. Duarte, F. Brasileiro, W. Cirne, and J. A. Filho. Collaborative fault diagnosis in grids through automated tests. In *20th International Conference on Advanced Information Networking and Applications. AINA 2006*, volume 1, page 6, 2006.

[25] Andrzej Duda. The effects of checkpointing on program execution time. *Information Processing Letters*, 16(5):221–229, 1983.

[26] P. Felber, X. Defago, R. Guerraoui, and P. Oser. Failure detectors as first class objects. In *1999. Proceedings of the International Symposium on Distributed Objects and Applications*, pages 132 –141, 1999.

[27] I. Foster. The anatomy of the Grid: enabling scalable virtual organizations. In *2001 Proceedings of First IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 6–7, 2001.

[28] I. Foster. The Grid: A new infrastructure for 21st century science. *Physics Today*, 55:42–47, 2002.

[29] Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., 2003. 996313.

[30] Ian T. Foster and Carl Kesselman. Computational grids. *VECPAR*, pages 3–37, 2000.

[31] James Frey, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke. Condor-g: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, 2002. 10.1023/A:1015617019423.

[32] Song Fu and Cheng-Zhong Xu. Quantifying temporal and spatial correlation of failure events for proactive management. In *SRDS '07: Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems*, pages 175–184, Washington, DC, USA, 2007. IEEE Computer Society.

[33] Song Fu and Cheng-Zhong Xu. Quantifying temporal and spatial correlation of failure events for proactive management. In *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*, pages 175 –184, 2007.

[34] A. Gara, M. A. Blumrich, D. Chen, L-T G. Chiu, P. Coteus, M. E. Giampapa, R. A. Haring, P. Heidelberger, D. Hoenicke, G. V. Kopcsay, T. A. Liebsch, M. Ohmacht, B. D. Steinmacher-Burow, T. Takken, and P. Vranas. Overview of the BlueGene/L system architecture. *IBM Journal of Research and Development*, 49(2):195–212, March 2005.

[35] S. Garg, A. Van Moorsel, K. Vaidyanathan, and K. S. Trivedi. A methodology for detection and estimation of software aging. In *ISSRE '98: Proceedings of the The Ninth International Symposium on Software Reliability Engineering*, page 283, Washington, DC, USA, 1998. IEEE Computer Society.

[36] Zheng Gengbin, Huang Chao, and V. Kal Laxmikant. Performance evaluation of automatic checkpoint-based fault tolerance for ampi and charm++. *SIGOPS Operation System.*, 40(2):90–99, 2006. 1131340.

[37] R. O. Gilbert. *Statistical methods for environmental pollution monitoring*. 1987.

[38] C. Gordon, C. Cooper, C. A. Senior, H. Banks, J. M. Gregory, T. C. Johns, J. F. B. Mitchell, and R. A. Wood. The simulation of SST, sea ice extents and ocean heat transports in a version of the Hadley Centre coupled model without flux adjustments. *SPRINGER*, 2000.

[39] Jiexing Gu, Ziming Zheng, Zhiling Lan, John White, Eva Hocks, and Byung-Hoon Park. Dynamic meta-learning for failure prediction in large-scale systems: A case study. In *ICPP '08: Proceedings of the 2008 37th International Conference on Parallel Processing*, pages 157–164, Washington, DC, USA, 2008. IEEE Computer Society.

[40] Xiaohui Gu, S. Papadimitriou, P. S. Yu, and Shu-Ping Chang. Toward predictive failure management for distributed stream processing systems. In *Proc. 28th*

International Conference on Distributed Computing Systems ICDCS '08, pages 825–832, 17–20 June 2008.

[41] Prashasta Gujrati, Yawei Li, Zhiling Lan, Rajeev Thakur, and John White. A meta-learning failure predictor for blue gene/l systems. In *ICPP '07: Proceedings of the 2007 International Conference on Parallel Processing*, page 40, Washington, DC, USA, 2007. IEEE Computer Society.

[42] Asela Gunawardana, Milind Mahajan, Alex Acero, and John C. Platt. Hidden conditional random fields for phone classification. In *9th European Conference on Speech Communication and Technology*, Lisbon, Portugal, September 2005.

[43] Greg Hamerly and Charles Elkan. Bayesian approaches to failure prediction for disk drives. In *ICML '01: Proceedings of the Eighteenth International Conference on Machine Learning*, pages 202–209, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[44] J.P. Hansen and D.P. Siewiorek. Models for time coalescence in event logs. In *Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on*, pages 221–227, Jul 1992.

[45] N. Hayashibara, A. Cherif, and T. Katayama. Failure detectors for large-scale distributed systems. In *Proceedings 21st IEEE Symposium on Reliable Distributed Systems.*, pages 404–409, 2002.

[46] Juergen Hofer, Juergen Hofer, and Thomas Fahringer. A multi-perspective taxonomy for systematic classification of grid faults. In Thomas Fahringer, editor, *16th Euromicro Conference on Parallel, Distributed and Network-Based Processing. PDP 2008.*, pages 126–130, 2008.

[47] Jiman Hong, Sangsu Kim, Yookun Cho, H. Y. Yeom, and Taesoon Park. On the choice of checkpoint interval using memory usage profile and adaptive time series analysis. In *Proc. Pacific Rim International Symposium on Dependable Computing*, pages 45–48, 17–19 Dec. 2001.

[48] J. Horning, H. Lauer, P. Melliar-Smith, and B. Randell. A program structure for error detection and recovery. In *Operating Systems*, pages 171–187. 1974. 10.1007/BFb0029359.

[49] Yennun Huang, Chandra Kintala, Nick Kolettis, and N. Dudley Fulton. Software rejuvenation: Analysis, module and applications. *Fault-Tolerant Computing, International Symposium on*, 0:0381, 1995.

[50] G.F. Hughes, J.F. Murray, K. Kreutz-Delgado, and C. Elkan. Improved disk-drive failure warnings. *IEEE Transactions on Reliability,*, 51(3):350 – 357, Sep 2002.

[51] Soonwook Hwang. A generic failure detection service for the grid, 2003. Ph.D. thesis, University of Southern California.

[52] B. Javadi, D. Kondo, J.-M. Vincent, and D.P. Anderson. Mining for statistical models of availability in large-scale distributed systems: An empirical study of seti@home. In *IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems, 2009. MASCOTS '09.*, pages 1 –10, sept. 2009.

[53] Feng Jiao, Shaojun Wang, Chi-Hoon Lee, Russell Greiner, and Dale Schuurmans. Semi-supervised conditional random fields for improved sequence segmentation and labeling. In *ACL-44: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 209–216, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

[54] Jeffrey O. Kephart. Research challenges of autonomic computing. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 15–22, New York, NY, USA, 2005. ACM.

[55] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003. 0018-9162.

[56] Kenneth R. Mayes John R. Gurd Keping Chen. Autonomous performance control of distributed applications in a heterogeneous environment. In *1st International Conference on Autonomics*, Rome, Italy, 2007.

[57] E. Kiciman and A. Fox. Detecting application-level failures in component-based internet services. *IEEE Transactions on Neural Networks*, 16(5):1027–1041, 2005. 1045-9227.

[58] Jorg Kienzle. Software fault tolerance: An overview. In *Reliable Software Technologies - Ada-Europe 2003*, pages 641–641. 2003. 10.1007/3-540-44947-7_3.

[59] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI'95: Proceedings of the 14th international joint conference on Artificial intelligence*, pages 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[60] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proc. 18th International Conf. on Machine Learning*, pages 282–289, 2001.

[61] Ronjeet Lal and Gwan S. Choi. Error and failure analysis of a unix server. In *HASE '98: The 3rd IEEE International Symposium on High-Assurance Systems Engineering*, pages 232–239, Washington, DC, USA, 1998. IEEE Computer Society.

[62] Zhiling Lan, Jiexing Gu, Ziming Zheng, Rajeev Thakur, and Susan Coghlan. A study of dynamic meta-learning for failure prediction in large-scale systems. *Journal of Parallel and Distributed Computing*, In Press, Corrected Proof, 2010.

[63] Ziming Li and Dabin Zheng. Computation with hyperexponential functions. *SIGSAM Bull.*, 39(3):84–85, 2005.

[64] Y. Liang, Y. Zhang, M. Jette, Anand Sivasubramaniam, and R. Sahoo. BlueGene/L failure analysis and prediction models. In *International Conference on Dependable Systems and Networks, 2006. DSN 2006.*, pages 425–434, June 2006.

[65] Yinglung Liang. *Failure Analysis, Modeling and Prediction for BlueGene/L.* PhD thesis, The State University of New Jersey, 2007.

[66] Yinglung Liang, Anand Sivasubramaniam, and Jose Moreira. Filtering failure logs for a bluegene/l prototype. In *DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks*, pages 476–485, Washington, DC, USA, 2005. IEEE Computer Society.

[67] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra Sahoo. An adaptive semantic filter for blue gene/l failure log analysis. *Parallel and Distributed Processing Symposium, International*, 0:445, 2007.

[68] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra Sahoo. Failure prediction in IBM BlueGene/L event logs. In *ICDM '07: Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, pages 583–588, Washington, DC, USA, 2007. IEEE Computer Society.

[69] Dong C. Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45:503–528, 1989. 10.1007/BF01589116.

[70] Yang Liu, Andreas Stolcke, Elizabeth Shriberg, and Mary Harper. Using conditional random fields for sentence boundary detection in speech. In *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 451–458, Morristown, NJ, USA, 2005. Association for Computational Linguistics.

[71] Yudan Liu, R. Nassar, C. Leangsuksun, N. Naksinehaboon, M. Paun, and S.L. Scott. An optimal checkpoint/restart model for a large scale high performance computing system. In *IPDPS 2008: IEEE International Symposium on Parallel and Distributed Processing, 2008*, pages 1 –9, 14-18 2008.

[72] John Makhoul, Francis Kubala, Richard Schwartz, and Ralph Weischedel. Performance measures for information extraction. In *In Proceedings of DARPA Broadcast News Workshop*, pages 249–252, 1999.

[73] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Min. Knowl. Discov.*, 1(3):259–289, 1997.

[74] Andrew McCallum, Dayne Freitag, and Fernando C. N. Pereira. Maximum entropy markov models for information extraction and segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 591–598, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[75] R. Medeiros, W. Cirne, F. Brasileiro, and J. Sauve. Faults in grids: why are they so bad and what can be done about it? In *Proceedings. Fourth International Workshop on Grid Computing.*, pages 18–24, 2003.

[76] In Jae Myung. Tutorial on maximum likelihood estimation. *Journal of Mathematical Psychology*, 47(1):90 – 100, 2003.

[77] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization.* Nueva York, EUA : Springer, 1999.

[78] Daniel Nurmi, John Brevik, and Rich Wolski. Modeling machine availability in enterprise and wide-area distributed computing environments. In *Euro-Par 2005 Parallel Processing*, pages 432–441. 2005. 10.1007/11549468_50.

[79] Daisuke Okanohara, Yusuke Miyao, Yoshimasa Tsuruoka, and Jun'ichi Tsujii. Improving the scalability of semi-markov conditional random fields for named entity recognition. In *ACL-44: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 465–472, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

[80] Adam Oliner and Jon Stearley. What supercomputers say: A study of five system logs. In *DSN '07: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 575–584, Washington, DC, USA, 2007. IEEE Computer Society.

[81] Adam J. Oliner, Larry Rudolph, and Ramendra K. Sahoo. Cooperative checkpointing: a robust approach to large-scale systems reliability. In *ICS '06: Proceedings of the 20th annual international conference on Supercomputing*, pages 14–23, New York, NY, USA, 2006. ACM.

[82] David Lorge Parnas. Software aging. In *ICSE '94: Proceedings of the 16th international conference on Software engineering*, pages 279–287, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[83] Karthik Pattabiraman, Christopher Vick, and Alan Wood. Modeling coordinated checkpointing for large-scale supercomputers. In *DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks*, pages 812–821, Washington, DC, USA, 2005. IEEE Computer Society.

[84] Tao Qin, Tie-Yan Liu, Xu-Dong Zhang, De-Sheng Wang, and Hang Li. Global ranking using continuous conditional random fields. In *NIPS*, pages 1281–1288, 2008.

[85] Ariadna Quattoni, Michael Collins, and Trevor Darrell. Conditional random fields for object recognition. In *In NIPS*, pages 1097–1104. MIT Press, 2004.

[86] F. Ramos, D. Fox, and H. Durrant-Whyte. Crf-matching: Conditional random fields for feature-based scan matching. In *Proceedings of Robotics: Science and Systems*, Atlanta, GA, USA, June 2007.

[87] C. R. Rao and Y. Wu. Linear model selection by cross-validation. *Journal of Statistical Planning and Inference*, 128(1):231 – 240, 2005.

[88] X. Ren, S. Lee, R. Eigenmann, and S. A. Bagchi S. Bagchi. Resource availability prediction in fine-grained cycle sharing systems. In S. Lee, editor, *15th IEEE International Symposium on High Performance Distributed Computing,*, pages 93–104, 2006.

[89] Brent Rood and Michael J. Lewis. Resource availability prediction for improved grid scheduling. In *ESCIENCE '08: Proceedings of the 2008 Fourth IEEE International Conference on eScience*, pages 711–718, Washington, DC, USA, 2008. IEEE Computer Society.

[90] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam. Critical event prediction for proactive management in large-scale computer clusters. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–435, New York, NY, USA, 2003. ACM.

[91] F. Salfner. Predicting failures with hidden markov models. In *Proceedings of the 5th European Dependable Computing Conference (EDCC-5), Student Forum*, Budapest, Hungary, April 2005.

[92] F. Salfner, M. Schieschke, and M. Malek. Predicting failures of computer systems: a case study for a telecommunication system. In *Proc. 20th International Parallel and Distributed Processing Symposium IPDPS 2006*, page 8, 25–29 April 2006.

[93] Felix Salfner, Maren Lenk, and Miroslaw Malek. A survey of online failure prediction methods. *ACM Computing Surveys*, 42(3):1–42, 2010.

[94] Sunita Sarawagi and William W. Cohen. Semi-markov conditional random fields for information extraction. In *In Advances in Neural Information Processing Systems 17*, pages 1185–1192, 2004.

[95] D. R. Schertz. Fault-tolerant computing: An introduction. *IEEE Transactions on Computing*, 23(7):649–650, 1974.

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

[97] B. Schroeder and G. A. Gibson. Understanding failures in petascale computers. *Journal of Physics Conference Series*, 78(1):012022–+, July 2007.

[98] L. M. Silva and J. G. Silva. System-level versus user-defined checkpointing. In *Proceedings 17th IEEE Symposium on Reliable Distributed Systems*. IEEE Computer Society, Proceedings 17th IEEE Symposium on Reliable Distributed Systems, 1998. 830990 68.

[99] Krishnan Sriram and D. Gannon. Checkpoint and restart for distributed components in xcat3. In D. Gannon, editor, *Proceedings Fifth IEEE/ACM International Workshop on Grid Computing.*, pages 281–288, 2004.

[100] P. Stelling, I. Foster, C. Kesselman, C. Lee, and G. Von Laszewski. A fault detection service for wide area distributed computations. In *The Seventh International Symposium on High Performance Distributed Computing, 1998. Proceedings*, pages 268–278, 1998.

[101] Xiao Sun, Degen Huang, and Ren Fuji. Chinese lexical analysis based on hidden semi-crf. In *International Conference on Intelligent Computing*, 2009.

[102] Charles Sutton and Andrew McCallum. *Introduction to Conditional Random Fields for Relational Learning*. MIT Press, 2006.

[103] Youhei Tanaka, Naohiro Hayashibara, Tomoya Enokido, and Makoto Takizawa. A fault-tolerant transactional agent model on distributed objects. In *ISORC '06: Proceedings of the Ninth IEEE International Symposium on Object and*

Component-Oriented Real-Time Distributed Computing, pages 279–286, Washington, DC, USA, 2006. IEEE Computer Society.

[104] Youhei Tanaka, Naohiro Hayashibara, Tomoya Enokido, and Makoto Takizawa. Fault detection and recovery in a transactional agent model. In *AINA '07: 21st International Conference on Advanced Information Networking and Applications*, pages 126–133, 2007.

[105] The BlueGene/L Team, T Domany, and et al. An overview of the BlueGene/L supercomputer, 2002.

[106] R. Teodorescu and F. Blaabjerg. Flexible control of small wind turbines with grid failure detection operating in stand-alone and grid-connected mode. *IEEE Transactions on Power Electronics*, 19(5):1323–1332, 2004. 0885-8993.

[107] R. Vilalta, C. V. Apte, J. L. Hellerstein, S. Ma, and S. M. Weiss. Predictive algorithms in the management of computer systems. *IBM Systems Journal*, 41(3):461–474, 2002.

[108] R. Vilalta and Sheng Ma. Predicting rare events in temporal domains. In *ICDM 2002 Proceedings: IEEE International Conference on Data Mining, 2002.*, pages 474–481, 2002.

[109] Sy Bor Wang, Ariadna Quattoni, Louis-Philippe Morency, and David Demirdjian. Hidden conditional random fields for gesture recognition. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1521–1527, Washington, DC, USA, 2006. IEEE Computer Society.

[110] Gary M. Weiss and Haym Hirsh. Learning to predict rare events in event sequences. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 359–363, 1998.

[111] Shi Xuanhua, Jin Hai, Han Zongfen, Qiang Weizhong, Wu Song, and Zou Deqing. Alter: adaptive failure detection services for grids. In *2005 IEEE International Conference on Services Computing*, volume 1, pages 355–358 vol.1, 2005.

[112] Zhenghua Xue, Xiaoshe Dong, Siyuan Ma, and Weiqing Dong. A survey on failure prediction of large-scale server clusters. In *Eighth ACIS International*

Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007., volume 2, pages 733–738, Aug. 2007.

[113] John W. Young. A first order approximation to the optimum checkpoint interval. *Communications of the ACM*, 17(9):530–531, 1974.

[114] Jia Yu and Rajkumar Buyya. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Record.*, 34(3):44–49, 2005. 1084814.

[115] Yanyong Zhang and A. Sivasubramaniam. Failure prediction in ibm bluegene/l event logs. In *IEEE International Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008.*, pages 1–5, April 2008.

[116] Hai Zhao, Chang-Ning Huang, and Mu Li. An improved chinese word segmentation system with conditional random field. In David Abramson, editor, *Proceedings of Fifth SIGHAN Workshop on Chinese Language Processing*, pages 162–165, 2006.

[117] Ziming Zheng, Zhiling Lan, B.H. Park, and A. Geist. System log pre-processing to improve failure prediction. In *IEEE/IFIP International Conference on Dependable Systems Networks, 2009. DSN '09.*, pages 572 –577, june 2009.

Appendix A

Experiments on prediction model 2

A.1 Parameter space and prediction results

For the two prediction models shown in Figure 5.7, both have the same parameter space, but use different settings. Similar experiments to those described for prediction model 1 in Section 6.4.2 have been performed on prediction model 2; the detailed configurations are listed in Table A.1. In order to assess prediction quality, 100 repetitions of the experiments have been performed according to the default configurations. Results are shown in Table A.2.

Parameters	Model 2
Sequence label number	40
Event category number	62
Maximum optimization tries	100
Underlying graphical model	semi-Markov CRF

Table A.1: Configurations of the prediction model 2.

It can be seen that the prediction accuracy of model 2 at 41.38% is much lower than that for prediction model 1. There are two reasons behind this. Firstly, failures are such rare events in the RAS logs that they take up less than 10% of the entire events based on the statistics derived from the raw event logs mentioned in Chapter 5. Prediction model 2 is trained using only failure sequences and the small data volume naturally results in lower prediction accuracy. Secondly, as described above, 40 failure patterns (categories) have been identified and they are not normally distributed. For some failure patterns, the prediction precision will reach 90%; in contrast, in terms of rare patterns, the accuracy may be as low as 20%. Thus the precision of model 2, which

		Prediction model 2
Parameters	Data window Δt_d	5 min
	Lead-time Δt_l	5 min
	No. of states	40
Prediction	Precision	41.38%
	Recall	44.08%
	$F_{0.5}$	40.02%
Variance	Precision	2.16%
	Recall	1.82%
	$F_{0.5}$	1.56%

Table A.2: Prediction accuracy and its variance for prediction model 2 according to the default parameter settings.

is calculated as the mean precision of the overall 40 failure patterns, is approximately 40%.

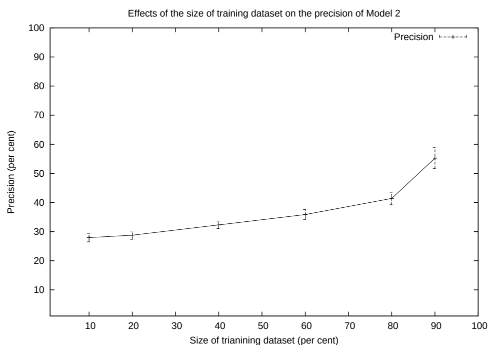
With consideration of result stability, the variance of precision reaches 2.16%, which means the prediction results have greater variance and the prediction quality of the model depends heavily on the target dataset. This situation explains the low precision of model 2 from another perspective.

A.2 Dependency analysis

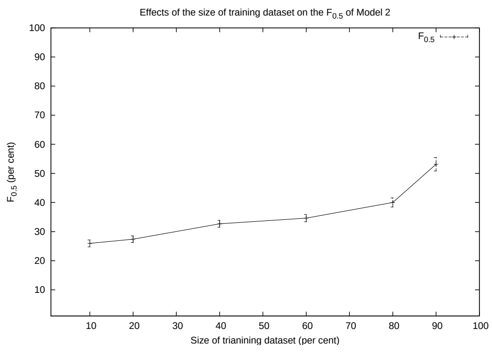
Similarly to the analysis of prediction model 1 in Section A.2, dependency analysis of the prediction model 2 has been performed according to three perspectives: *data specific issues*, Δt_d and Δt_l.

Dependency on data specific issues

To investigate the effects of data specific issues, an experiment has been conducted that uses training data sizes of 10%, 20%, 40%, 60%, 70%, 90% of the entire data; correspondingly, the remaining 90%, 80%, 60%, 40%, 30%, 10% of the dataset is used for testing. Figure A.1 presents the effects on precision and $F_{0.5}$ of model 2. It can be concluded that, for both precision and $F_{0.5}$, the accuracy rises sharply to 55.24% and 53.14% respectively when the size of the training dataset reaches 90%, however, they have great variability because of the limited size of the testing dataset.



(a) Data effect on precision

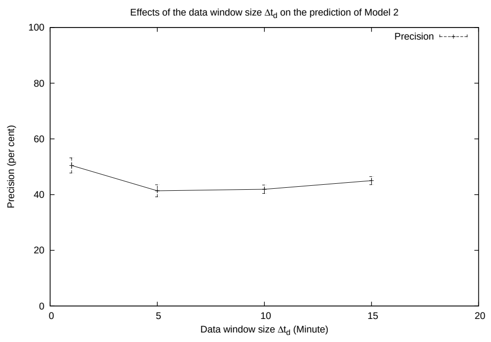


(b) Data effect on $F_{0.5}$

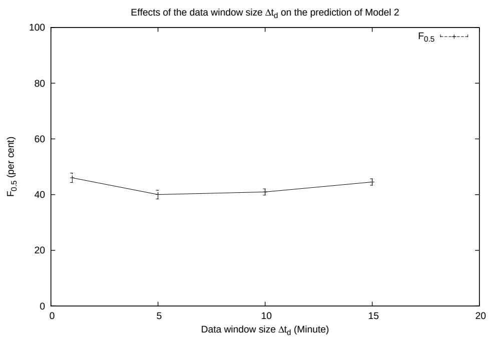
Figure A.1: Effects of the size of the training dataset on the prediction of model 2. Figure A.1(a) plots the effects of training dataset on the precision of model 2 with error bars, whilst Figure A.1(b) presents the effects on the $F_{0.5}$.

Dependency on data window Δt_d

Figure A.2 presents the effects of Δt_d. For the experiment, values of 1 minute, 5 minutes, 10 minutes and 15 minutes for Δt_d are considered. For the effects on precision and $F_{0.5}$, shown in Figure A.2(a) and Figure A.2(b), respectively, the charts show that accuracy first decreases when Δt_d changes from 1 minute to 5 minutes, then increases as a long-term trend.



(a) Effect on precision

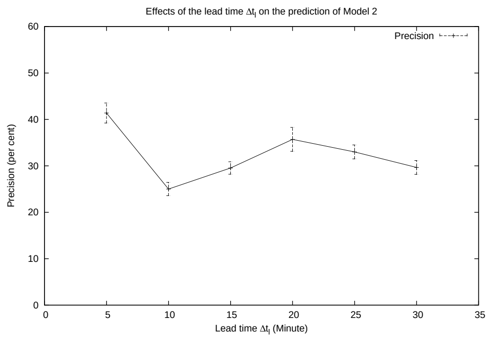


(b) Effect on $F_{0.5}$

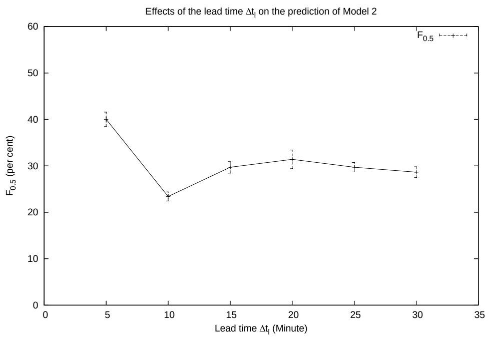
Figure A.2: Effects of the length of data window Δt_d on the prediction of model 2. Figure A.2(a) demonstrates the changes of the precision over three values of Δt_d, whilst, Figure A.2(b) presents the effects of Δt_d on the $F_{0.5}$.

Dependency on lead time Δt_l

Figure A.3 presents the effects of the length of Δt_l in terms of precision and $F_{0.5}$, shown in Figure A.3(a) and Figure A.3(b), respectively. Precisely, experiments with lengths of 5 minutes, 10 minutes, 15 minutes, 20 minutes, 25 minutes and 30 minutes on Δt_l have been performed. It can be concluded from the charts that both precision and $F_{0.5}$ fluctuate according to Δt_l. Similarly, they decrease with Δt_l ranging from 5 minutes to 10 minutes, then increase in the middle and later decrease again as Δt_l ranges from 20 minutes to 30 minutes. They both show a decline as a long-term trend.



(a) Effect on precision



(b) Effect on $F_{0.5}$

Figure A.3: Effects of the length of lead time Δt_l on the prediction of model 2. Figure A.3(a) presents the relationship between precision and Δt_l, whilst, Figure A.3(b) shows the effects on $F_{0.5}$.