View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Queen Mary Research Online

Flexible distributed computing with volunteered resources

Zhang, Jun

The copyright of this thesis rests with the author and no quotation from it or information derived from it may be published without the prior written consent of the author

For additional information about this publication click this link.
https://qmro.qmul.ac.uk/jspui/handle/123456789/358

Information about this research object was correct at the time of download; we occasionally make corrections to records, please therefore check the published record when citing. For more information contact scholarlycommunications@qmul.ac.uk

Flexible Distributed Computing with Volunteered Resources

By

Jun Zhang

SUBMITTED FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Supervised by Dr. Chris Phillips

School of Electronic Engineering and Computer Science

Queen Mary, University of London

June 2010

Acknowledgement

I would like to express my most sincere appreciation to my supervisor Dr. Chris Phillips for his expert supervision, massive support and continuous encouragement throughout my PhD study.

Many thanks to Dr. John Schormans, Dr. John Bigham, Dr. Raul Mondragon, Prof Laurie Cuthbert, Dr. Na Yao, Dr. Yue Chen and many others for their valuable suggestions over this period.

In addition, I would like to thank the support staff at Queen Mary, including Melissa Yeo, Kok Ho Huen, Rhainnon Thomspon, Peter Smith, Sharing Cording, Phil Willson, Theresa Willis and many others for their support and help.

My PhD has been partly supported by British Telecom, so I would like to express my thanks to the staff in the Networks Research Centre of British Telecom, including Gabriele Corliano, Peter Hovell, Ben Strulo, Edmund Kirkham, Laurence Forgiel, Charalambos Oxinos, Marc Wenink, Phillip Eardley, Arnauld Jacquet, Bob Briscoe, Linda Saddick and many others for their friendliness and help.

Thanks to all the friends in the UK and China.

Finally, my love and gratitude go to my family, especially to my parents. They are the most important people in my life. Without them, this thesis would have not been possible.

Abstract

Nowadays, computational grids have evolved to a stage where they can comprise many volunteered resources owned by different individual users and/or institutions, such as desktop grids and volunteered computing grids. This brings benefits for large-scale computing, as more resources are available to exploit. On the other hand, the inherent characteristics of the volunteered resources bring some challenges for efficiently exploiting them. For example, jobs may not be able to be executed by some resources, as the computing resources can be heterogeneous. Furthermore, the resources can be volatile as the resource owners usually have the right to decide when and how to donate the idle Central Processing Unit (CPU) cycles of their computers.

Therefore, in order to utilise volunteered resources efficiently, this research investigated solutions from different aspects. Firstly, this research proposes a new computational Grid architecture based on Java and Java application migration technologies to provide fundamental support for coping with these challenges. This proposed architecture supports heterogeneous resources, ensuring local activities are not affected by Grid jobs and enabling resources to carry out live and automatic Java application migration.

Secondly, this research work proposes some job-scheduling and migration algorithms based on resource availability prediction and/or artificial intelligence techniques. To examine the proposed algorithms, this work includes a series of experiments in both synthetic and practical scenarios and compares the performance of the proposed algorithms with existing ones across a variety of scenarios. According to the critical assessment, each algorithm has its own distinct advantages and performs well when certain conditions are met.

In addition, this research analyses the characteristics of resources in terms of the availability pattern of practical volunteer-based grids. The analysis shows that each environment has its own characteristics and each volunteered resource's availability tends to possess weak correlations across different days and times-of-day.

Table of Contents

ACKNOWLEDGEMENT **2**
ABSTRACT **3**
TABLE OF CONTENTS **4**
LIST OF FIGURE **7**
LIST OF TABLE **9**
GLOSSARY **10**
CHAPTER 1 INTRODUCTION **11**
1.1 Motivation 11
1.2 Objectives 12
1.3 Contributions and Publications 13
1.4 Thesis Organisation 14
CHAPTER 2 BACKGROUND **16**
2.1 Grid Computing 16
2.1.1 Overview *16*
2.1.2 Components *18*
2.1.3 Main Procedures *20*
2.1.4 Grid Computing with Volunteered Resources *21*
2.1.5 Applications *23*
2.1.6 Projects *23*
2.2 Challenges in Volunteered Resources based Grid Computing 28
CHAPTER 3 PROPOSED SYSTEM ARCHITECTURE **30**
3.1 Java Technology 30
3.1.1 Technology Overview *30*
3.1.2 Features of Java Program *31*
3.1.3 Java Platform *32*
3.2 Overview of the Proposed System Architecture 34
3.3 System Components 35
3.3.1 User/Resource Management Level Components *35*
3.3.2 Resource Level Components *36*
3.3.3 Java Application Migration Technologies *36*
3.4 System Operations Procedures 40
3.4.1 Job Execution Monitor *40*
3.4.2 Job Migration *41*
3.5 System Messages 41
CHAPTER 4 JOB-SCHEDULING AND JOB MIGRATION **42**
4.1 Job-Scheduling Introduction 42
4.1.1 Job-Scheduling Overview *42*
4.1.2 Job-Scheduling Components and Procedures *42*
4.1.3 Taxonomy of Job-Scheduling Algorithms *44*
4.1.4 Open Issues in Job-Scheduling Algorithms *46*
4.2 Proposed Job-Scheduling Algorithms 50
4.2.1 Adopted Prediction Technique *50*
4.2.2 Resource Availability *53*
4.2.3 FCFS plus Predictor (FCFSPP) Algorithm *54*

4.2.4 Fuzzy Logic plus Predictor (FLP) Algorithm 57
4.2.5 Particle Swarm Optimisation plus Predictor (PSOPP) Algorithm 62
4.3 Job Migration 66
4.3.1 Reactive Job Migration 67
4.3.2 Proactive Job Migration 68
CHAPTER 5 ANALYSIS OF PROPOSED ALGORITHMS 75
5.1 Analysis of the Adopted TDE Prediction Method 75
5.2 Analysis of the FCFSPP Algorithm 76
5.2.1 Features of the FCFSPP Algorithm 76
5.2.2 Influences on the FCFSPP Algorithm 78
5.3 Explanation of the FLP Algorithm 89
5.3.1 Features of the FLP Algorithm 89
5.3.2 Influences on the FLP Algorithm 91
5.4 Analysis of the PSOPP Algorithm 92
5.4.1 Features of the PSOPP algorithm 92
5.4.2 Influence of Workload 93
5.4.3 Influence of Fitness Function 93
5.4.4 Influence of Resource Reliability 93
5.4.5 Influences of the PSO Algorithm 94
5.5 Analysis of the PSPP Migration Algorithm 95
5.5.1 Features of the PSPP Algorithm 95
5.5.2 Influences on the PSPP Algorithm 95
5.6 Analysis of the CBR Migration Algorithm 98
5.6.1 Features of the CBR Migration Algorithm 98
5.6.2 Influence of CPU Availability Percentage 99
5.6.3 Influence of CPU Migration Threshold 99
5.6.4 Influence of Migration Prediction Interval 100
5.6.5 Influence of Adjustment Percentage 100
CHAPTER 6 CHARACTERISTICS OF REAL RESOURCES 101
6.1 Data Traces Overview 101
6.1.1 Analysed Data Sets 102
6.1.2 Data Trace Formats 103
6.1.3 Job Execution Availability Characterisation 104
6.1.4 Job Execution Availability Correlations 108
CHAPTER 7 SIMULATION AND EVALUATION 122
7.1 Simulation Environment 122
7.1.1 Components 122
7.1.2 General Evaluation Approach 126
7.2 Evaluation of FCFSPP Algorithms 130
7.2.1 Evaluation of the Number of Checking Days 130
7.2.2 Evaluation of Resource Availability Probability Threshold T 131
7.2.3 Evaluation of Different Weights on TDE Prediction 133
7.2.4 Influence of Similarity of Job Execution Availability between Days 135
7.3 Evaluation of FLP Algorithm 143
7.4 Evaluation of the PSOPP Algorithm 146
7.4.1 Influence of Workload 146

7.4.2 Influence of PSO Fitness Function 149
7.5 Evaluation of PSPP Migration Algorithm 152
7.5.1 Evaluation with Data Set UCB 154
7.5.2 Evaluation with Data Set SDSC 156
7.5.3 Evaluation with Data Set LRI 158
7.5.4 Evaluation with Data Set DEUG 160
7.5.5 Summary 161
7.6 Evaluation of CBR Migration Algorithm 162
CHAPTER 8 DISCUSSION, CONCLUSION AND FUTURE WORK 164
8.1 Discussion 164
8.2 Conclusion 167
8.3 Future Work 167
REFERENCES 170
APPENDIX A 175
I. Registration Message 175
II. Registration Acknowledgement Message 175
III. Resource Information Message 176
IV. Request Resource Information Message 177
V. Job Submission/Allocation Message 177
VI. Job Submission/Allocation Acknowledgement Message 178
VII. Job Submission/Allocation Completion Message 178
VIII. Job Submission/Allocation Completion Acknowledgement Message 178
IX. Job Information Message 179
X. Migration Notification Message 180
XI. Migration Notification Acknowledgement Message 180
XII. Migration Connection Request Message 180
XIII. Migration Connection Acknowledgement Message 181
XIV. Migration Completion Message 181
XV. Migration Completion Acknowledgement Message 182
XVI. Resource Unavailable Message 183
APPENDIX B 184
I. FCFSPP Algorithm with Synthetic Data 184
II. FLP Algorithm with Synthetic Data 193
III. PSPP Algorithm with Synthetic Data 197
IV. CBR Migration Algorithm with Synthetic Data 198

List of Figures

Figure 1-1: Challenges, Motivation and Contributions of this Research 13
Figure 2-1: A generic high architecture of a Grid system 18
Figure 3-1: The Java Programming Environment 31
Figure 3-2: A Basic Block Diagram of the Java Virtual Machine 33
Figure 3-3: Platform Specific Invocation 33
Figure 3-4: A platform-Independent Java Program 34
Figure 3-5: Resource Level of the Proposed Grid System Architecture 34
Figure 3-6: Classification of Different Migration (Adapted from [Illmann00]) 37
Figure 4-1: Global and local scheduling in a computational Grid 43
Figure 4-2: Task Scheduling Characteristics (Adapted from [Casavant88]) 44
Figure 4-3: Influence of Availability in Volunteered Resource-Based Systems 47
Figure 4-4: Availability States and Transitions (Adapted from [Rood08]) 51
Figure 4-5: FCFSPP Algorithm *Job Submission* Procedure 55
Figure 4-6: FCFSPP Algorithm *Job Allocation* Procedure 56
Figure 4-7: Traditional Boolean Logic in Computer System 57
Figure 4-8: Fuzzy Logic in Fuzzy World 58
Figure 4-9: Fuzzy Set A and B 58
Figure 4-10: Result of A U B, A ∩ B and ¬A 59
Figure 4-11: FLP Algorithm Membership Function 60
Figure 4-12: *Resource Availability Probability Threshold Adjustment* Procedure 62
Figure 4-13: PSOPP Algorithm Procedure 66
Figure 4-14: PSPP Algorithm Procedure 70
Figure 4-15: The CBR Cycle (Adapted from [Watson94]) 71
Figure 4-16: Main Procedure of CBR Migration Algorithm 74
Figure 4-17: Procedure of CPU Migration Threshold Adjustment 74
Figure 5-1: Example Resource *Job Execution Availability* Pattern 75
Figure 5-2: Extracted and Combined *Job Execution Availability* Pattern (Type 1) 78
Figure 5-3: Extracted and Combined *Job Execution Availability* Pattern (Type 2) 79
Figure 5-4: Examples of *Job Execution Availability* in 6 Cases 80
Figure 5-5: Transitions among all cases 81
Figure 5-6: Two Examples of "Exactly the Same" Checking and Prediction Period 83
Figure 5-7: Third Example of "Exactly the Same" Checking and Prediction Period 84
Figure 5-8: Checking Past 2 Days for Prediction 85
Figure 6-1: Number of Available Resources for Different Days (UCB) 105
Figure 6-2: Average Number of Available Resources over Time (UCB) 105
Figure 6-3: Number of Available Resources for Different Days (SDSC) 106
Figure 6-4: Average Number of Available Resources over Time (SDSC) 106
Figure 6-5: Number of Available Resources for Different Days (LRI) 107
Figure 6-6: Average Number of Available Resources over Time (LRI) 107
Figure 6-7: Number of Available Resources for Different Days (DEUG) 107
Figure 6-8: Average Number of Available Resources over Time (DEUG) 108
Figure 6-9: *Daily Series* PMCC Range of Results in UCB 111
Figure 6-10: *Daily Series* PMCC Distribution in UCB 111

Figure 6-11: *Hourly Sub-Series* PMCC Range of Results in UCB 112
Figure 6-12: *Hourly Sub-Series* PMCC Distribution in UCB 112
Figure 6-13: *Daily Series* PMCC Range of Results in SDSC 113
Figure 6-14: *Daily Series* PMCC Distribution in SDSC 113
Figure 6-15: *Hourly Sub-Series* PMCC Range of Results in SDSC 113
Figure 6-16: *Hourly Sub-Series* PMCC Distribution in SDSC 114
Figure 6-17: *Daily Series* PMCC Range of Results in LRI 114
Figure 6-18: *Daily Series* PMCC Distribution in LRI 115
Figure 6-19: *Hourly Sub-Series* PMCC Range of Results in LRI 115
Figure 6-20: *Hourly Sub-Series* PMCC Results Distribution in LRI 115
Figure 6-21: *Daily Series* PMCC Range of Results in DEUG 116
Figure 6-22: *Daily Series* PMCC Results Distribution in DEUG 116
Figure 6-23: *Hourly Sub-Series* PMCC Range of Results in DEUG 117
Figure 6-24: Hourly Sub-Series PMCC Results Distribution in DEUG 117
Figure 7-1: Structure of the Simulation Environment 122
Figure 7-2: *Number of Checking Days* in UCB and DEUG 130
Figure 7-3: *Resource Availability Probability Threshold* in UCB and DEUG 132
Figure 7-4: Fourth Day Simulation Performance with SDSC 135
Figure 7-5: Fourth Day Simulation Performance with LRI 135
Figure 7-6: *Average Allocated Jobs* in the Second Simulation Day 137
Figure 7-7: *Job Allocation Proportion* in the Second Simulation Day 138
Figure 7-8: *Average Succeeded Jobs* in the Second Simulation Day 139
Figure 7-9: *Job Success Proportion* in the Second Simulation Day 140
Figure 7-10: *Average Failed Jobs* in the Second Simulation Day 141
Figure 7-11: *Job Failure Proportion* in the Second Simulation Day 141
Figure 7-12: *Job Success Percentage* in the Second Simulation Day 142
Figure 7-13: *Total Allocated Jobs* with Margin of Error in the Second Simulation Day 142
Figure 7-14: *Total Succeeded Jobs* with Margin of Error in the Second Simulation Day 143
Figure 7-15: *Total Succeeded Jobs* with Margin of Error in the Second Simulation Day 143
Figure 7-16: *Total Allocated Jobs* in the Second Simulation Day 144
Figure 7-17: *Total Succeeded Jobs* in the Second Simulation Day 144
Figure 7-18: *Total Failed Jobs* in the Second Simulation Day 144
Figure 7-19: *Job Success Percentage* Comparison 145
Figure 7-20: *Total Allocated Jobs* in the Second Simulation Day 147
Figure 7-21: *Total Succeeded Jobs* in the Second Simulation Day 147
Figure 7-22: *Job Process Percentage* in the second simulation day 148
Figure 7-23: *Average Job Makespan* in the Second Simulation Day 149
Figure 7-24: *Total Succeeded Jobs* in the Second Simulation Day 150
Figure 7-25: *Job Process Percentage* in the Second Simulation Day 151
Figure 7-26: Effect of Different Values of *N* and *P* in UCB 155
Figure 7-27: Effect of Different Values of *N* and *P* in SDSC 157
Figure 7-28: Effect of Different Values of *N* and *P* in LRI 159
Figure 7-29: Effect of Different Values of *N* and *P* in DEUG 161
Figure 7-30: Comparison of PSPP and CBR Algorithms 163

List of Tables

Table 5-1: Prediction Accuracy....96
Table 6-1: UCB, SDSC, LRI and DEUG Data Sets....108
Table 6-2: *Non-zero Standard Deviation Daily Series* PMCC Results....118
Table 6-3: *Non-zero Standard Deviation Hourly Sub-Series* PMCC Results....118
Table 6-4: Rho Mean of Different *Day Intervals*....119
Table 6-5: *Zero Standard Deviation Available Daily Series* Results....120
Table 6-6: *Zero Standard Deviation Unavailable Daily Series* Results....120
Table 6-7: *Zero Standard Deviation Available Hourly Sub-Series* Results....121
Table 6-8: *Zero Standard Deviation Unavailable Hourly Sub-Series* Results....121
Table 7-1: Common Experimental Setup for Simulations....129
Table 7-2: Experimental Setup for Simulations of *N*....130
Table 7-3: Experimental Setup for Simulations of *T*....132
Table 7-4: Experimental Setup for Simulations of Different Weight Schemes....134
Table 7-5: Experimental Setup for Simulations of ρ....137
Table 7-6: Experimental Setup for Simulations of FLP....143
Table 7-7: Experimental Setup for Simulations of Different *Workload* in PSOPP....146
Table 7-8: Experimental Setup for Simulations of Fitness Function of PSOPP....150
Table 8-1: Comparison of Proposed Job-Scheduling Algorithms....165
Table 8-2: Comparison of Proposed Proactive Job Migration Algorithms....167

Glossary

AI	-	Artificial Intelligence
API	-	Application Programming Interface
BOINC	-	Berkeley Open Infrastructure for Network Computing
CBR	-	Case Based Reasoning
CMCL	-	Computers, Media, and Communication Laboratory
CPU	-	Central Processing Unit
DEUG	-	Diplôme d'Enseignement Universitaire Général. Desktop PCs in classrooms used by first-year undergraduates at the University of Paris South and ran the open source XtremWeb.
EDF	-	Earliest-Deadline-First
FCFS	-	First-Come-First-Served
FCFSPP	-	First-Come-First-Served plus Predictor
FL	-	Fuzzy Logic
FLP	-	Fuzzy Logic plus First-Come-First-Served plus Predictor
GHz	-	Giga Hertz
GRAM	-	Globus Resource Allocation Manager
HTC	-	High Throughput Computing
I/O	-	Input and Output
Java EE	-	Java Platform Enterprise Edition
Java ME	-	Java Platform Micro Edition
Java SE	-	Java Platform Standard Edition
JVM	-	Java Virtual Machine
JVMDI	-	JVM Debugger Interface
JVMTI	-	JVM Tool Interface
LRI	-	Laboratoire de Recherche en Informatique. A cluster used by a computer science research for running parallel applications and benchmarks at the University of Paris South and ran the open source XtremWeb
LSF	-	Load Sharing Facility
LTTR	-	Least-Time-To-Run-First
NP	-	Non-deterministically Polynomial
PBS	-	Portable Batch System
PDF	-	Probability Density Function
PMCC	-	Pearson Product-Moment Correlation Coefficient
RPC	-	Remote Procedure Call
PSO	-	Particle Swarm Optimisation
PSPP	-	Periodical Scanning plus Predictor
PSOPP	-	Particle Swarm Optimisation plus Predictor
PSC	-	Pittsburgh Supercomputing Centre
SDSC	-	San Diego Super Computing Centre
TCP	-	Transmission Control Protocol
TDE	-	Transitional N-Day with Equal transition weights
UCB	-	University of California, Berkeley

Chapter 1 Introduction

1.1 Motivation

Grid computing can be described as a type of distributed system consisting of a group of networked computers that is presented as one virtual computing resource. This virtual computing resource can be used to solve large-scale problems, such as computational and data storage problems.

The motivation for this research is based on observations at the system architecture level and user/resource management level of the Grid environment, including challenges in existing Grid computing implementations and potential benefits brought by some current technologies. At the system architecture level, there are two challenges with many existing Grid computing environments:

- Resources may not be able to run all kinds of jobs. This is because of the heterogeneity of the computer operating system. The operating system of the resources that are used to run the job can be different from the system that is used to create the jobs. For example, a job created by Windows system may have difficulty on running on a Linux machine.
- Resources may not support live and automatic (reactive and proactive) job migration, especially migrations between heterogeneous computers. Here, job migration is an approach to enable fault-tolerant environment, especially in a volatile (resources may appear and disappear at any time) environment. It is more beneficial if the job can be migrated live between heterogeneous systems. For example, if a job running on an unreliable resource can be migrated to a reliable resource, then the probability of job failures brought by the unreliable resource can be reduced.

Java technology [Sun10a][Arnold05] provides many useful features, e.g. simple, robust, secure, architecture neutral, portable, interpreted, threaded and so on [Gosling96]. Among all these features, architecture neutrality and portability are two key aspects that provide a solution to the two main challenges mentioned above. Architecture neutral and portable means that the Java programs are able to run on multiple kinds of operating systems, especially the mainstream operating systems such as Windows, Linux and Mac. Therefore, if a job is represented as a Java application(s), it should be able to run on multiple resources without worrying about heterogeneity between systems. In addition, architecture neutral and portable also enables job migration. If a job is represented as a Java application(s), it will be able to migrate between resources without worrying about the heterogeneity between systems. Though Java technology does not explicitly support live job migration, existing Java migration technologies can easily be leveraged to fulfil this requirement. Furthermore, other features of Java bring many benefits. For example, Java applications can be “simple”, allowing users to write a Java program with little difficulty, and “secure”, ensuring both jobs and the resources used to run the jobs will be safe.

At the user/resource management level, one big challenge for effective job allocation for

volunteered resources based Grid environments (such as desktop Grid computing or volunteer computing environment) is resource volatility. In such an environment, the Grid can comprise many heterogeneous computing resources owned by different individual users and institutions. As the owners of the resources usually have the right to decide when and how to donate idle CPU cycles of their computers, availability of these resources can be hard to predict. If a resource that becomes unavailable before processing a job, then this job will have to be suspended or even failed, requiring the job scheduler to reallocate the job to another resource for processing again from scratch. This is a waste of resource CPU cycles and it lengthens the job's *Makespan* (the time to "make" or complete a job).

One approach to solve the problem of job failures caused by resource volatility is to characterise what will happen to the computing resources and to make reasonable job-scheduling and migration decisions accordingly. To anticipate what will happen to the computing resources, some resource availability prediction methods have been proposed, such as [Rood07][Rood08] [Mickens06][Dinda99][Ren06a]. For example, in [Rood07][Rood08], the authors propose a method to predict resource availability based on a multi-state model and the resource's nearest past few days' availability information. This multi-state model represents the states of volunteered resources in Grids appropriately and the prediction method performs well if the resource owner's behaviour has displays pattern(s) across different days.

Taking this into account, in this thesis new job-scheduling and job migration algorithms are created to improve scheduling performance in volunteered resources based Grid environments, especially in terms of avoiding job failures caused by resource volatility.

1.2 Objectives

There are two main objectives in this research:

The first objective is to achieve an intelligent, ubiquitous, flexible, secure and distributed computing environment by proposing a new Grid architecture based on existing Java technology. The architecture is mainly focused on using volunteered resources so it should not only support multiple operating systems, but also support live and automatic Java application migration. In addition, it also ensures resources' local activities are not affected by the execution of Grid jobs.

The second objective is to provide new job-scheduling and migration algorithms, which ensuring *reliability* (ensuring jobs are processed successfully) with acceptable *speed* (i.e. getting jobs processed quickly) by proposing resource availability aware job-scheduling and job migration algorithms. According to [Dogan02], for a job-scheduling/migration algorithm, there is a trade-off between *speed* and *reliability* in a heterogeneous distributed computing system. Here, *speed* can be represented by the total number of jobs successfully completed by resources within a period of time and *reliability* can be represented by the percentage of jobs successfully completed by resources within a period of time. Therefore, ensuring *reliability* will be the primary goal of these algorithms.

1.3 Contributions and Publications

In general, this research mainly focuses the following aspects: development of volunteered resources based computing architecture, research into resource management in a desktop Grid computing environment, analysis of the resources' characteristics in real computing environments. Figure 1.1 summarises these contributions.

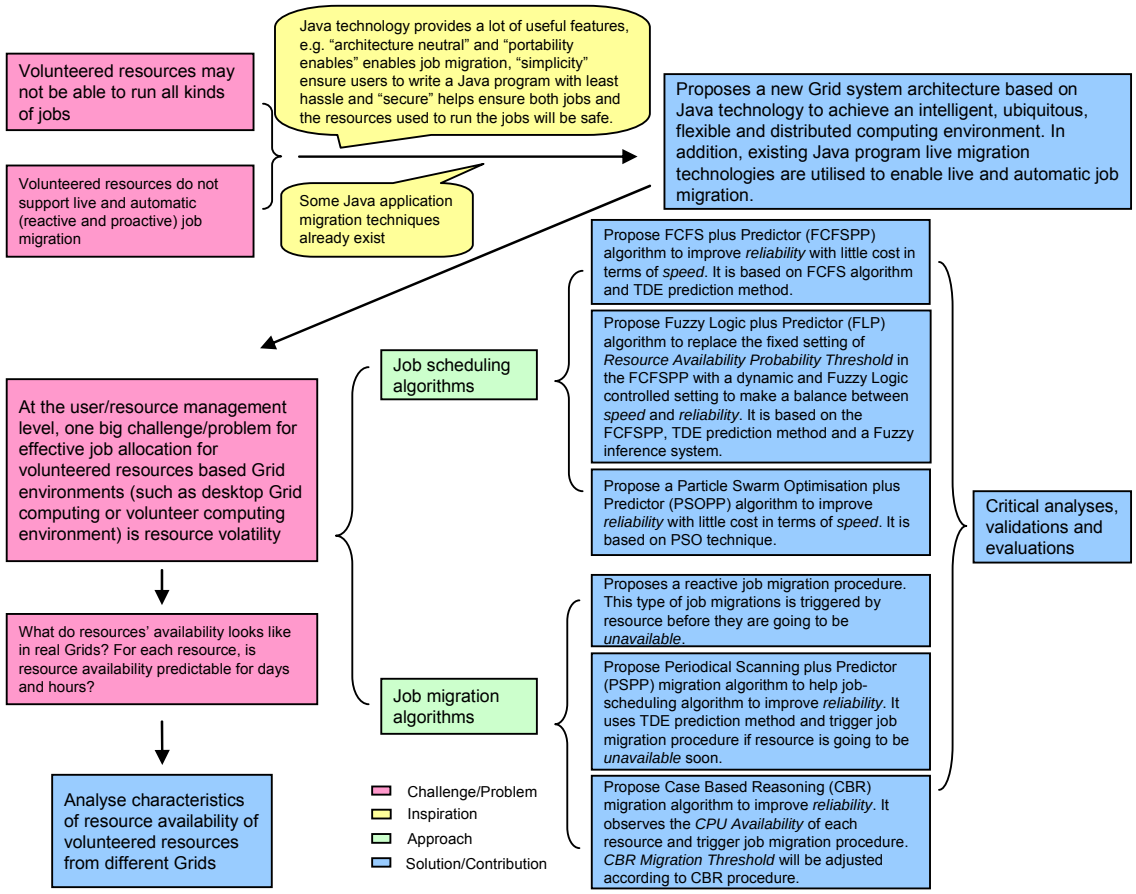


Figure 1-1: Challenges, Motivation and Contributions of this Research

The challenges and motivation are introduced in Section 1.1 and 1.2 already. Based on these observations, this research addresses different aspects to solve the problems, which include proposing a new system architecture supporting live and automatic job migration, proposing new job-scheduling/migration algorithms to ensure jobs process successfully and critical analyses, validations and evaluations of the proposed algorithms. Therefore, the major contributions of this research can be summarised as follows:

1. This research proposes a novel, ubiquitous, flexible and distributed Grid system architecture to utilise volunteered and heterogeneous resources and support live and automatic (reactive and proactive) job migration between them. This architecture also leverages features of Java technology to ensure a Grid can work well in a heterogeneous and volatile environment.
2. This research proposes novel job-scheduling algorithms to improve the chances of jobs being processed successfully with little cost in terms of *speed*. All the algorithms are based on a resource availability prediction and some of them employ Artificial Intelligence (AI)

techniques to achieve the objective via different approaches.

3. This research proposes novel proactive job migration algorithms to assist job-scheduling in order to ensure jobs are processed successfully. Again, these job migration algorithms try to achieve the objective via different approaches.
4. This research analyses the characteristics of resources in real volunteered computing environments. In this research work, several sets of real resource availability data collected from different institutions have been analysed, especially in terms of correlations of resource availability within each resource.
5. This research evaluates the job-scheduling and job migration algorithms. After proposing the algorithms, the research critically analysed them under different conditions.

Publications List:

- Jun Zhang, Chris Phillips, "Ubiquitous, Flexible and Distributed Computing", PGNet 2007
- Jun Zhang, Chris Phillips, "Intelligent Roaming for Nomadic Computing", Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on 7-11 April 2008 Page(s):1 - 6 Digital Object Identifier 10.1109/ICTTA.2008.4530179
- Jun Zhang, Chris Phillips, "Job-Scheduling with Resource Availability Prediction for Volunteer-Based Grid Computing", London Communications Symposium 2009

1.4 Thesis Organisation

The thesis is organised as follows:

Chapter 2 provides background knowledge to the thesis. Firstly, it introduces Grid computing, especially desktop Grids and volunteer computing Grids. Next, some open issues in the existing Grid computing environment are described.

Chapter 3 presents a novel Grid computing system architecture. Firstly, general background knowledge about Java application and migration technologies are described as Java technology plays a very important role in the new architecture. Next, all the components of the proposed system architecture are functionally described in detail. In addition, the system operation is introduced. Furthermore, messages used in the system architecture are briefly described; further details about the messages are provided in Appendix A.

Chapter 4 introduces job-scheduling and migration algorithms in the proposed system. Before describing the novel job-scheduling algorithms, the issues associated with volunteered resource-based Grid system are considered in detail along with prediction techniques.

Chapter 5 critically analyses the proposed job-scheduling and migration algorithms. In this chapter, some important features of each algorithm will be considered and the performance of the proposed job-scheduling algorithms in different scenarios are analysed.

Chapter 6 provides some information pertaining to four sets of real volunteered resource

based Grids and ascertains their characteristics.

Chapter 7 described the simulation setup and evaluates the results. Several sets of real resource availability data are adopted for the simulation experiments. The simulation environment is described and evaluation results of the proposed job-scheduling and job migration algorithms are presented and discussed for the different scenarios.

Chapter 8 provides some discussion of the salient results concludes the whole thesis. As many results are provided in Chapter 7, this chapter focuses on some important or interesting ones. In addition, future work is discussed.

Finally, appendices provide detailed information about the messages used in the proposed architecture and validation work of job-scheduling and job migration algorithms.

Chapter 2 Background

To better understand this thesis, this chapter will firstly give some background knowledge to the research.

2.1 Grid Computing

In this section, some knowledge about Grid computing will be viewed from some important aspects, including its system architecture, working procedures, potential applications, and existing Grid projects.

2.1.1 Overview

There are many definitions of Grid computing. Typically, Grid computing can be defined as a collection of networked computing resources used for solving common tasks. Here, a common task can be a computational job (a job mostly requires computational resources such as CPU processors), or a storage job (a job mostly requires storage resources such as hard disk and memory) or any other types of jobs. Different designs to the Grid system are required for coping with different types of tasks. In this thesis, the main focus is the computational Grid – the Grid that handles computational tasks. If it is not specified further, both Grid computing and computational Grid will have the same meaning in this thesis.

Grid computing is a type of distributed computing. Different from conventional cluster computing, these computing resources are usually heterogeneous and from multiple geographical sites, connecting by the Internet or a local area network. The computational tasks in the Grid are created by the users and allocated to resource(s) according to specific requirements by the Grid.

Grid computing was motivated by facts observed from some previous research work. According to many research [Jacob02][Smith][Mutka92][Acharya97], most computers are idle for over 60% (some research shows the result is over 90%) of the time. Therefore, many CPU cycles are idle. Meanwhile, many computers are easily accessible as computers typically connected to the Internet nowadays. Therefore, motivated by these easily accessible idle computing resources, the Grid computing technology has been proposed.

According to suggestions from the father of Grid computing [Foster02], a Grid system should possess the following features:

1. The system should “coordinate resources that are not subject to centralized control” [Foster02]. This means the resources in the Grid could be from different domains, e.g., different universities or different departments in a university. In addition, these resources could be controlled by different users, e.g., different staff or students in the university. Otherwise, if the resources are in the same domain, then it is more like a local management system.
2. The system should use “standard, open, general-purpose protocols and interfaces”

[Foster02]. This is because fundamental problems exist like authentication and authorization that need to be addressed irrespective of the technologies in use.

3. The system can "deliver nontrivial qualities of service" [Foster02]. This means the resources can be used to provide service with quality assured, like response time and throughput in terms of *speed* and success rate in terms of reliability. Therefore, a Grid – a combination of resources is much better than the sum of each single resource.

According to the descriptions in 1, 2 and 3 the computational Grid was motivated by under utilised resources and used for computational tasks. However, in addition to this, a computational Grid can provide useful functionality. So some capabilities of Grid computing can be summarized as follows:

1. Exploiting idle CPU cycles. This is one of the most important motivations for Grid computing and it is also one of the most important goals of Grid computing. Though there are many resources with idle CPU cycles in a private network or in the Internet, it may not be straightforward to utilise their idle CPU cycles as their CPU may not always idle. So if the Grid not only uses resources' idle CPU cycles but also tries to occupy resources' CPU when they are busy, then it will interfere with resources owners' activities. This is an important issue, especially for desktop Grids and volunteer computing. This will be described further in the following sections.
2. Balancing resources' load. Generally speaking, it is better if all the resources in a Grid have more or less the same load as this will ensure all resources are efficiently utilised and the jobs can be quickly processed. However, due to various reasons, like resources owner's behaviours, job-scheduling algorithm, some resources in the Grid may have extremely high loads while some other resources burden is relatively low. Therefore, it will be desirable to have mechanisms to balance resources' load. Fortunately, it is possible to achieve this goal in a Grid via multiple approaches, e.g., different job allocation strategies and live job migration. In terms of live job migration, this will be covered in more detail in the following chapters.
3. Providing extra reliability. Reliability is a part of quality of service and it is always a big issue in many fields. Reliability can be viewed from different angles, e.g., job completion reliability, data storage reliability and resource access reliability. Conventionally, extra reliability was mainly achieved by increasing the number of hardware equipments, e.g. buying more computers. However, this is an expensive solution and may be unnecessary in many situations. Grid provides another solution via a different approach, which is mainly based on software, rather than hardware. With Grids, extra reliability can be achieved by techniques such as job replication and job migration, which are, to some extent, more cost-effective than providing additional hardware.

2.1.2 Components

Typically, a Grid has three main levels: user level, resource/user management level and resource level. At each level, there are multiple components. Figure 2.1 shows a common high level architecture of a Grid system:

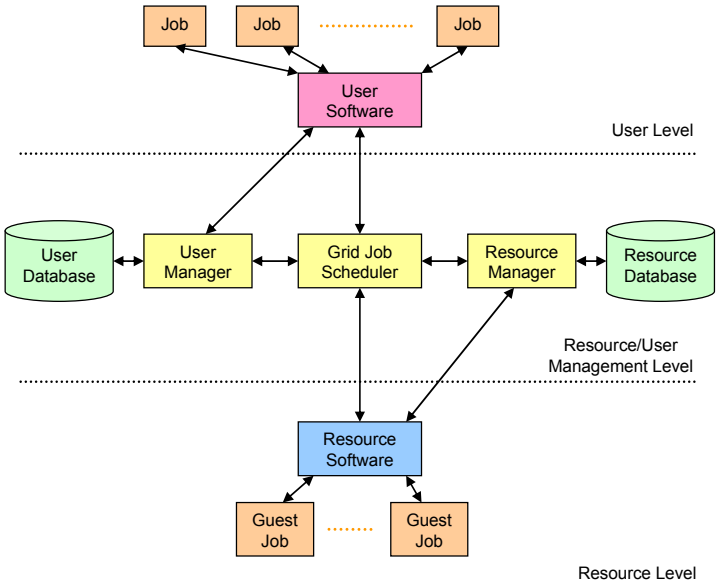


Figure 2-1: A generic high architecture of a Grid system

According to the Figure 2.1, there are multiple components at each level. Each component and its functionalities are described as follows:

- **User Level**

At the user level, there are two main components - job and the user software:

Job: A job is a task created by a user that will be submitted to the Grid and executed by the Grid resource(s) later. As the jobs have to be executed by the resource(s), the job(s) should have format(s) that can be understood by the resource(s). For example, a job could be a self-contained executable file that can be executed by a resource, or it could be a predefined specific format file that can be understood and executed by the resource(s). Different jobs may have different priorities.

User software: It is the software for communicating with the user manager and the Grid job scheduler. Typically, it has three main functions: the first one is to register the user manager at the user/resource management level. Secondly, it is responsible for submitting the created jobs to the Grid job scheduler at the user/resource management level. Thirdly, it is responsible for receiving results from the Grid job scheduler after the submitted jobs are completed by resource(s). So if a user wants to use the Grid to execute job(s), then user software will execute the two functions in sequence. Firstly, it has to register at the user manager to gain the authorisation to submit jobs. Then after getting authorization, it sends the job(s) to the Grid job scheduler. Later, when the job is complete, the user software will receive the result(s) from the Grid job scheduler.

- **User/Resource Management Level**

At the user/resource management level, there are five main components: Grid job scheduler, user manager, resource manager, user database and resource database.

Grid job scheduler: It is responsible for sorting and allocating coming jobs to different resources and redirecting job results to user software when the jobs are completed by resources. Firstly, as many jobs from different users will be sent to the Grid job scheduler, the Grid job scheduler needs to sort the jobs in preparation of job allocation. In the phase of sorting, the Grid job scheduler can use one or multiple job queues to sort all the jobs. As mentioned earlier, different jobs can have different priorities. Therefore, jobs with high priorities may be put in front of a queue or put in a specific queue exclusively used for high priority jobs. For each queue, the Grid job scheduler can sort the jobs according to predefined sorting algorithm, e.g., First-Come-First-Served (FCFS) [Esklcloglu01], Earliest-Deadline-First (EDF) [StanKovic98], backfilling[Mu'alem01] and Least-Time-To-Run-First (LTTR) [Lazarevic06]. In addition, the Grid job scheduler can sort the jobs with specific requirements. For example, jobs with high priorities will always be in front of jobs with low priorities.

In the phase of allocating, the Grid job scheduler also allocates the jobs to resources with a predefined allocation algorithm with/without specific requirements. For example, the Grid job scheduler can allocate jobs to the resource by using algorithm like FCFS [Esklcloglu01] and Matchmaker [Thain05]. The Grid job scheduler can also allocate jobs to the resource with some specific requirements so that high priority jobs are allocated to resources that have the shortest response time.

When a job is finished, the resource will return the result(s) to the Grid job scheduler and the Grid job scheduler will redirect the result(s) to the original user.

User manager: It is responsible for authenticating and authorising users. As mentioned above, the user software should communicate with the user manager to register at the user manager. After getting the registration request, the user manager will check its user database and/or predefined policies to make authentication decisions. If the user is authenticated, then the user manager authorises the user with predefined policies. For example, a regular user is allowed to use all the available resources while a restricted user can use limited number of resources.

Resource manager: It is responsible for registering resources. When a resource would like to join the Grid, then the resource will need to send out a registration request to the resource manager. After getting the registration request, the resource manager will check the resource database and/or predefined policy to make authentication decisions. If the resource is authenticated, then the resource manager will allow the resource to join the Grid.

User Database: It is a database used to store user related information, such as user ID, user authentication status, user authorisation status and so on. When the user manager gets the registration request from a user, the user manager will check the user database to make

authentication decisions and then update related records in the user database.

Resource Database: It is a database used to store resource related information, such as resource ID, resource authentication status, resource availability history and so on. When the resource manager gets the registration request from a resource, the resource manager will check resource database to make authentication decisions and then update related records in the resource database.

- **Resource Level**

At the resource level, there are two main components: resource software and guest jobs.

Resource software: It is a program running on the resource and it is responsible for communicating with the resource manager and the Grid job scheduler. Typically, it has up to four or five main functions: the first one is to register the resource at the resource manager at the user/resource management level. Second, it is responsible for receiving jobs allocated by the Grid job scheduler at the user/resource management level. Third, if the resource currently has more than one job, the resource software is responsible for scheduling all these jobs. Similar to the Grid job scheduler, the resource software can use one or multiple queues to sort all the jobs and then decide which job(s) to run next. Fourth, it is also responsible for monitoring the job execution state. Fifth, it is responsible for sending results to the Grid job scheduler after the submitted jobs are completed by resource.

Therefore, if a resource wants to join the Grid to provide job execution service, then the resource software will carry out these two functions in sequence. Firstly, it has to register at the resource manager to gain the authentication to join the Grid. Then after getting the authentication, it waits for the Grid job scheduler allocating jobs to itself. Once the Grid job scheduler decides to allocate a job to the resource, then the resource software will receive it.

After a job is completed, the resource software will return the results to the Grid job scheduler and then the Grid job scheduler will redirect the results to the users.

Guest job: A guest job is the job created by the users and received by the resource software. A guest job is the exactly the same as the component “job” at the user level. As the jobs have to be executed by the resource(s), the job(s) should have format(s) that can be understood by the resource(s). For example, a job could be a self-contained executable file which can be executed by a resource, or it could be a predefined specific format file that can be understood and executed by the resource(s).

2.1.3 Main Procedures

Though different Grid systems might have more or fewer elements for providing extra or less functionality, typically the key functions in a Grid are now considered. All these components work together to achieve the goals of the Grid. In terms of work procedures, generally there are three main procedures:

User registration procedure: user registration procedure is the procedure used to register

users at the user manager and the procedure can be outlined as follows:

1. A user sends out a registration request to the user manager by using the user software.
2. The user manager checks its user database and/or predefined policies to make authentication decisions.
3. If the user is not authenticated, then the user is not allowed to use the Grid. If the user is authenticated, then the user manager makes authorisation decisions to the user with predefined policies.
4. The user manager sends the authentication and authorisation decisions back to the user.

Resource registration procedure: resource registration procedure is the procedure used to register the resources at the resource manager and the procedure can be outlined as follows:

1. A resource sends out a registration request to the resource manager by using the resource software.
2. The resource manager checks its resource database and/or predefined policies to make authentication decisions.
3. If the resource is authenticated, then the user is allowed to join the Grid. Otherwise, the resource will be not allowed to join the Grid.
4. The resource manager sends the authentication decisions back to the resource.

Job execution procedure: job execution procedure is the procedure describing the whole job life cycle from creation to completion. The procedure can be outlined as follows:

1. A user creates and submits a job by using the resource software. The job will be submitted to the Grid job scheduler.
2. When the Grid job scheduler receives the job, the Grid job scheduler puts the job into a job queue and sorts it with a predefined job sorting algorithm.
3. If the job is in front of the job queue, the Grid job scheduler allocates the job to a resource by using a predefined job allocation algorithm.
4. When the resource receives the job via the resource software, it executes the jobs with the resource policies. After completing the job, the resource software returns the result(s) to Grid job scheduler.
5. When the Grid job scheduler receives the result(s), it checks the original user of the job from the user database and then returns the result(s) to the original user.

2.1.4 Grid Computing with Volunteered Resources

Conventionally, distributed computing usually utilises resources that are completed owned and controlled by the distributed computing system. Different from this, Grid computing usually does not have this constraint and a Grid may be composed of fully controlled resources and/or volunteered resources from different places. In Grid computing, there are two represented types of Grid systems designed to utilise volunteered resources on purpose:

The first type of Grid is desktop Grid computing. “Desktop Grid computing, exploiting

unused resources in the Intranet environments and across the Internet; it can provide considerable computational power, enabling the investigation of complex and demanding problems in a variety of different scientific fields." [DGRID03] It is a type of Grid computing. In a desktop Grid computing environment, the resources can be composed by a group of computers within an organisation and they can come from PCs all over the Internet.

The second type of Grid is volunteer computing. "Volunteer computing is a type of distributed computing in which computer owners donate their computing resources (such as processing power and storage) to one or more projects" [Volwiki10]. Volunteer is also a type of Grid computing. Therefore, these computers are also heterogeneous and geographically dispersed, connecting by the Internet or a local area network.

Some researchers point out that the most important differences between desktop Grid and volunteer computing are accountability and anonymity [BOINC10a]. In desktop Grid computing, a resource is assumed to behave mannerly (not creating fake or malicious results) and the resource's identity is known in advance. However, no matter whether it is a desktop Grid computing environment or a volunteer computing environment, the computing resources volunteer to join the grid and donate their idle CPU cycles to the Grid for finishing jobs of the project(s), e.g. donating idle CPU cycles for computational jobs or donating spare space for storing data. In this thesis, the focus will be mainly on contributing resources for computational tasks. Therefore, this kind of volunteer computing will discussed throughout this thesis.

Here, one of the most important common characteristics of both desktop Grid computing and the volunteer computing environment is that the resources donated to the Grid are volunteer based, which brings both benefits and challenges to the Grid.

In terms of benefits, volunteered can attract more resource owners to donate their idle CPU cycles of their computers as the owners of the resources are always welcome to join the Grid and they can decide when and how to donate their resources. In addition, their activities (e.g. processing local tasks, mouse and keyboard events) on the computers are usually ensured not be affected by the jobs from the Grid.

For some projects, they are especially attractive as the joiner can collect credits, awards and even money from the projects. For example, in the projects [SETI10][Climate10], joiners can get credit scores when they successfully finish jobs. Computers collecting highest scores will be listed on the websites of the projects. In [GIMPS10], the owner of the computer that finds a new prime number can get a monetary award. For example, the department of mathematics of University of California Los Angeles was awarded 50,000 US dollars as the 45^{th} known Mersenne prime was found on one of their computers.

In addition, "volunteer" also brings inexpensive and efficient Grids. As the owners donate resources freely, it is a cheap approach to gather many accessible computing resources. As the jobs only execute when a computer's CPU is idle, the computer resources can be efficiently utilised while the resource owners' activities will not be affected. On the other hand, "volunteer"

also brings some problems and challenges for the Grid. As the owners of the resources usually have the right to decide when and how to donate idle CPU cycles of their computers, these resources may be volatile (they may appear and disappear at any time).

The volatility of computing resources brings a big challenge for allocating jobs effectively. If a job is allocated to a resource that becomes unavailable before finishing the job is processed, then this job will be suspended or may even fail, requiring the job to be sent to another resource for processing again from the start. This is a waste of resource CPU cycles and it lengthens job's *Makespan* (the time to "make" or complete a job). Therefore, this makes the work of job allocation complex. This is one of the most important issues that this thesis focuses on. More discussion on this issue is provided in Section 4.1.4.

2.1.5 Applications

According to the descriptions above, Grid computing is used for solving common computational tasks. As it is composed of a collection of loosely coupled computers, it can be considered as a "virtual super computer". Therefore, it is not only useful for small and medium size computation tasks but is also suitable for large-scale computational tasks. So Grid computing can be applied to areas which require computing resources, and is especially suitable for the areas which need huge amount of computing resources.

"This technology has been applied to computationally intensive scientific, mathematical, and academic problems through volunteer computing, and it is used in commercial enterprises for such diverse applications as drug discovery, economic forecasting, seismic analysis, and back-office data processing in support of e-commerce and Web services." [Physorg10]. In Ian Foster and Carl Kesselman's book "The GRID: Blueprint for a New Computing Infrastructure (Second Edition)" [Foster09], many kinds of practical applications have been collected and described in detail.

According to the authors' description, Grid computing is applicable to a wide range of areas. "They cover compute-, data-, sensor-, knowledge-, and collaboration-intensive scenarios and address problems ranging from multiplayer video gaming, fault diagnosis in jet engines, and earthquake engineering to bioinformatics, biomedical imaging, and astrophysics" [Foster09]. For example, Grid computing can be used to gather telescope data from hundreds of telescopes, allowing astronomers to carry out analysis in a large scale. Grid computing can be also used by enterprise to improve efficiency and flexibility in terms of resource management. Grid computing can be also used to federate data for analysing and discovering new drugs. More details about practical applications of Grid computing are given by Foster [Foster09].

2.1.6 Projects

In the real world, extensive Grid projects have been designed and implemented in the past decade. In this section, some well known projects will be reviewed, especially their distinct features, advantages and disadvantages.

Condor [Thain05][Condor10a] is a project developed by the computer science department of University of Wisconsin-Madison. It is a project to implement and deploy High Throughput Computing (HTC) environments by utilising large collections of distributed computing resources. In practice, many researches rely on the number of computing results so scientists need a computing environment that can deliver a huge amount of computational capabilities over a long period of time. Therefore, the HTC environment was developed to fulfil this requirement. It is an environment that aims for high computing throughput. There are a couple of approaches to building such a HTC environment such as using mainframe computers, groups of personal computers. With the observations that many small, fast and inefficiently utilised personal computers are accessible in the Internet, Condor tries to achieve this goal by using distributed computing power all over the Internet and it aims to take " this wasted computation time and puts it to good use" [Condor10b].

In Condor, jobs are assumed as long running tasks that do not require user interactions. Initially, users create a job and submit the job to the system via a end-user software. Unlike many other designs in Grid computing, users in Condor environment can specify their requirement for the job by using an advertisement mechanism - ClassAd. For example, the user can specify CPU speed, memory and storage space required for a single job. On the other hand, the owners of the donated resources can also specify the requirements they can fulfil, i.e. the CPU speed, memory and storage that their resources can provide. The system streamlines all the jobs and the Grid job scheduler (called "Matchmaker" in Condor) makes allocation decisions by using a match mechanism. For a single job, if the system can find a resource that can fulfil its requirements, then the job will be allocated to the resource [Thain05][Condor10c].

In addition, Condor software on the resources provides some useful functions. Firstly, it checkpoints job(s) periodically, so that the job can resume from the checkpoint if the resource crashes. It also suspends job(s) when the resources are busy with processing local tasks. It also restarts jobs if the resource reboots. It also supports job migration, so that a job can resume on another resource from the checkpoint. However, it does not provide any mechanism to support proactive job migration.

Berkeley Open Infrastructure for Network Computing (BOINC) is an "open-source software for volunteer computing and grid computing" [BOINC10b] developed by The University of California. Any individual user or institution can contribute idle computer time to one or multiple projects on the platform at a time and a number of interesting public projects are running on this platform. For example, SETI@Home [SETI10] is one of the famous projects and it aims to detect intelligent life outside Earth. It uses radio telescopes to listen to narrow-bandwidth radio signal from space and the collected data is distributed and analysed by distributed resources. Climateprediction.net [Climate10] is another interesting project. It aims to make predictions of the Earth's climate up to 2080. In this project, each resource runs the model distributed by the project server and calculates a unique version of the climate change in the

future.

In BOINC, participant resources can claim credit when finishing a job and the procedure of finishing a job can be described as follows: Firstly, the client software on a resource requests one or multiple jobs from the scheduling server rather than waiting for the scheduling server to distribute jobs. Here, the jobs are preloaded to the data server by the project administrator. The scheduling server allocates job(s) to the resource according to the hardware of the resource. For example, the server will not give a job that requires more memory than the resource has. After that, the data server will send the executable and input files to the resource. Then the resource will start to execute the job(s) and produce output files. After completing the job, the resource will upload the results and request new jobs.

One of the most interesting features is the resources can get credit when completing jobs. To claim credit for each job, the correct result should upload to the server before the deadline. The claim credit is usually dependent on the CPU time and CPU benchmark of the resource. Task replication is used to guarantee if the result is correct. If both results match, then both resources will be given the minimum claimed credit. Otherwise, the job will be allocated to a new resource until the matching result is found.

Another interesting feature is its local scheduling policies. As mentioned above, a resource can download multiple jobs from different projects at a time. However, the number of CPU and the memory space is limited on a resource. In addition, there is a single deadline for each job. Therefore, how to schedule these jobs effectively is an important issue in such circumstances. In [Anderson07], the author proposes multiple scheduling policies - debt notion, deadline scheduling and job completion time estimation to “maximize CPU utilization and to avoid missed deadlines (and to balance these goals when they conflict)”.

In addition to contributing idle CPU cycles to the existing projects, BOINC also allows users to create their own projects (especially public projects) with the platform software, even making them public projects which allow the public to join and contribute their computing resources. For a university, BOINC can be used to create a “Virtual Campus Supercomputing Centre” [VCSC10], which can provide a clustered computing power to the researchers. For a company, BOINC can be used for desktop Grid computing in dealing with long running computational tasks. For a scientist, BOINC can be used to create volunteer computing projects, like the SETI@Home project and Climateprediction.net mentioned above.

Xgrid is a “technology in Mac OS X Server and Mac OS X, simplifies deployment and management of computational grids” [Xgrid10a]. It is “a proprietary software program and distributed computing protocol developed by the Advanced Computation Group subdivision of Apple Inc that allows networked computers to contribute to a single task” [Xgrid10b]. With this technology, researchers and scientists can build up their own computational grids for high throughput computing. The same as other Grid projects, Xgrid also tries to utilise idle CPU time on the networked computers. One of the most distinct features of Xgrid is that it only supports

the Mac operating system, which blocks users of other types of operating systems from joining or implementing a grid above Xgrid.

The work procedure of Xgrid can be summarised as follows: firstly, a user (it is called "Client" in Xgrid) creates and submits a job to the Grid job scheduler (it is called "Controller" in Xgrid). Then the Grid job scheduler divides the job into small tasks and allocates them to multiple resources (it is called "Agent" in Xgrid). Here, each resource CPU executes at most one task at a time and a multiple-CPU machine can execute multiple tasks at a time. Later, the resources return the results to the Grid job scheduler when they finish the tasks. Then the Grid job scheduler compiles individual task results into job results and sends the results to the original user. During the task execution, the Grid job scheduler will monitor the task status. If a task fails to complete on a resource (e.g. the resource crashes), then the Grid job scheduler will reassign the task to another resource.

XtremWeb is an open source platform for desktop grids designed by IN2P3 (CNRS), INRIA and Universisty Paris XI [Neri00]. It is designed to help for building users' own Grid based on PCs within an institution or over the Internet. Like other Grid computing environment, XtremWeb utilises resources' idle CPU cycles to accomplish computational tasks.

The work procedure of XtremWeb can be summarized as follows: Firstly, the users (it is called "Client" in XtremWeb) submit jobs to the Grid job scheduler (called "Coordinator" in XtremWeb). Then, similarly to BOINC, the resources (it is called "Worker" in XtremWeb) sends requests to the Grid job scheduler and the Grid job scheduler will send the jobs (the job may have already been stored in the resources. If so, the Grid job scheduler will send a set of parameters to the resources. Otherwise, the Grid job scheduler will send the whole job to the resources). Later, the resources complete the job and reply with the results to the users via the Grid job scheduler.

One of the most distinct features of XtremWeb is that it "can be used to build centralized Peer-to-Peer Systems such as some well known projects related to audio file exchange" [Xtremweb08a]. Therefore, a computer is considered as a resource and a user at the same time.

Comparing with other systems, "XtremWeb is somewhere in between pure Desktop Grid system, a la Condor and pure Volunteer Computing system, a la BOINC." [Xtremweb08b]. In Condor, the Grid job scheduler (Matchmaker) finds the best resource and allocates the job to it. Different from Condor, in XtremWeb, the resource request jobs rather than waiting for jobs to come. This is similar to the mechanism being used in BOINC. Compare with BOINC, XtremWeb allows each user has the right to submit jobs if it is authorised, while only project creator can upload jobs in BOINC. In addition, unlike BOINC, XtremWeb does not provide a credit system to award the resources.

Globus® Toolkit [Globus10a] is open source software toolkit for building Grids, "letting people share computing power, databases, and other tools securely online across corporate, institutional, and geographic boundaries without sacrificing local autonomy" [Foster06]. It is

being developed by the Globus Alliance, an international association that is mainly based at Argonne National Laboratory [Globus10b], USA. The toolkit has a series of software services and libraries to provide functionalities covering different aspects in Grid computing, from resource discovery, monitoring to protecting resources' safety.

"Its core services, Interfaces, and protocols allow users to access remote resources as if these resources were located within the users' own machine room while simultaneously preserving local control over who can use resources and when" [Garritano03].

The work procedure in Globus can be summarised as follows: Firstly, a user obtains authentication from the system. After that, the user queries the system to see if there is any resource available. If yes, then the user submits the job to the Grid. Then the Grid sends it to the resource for execution. During the job execution, the Grid monitors the progress of the job and notify the user if the job is finished, failed, or being delayed.

In Globus, the Globus Resource Allocation Manager (GRAM) [Feller07] is a software component used to manage jobs and resources. Interestingly, "GRAM is not a scheduler itself -- but a standardized, front end interface to different existing scheduler components, such as PBS (Portable Batch System) [Corbatto00] and Platform's LSF (Load Sharing Facility) [UNC09]" [Czajkowski10]. Therefore, Globus leaves the choice of Grid job scheduler to the users of Globus Toolkit. In addition to this, Globus Toolkit also supports a number of third-party software. For example, Condor can build upon Globus Toolkit and Condor-G [Frey01] can be utilised as a job management mechanism to manage job submission.

According to the above descriptions about different projects, all these representative projects have some common features:

- All the Grid systems developed in the projects have three level components: user level components, user/resource management level components and resource level components.
- User level components are mainly responsible for submitting jobs to the user/resource management level components for execution.
- User/resource level management components are mainly responsible for managing users/resources and scheduling jobs to different resources.
- Resource level components are mainly responsible for executing jobs and returning results back to the user/resource management level components.

However, they have many differences if looking into details:

- The definition and the range of users are different. In BOINC, the users are a limited number of system administrators or project managers that can submit jobs to the Grid system. In other projects, they generally do not have this kind of restriction, which means public users can also submit their jobs to the Grid system.
- Job allocation mechanisms are different. In BOINC and XtremWeb, the resources will try to fetch jobs from the Grid job scheduler while the Grid job scheduler will try to allocate jobs

to resources.

- Job-scheduling algorithms are different. FCFS algorithm is widely used, such as BOINC, XGrid and XtremWeb. In Globus, job-scheduling algorithm is not specified and the developers can decide what job-scheduling algorithm to be used when building a real Grid system. In Condor, it is based on FCFS and a matchmaker to scheduling jobs.
- Operating systems supported by the Grid are different. XGrid only support Mac OS system, which means only the computers with Mac OS can become the resources of the Grid. For other projects, though multiple operating systems are supported by the Grid in theory, operating system specific resource software has to be run on a different operating system to make sure operating system specific jobs can be executed on different platform. In this thesis, Java technology is utilised to support multiple operating systems and resource software or jobs will be platform independent (this will be discussed in more details in following chapters).
- Assumptions about resource availability are different. Some Grid systems assume the resources are available once they are in a Grid, such as Xgrid. However, for some Grid systems, resource owners can decide when and how to donate their resources to the Grid. For example, in Condor, a resource is considered as available to the Grid when the CPU load is lower than 25%. In BOINC, resource owners can define when and how to donate the CPU and memory resources with different parameters. In XtremWeb, it has a strict assumption in terms of availability: a resource is fully controlled by the resource owner and it is not available if some local activities occurs (e.g. mouse moved, keyboard touch or local process launched). In this thesis, Resource availability will have the same definition as XtremWeb and try to explore how job-scheduling algorithms can perform under such difficult circumstances (more details will be discussed in following chapters).
- The approaches for tackling the problem of resource volatility are different. Resource volatility means the resources may come and go at any time so that jobs may be lost if the resources leave the Grid (more details about this problem will be discussed in Section 2.2). Here, different approaches are taken by different projects. For some projects, it simply allocates the job to a new resource for execution if a job fails, such as XGrid and BOINC. For some projects, it checkpoints jobs regularly so that jobs can be executed from the checkpoint if it fails, such as Condor. However, none of these projects provides live and automatic job migration algorithm so that jobs can be reactively or proactively migrated before jobs get lost. This is another focus of this thesis. This thesis uses the approach of reactive and proactive job migration algorithms to avoid jobs getting lost in a volatile Grid environment (more details will be discussed following chapters).

2.2 Challenges in Volunteered Resources based Grid Computing

Resource management is the core part of a Grid system. “At the heart of the Grid is the

ability to discover, allocate, and negotiate the use of network-accessible capabilities—be they computational services offered by a computer, application services offered by a piece of software, bandwidth delivered on a network, or storage space provided by a storage system" [Foster09]. In the Grid, resource management includes work like resource discovery, resource state monitoring, job-scheduling and job state monitoring, most of which are managed by Grid job scheduler at the user/resource management level and resource software at the resource level.

Among the functionalities of the resource management, job-scheduling is an important part. In a Grid, especially in a volunteer resources based Grid (such as a desktop Grid computing or a volunteer computing environment), a resource may not always be available to the Grid and the job(s) may not always be allowed to run on them all the time. This is due to some characteristics of these systems:

Firstly, resources in such systems are assumed to be controlled by the resource owners rather than the Grid system. This characteristic makes the resources volatile. In a volunteer resources based Grid, resources may join and leave at any time, even without any precaution. A resource joining the Grid is good for the Grid as more computing power is available and potentially more jobs can be processed within a period of time. However, a resource leaving the Grid is bad news for the Grid, as the jobs running on the resources will be lost. The case will be worse if the resource leaves without any precaution. With precaution, it is possible to migrate jobs beforehand. Without precaution, it is difficult for the job scheduler to know when to migrate jobs.

Secondly, resources in such systems are typically assumed not to interfere with local activities. Every time the owner reclaims the resource, the guest job(s) from the Grid will have to be suspended. This characteristic also makes the resources volatile. Furthermore, if the suspension exceeds the predefined time-out, the job will be terminated and lost.

Therefore, both characteristics makes the resources volatile and the volatility of computing resources brings a big challenge for the job-scheduling algorithm to a volunteer resources based Grid in terms of how to get higher job throughput and how to ensure jobs are processed successfully.

One potential solution to this challenge is enabling the job scheduler to allocate jobs effectively with the information about resources' future availability. This is one of the focuses of this research and this type of solution will be discussed in details in Chapter 4, 5 and 6.

Another potential solution to this challenge is enabling the job scheduler to anticipate all resources' future availability so that the system can trigger proactive job migrations when resources are about to become unavailable. This is another one of the focuses of this research and this type of solution will be discussed further in details in Chapter 4, 5 and 6 as well.

Chapter 3 Proposed System Architecture

In this chapter, a novel Grid computing system architecture will be presented. In the system architecture, Java plays an important role as it has been used at both the user and resource level. Therefore, to make it easier to understand the proposed system architecture, a brief introduction about Java technology is provided.

3.1 Java Technology

According to the discussions in Chapter 2, the heterogeneity between resources and the volatility of each resource are the two main challenges faced by a computational Grid computing environment that is composed of unreliable resources (e.g. a desktop Grid or a volunteer computing Grid system). Therefore, this research proposes a new Grid system architecture to provide a solution. In terms of resource heterogeneity, and in order to tackle this challenge the proposed system architecture uses Java technology. In terms of resource volatility, this proposed system architecture utilises existing Java application migration technology to enable live and automatic job migration between heterogeneous resources. To provide a better understanding of the proposed system, some general knowledge about Java will be introduced.

3.1.1 Technology Overview

Java technology is "both a programming language and a platform" [Sun10b]. It was originally developed by Sun Microsystems to provide a modern programming paradigm. One of the important motivations for developing this technology was based on the observed difficulties of running applications on different operating systems. This was caused by the incompatibility of different operating systems. Therefore, before Java, if an application developer wanted to create an application running on multiple operating systems, then the developer may have to cope with one operating system at a time.

To mitigate the problem brought about by incompatible operating systems and to accelerate the development process of applications, especially the applications working on a distributed environment, Sun Microsystems proposed Java technology, a new programming language and the platform to support this language.

As a whole, Java's architecture consists of four components:

- The Java programming language:
- The Java class file format.
- The Java Application Programming Interface (API).
- The Java Virtual Machine (JVM)

The Java programming Language and the Java class file format can be considered to be in the category of the Java programming language whilst The Java API and the JVM can be regarded as components of the Java platform. Though these four components are distinct, they are related. During the process of writing and running a Java program, all of them cooperate to get the Java

program running successfully. Figure 3.1 shows the relationships of these four components:

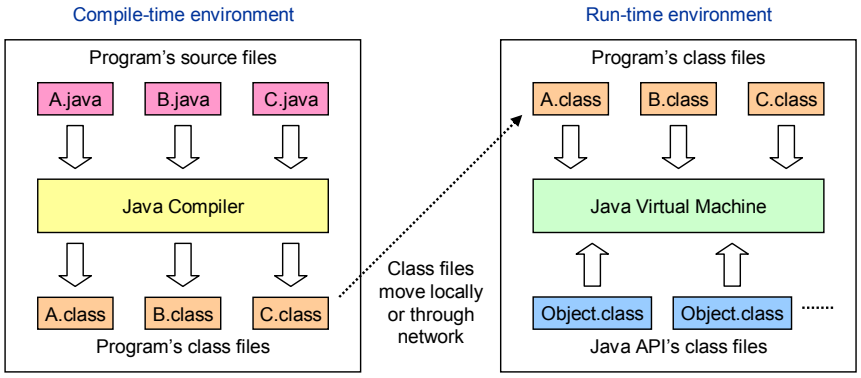


Figure 3-1: The Java Programming Environment

In order to run a Java program, firstly the source code should be written and saved in a .java format file. Here, the Java programming language is used. Then, a Java compiler should be used to compile the .java file to .class format file. This .class file is composed of the bytecodes which can be understood and executed by the JVM. Here, the Java class file format is used. Later, The JVM will be used to run the Java program. During the running period, the Java program may want to use system resources (such as I/O), the standard Java APIs can be used to access the required resources.

3.1.2 Features of Java Program

According to Java designers' idea, Java programming was designed to have a number of beneficial features like simple, object oriented, familiar, interpreted, multithread, dynamic, architecture neutral, portable, robust and secure. Here, these features will be explained briefly. More detailed information can be found on [Gosling96].

- Java was designed to be a simple language so that programmers can understand the fundamental concepts and be productive quickly without much training.
- The Java programming language is an object-oriented language. Before Java, the concept of object-oriented programming had been developed and become the mainstream. Java simply adopted this idea.
- The "look and feel" of Java programming language was designed to be similar to C++ [Stroustrup04], a mainstream programming language before Java. So the programmers would feel familiar with Java when they started to use Java.
- Java was designed as an interpreted language. Compiled and interpreted are two different approaches in the programming world. In compiled language, the source code is written in compiled language that is compiled into machine code (understandable and executable by the specific Resource) [Haas10]. On the other hand, in interpreted language, an interpreter is used to execute (interpret) the source code written in such a language [IBM08]. Java used the interpretation approach and this was based on the consideration of supporting heterogeneous operating systems. With the support of interpreter, Java program can be

quickly deployed on multiple platforms without worrying too much about the underlying operating systems.

- Java is a multithreaded language. Multithreaded programs allow several tasks to be carried out parallel. For example, a web browser application may be needed to download files while displaying contents on the screen. Therefore, to enable applications with the ability of dealing with concurrent activities, Java was designed to be a multithreaded language.
- Java is a dynamic language. At the linking stage, the Java class files are linked when they are needed. Therefore, new code can be loaded dynamically, which enables a more flexible approach to run applications, especially for network-based applications. For network-based applications, new code can be downloaded from somewhere else in the network and then start to run directly.
- Java is an architecture neutral programming language. In Java, the source file will be compiled to bytecodes, a format that will be understood and executed by the Java interpreter - JVM running upon different operating systems. Therefore, with the representation format and the help of JVM, Java programming language is independent from underlying operating systems.
- Java was designed to be portable. The feature of architecture neutral partially enables portability. In addition, Java also specifies that the size of its basic data types and arithmetic operators are the same on each platform, which eliminates the problem of data type incompatibilities on different platforms.
- Java is a robust programming language. It provides compile-time checking and run-time checking to ensure the source code is compiled correctly. Furthermore, it has a simple memory management model to eliminate potential memory related program errors.
- Java language is secure. As Java technology was designed to run in distributed and complex environments, security is of importance for applications and underlying operating systems. With Java bytecodes and the protection provided by JVM, Java applications will not be able to be invaded from outside. In addition, JVM prohibits malicious Java code from invading underlying operating systems.

3.1.3 Java Platform

Conventionally, most platforms consist of the operating system and underlying hardware. Unlike this convention, the Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms [Sun10c].

The Java platform has three editions: Java Platform Standard Edition (Java SE), Java Platform, Enterprise Edition (Java EE), and Java Platform Micro Edition (Java ME) [Sun10d]. Each edition has the JVM and the main difference between each edition is that the Java API they provide. Java SE provides the standard API whilst Java EE provides a superset of the standard API and Java ME provides a subset of the standard API. In this research, the main

focus will be on Java SE.

According to the description earlier in this section, the JVM and the Java API comprise the Java platform. The JVM is an abstract computer and its main task is to load and execute the .class files. Figure 3.2 shows the basic structure of the JVM.

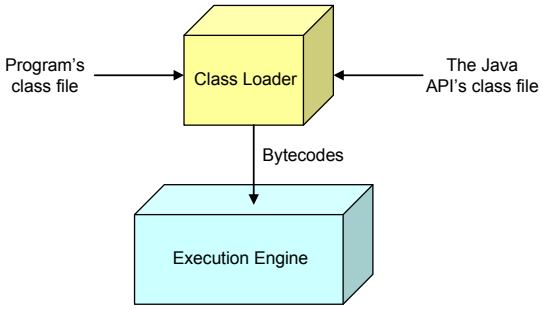


Figure 3-2: A Basic Block Diagram of the Java Virtual Machine

The JVM contains a class loader and an execution engine. The main job of the class loader is to load all the class files that are needed. The main job of the execution engine is to execute the bytecodes that are loaded by the class loader.

During executing the Java programs, some system resources of the computer may be required, such as reading a file from the hard disk and setting up a TCP connection with another computer.

There are two approaches to fulfil these requirements. The first approach is invocating native methods, which are written in some other languages directly, such as C or Pascal. These native methods are platform specific. As a result, using this approach leads the Java programs to be platform specific. Figure 3.3 shows this approach.

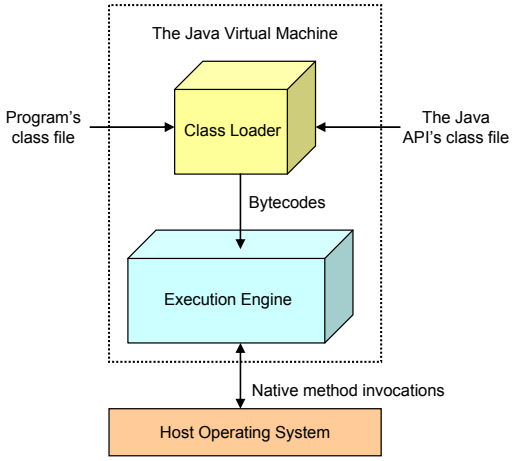


Figure 3-3: Platform Specific Invocation

The second approach is invocating Java methods that are written in Java programming language. These Java methods provide a standard way to access the system resources of the Resource computer and there are called Java API. Using this approach, Java programs will become platform independent. Figure 3.4 shows this approach.

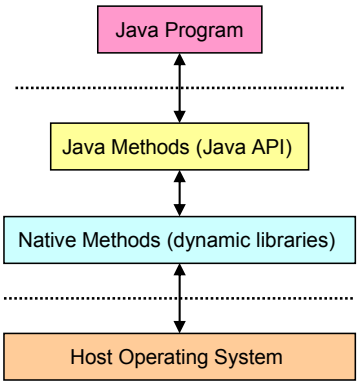


Figure 3-4: A platform-Independent Java Program

3.2 Overview of the Proposed System Architecture

This research proposes a new high-level Grid system architecture. As with the generic architecture described in Section 2.1.1, this new architecture has three levels and the same components. The novel parts of this architecture are the new features at the resource level. Figure 3.5 shows the resource level of the new system architecture.

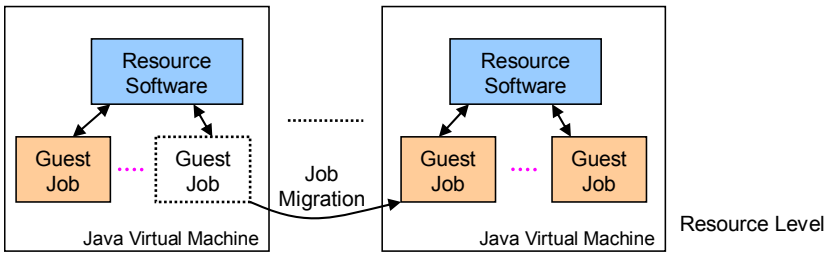


Figure 3-5: Resource Level of the Proposed Grid System Architecture

There are three important features in this proposed system architecture:

1. The proposed system architecture is based on Java technology [Sun10a] so that the resource software and guest jobs are running in JVM environment. Among Java technology's features, architecture neutrality and portability are the most important two:

   Firstly, these two features ensure the Grid system to be able to utilise heterogeneous computing resources. According to the descriptions in Chapter 2, a source file of a program will be compiled to a .class file and the .class file is composed of bytecodes (the machine language which can be understood and executed by JVM). As JVM is available on multiple platforms, so the .class file can be executed on different computing systems. Therefore, if a job in a Grid system is represented as a .class file, then it will be able to run on heterogeneous computing resources and the idle CPU cycles on these computing systems can be utilised.

   Secondly, these two features raise the possibility of job migrations between these heterogeneous operating systems. As the .class file is in a standardised format and understood by JVMs on multiple platforms, the .class file can be executed by the new resource if it is migrated from another resource. Therefore, if a job is represented as a .class file, then it can be migrated between heterogeneous resources without worrying about

whether they can be executed by the resources.

In addition, using Java technology can bring some extra benefits. For example, in terms of security, JVM can provide protection to the resources and guest jobs as guest jobs are running within a sandbox that is isolated from the underlying operating system. The activities of the guest jobs are restricted within the sandbox and any harmful activities to the resource will be blocked by the JVM. On the other hand, as the guest jobs are isolated from the resource, the resource cannot access the guest jobs either. As a result, both the resource and the guest jobs will be kept safe because of the JVM. In terms of programming, Java is an advanced programming language with features like simple, object-oriented and multi-thread, these features ensure Grid users can utilise Java to create powerful jobs easily.

2. The proposed system architecture enables resources to support live and automatic (reactive and proactive) job migration. This is based on the Java application migration technologies investigated by other research work (more details are provided in Section 3.3.4) and the Job migration algorithms proposed in this research (more details are provided in Section 4.3. With live and automatic job migration, potential job failures can be avoided.
3. The proposed system architecture is that it ensures resources' local activities are not affected by the guest jobs. If any local activity occurs, the Guest Job(s) will be terminated by the Resource Software.

3.3 System Components

As mentioned in Section 3.1, there are three levels in the proposed system architecture and there are multiple components at each level. Some system components have the same functionalities as the components' functionalities described in Section 2.1.2. This section will focus on the new functionalities provided by the new system architecture.

3.3.1 User/Resource Management Level Components

Grid job scheduler: It is the component used to deal with jobs. Specifically, it has the following functions:

- It is responsible for making job migration decisions. When the jobs are running on the resources, the Grid job scheduler monitors both the progress of the job and the state of the resource. In addition, the Grid job scheduler can also predict resources' future availability by using current/past information. Then with the monitoring information and migration decision algorithm(s), the Grid job scheduler makes job migration decisions.
- It is responsible for notifying related resources and monitoring the migration state of the job migration. After the Grid job scheduler makes a job migration decision, the Grid job scheduler should notify both the original and the destination resources to let them carry out the job migration process. During the migration process, both resources will send updated information to the Grid job scheduler and the Grid job scheduler will change the resource and job states in the resource and job databases accordingly.

3.3.2 Resource Level Components

Resource software: In addition to the functionalities described in Section 2.1.2, the resource software in this proposed system architecture has the following function:

- It is responsible for carrying out job migrations and reporting the migration progress to the resource manager and Grid job scheduler. If the resource software on the original resource receives the job migration notification from Grid job scheduler or resource owner, it will communicate with the destination and then carry out the job migration procedure.

Guest job: A job is composed of a .class file(s) with/without additional files used by the .class files. The additional files can be files used for input or output. For example, it can be an input file describing settings of parameters for running the job and it can also be an output file recording job results.

There are multiple approaches to create a guest job. The most common approach is that original user creates a Java source file(s) and then gets the source file(s) compiled. This approach requires the user has knowledge about Java and implement the jobs in Java programming language.

Another approach is creating source file(s) in other programming language and then converting the source file(s) into a Java source file(s) or converting into .class file(s) directly. For example, C2J converter [C2J01] is software to translate C-code source files into Java .class files. BEELUCID [Beelucid09] is software to convert VB.Net source files into Java source files. With this approach, users can use their preferred programming language to create source file(s) and then convert it to Java format. In addition, this approach enables the reuse of legacy applications that were written in other programming languages.

3.3.3 Java Application Migration Technologies

In the state of art, to enable distributed computing and fault tolerance, Java process / thread migration technology has been introduced in recent years. In [Illmann00], they examine the migration problem of migrating Java applications and classify different types of migration.

At the top level, Java applications can be divided into two kinds: those supporting strong or weak migration. Strong migration is “a migration technique which realizes code migration and strong state migration” [Illmann00]. This means not only the source code of the application but also the execution state and data information such as the program counter and Java stack should be migrated as well. Any kind of migration that cannot achieve both code and strong state transfer together is considered weak migration.

Strong migration can be classified into different sub-types and Figure 3.6 shows the classification under strong migration:

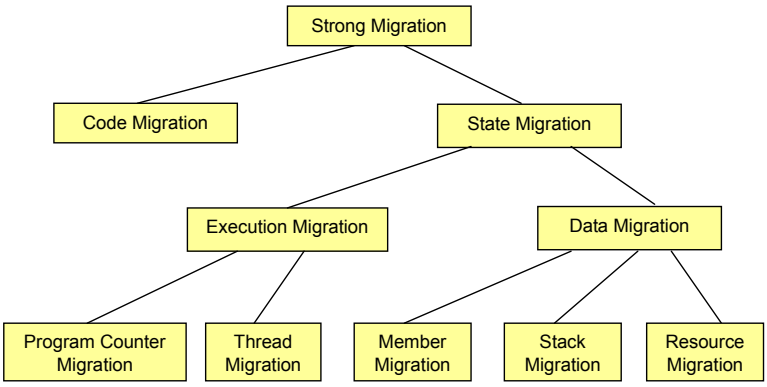


Figure 3-6: Classification of Different Migration (Adapted from [Illmann00])

According to the classification, a strong migration is described as mixture of two aspects: code migration and state migration. Code migration means that all the source code could be migrated from the source to the destination. State information includes execution state and data that is being used. Therefore, state migration consists of two aspects – execution migration and data migration.

Each thread has a program counter and its own state information, i.e. running, suspended and blocked. As a result, execution migration is composed of two aspects – program counter migration and thread migration.

In terms of data, it includes member variables, local variables, operands and external resources (e.g. network connections and files), which are being used by the code. Therefore, data migration is composed of member migration, stack migration and resource migration.

According to [Bouchenak00a], the key techniques in Java process / thread migration mainly include how to capture and restore Java process or thread's execution state and data, which is represented by execution migration and data migration in Figure 3.6. In order to achieve this objective, some solutions through different approaches have been proposed in earlier research work:

- The first approach is to pre-process the source-code of Java [Truyen00][Huang02]. In this approach, some extra Java statements are inserted into the source code before execution. During the execution time, the execution state of the Java program will be captured and the Java application with the current execution state can be stored and migrated.
- The second one is adding a middleware between Java application and JVM. This middleware is capable of capturing Java application execution states and data by using JVM Debugger Interface (JVMDI) [Ma00][Ma02]. Note that currently JVMDI [Sun04] has already replaced by JVM tool interface (JVMTI) and the functions of JVMDI are inherited by JVMTI [Sun06]. JVMDI/JVMTI is on top of the JVM and is a native interface for debuggers. It defines the standard services that a JVM must provide for debugging. When starting a Java application, a JVMDI/JVMTI client is started as well. By using JVMDI/JVMTI, the runtime information of threads, stack frames, local variables, classes, objects and methods can be obtained. In addition, some Java statements will be inserted to

the source code to make sure that the operand stacks is empty when the migration occurs.

- The third approach is to extend the JVM [Bouchenak00b]. In their mechanism, the JVM was extended to be able to extract the Java thread's execution state and store it in a Java object during the execution time. This object can then be stored in a file and sent to the destination Resource to resume the Java application from the current execution state.

Now, how to apply one or more of these proposed Java migration technologies in the proposed system architecture becomes a problem:

- For the first approach, it needs to pre-process the Java source code to insert additional statements for the purpose of capturing execution states and data while the Java application is running. When the Java application reaches the point of those specially inserted statements, the execution states and data can be accessed and recorded (checkpointed). Later, when job migration is needed, the original job (the Java applications) with the latest execution states and data can be transferred to a new resource and resume execution from the checkpoint.

  In theory, the work of inserting additional statements can be done at any level of the proposed system. At the user level, the additional statements can be added when the users create the Java source code. At the user/resource management level, if the users submit Java source code rather than submitting compiled .class files, then the Grid job scheduler can insert these additional statements and then get the source code compiled. At the resource level, if the users submit Java source code and the Grid job scheduler allocates the source code to the resource software, rather than the compiled .class files, then the resource software can insert additional statements and get the source code compiled.

  In terms of implementation, this approach could be simpler than the other two as it does not require any modification to the existing systems (e.g. extending JVM or adding a new middleware). What it requires is the ability to insert additional and special statements in the source code at the level of user, resource management or resource.

  In the use phase, this approach can be more complex than the other two approaches. Additional special statements have to be inserted into the Java source code. Therefore, where and how to insert this type of code is important and can be difficult to decide. So compared with the other two approaches, this approach can be more complex. In addition, as it requires the source code to be pre-processed before running, it may add significant overheads to application performance because of the inserted code. Furthermore, as the statements are inserted at certain points, this means the program states can be captured at these points only. Another potential problem is that many legacy applications' source code is not available anymore. This means it may be impossible to insert the required statements in the source code for the purpose of job migration.
- For the second approach, it needs to implement a middleware between the JVM and Java applications to capture Java applications' execution states and data. The middleware should

be capable of utilising JVMDI/JVMTI to achieve this goal. To apply this approach, the resource software should play the role as the middleware and be capable of invoking JVMDI/JVMTI to execute Java applications in a debugger mode. Except for this, there is no requirement to modify JVM or pre-processing Java source code. As JVMDI/JVMTI can capture applications execution states and data at any time, the resource software can access the updated job execution states and data when the migration is needed. After getting the necessary information, the resource software can transfer the original job and the update job execution states and data to a new resource.

In terms of implementation, this approach could be more complex compared to the first approach as it requires the implementation of an additional layer between Java application and JVM. However, compare with the third approach, it could be simpler as the work is to writing software to utilise existing JVMDI/JVMTI rather than modifying underlying infrastructure.

In the use phase, it could be simpler than the first approach as no further modification is needed after this approach is implemented and deployed. The resource software can deal with all the migration work without any pre-processing work to the Java source code.

One potential disadvantage of this approach is that the previous study in [JPC10] shows that a Java application running in the debugger mode is much slower than running in the normal mode. Therefore, this approach may significantly affect the jobs' *Makespan*.

- For the third approach, it needs to extend existing JVM. After extending the JVM, the modified JVM will be able to provide support for capturing Java applications' execution state and data. If the modified JVM is installed in the resource, then external applications (e.g. the resource software) can access the current state and data information of the running Java applications (e.g. the guest jobs) via the Application Programming Interface (API) provided by the extended JVM. When a job needs to be migrated, the resource software can access current job execution states and data by invoking specific functions provided by the modified JVM. Then the resource software can transfer the original job with its current execution states and data to a new resource.

  Compared with the other two approaches, this approach could be more complex in terms of implementation and deployment as it requires modifications to the existing JVM and it also requires the resources to install this modified version of JVM.

  In the use phase, similarly to the second approach, once it is implemented and deployed, then it will be a standard and easy way for resource software to access running jobs information and carry out job migrations. It could be even simpler than the second approach in terms of use, as the underlying infrastructure will provide enough support so that the resource software can access what it needs by using standard APIs.

  One concern for this approach is the security. As the modifications to the JVM enable resource software to access running jobs' execution information, it may also enable other

programs to fetch the information as well. This might be a threat to the running jobs if this program has malicious purposes. Therefore, security issues should be considered carefully when extending the existing JVM.

Therefore, none of these approaches has been specified as the only way to achieve job migration in the proposed system architecture. This is due to the following reasons:

Firstly, each technology is compatible with the system architecture. The proposed system architecture defines the broad framework for the whole system. However, for further details like choosing a job migration technology, the proposed system architecture is quite open (it is possible to apply any job migration technology compatible with the system architecture to the system architecture). According to the preceding discussions, all approaches are compatible with the system architecture, which means each of them can be applied to the proposed system architecture.

Secondly, each approach has its own advantages and disadvantages. To choose a certain technology, the chosen technology should provide more benefits than others. However, according to the above discussions, all technologies have their own distinct advantages and inherent disadvantages at the same time. Therefore, for the current stage, none of these technologies is specified as the only technology for job migration in the proposed system architecture.

3.4 System Operations Procedures

In order to make sure the proposed system architecture work properly, some important system operations were designed. As some system operations have been described in Section 2.1.3, this section will focus on the new operation procedures provided by the new system architecture.

3.4.1 Job Execution Monitor

After the job is distributed to a resource, the resource software will start the job. During the execution of the job, the resource software will monitor the execution progress and report to the job manager when important events occur. The procedure can be described as follows:

1. The resource software sends out job's latest information (e.g. job started, job finished 50% or job delayed, etc) to the job manager by using *Job Information Message*.
2. If the resource manager receives this updated job state information, it will store the data in the resource database.

In the way as the procedure of monitoring resource state, the job manager can also initiate the procedure of monitoring job execution progress. If the procedure is initiated by the job manager, then the procedure can be described as follows:

1. The resource manager sends a *Request Resource Information Message* to the resource software.
2. The resource software sends out job's latest information (e.g. job started, job finished 50% or job delayed) to the job manager by using *Resource Information Message*.

3. If the resource manager receives this updated job state information, it will store the data in the resource database.

3.4.2 Job Migration

Job migration is one of the most important features provided by the new system architecture, the benefits and the technique used to carry out this migration has been discussed in earlier chapters. In general, the whole procedure can be divided into two parts: the first part is to make a job migration decision and the second part is to carry out the job migration.

1. If the Grid job scheduler determines that the job needs migration or the resource owner notify the Grid job scheduler that the resource is going to be unavailable, the Grid job scheduler will look for a destination resource (the resource which the job will be migrated to) for the job according to a job-scheduling algorithm.
2. If a suitable resource is not found, then the Grid job scheduler will not make a job migration decision. Otherwise, the Grid job scheduler will make the job migration decisions.

After making the decision, the next step is to carry out the job migration. The steps are as follows:

1. The Grid job scheduler sends a *Migration Notification Message* to the origin and destination resources separately.
2. Once the origin resource software receives this message, it will send a TCP connection request to the destination resource software and setup a TCP connection with the destination resource software.
3. After this, the original resource will checkpoint the job and send a copy of the checkpointed job to the destination resource.
4. When the destination resource software receives this checkpointed job, it will resume the job on the destination resource. The origin resource software will terminate the job running on the origin resource.

3.5 System Messages

In order to support communication between different components in the proposed system architecture, some messages are designed to be used in different scenarios. These messages cover various aspects to ensure the system operation procedures can be carried out correctly, such as the *Registration Message* and *Registration Acknowledgement Message* ensure users and resources can register with the Grid system. The *Resource Information Message* lets resources report their updated system information to the Grid job scheduler, etc. The most important messages are the job migration related messages as they ensure live and automatic job migration procedures are carried out properly. For more details about the system messages, please refer to Appendix A.

Chapter 4 Job-scheduling and Job Migration

In this chapter, some novel job-scheduling and migration algorithms will be proposed.

4.1 Job-Scheduling Introduction

In this section, some background knowledge about job-scheduling will be introduced, including scheduling procedures, job-scheduling taxonomy, challenges of job-scheduling and so on.

4.1.1 Job-Scheduling Overview

Job-scheduling is a part of resource management in a Grid system. Resource management is a core part of a Grid system. In practice, there are several kinds of Grids, e.g. computational Grids and data Grids. For different kind of Grids, different objectives are defined. For computational Grids, one of the main objectives of resource management is to manage the resources and let them process computational jobs efficiently, such as improving job throughput and reducing job *Makespan*.

For a computational Grid like a desktop Grid or volunteer computing Grid, the main concern is to process computational jobs using the computing power of CPU(s) on each computer comprising the Grid. Therefore, in a computational Grid, resource management includes but is not limited to the following tasks: resource registration, resource state monitoring, job-scheduling and job state monitoring.

Among all these tasks, job-scheduling is one of the most important and complex. In a computational Grid, Job-scheduling is the process of mapping computational jobs to available resources with one or more objectives (e.g. achieving the highest throughput or the shortest job *Makespan*). Therefore, an efficient job-scheduling algorithm is important to achieve these objectives.

The reason for its complexity is because it can be difficult or even impossible to achieve some objectives at a given time. For example, conventionally, the Shortest Job First (SJF) [Thomas56] scheduling algorithm is proven optimal [Dusseau09] in terms of maximising job throughput when all jobs are simultaneously available [Bridgeport01]. However, SJF may starve long jobs as short jobs may occupy all available CPU cycles. In addition, to obtain optimal results, SJF requires all jobs to be available simultaneously, which is not usually practical in a computational Grid as new jobs may arrive at the job scheduler at any time. Furthermore, as the resources in desktop Grids and volunteer computing Grids are dynamic and volatile, it is difficult for the Grid job scheduler to ensure all jobs are completed successfully.

4.1.2 Job-Scheduling Components and Procedures

In Chapter 2 and 3, job-scheduling components and procedures have been described briefly. In terms of components, the Grid job scheduler and resource software are involved in the job-scheduling and they are responsible for different levels of job scheduling. Figure 4.1 shows

how these two components work together and interact with other system components in a computational Grid:

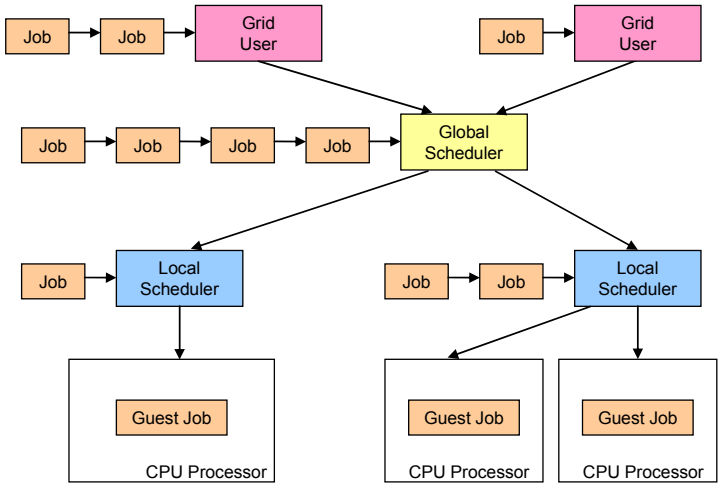


Figure 4-1: Global and local scheduling in a computational Grid

After the jobs are submitted to the Grid system, a job needs two levels of job-scheduling before it is executed by the CPU resource:

The first level is global scheduling, (deciding where to allocate jobs). In the proposed system architecture described in Chapter 3, the Grid job scheduler plays the role of global scheduler. As a global scheduler, it receives submitted jobs from users and decides where to allocate the jobs. Then according to its allocation decision, the job will be transferred to the resource.

The second level is local scheduling, (deciding the jobs' execution sequence) and is the responsibility of the local scheduler. In the proposed system architecture described in Chapter 3, the resource software is a local scheduler. As a local scheduler, it receives jobs allocated by the global scheduler and then decides which job to run first. A conventional computer typically has one CPU. Nowadays, however along with the development of computer hardware, more and more machines have more than one CPU. Therefore, in such a computer system, the local scheduler is not just responsible for deciding which job(s) to run next, but also responsible for deciding which job(s) to run on which CPU. After these two levels of job scheduling, a job will become a guest job on a resource and will be executed by the CPU of that resource. Note, in this thesis the job-scheduling decisions are based purely on the global scheduler, (the Grid job scheduler). The local scheduler (the resource software) always schedules jobs in FCFS order. In addition, the job migration decisions are based purely on the global scheduler and the local scheduler will get the jobs migrated according to the global scheduler's decisions.

In a Grid system, especially in the proposed Grid system architecture, the whole lifetime of a job from creation to termination can be summarised as follows:

1. The job is created and submitted by the Grid user. In the proposed Grid system architecture, a job is Java source/compiled files with/without additional input/output files, created and submitted by the user via the user software.
2. The job is scheduled by the global scheduler. In the proposed Grid system architecture, this

scheduling procedure can be divided into several parts: job submission, job allocation and job distribution. In the job submission, the job will be put into a job queue and sorted with a sorting algorithm. Next, the job will be mapped to an available resource in the Grid with a job-scheduling algorithm. Later, in job distribution, the job will be transferred to the resource.

3. The job is scheduled by the local scheduler. In the proposed Grid system architecture, resource software plays the role of local scheduler and is responsible for sorting incoming jobs and deciding which job should be run by which CPU if there is more than one CPU on the resource.
4. The job is executed by the resource CPU. In the proposed Grid system architecture, the resource plays the role of job processor and it is responsible for executing the jobs.

4.1.3 Taxonomy of Job-Scheduling Algorithms

In [Casavant88], the author proposed a hierarchical taxonomy for job-scheduling algorithms in general distributed systems. The taxonomy is shown in Figure 4.2

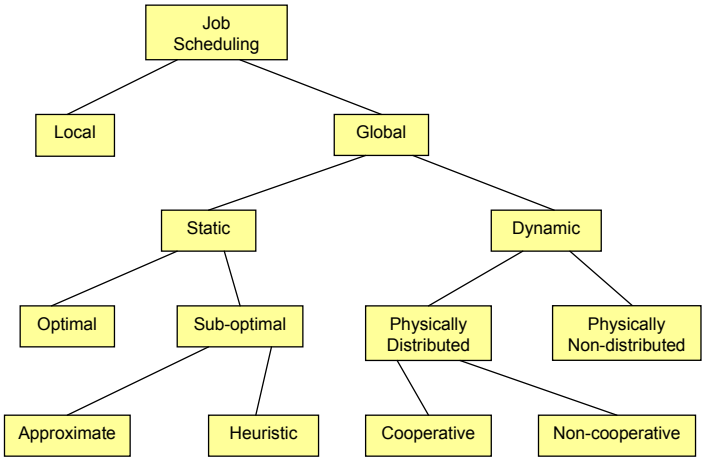


Figure 4-2: Task Scheduling Characteristics (Adapted from [Casavant88])

- *Local versus Global:* At the top level, the job-scheduling algorithm is divided into local and global scheduling. Local scheduling is used for scheduling jobs on a single CPU while global scheduling is used for scheduling among multiple CPUs. In Grid computing, the Grid job scheduler is mainly responsible for the global scheduling as it schedules jobs among different resources (CPU). Resource software is mainly responsible for local scheduling as it schedules jobs on the local CPU processor.

  However there are some exceptions here. Nowadays, computers with multiple CPUs are becoming more prevalent [Intel10][AMD10]. Therefore, if the resource software is working on such a computer, it will also be responsible for job-scheduling among multiple CPU so in fact the resource software will be not only responsible for local scheduling but also partially in charge of global scheduling.
- *Static versus Dynamic:* Global scheduling can be further divided into static scheduling and dynamic scheduling. According to [Casavant88], static scheduling has the same meaning as

deterministic scheduling described [Lo84]. In static scheduling, the information about jobs and resources is known by the job scheduler in advanced and the job-scheduling decisions will be made before the jobs are being executed on the resources. When decisions are made, jobs will be transferred to the resources and they will not be rescheduled to other resources whilst they are being executed. Many job-scheduling algorithms belong to static scheduling, such as FCFS [Esklcloglu01], EDF [StanKovic98] and so on. The job-scheduling algorithms proposed in this thesis also belong to static scheduling.

After the initial decisions are made, dynamic scheduling will still change the job-scheduling decisions when necessary. This is for the purpose of balancing loads between resources and avoiding potential job failures. In addition to job-scheduling algorithms, this thesis also proposes some job migration (rescheduling) algorithms as aids to these proposed job-scheduling algorithms. After adding these job migration (rescheduling) algorithms, the job-scheduling algorithms proposed in this thesis will become dynamic scheduling.

Compared with static scheduling, dynamic scheduling is more complex. However, as the decisions can be adjusted according to real time information, dynamic scheduling will be useful in a dynamic environment, such as desktop Grid and volunteer computing environments.

- *Optimal versus Suboptimal:* Static scheduling can be divided into optimal and suboptimal scheduling. Optimal scheduling means all related information about the job and the resource is known by the job-scheduler and optimal allocation decisions can be calculated within a feasible period of time. If these problems are not computational feasible and / or some related information is unknown, suboptimal allocation will be a more practical approach.
- *Approximate versus Heuristic:* In approximate scheduling, instead of searching the entire solution space, the algorithm will terminate when it finds a "good" solution. Next, the solution will be evaluated by an objective function and the job scheduler will decide whether to pursue this solution for later jobs, based on the results of the evaluation. Heuristic scheduling uses the most realistic assumptions about the jobs and resources to make a "reasonable" solution. Though heuristic algorithms use assumptions about jobs and resources, they are not restricted by the assumptions nor evaluated by an objective function. Therefore, they can make more flexible and adaptive decisions within an acceptable time given a certain computational complexity.
- *Physically distributed versus Physically non-distributed*: Dynamic scheduling can be further divided into distributed and non-distributed (centralised). For distributed scheduling, dynamic scheduling decisions will be made at different places. For example, if a resource considers a job is not suitable to run on itself any longer, it can reschedule the job to another resource. For non-distributed scheduling, all the dynamic scheduling decisions are made at a centralised place, such as the centralised job scheduler at the user/resource management

level. Therefore, jobs will be rescheduled when the job scheduler thinks jobs needs rescheduling. The job migration algorithms proposed in this thesis belong to non-distributed scheduling as all dynamic scheduling will be done by the centralised Grid job scheduler.

- *Cooperative versus Non-cooperative:* Distributed scheduling can be further divided into cooperative scheduling and non-cooperative scheduling. For cooperative scheduling, multiple resources will work together to make dynamic and distributed job-scheduling decisions toward a common system-wide goal. For example, multiple resources work together to decide which resources should run each job in order to maximise the job throughput. For non-cooperative scheduling, each resource works alone making job-scheduling decisions about how it should be used.

4.1.4 Open Issues in Job-Scheduling Algorithms

Resource management is the core part of a Grid system. In the Grid, resource management includes functions such as: resource discovery, resource state monitoring, job scheduling, and job state monitoring.

Among the functionalities of the resource management, job-scheduling is an important piece of work. In a Grid environment, especially in a volunteer resource based Grid job-scheduling may be more complex than a conventional distributed computing system. A volunteer resource based Grid may have an important issue that rarely appears in conventional distributed computing systems. In a conventional distributed computing system, all the resources in the system are typically assumed to be available to the system and jobs can run on them all the time (except when the system does not work properly, e.g. resources crash or the network connection is down). For a desktop Grid or a volunteer computing environment, resources may not always be available to the Grid and the job(s) may not always be allowed to run on all resources all of the time. This is due to the characteristics of these systems:

Firstly, resources in these systems may be assumed to be fully controlled by the resources owners. In such a case, each resource's owner can decide when and how to donate their resources. In terms of when to donate the resource, each resource's owner can specify some policies through the user software. For example, a resource owner can define a policy that the resource only works for a Grid during the night, from 12am to 8am. In terms of how to donate the resource, resources owners can also specify some related policies as well.

In some Grid systems, default policies are predefined by the system. For example, in the Condor Grid system, the resource can only execute guest jobs when the CPU load is not above 25% and the job suspension time is at most 10 minutes [Kondo05]. If the CPU load of a resource is over the threshold (i.e. 25% in Condor), the guest jobs will be temporarily suspended first. Later, if the resource CPU load reduces to less than the threshold before a predefined timeout (10 minutes in Condor), the job will be able to resume from the point where it was suspended. However if the CPU load is not lowered to less than the threshold before the

predefined timeout, the job will fail.

Secondly, resources in such systems may be assumed not to interfere with local activities. Here, local activities are the activities initialised by the user of the local resource, such as mouse/keyboard activity, initiating local processes and accessing the hard disk. So if any kind of local activity occurs, the guest job(s) running on the resources will be suspended. If the resources do not become idle after a predefined time-out, the job will be terminated and lost. Figure 4.3 illustrates this problem.

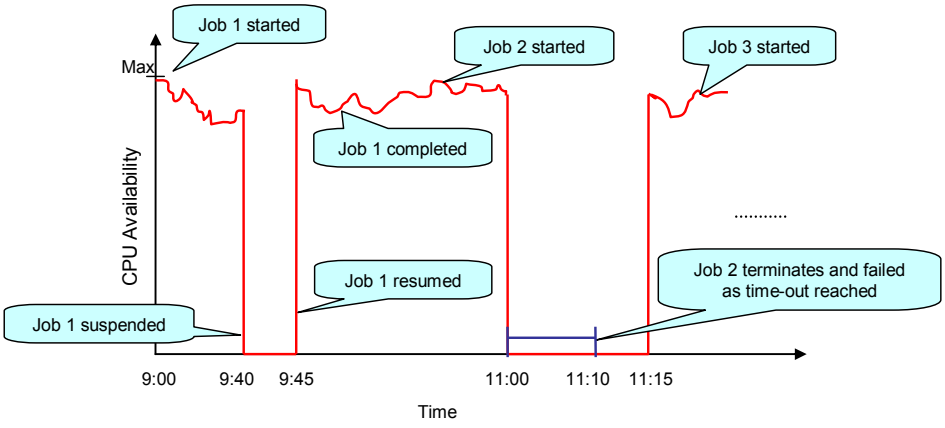


Figure 4-3: Influence of Availability in Volunteered Resource-Based Systems

(Adapted from [Kondo07])

In Figure 4.3, Y axis "CPU Availability" shows the number of CPU cycles delivered to the Grid by a resource per second. Max means the maximum CPU speed. CPU availability may vary along with the time as some CPU cycles are used by the local processes. When a local activity occurs, the CPU availability will become 0 and the guest job(s) will be suspended. After a short while, if the local activity finishes, then the CPU availability will be above 0 again and the guest job(s) will be resumed. However, if the suspension period is too long and exceeds the predefined time-out, the job will be terminated and fail.

Note these two characteristics depend on the definitions and assumptions of a specific Grid system. For example, in a volunteer computing environment BOINC [ANDERSON05], resource owners can decide when and how to donate their resource and jobs are allowed to run while local activities are being carried out. For Grid system Condor [Thain05], resource owners can decide when and how to donate their resource and jobs are NOT allowed to run if resource's CPU load is over 25%. For Grid system XtremWeb [Neri00], it has both two assumptions mentioned above: resource owners can decide when and how to donate their resource and jobs are allowed to run while local activities are being carried out. In this thesis, these two important assumptions are used and the predefined suspension time-out shown in Figure 4.3 is assumed to be 0. This means the guest job will be terminated and failed once the resource owners reclaim their machines. These assumptions are based on the following considerations:

- Firstly, the influences of the Grid system's activities to volunteered resources are controlled at the lowest level. The activities of the Grid system are almost transparent and invisible to

the resource owners so that guest jobs from the system will not slow down their resources. Therefore, these conditions protect the resource owners' rights and this should attract more people to join the Grid system.

- Secondly, these assumptions present a big challenge to the job-scheduling algorithm, in particular how to ensure guest jobs are processed successfully as well as quickly. Conventionally, job-scheduling algorithms are typically assumed to work with looser assumption(s); very limited work has considered this issue though one example is [Kondo05], which is discussed in the next paragraph. Therefore, more research work is needed in this area.

[Kondo05][Kondo07] propose a number of resource prioritisation/exclusion methods for resource selection in a Grid computing context, especially in the Grid context where volunteer and unreliable resources are utilised. Some of these methods simply use static information, e.g. resources' clock rate to prioritise resources and some use resources' past performance to predict its future performance and prioritise the resources accordingly. Based on the prioritisation results, the Grid job scheduler allocates jobs to the resource that has the highest priority. They evaluate the job-scheduling algorithm with a couple of resource availability data traces collected from real desktop Grids (these data traces are also utilised in this research, more detail about the data traces are provided in Chapter 6 and 7). According to their results, though FCFS job-scheduling algorithm is a simple and static algorithm, it works relatively well in many different scenarios, especially in the scenarios where the number of jobs are more than the number of resources. Therefore, in this thesis FCFS algorithm is used to compare the proposed algorithms in many scenarios (more details are provided in Chapter 7).

An approach to solve this problem is that the job-scheduling algorithm allocates jobs without reference to resource availability at all. This is the approach used mainly by existing job-scheduling algorithms. Instead of reference to resource availability, some existing job-scheduling algorithms use other approaches to mitigate the problem. For example, in one approach the Grid job scheduler simply ignores the problem and reallocates the job to another resource if the job fails (e.g. FCFS algorithm).

Another approach to the problem is for the Grid job scheduler to use job replication (e.g. derrick [Kondo07]). Here, job replication means allocating the same job to multiple resources at the same time. However, these solutions also present difficulties.

The first approach cannot provide any reliability so that jobs' *Makespan* may be delayed and job throughput may be reduced as a result of resource volatility. For example, if a job runs on a resource and then fails before completion, the job needs to be allocated to another resource and will start from the beginning. Furthermore, though the first resource cannot finish the job before it becomes unavailable, it has already used some CPU cycles, as the job fails, so the used CPU cycles can be considered wasted.

The second approach, though it provides extra reliability via job replication, some CPU

cycles will be wasted as a result of replication. For example, even if a job is allocated to two resources at the same time, then only job will be completed. If two jobs are allocated to two resources separately, then two jobs may be completed. Therefore, job replication is way of providing extra reliability by sacrificing the job throughput.

In an ideal situation, one resource should have one job at a time and it should be able to migrate the job to another resource just prior to the resource becoming unavailable. In this case, the jobs' *Makespan* will not be delayed due to the resource unavailability nor will the resources' CPU cycles be wasted. To approach this ideal state, some important requirements are needed:

Firstly, the Grid system should support live and automatic job migration between heterogeneous systems so that jobs can be migrated proactively or reactively when necessary. However, existing Grid systems lack such a mechanism for providing support for live job migration between heterogeneous systems. Only a few of existing systems support job migration, e.g. Condor [Thain05], MOSIX [Barak05][Barak08] and vOS [Boyd02]. However, very few of them support heterogeneous job migration between heterogeneous systems. For example, Condor supports live job migration, but it does not provide any automatic mechanism and it can only support job migration within Unix system. Therefore, to provide support for live and automatic job migration between heterogeneous systems, a new Grid system architecture is proposed in this research. In addition to providing a job migration mechanism, this system can also provide other benefits. More detailed information is given in Chapter 3.

Secondly, the Grid system should have a job-scheduling algorithm, with the information about resources' future availability and/or reliability, which can allocate jobs effectively. Therefore, job migration will delay the jobs' execution. Therefore, effective job allocation decisions should allocate jobs to resources that require minimum times of migrations (0 times in the ideal case). To make effective job allocation decisions, this research work proposes some novel job-scheduling algorithms. More discussions about this job-scheduling algorithm will be provided in Section 4.2.

Thirdly, the Grid system should be able to carry out job migration after jobs have been allocated to resources. There are two types of job migration algorithms. If resources notify the job scheduler when they are going to be unavailable, the job scheduler can trigger these job migrations; it will provide reactive job migrations, which is straightforward as the job scheduler can trigger job migrations according to resources notifications. However, if resources do not notify the job scheduler, proactive job migrations are required and the job scheduler needs to anticipate all resources' future availability so that it can trigger job migrations when resources are about to become unavailable. To anticipate resources' future availability, two types of measurements can be carried. The first measurement is uses the user's defined policies and the second one uses prediction techniques. To make predictions, this research work adopted and examined a prediction technique. More details about these two types of measurements are discussed in Section 4.3.

4.2 Proposed Job-Scheduling Algorithms

As mentioned in Section 4.1, one efficient approach to solve the problems caused by resource volatility in the volunteered resources based Grid is to enable the Grid job scheduler to allocate jobs effectively with the information about resources' future availability and/or reliability. Therefore, this research proposes a couple of novel job-scheduling algorithms based on considerations about resources' availability and/or reliability. In this section, all the algorithms proposed in this research will be introduced.

Generally, in addition to the two important assumptions mentioned in Section 4.1 (resource owners can decide when and how to donate their resource and jobs are allowed to run while local activities are being carried out), all job-scheduling and job migration algorithms proposed in this research work have the following assumptions: Firstly, there is a centralised Grid job scheduler in the Grid and users submit their self-contained executable jobs to the Grid job scheduler. When the job arrives at the Grid job scheduler, the Grid job scheduler puts the job into a job queue. Later, the Grid job scheduler decides where to allocate these jobs. Jobs can fully exploit resources' CPU cycles donated to the grid. After execution, resources return the results to the original users that submitted those jobs. The Grid job scheduler is presumed to know the execution time for each job before making allocation decisions.

All the job-scheduling algorithms proposed in this thesis utilise a resource availability predictor and this resource availability predictor is based on a resource availability technique proposed in [Rood07][Rood08]. In fact, some prediction techniques have been proposed as well, such as Saturating and History Counter predictor [Mickens05][Mickens06][Mickens07], Ren predictor [Ren06a][Ren06b][Ren07a][Ren07b] and Multi-State and Single State Sliding Window predictor [Dinda99]. The reasons why the prediction technique was adopted in this research was: firstly, their model describes resource states clearly for a Grid, especially for a volunteer Grid. Secondly, according to their simulation results, their prediction technique was more accurate than some other existing prediction techniques, including those prediction techniques proposed in [Mickens05][Dinda99][Ren06a]. In order to better understand the proposed algorithms, some background knowledge about the adopted prediction technique is given.

4.2.1 Adopted Prediction Technique

The prediction technique uses a multi-state model to describe a resource's state. Figure 4.4 shows this model:

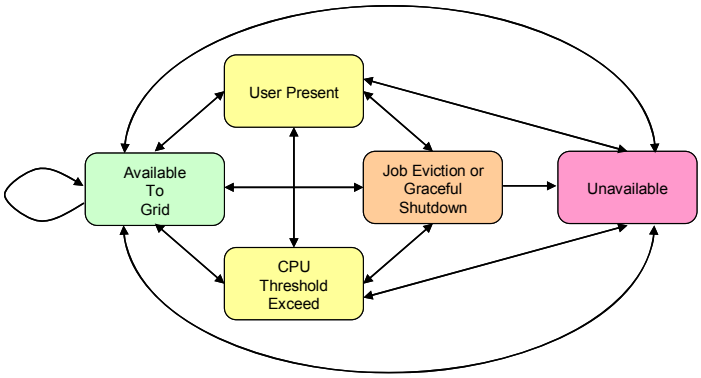


Figure 4-4: Availability States and Transitions (Adapted from [Rood08])

In this model, there are five states to describe the state of a volunteered resource in the Grid:

1. *Available to Grid*: *Available to Grid* means the resource is accessible and exploitable by the Grid at the moment. When the resource is in this state, the guest job(s) allocated by the Grid is allowed to utilise the idle cycles of the resource's CPU.
2. *User Present*: *User Present* means the resource is being used by the owner at the moment so that it is accessible by the Grid but not available to the Grid. When the resource is in this state, the guest job(s) will have to be suspended.
3. *CPU Threshold Exceeded*: *CPU Threshold Exceeded* means the CPU load of the resource is over a predefined threshold so that it is available to the Grid but not exploitable by the Grid. When the resource is in this state, the guest job(s) on the resource will have to be suspended.
4. *Job Eviction or Graceful Shutdown*: *Job Eviction or Graceful Shutdown* means the resource has notified the Grid that it is going to leave the Grid or it is not going to allow guest job(s) to run any longer so that the resource is available (and become unavailable soon) to the Grid but not exploitable by the Grid. When the resource is in this state, the guest job(s) will have to be migrated, otherwise the job(s) will be lost if the resource enters *Unavailable* state later.
5. *Unavailable*: *Unavailable* means the resource is not in the Grid at the moment so that it is neither available nor exploitable by the Grid. When the resource is in this state, the guest job(s) will not be able to allocate to the resource. If any guest job(s) is still on the resource when the resource is in this state, the guest job(s) will be lost.

With this resource availability model, they then consider a number of multi-state prediction algorithms. In brief, a multi-state algorithm works as follows: it takes a length of time as an input (this is called *Checking Period*) and uses a resource's availability history to predict the probability of that resource remain in the state of *Available to Grid* throughout the interval (this interval is called *Prediction Period*). This probability is called *Resource Availability*.

To calculate this probability, they employ several techniques and the one used in this research is Transitional N-Day with Equal transition weights (TDE). "Transitional" means the prediction technique calculates the output probabilities by counting both the number of transitions from *Available to Grid* to other states and how many times the job could be processed between two

transitions. "N-Day" means checking the most recent past N days' transitions. *Number of Checking Days* is a parameter to define the number of N here. "Equal transition weights" means the prediction method considers each transition within the different checking days equally. According to their results, TDE is the most successful technique among all the prediction techniques they tested when the prediction length is no longer than 42 hours, especially when it is shorter than 19 hours. According to research in [Lazarevic06] [Li04][Iosup06][Medernach05], the *Job Execution Time* in a grid context is typically less than 10^5 seconds (around 27.8 hours), so TDE was adopted in this thesis.

A day to which the *Checking Period* belongs is called a *Checking Day* and a day to which the *Prediction Period* belongs is called a *Prediction Day*. If the value of *Number of Checking Days* is larger than 1 (which means the prediction method will check more than one day to make a prediction), each day being checked is a *Checking Day*.

Their TDE prediction method checks the resource's most recent past N days' availability state, transition history, to get a prediction result. Briefly, the TDE prediction method works as follows: assume now the current time is $T_{current}$ and predictor is examining a resource for a job that is expected to run for L hours. Here, the length that a job is expected to run is called *Job Execution Time*. The predictor will check the resource' state transitions history (only state transitions exit from the state of *Available to Grid*) in the *Checking Period* – time between $T_{current}$ and $T_{current}$ + M*L in the past N days.

Here, M is a *Multiply Factor*; a positive number equal to or bigger than 1, like 1, 1.3, 6, etc. After checking, the predictor calculates the *Resource Availability Probability* result in the *Prediction Period* – between $T_{current}$ and $T_{current}$ + M*L in the current day. The length of *Prediction Period* is the same as the length of *Checking Period*.

In this thesis, the definition of "past N days" is slightly different to the definition of used by the proposer of the TDE prediction method [Rood08]. In this thesis, "past N days" is considered as the most recent past N days by default, but it could be any N days in the past. This is based on the following assumptions:

Firstly, no evidence has been found to show that the prediction will be more accurate if the most recent, past N days data is used (this will be discussed more in Chapter 6). If resource owners' behaviours have a regular pattern everyday, then using the nearest "past N days" data will be helpful for prediction. However, if resources owners' behaviours have some other patterns or no patterns, then using the nearest "past N days" data may not be helpful for prediction. For example, assume some resources are owned by company staff and the resources are usually used intermittently during 9am to 5pm in the working days but left free during the weekend. So if now is Monday and the prediction method uses past N days' data (including the weekend's data) to make a prediction, the accuracy of the prediction will be doubtful.

Secondly, though job's *Job Execution Time* is typically less than 24 hours, it could be longer than that time in principle. If a job's *Job Execution Time* is longer than 24 hours, the length of

Checking Period will overlap with the *Prediction Period*, which is problematic. For example, assume the time is 9am and the prediction method uses past 1 day's data. If the job's *Job Execution Time* is 26 hours, then the length of *Checking Period* will be 26 hours - from 9am yesterday to 11am today, which is beyond the current time. Therefore, this will influence the accuracy of the prediction.

Let $P_{dayi}(r)$ denote the *Resource Availability Probability* of resource r in day i; $P_{dayi}(r)$ is calculated by the following equation:

$$P_{dayi}(r) = T_{dayi} / T_{all} * 100\% \qquad \text{(Equation 4.1)}$$

where T_{toA} denotes the number of times that resource r transits from the state of *Available to Grid* to *Available to Grid* in the *Prediction Period* and Tall denotes the number of times that resource r transits from the state of *Available to Grid* to all states in the *Prediction Period*. P(r) denotes the final output of *Resource Availability Probability*; it can be calculated by the following equation:

$$P(r) = \frac{\sum_{i=1}^{N} P_{dayi}(r)}{N} * 100\% \qquad \text{(Equation 4.2)}$$

where N is the total number of *Checking Days*.

4.2.2 Resource Availability

As discussed in Section 4.1.4, "resource availability" is an important term in a thesis since resource volatility is a distinct characteristic in a volunteer resource based Grid environment and this characteristic brings a big challenge to the job-scheduling algorithms. Therefore, "resources availability" should be one of the main concerns for the proposed job-scheduling and job migration algorithms. Before discussing the proposed job-scheduling and job migration algorithms, some definitions and clarifications related to "resource availability" are introduced.

"Resource availability" can be defined at different levels. In [Kondo05], the author defines three levels of availability in the Grid computing environment: *Host Availability*, *Job Execution Availability* and *CPU Availability*.

Host Availability: *Host Availability* indicates whether the resource is reachable by the Grid. If a resource is in a Grid, then *Host Availability* is true. Otherwise, *Host Availability* is false. In the multi-state model described in Section 4.2.1, if a resource is in the state of *Available to Grid* or *User Present* or *CPU Threshold Exceeded* or *Job Eviction or Graceful Shutdown*, the resource's *Host Availability* will be true. If a resource is in the state of *Unavailable*, the resource's *Host Availability* is false.

Job Execution Availability: *Job Execution Availability* indicates whether guest jobs from the Grid are currently allowed to execute on the resource. This is based on the resources' recruitment policy. Therefore, if the resource's current condition is in line with the recruitment policy, *Job Execution Availability* will true. Otherwise, the *Job Execution Availability* will be

false. If *Resource Unavailability* is false, *Resource Execution Unavailability* will be false as well because *Job Execution Availability* cannot be true if the resource is unavailable. On the other hand, if *Job Execution Availability* is true, *Host Availability* will be true. In the multi-state model described in Section 4.2.1, if a resource is in the state of *Available to Grid,* the resource's *Job Execution Availability* will be true. Otherwise, the resource's *Job Execution Availability* will be false if it is in any other state.

CPU Availability: *CPU Availability* indicates current CPU speed (number of CPU cycles delivered to the Grid per second). It is directly influenced by the resource's recruitment policy and activities of local processes on the resource. For example, if a resource recruitment policy defines that the guest job is not allowed to run when the owner reclaims the resource, then CPU cycles delivered to the Grid will becomes 0 when the owner reclaims the resource. The difference between *CPU availability* and *Job Execution Availability* is that *Job Execution Availability* only indicates whether a resource is currently allowing guest jobs to run on it or not, whilst *CPU Availability* not only indicates this but also shows the number of CPU cycles the resource contributes to the Grid in each second. Therefore, if *Job Execution Availability* is false, *CPU Availability* will be 0. On the other hand, *Job Execution Availability* can be derived from the value of *CPU Availability*. If *CPU Availability* is above 0, *Job Execution Availability* will be true. If *CPU Availability* is 0, *Job Execution Availability* will be false. As for *CPU Availability*, in the multi-state model described in Section 4.2.1, if a resource is in the state of *Available to Grid,* the resource's *CPU Availability* will be true. Otherwise, the resource's *CPU Availability* will be true if it is in any other state.

In this thesis, unless stated, resources are volunteered members of a Grid. Furthermore, if a resource is *available* it means the resource's *Job Execution Availability* is true and the resource is prepared to allow guest jobs to run on it. Conversely, when a resource is *unavailable* it means the resource's *Job Execution Availability* is false then the resource does not currently allow any guest jobs to run on it. A resource may already have had old guest job(s) when the job scheduler tries to allocate new guest job(s) to it. If a resource is *available* and it does not have any guest job(s), it will be called *idle* whereas a *busy* resource is already running a guest job. A resource may be considered as suitable when the Grid Job Scheduler tries to make a job allocation. A *qualified* resource is considered to be a suitable resource for the first job in the job queue so a job can be directly allocated to the resource. An *unqualified* resource is not considered to be a suitable resource for the first job in the job queue and so a job will not be allocated to the resource directly.

4.2.3 FCFS plus Predictor (FCFSPP) Algorithm

The first algorithm is called the FCFS Plus Predictor (FCFSPP) algorithm. This algorithm is based on a simple and widely used algorithm FCFS and to it is added an advanced part – a resource availability predictor. In FCFS algorithm, jobs will be allocated to *available* resources

in turn. In terms of prediction, this algorithm uses a resource availability predictor based on the prediction technique described in Section 4.2.1.

According to [Dogan02], for a job-scheduling algorithm, there is a trade-off between *speed* and *reliability*. This means that it is impossible to achieve both objectives at the same time in most cases. Extensive research has been carried out to achieve the objective of *speed*, this proposed FCFSPP scheduling algorithm focuses on the second objective whilst not ignoring the first objective – trying to ensure the *reliability* as much as possible with as least as possible cost in terms of *speed*.

The basic idea of this job-scheduling algorithm is to avoid allocating jobs to the resources that are considered to be *unqualified*. The resource availability predictor is used to judge whether a resource is *qualified* or not. In general, the procedure of the job-scheduling algorithm can be divided into two separate parts. The first part is *Job Submission*. Figure 4.5 shows the *Job Submission* procedure.

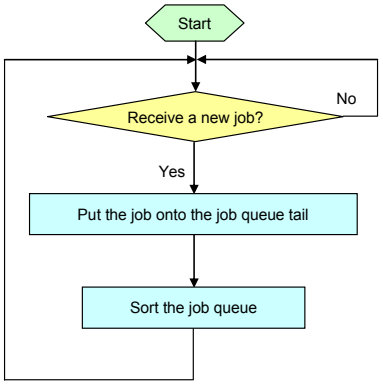


Figure 4-5: FCFSPP Algorithm *Job Submission* Procedure

Firstly, the Grid job scheduler waits for jobs all the time. The jobs are self-contained jobs sent by the Grid users. Secondly, if the job scheduler receives a job, it will put the job onto the end of the job queue. Thirdly, the job scheduler uses a predefined algorithm (e.g. FCFS or EDF) to sort the job queue.

After *Job Submission*, the next part of job-scheduling algorithm is *Job Allocation*. Figure 4.6 shows the *Job Allocation* procedure.

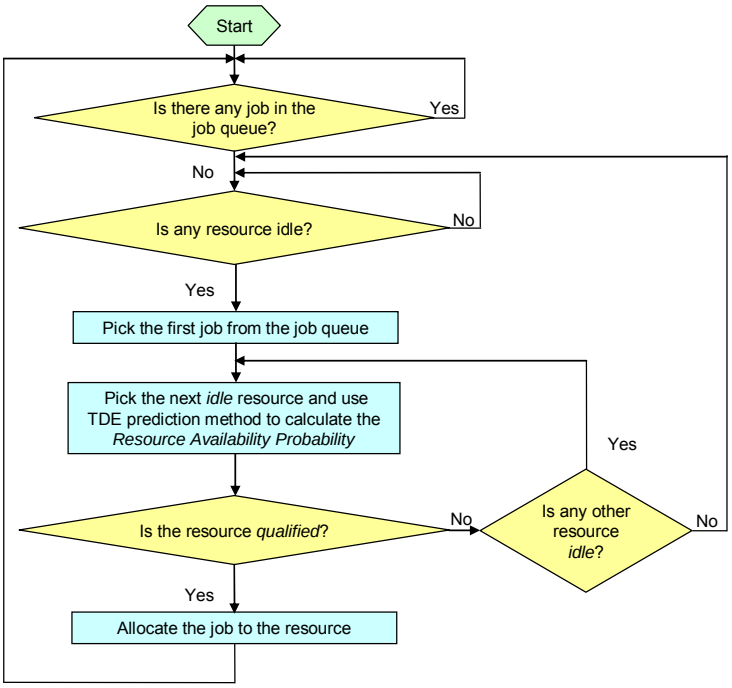


Figure 4-6: FCFSPP Algorithm *Job Allocation* Procedure

First, at regular intervals, if the Grid job queue is not empty and there are idle resource(s) in the Grid, the Grid job scheduler picks the first job in the job queue.

Second, the Grid job scheduler picks the next *idle* resource from the resource list as a candidate resource and uses the TDE prediction method to calculate the *Resource Availability Probability* for the waiting job.

Third, the Grid job scheduler checks to see if the picked candidate resource is *qualified* or not. Here, *qualified* means the *Resource Availability Probability* is over a predefined threshold – *Resource Availability Probability Threshold* after checking the resources nearest past few days' *Job Execution Availability* history. For example, if the *Resource Availability Probability* is 80% and the predefined threshold is 70%, then the candidate's resource is considered to be qualified as the *Resource Availability Probability* is higher than the predefined threshold. In addition to the parameter *Resource Availability Probability Threshold*, there is another important parameter *Number of Checking Days* here. It means the number of days checked by the predictor. So if the *Number of Checking Days* is 3 in the above example, the value of *Resource Availability Probability* 80% is calculated after checking the resource's past 3 days' *Job Execution Availability* history.

Fourth, based on the checking result, the job scheduler makes a decision about whether to allocate the job to the candidate resource. If yes, then the job will be allocated to the resource and the Grid job scheduler goes back to the first step of *Job Allocation*. If no, the Grid job scheduler will check if there is any other *idle* resource(s) in the Grid at the moment. If yes, the Grid job scheduler will go back to the third step. If no, the Grid job scheduler will go back to the first step.

4.2.4 Fuzzy Logic plus Predictor (FLP) Algorithm

The second algorithm proposed in this research is based on Fuzzy Logic (FL). FL is a type of Artificial Intelligence (AI) technique used in certain areas, including air conditioners, automobile, digital image processing amongst others.

Fuzzy Logic Introduction

The concept of FL was proposed in [Zadeh73]. It was derived from fuzzy set theory [Zadeh65]. "Basically, Fuzzy Logic is a multi-valued logic that allows intermediate values to be defined between conventional evaluations like true/false, yes/no, high/low, etc."[Hellmann01]. FL is not only widely used in building real products (e.g. washing machines, fridges and so on), but it has also one of the most active and fruitful area of research in the past few decades. Furthermore, "the use of fuzzy systems makes a viable addition to the field of Artificial Intelligence" [Brule05]

- **Fuzzy Sets**

FL allows people to encode linguistic expressions to numeric form and therefore build a more flexible rule based computer system. As with traditional logic, a computer is a binary based system so Boolean logic of computer programming only has the two values: true (1) and false (0). Although there are many advantages of two-value based logic, it is difficult to describe some terms in the real world. In reality, there are many imprecise concepts for instance statements like "Resource A is fast", "Resource B is reliable" and so on. Figure 4.7 illustrates how the term "fast" could be represented in a computer system.

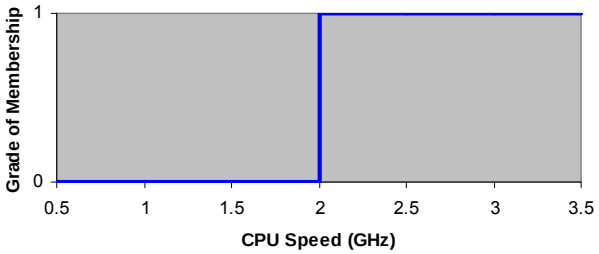


Figure 4-7: Traditional Boolean Logic in Computer System

A computer with CPU speed 2GHz is considered fast but a computer with CPU speed 1.9GHz is not. This does not really reflect the way people think or make comparative judgements. To let the computer reflect the way people think, FL introduces multiple values between the truth (1) and false (0). As a result, though "grade of membership" can only be 0 or 1 in the computer world, "grade of membership" can have values between 0 and 1 in the fuzzy world. Therefore, the term "fast" could be represented in the fuzzy world as shown in Figure 4.8.

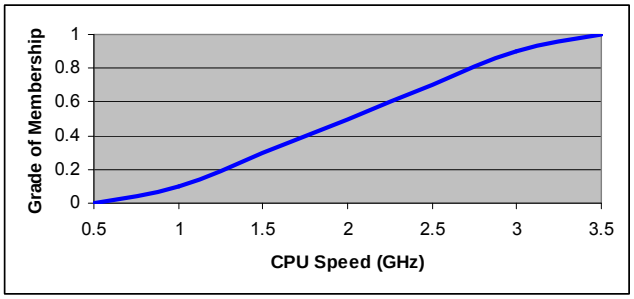


Figure 4-8: Fuzzy Logic in Fuzzy World

So a computer that has 2GHz CPU and the computer that has 1.9GHz will have the values of "grade of membership" of around 0.5 in FL, which is much closer to the way people think rather than Boolean logic.

"The notion central to fuzzy systems is that truth values (in fuzzy logic) or membership values (in fuzzy sets) are indicated by a value on the range [0.0, 1.0], with 0.0 representing absolute falseness and 1.0 representing absolute Truth."[Brule05]. To determine the membership value for an element x, membership function $f(x)$ is used. "A fuzzy set (class) A in X is characterized by a membership (characteristic) function $f_A(x)$ which associates with each point in X a real number in the interval [0,1], with the value of $f_A(x)$ at x representing the "grade of membership" of x in A" [Zadeh65]. Take Figure 4.8 for example, if "Resource's CPU speed is 2GHz", then the result of membership function f(2GHz) is 0.5 in this example, which means the resource's grade of membership within the set of fast computers is 0.5.

- **Fuzzy Set Operations**

Besides the concept of fuzzy sets, some operations are also used to express the fuzzy "thing". Three Boolean logic operators are used FL: OR, AND and NOT. Assuming A and B are two fuzzy sets, then the operations results are defined as follow:

OR: A U B = MAX (A, B)

AND: A ∩ B = MIN (A, B)

NOT: ¬A = 1 - A

For example, assume that fuzzy sets A and B have the shapes shown in Figure 4.9:

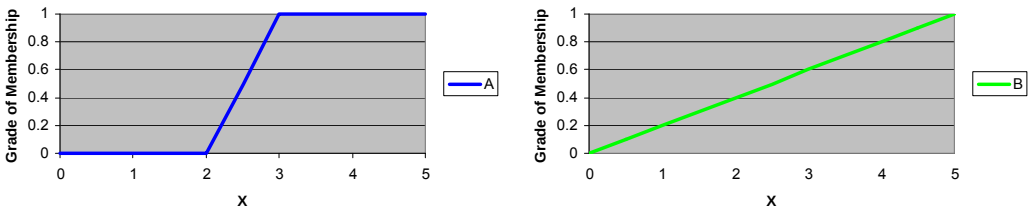


Figure 4-9: Fuzzy Set A and B

Then the results of A U B, A ∩ B, and ¬A are shown in Figure 4.10:

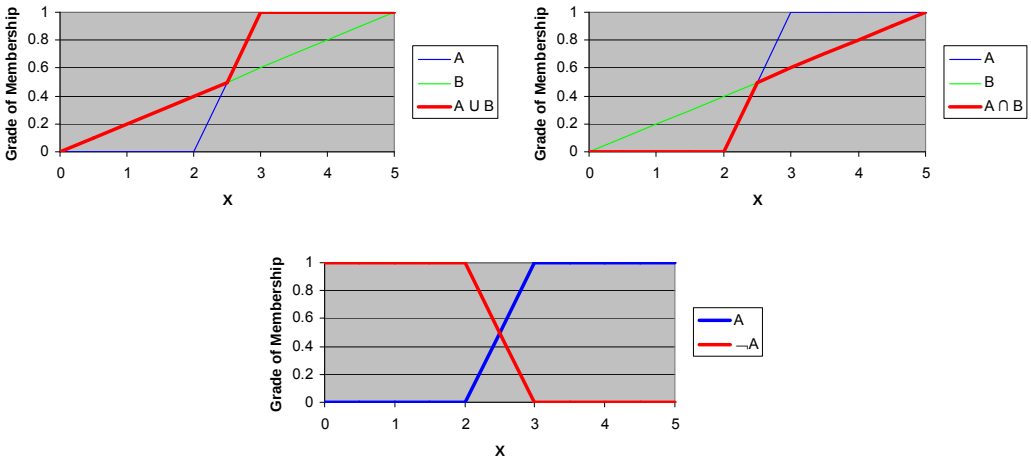


Figure 4-10: Result of A ∪ B, A ∩ B and ¬A

- **Fuzzy Rules**

In addition to the concept of fuzzy sets and fuzzy set operations, rules for inference are defined. The set of rules is usually expressed in the form:

"`IF` *`variable`* `IS` *`set`* `THEN` *`action`*" [Synaptic06]

For example, a simple set of rules for controlling a heater could be defined as follows:

```
IF temperature IS cold THEN start heating.
IF temperature IS hot THEN stop heating.
```

With these defined rules, a fuzzy control system can influence the output according to the input variable(s). Similar to membership functions, this kind of rules could be modified according to the design requirement. For example, the rules for the heater be refined as follows:

```
IF temperature IS cold THEN speed up heating.
IF temperature IS normal THEN keep the speed.
IF temperature IS hot THEN stop heating.
```

Overall, FL provides a different way to solve a control problem and it focuses on what the system should do with a set of rules rather than using a complex mathematic model. As our research will focus on a large-scale network, the use of FL will provide benefits such as getting the system to work correctly without worrying too much about the complex mathematic model of the network.

- **Fuzzy Inference System**

Next, if fuzzy sets, fuzzy set operations and fuzzy rules are put together, a FL based system – A Fuzzy inference system can be built up. A fuzzy inference system consists of four distinct steps:

1. Fuzzification: In this step, a "crisp" numerical value will be translated to a fuzzy variable with membership function.
2. Rule Evaluation: In this step, some fuzzy rules are defined and the fuzzy sets' truth values will be applied to each rule to get outputs.
3. Aggregation: In this step, all outputs are aggregated.

4. Defuzzyfication: In this step, the aggregated output will be translated to a "crisp" numerical value as the final output.

To explain how a fuzzy inference system operates, consider an example where FL is used to control the *Resource Availability Probability Threshold*. Here, *Dispose Jobs Dot* is used to describe the difference between the number of disposed jobs in the last time interval N_{t-1} and the number of disposed jobs in the current interval N_t: *Dispose Jobs Dot* = $N_t - N_{t-1}$. Therefore, if the numbers of disposed jobs are 12 and 7 in the last and the current time interval, respectively, *Dispose Jobs Dot* will be 7 – 12 = -5. As a result, the *Resource Availability Probability Threshold* will be lowered as resources tend to dispose of fewer jobs (*Dispose Jobs Dot* is smaller than 0).

In this fuzzy inference system, the input is *Dispose Jobs Dot* and the output is the adjustment value to the *Resource Availability Probability Threshold.* To describe *Dispose Jobs Dot*, three fuzzy sets are defined – Negative, Zero and Positive.

The first step is fuzzification. In this step, the "crisp" numerical values of *Disposed Jobs Dot* will be translated to fuzzy variables with the membership function shown in Figure 4.11.

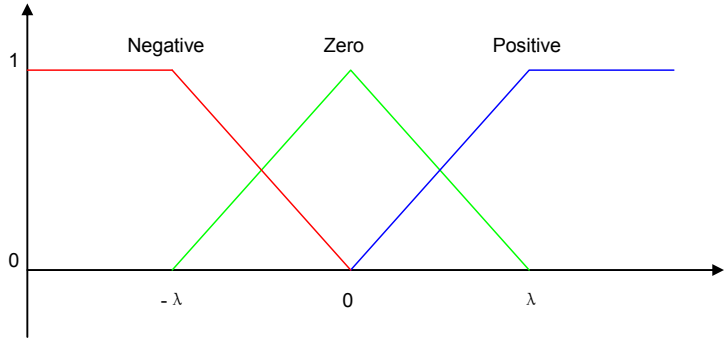


Figure 4-11: FLP Algorithm Membership Function

In this membership function, two important thresholds are λ and – λ. λ is a numerical value, such as 1, 5, etc. If the value of *Disposed Jobs Dot* is smaller than 0, *Disposed Jobs Dot*'s truth value of Negative is above 0 and becomes 1 when the value of *Disposed Jobs Dot* is smaller than – λ. If the value of *Disposed Jobs Dot* is between – λ and λ, *Disposed Jobs Dot*'s truth value of Zero is above 0 and becomes 1 when the value of *Disposed Jobs Dot* is 0. Finally, if the value of *Disposed Jobs Dot* is larger than 0, *Disposed Jobs Dot*'s truth value of Positive is above 0 and becomes 1 when the value of *Disposed Jobs Dot* is larger than λ.

In the second step of rule evaluation, some fuzzy rules are defined:

1. If *Disposed Jobs Dot* = Negative, then *Resource Availability Probability Threshold* changes = Negative%.
2. If *Disposed Jobs Dot* = Zero, then *Resource Availability Probability Threshold* changes = Negative%.
3. If *Disposed Jobs Dot* = Positive, then *Resource Availability Probability Threshold* changes = Positive%.

Here, the rules can be translated into the following natural language. Take rule 1 for example;

if the *Disposed Jobs Dot*'s grade of membership within the set of Negative is above 0, then the value of *Resource Availability Probability Threshold* will be lowered by *Disposed Jobs Dot*'s truth value of Negative. For example, if *Disposed Jobs Dot*'s truth value of Negative is 0.5, then the value of *Resource Availability Probability Threshold* will be lowered by 0.5%. So if the value of *Resource Availability Probability Threshold* is 90%, it should be changed to 90% – 0.5% = 89.5% after applying this rule. The other two rules can also be translated into natural language in the same way.

The third step is rule aggregation. In this step, the fuzzy variable will apply to each rule and the result of each rule is aggregated.

The fourth step is defuzzification. In this step, the aggregated fuzzy results will be translated to a "crisp" numerical value. Let O_{final} denotes the final output; O_{final} can be calculated by the following centroid computation equation:

$$O_{final} = \frac{(T_N * O_1 + T_Z * O_2 + T_P * O_3)}{(T_N + T_Z + T_P)} \qquad \text{(Equation 4.3)}$$

Where T_N, T_Z and T_P denote the *Disposed Jobs Dot*'s truth value of Negative, Zero and Positive respectively and O_1, O_2 and O_3 denote the output of rule 1, 2 and 3, respectively.

Later, let $P_{new}(r)$ denotes the new value of *Resource Availability Probability Threshold*; $P_{new}(r)$ will be adjusted according to the value of final output by the following equation:

$$P_{new}(r) = P_{old}(r) + O_{final} \qquad \text{(Equation 4.4)}$$

Where $P_{old}(r)$ is the old value of *Resource Availability Probability Threshold*.

i. FLP Algorithm

In general, the FLP algorithm is very similar to the FCFSPP algorithm proposed in Section 4.2.2. The distinct difference between them is that FLP uses FL to adjust the *Resource Availability Probability Threshold* of candidate resource(s) according to the trend of overall resources reliability. Therefore, the FLP algorithm can be considered to be a modification of the FCFSPP algorithm.

The basic idea is to replace the fixed setting of *Resource Availability Probability Threshold* with a dynamic and artificial intelligently controlled setting in order to achieve a better balance between *speed* and *reliability*. Here, "a better balance" means achieving a better result than the FCFSPP algorithm in terms of *speed* (but the result should still be lower than FCFS algorithm) and a better result than the FCFS algorithm in terms of *reliability* (but the result should be lower than the FCFSPP algorithm). In FCFSPP, if the value of *Resource Availability Probability Threshold* is high (e.g. 100%), many resources (including some relatively reliable resources) may be considered as unqualified so that FCFSPP will not allocate any job to them and their idle CPU cycles will be wasted. On the other hand, if the value of *Resource Availability Probability Threshold* is low (e.g. 5%), many resources (including some volatile resources) may be considered as qualified so the FCFSPP algorithm will allocate jobs to some volatile resources and many jobs will be failed due to resources' volatility.

The same as the FCFSPP algorithm, the procedure of the FLP algorithm has the separate parts of *Job Submission* and *Job Allocation*. The procedures for these two parts in FLP are exactly the same as it is in the FCFSPP algorithm. Figure 4.12 shows the third part of FLP *Resource Availability Probability Threshold Adjustment*:

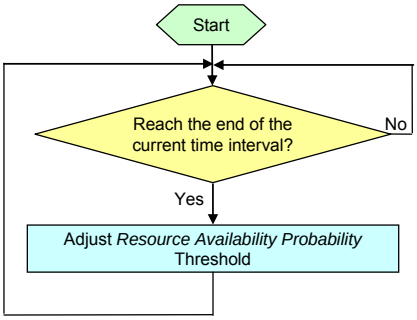


Figure 4-12: *Resource Availability Probability Threshold Adjustment* Procedure

Firstly, the Grid job scheduler checks the time interval. The time interval is called *Resource Availability Probability Threshold Adjustment Interval*, which is a predefined value, such as 1 minute, 10 minutes and so on.

Secondly, if the time interval has been reached, the Grid job scheduler will trigger the adjustment of *Resource Availability Probability Threshold*. If not, the Grid job scheduler will keep on repeating the first step. Therefore if the time interval is set as x minutes (such as 10 minutes), the procedure of *Resource Availability Probability Threshold Adjustment* is triggered every x minutes (such as 10 minutes).

In the proposed the FLP algorithm, the *Resource Availability Probability Threshold Adjustment* uses FL to control the adjustment. The basic idea of this adjustment is to compare the number of disposed jobs in the current time interval to the last time interval, and then decreases/increase the *Resource Availability Probability Threshold* if resources tend to drop fewer/more jobs. Here, *Dispose Jobs Dot* mentioned earlier in this section is used to describe this trend. As a result, the *Resource Availability Probability Threshold* will be lowered as resources tend to dispose of fewer jobs (*Dispose Jobs Dot* is smaller than 0).

4.2.5 Particle Swarm Optimisation plus Predictor (PSOPP) Algorithm

The third job-scheduling algorithm proposed in this research is called Particle Swarm Optimisation plus Predictor (PSOPP) algorithm. Different from FCFSPP and FLP job-scheduling algorithms, the PSOPP algorithm is not based on the FCFS algorithm. Instead, it is mainly based on an AI algorithm - Particle Swarm Optimisation (PSO) [Hu06]. It is a relatively new AI algorithm so very little research work has been done in terms of applying the PSO algorithm to job-scheduling algorithm. Therefore, to check if this new AI algorithm can bring any benefits to job-scheduling algorithm in terms of *speed* and *reliability*, this new PSO based job-scheduling algorithm is proposed. To better understand this proposed algorithm, some general background knowledge about PSO will be given.

i. PSO Introduction

PSO is an artificial intelligence algorithm proposed by James Kennedy and Russell Eberhart [Kenndy95] in 1995. It is based on swarm intelligence and used to find a solution to an optimisation problem in a specific search space.

PSO was motivated by the social behaviour of birds flocking and fish schools. One scenario can be used to explain this kind of behaviour in a Grid computing context. In this example, the aim of PSO is to find the resource the fastest CPU in the Grid. Initially, PSO initialises a group of particles with random values in the search space. This can be understood as each particle selects a resource randomly in the Grid. Later, PSO evaluates the resource that was selected by each particle with a predefined fitness function and records the fitness value of each resource. Here, the fitness value of a resource is determined by the CPU speed of that resource. Next, PSO updates each particle with the two "best" values. The first "best" value is the fastest resource a particular resource has found so far. This value is called pbest. The second "best" value is the fastest resource any particle in the group has found so far. After getting these two values, each particle updates its value with the following two equations:

$$v[i] = v[i] + c_1 r_1(pbest[i] - present[i]) + c_2 r_2(gbest - present[i]) \quad \text{(Equation 4.5)}$$

$$present[i] = present[i] + v[i] \quad \text{(Equation 4.6)}$$

v[i] is the velocity of particle i. The maximum velocity can be restricted by a parameter V_{max}. pbest[i] is best value that particle i has got so far and gbest is the best value that any particle in the group has got so far. present[i] is the present value of particle i. c_1 and c_2 are learning factors and usually $c_1 = c_2 = 2$. r_1 and r_2 are random numbers and their values are between 0 and 1. After an update, the new resources selected by each particle will be evaluated by the fitness function again. This process will continue until achieving the predefined objective(s) or reaching the maximum number of iterations. Pseudo code of the whole procedure of the PSO algorithm is shown as follows:

```
For each particle
  Begin
    Initialises particle' value present[i]
  End
Repeat
   For each particle
      Begin
        Calculate particle's fitness value f(present[i])
        If fitness value f(present[i]) > local best value f(pbest[i])
          Begin
            pbest[i] = present[i]
          End
        If fitness value f(present[i]) > global best value f(gbest)
          Begin
            gbest = present[i]
          End
        Calculate particle's velocity according to [1]
        Update particle's value according to [2]
      End;
Until predefined goal(s) achieved or the maximum number of iterations is
reached
```

ii. PSOPP Algorithm Procedure

In the PSOPP algorithm, the Grid job scheduler tries to allocate a job. However, unlike the FCFSPP and FLP job-scheduling algorithms, all (not just one resource at a time) *available* (not necessarily to *idle*) resources are candidates when the Grid job scheduler tries to make a job allocation decision.

In the PSOPP algorithm, there are P particles and each particle is represented by a unique number p, where $p \in [1, P]$. The total number of a resources at a given instant is represented by R and the number of a particular resource is represented by r, where $r \in [1, R]$. Particle p's current position at iteration t is represented by $P_p(t)$. If $P_p(t) = r$, this means the job is supposed to allocate resource r in the particle p's solution at iteration t. Here, the position represents the solution. For example, assuming there are 10 resources currently available and 5 particles are used to find out the optimal solution. At iteration 5, each particle's position can be represented as follows:

	Position
Particle 1	10
Particle 2	1
Particle 3	7
Particle 4	2
Particle 5	3

Take particle 5 for example, $P_3(5) = 7$ means particle 3's position at the 5th iteration is 7 - the job is allocating the job to resource 7.

$B_p(t)$ represents the best position that particle p has visited after t iterations. For example, assuming the personal best position $B_2(3) = 9$, then it means that particle 2's best solution is to allocate the job to resource 9.

G(t) represents the best position that all particles have visited after t iterations. For example, if G(3) = 6, it means that the best solution all particles have visited after 3 iterations is to allocate the first job to resource 6.

At each iteration, the value of $B_p(t)$ and G(t) will be updated with a fitness function. The fitness value $F_p(t)$ of particle p at iteration t can be represented by the following equation:

$$F_p(t) = x*P(r) + y*\frac{CPU_{current}}{N_{job}} \qquad \text{(Equation 4.7)}$$

Where P(r) means the resource r's *Resource Availability Probability* in the *Prediction Period*, $CPU_{current}$ means the current *CPU Availability* (unit is GHz) of resource r and N_{job} means the total number of jobs on resource r after adding the new job, x and y are two multiplication factors and their default values are 1. Here, the value of *Resource Availability Probability* is also calculated by the TDE prediction method introduced in Section 4.2.1. Assuming the value of P(r) is 70%, $CPU_{current}$ is 1.5(GHz) and N_{job} is 3, x and y are 1, the fitness value $F_p(t) = 0.7 + 1.5 / (2+1) = 1.2$. After calculating the fitness of each particle, the value of $P_p(t)$ will be updated by the highest value particle p has visited so far and G(t) will be updated by the highest value all

particles have visited so far.

After each iteration the position of each particle will be updated by the parameter of velocity. Each particle will have a separate velocity value. The velocity value is within the range of $[-V_{max}, V_{max}]$. After updating by the velocity, the position of a particle should always be between 1 and R. This is to ensure each job will be allocated to a valid resource, as there are total R resources in the Grid. If the position in a certain dimension exceeds the range, its value will be rounded to 1 or R. For each particle, the position will be updated by the following equations:

$$V_p(t+1) = V_p(t) + c_1r_1(B_p(t) - P_p(t)) + c_2r_2(G(t) - P_p(t)) \quad \text{(Equation 4.8)}$$

$$P_p(t+1) = P_p(t) + V_p(t+1) \quad \text{(Equation 4.9)}$$

$V_p(t)$ is the velocity of particle p at iteration t and $V_p(t+1)$ is the velocity of particle p at iteration t+1. c_1 and c_2 are learning factors and $c_1 = c_2 = 2$. r_1 and r_2 are random numbers between 0 and 1. $P_p(t)$ is the position of particle p at t iterations $P_p(t+1)$ is the position of particle p at t+1 iterations. The value of $P_p(t)$ is always within the range [1, R]. If $P_p(t)$ exceeds the range after adding $V_p(t+1)$, it will be rounded to 1 or R depending on which boundary it exceeds.

This is the procedure to use PSO to make allocation decisions for jobs:

1. Create and initialise a P-element array position[P] for recording all particles' positions. Each element in the array is initialised to a random number with a random number generator [6], which means particles are initially randomly scattered.
2. Create and initialise a P-element array velocity[P] for recording all particles' velocities. Each element in the array is initialised to zero, which means particles are initially stationary.
3. Create and initialise a P-element array pbest[P] for recording such particles' personal best positions visited so far. Each element in the array is initialised to zero.
4. Create and initialise a variable gbest for recording the best position that all particles have visited so far. The value of gbest is initialised to zero.
5. For each particle, if $F_p(t+1)$ is larger than pbest[p], then the value of pbest[p] will be updated to be the value of $F_p(t+1)$.
6. For each particle, if $F_p(t+1)$ is larger than gbest, then the value of gbest will be updated to the value of $F_p(t+1)$.
7. Update each particle's velocity with Equation 4.8.
8. Update each particle's position with Equation 4.9.
9. Repeat steps 6 to 9 until the predefined goal(s) are reached or the maximum number of iterations.

The procedure can be represented by the flowchart shown in Figure 4.13:

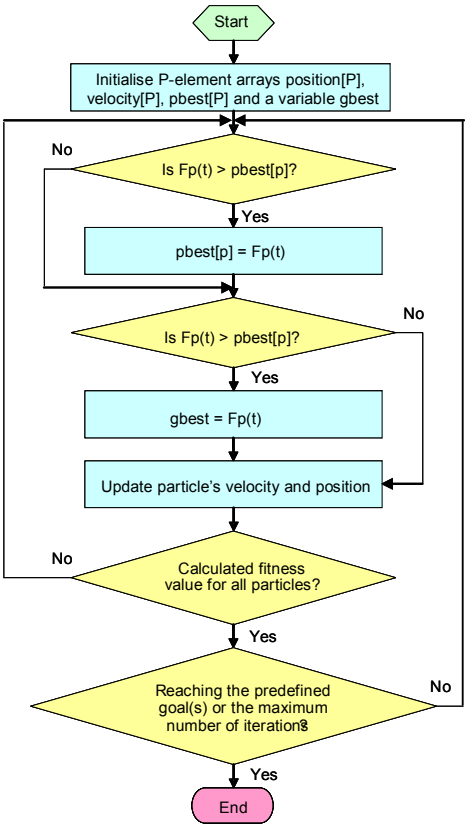


Figure 4-13: PSOPP Algorithm Procedure

4.3 Job Migration

As discussed in Section 4.1.5, in volunteer resource based environments like desktop Grids and volunteer computing environments, resources may come and go at any time. This means a resource may not be able to finish a job before it becomes unavailable. If the job keeps on running on the resource, then the job will fail when the resource becomes unavailable. This is harmful for both the job and the resource. For the job, it cannot finish as expected. For the resource, it is a waste of CPU cycles.

Therefore, it is necessary to take some actions to protect jobs from this kind of failure. In general, checkpointing is a technique to introduce fault tolerance. Basically it consists of storing a snapshot of the current application state and using it to restart the job in the case of failure [Microsoft10]. However, it may not be enough if the job needs to be processed as soon as possible. Imagine that a job runs on a resource and the job checkpoints before the resource becomes unavailable. After that, the resource does not become available for a long time. Then the job has to wait for a long time before it can resume. Therefore, job process time will take longer. As mentioned in Section 4.1, one solution to the volatility problem in the volunteer Grid is to enable the job scheduler to migrate jobs effectively after jobs have already been allocated to resources. Job migration can be considered a type of job rescheduling, this is an approach to assist job-scheduling by adjusting job-scheduling decisions dynamically. As discussed in Section 4.1, job-scheduling algorithms generally have two objectives: *speed* and *reliability*. To

assist job scheduling, the job migration algorithm's objective is to help achieve the *speed* or *reliability*. In a volunteered Grid, *reliability* is the main goal for a job migration algorithm.

They are two main types of job-scheduling algorithms: reactive migration and proactive migration. For each type of job migration algorithm, this research proposes some job migration algorithms to help achieve *reliability*. In this section, all the algorithms proposed in this research will be introduced.

4.3.1 Reactive Job Migration

Reactive migration is a type of job migration initiated by the volunteer resources. From the point of view of the Grid job scheduler, it is a passive approach as resources trigger this type of migration. When a resource is going to leave the Grid or it is no longer allowing any guest jobs from the Grid to run on it, the resource can notify the Grid job scheduler. When the Grid job scheduler receives this kind of notification, it will carry out the job migration.

The procedure of proposed reactive job migration algorithm can be described as follows:

1. When a resource is in any one of the following states, the resource should send out a job migration notification to the job scheduler. These states are:
   - The resource is leaving the Grid soon.
   - The resource is no longer allowing any guest job to run on it.
2. When the Grid job scheduler receives this job migration notification, it will trigger the job migration procedures (details about the whole procedure of job migration are described in Section 3.4.9). In general, the job migration procedure can be summarised in two steps: the first step is using a job-scheduling algorithm to find suitable resources for jobs, one by one. Next, if a *qualified* resource is found for a job, the job migration will be carried out.

Generally, compared with proactive migration (for details about proactive migration, please refer to Section 4.3.2), it has the following advantages:

- It is simpler and more straightforward. This is because in reactive migration the job scheduler needs to wait for notification from the resources' before triggering job migrations.
- It is more effective in terms of utilising resources' idle CPU cycles. This is because the resources' idle CPU cycles will be fully utilised until the resources raises notifications and so potential job failures brought by resource volatility will not occur.

At the same time, it has the following disadvantages when compared with proactive migration:

- It lacks intelligence. This is because it always waits for resources' notification to trigger job migrations rather than observing the performance (such as changes of CPU load or availability patterns) of resources and then makes intelligent job migrations decisions. Therefore if resources leave the Grid without any notification or do not allow guest jobs to run without any precautions, such as when resources crash or there is a network connection failure or the user reclaims the resources for a long time, reactive migrations will not

improve *reliability*.

- It may not have enough time to carry out the job migration before resources become *unavailable*. This is because the resource may leave shortly after sending out the notifications. If the time gap between the resource sending notification and the resource leaving the Grid, is shorter than the time required to migrate all guest jobs on the resource, then some or all of the jobs will not be able to migrate successfully.

4.3.2 Proactive Job Migration

Proactive migration is a type of job migration initiated by the Grid job scheduler. From the point of view of the job scheduler, it is an active approach as this type of migration is triggered by itself. When the job scheduler observes a resource is going to leave the Grid or it is no longer allowing any guest jobs from the Grid to run on it, the resource can trigger the job migration proactively. In general, compared with reactive migration, it has the following advantages:

- It is more intelligent. This is because it observes the performance of resources and then makes proactive job migration decisions rather than waiting for resource notifications to trigger job migrations. Therefore if resources leave the Grid without any notification or do not allow guest jobs to run without precautions, an efficient proactive job migration algorithm will be able to provide help to improve *reliability*.
- An efficient proactive job migration will have more time to carry out the job migration. As that the Grid job scheduler will take proactive migration before getting the migration notifications from resources. Therefore, with an efficient proactive migration algorithm, the job scheduler has more time to carry out job migrations. However there are disadvantages compared to reactive migration:
- It is more complex than reactive migration. This is because the job scheduler needs to observe the performance (such as changes of CPU load or availability patterns) of resources and then make migration decisions. As a resource's performance may change constantly, it is straightforward enough to make migration decisions, but it is not so straightforward to make efficient decisions.
- It may be less effective in terms of utilising resources' idle CPU cycles. This is because the resources' idle CPU cycles will not usually be fully utilised as it is difficult to get the ideal timing to carry out proactive job migrations. It is easy to migrate a job too early or too late.

Two proactive job migration algorithms have been proposed in this research. The first one is based on the resource availability predictor described in Section 4.2.1. The second one is based on an artificial intelligence technique Case Based Reasoning (CBR). In this section, these two algorithms will be described:

i. Periodical Scanning with Predictor Migration Algorithm

The first proactive job migration algorithm proposed in this research is Periodical Scanning with Predictor (PSPP) algorithm. The name of the algorithm shows this algorithm is based on

scanning resources periodically and judging whether job(s) on each resource needs migration or not using the prediction technique described in Section 4.2.1. The objective of this algorithm is to help the job-scheduling algorithm in terms of improving *reliability* – reducing the number of job failures caused by resources' unavailability. The procedure of PSPP algorithm can be described as follows:

First, at the end of a predefined regular interval, the job scheduler picks the first *busy* resource. This interval is called *Migration Prediction Interval*. The reason why the job scheduler only picks a busy resource is that only this kind of resource has the potential to experience job failures.

Second, the job scheduler checks *Resource Availability Probability* in the *Prediction Period*. Here, the *Prediction Period* can be a predefined length of time, such as 5 minutes, 10 minutes and so on.

Third, if the resource is predicted to exit *Available to Grid* state some time during the *Prediction Period* (*Resource Availability Probability* is lower than 100%), the resource will be considered as *unqualified*. A resource marked as *unqualified* means it is considered as becoming unavailable to the Grid soon. Therefore, all the job(s) on the resource will need migration to avoid failure.

Fourth, the job scheduler uses a job-scheduling algorithm to make job allocation decisions for job(s) on the *unqualified* resource one by one.

Fifth, if a job scheduler finds a *qualified* resource, it will migrate the job to the new resource at once. If not, the job will stay on the resource.

After this step, the job scheduler will pick the next *busy* resource and repeat steps one to five until all resources have been scanned. After all resources have been scanned, the job scheduler waits a predefined interval and starts this procedure again. Figure 4.14 shows this procedure:

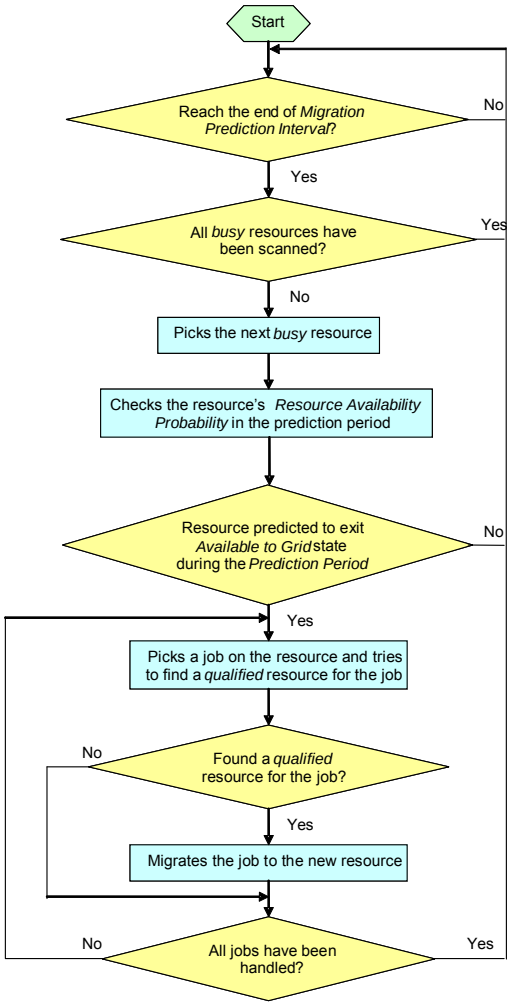


Figure 4-14: PSPP Algorithm Procedure

ii. Case Based Reasoning Migration Algorithm

The second migration algorithm proposed in this research is based on CBR. CBR is a type of AI technique and successfully applied in various areas, such as air conditioners, automobile, digital image processing and so on. To better understand the algorithm, some background knowledge on CBR is given.

Case Based Reasoning Introduction

CBR is a machine-learning approach that has received much attention over the last few years. "It is the process of solving new problems based on the solutions of similar past problems" [Richter06]. Typically, a CBR system consists of a database which records past cases and their solutions. With this database, past similar cases will be generalised and their solutions will be reused (with some modifications if necessary) for new problems.

The CBR approach is based on two main assumptions from the real world. The first one is that similar problems have similar solutions. "Consequently, solutions for similar prior problems are a useful starting point for a new problem" [Leake96]. The second one is that similar problems will recur again and again. Therefore, when a new problem occurs, it is likely to be similar to the old ones.

Similarly to some other problem solving techniques, especially rule induction algorithms [Slade91], a CBR system uses a database to record past cases and their solutions. However, unlike rule induction algorithms, a CBR system makes generalizations of past cases. "A rule-induction algorithm draws its generalizations from a set of training examples before the target problem is even known. This contrasts to CBR, which delays (implicit) generalization of its cases until testing time – a strategy of lazy generalization". [CBRwiki10]

In addition, CBR has the ability to learn from the past whilst rule induction algorithms do not have this ability. In rule induction algorithms, the same rules will be used unless they are modified manually. However, CBR "is an approach to incremental, sustained learning, since a new experience is retained each time a problem has been solved, making it immediately available for future problems." [Aamodt94].

Generally, the CBR cycle is composed of four steps: retrieve, reuse, revise and retain. Figure 4.15 shows this cycle.

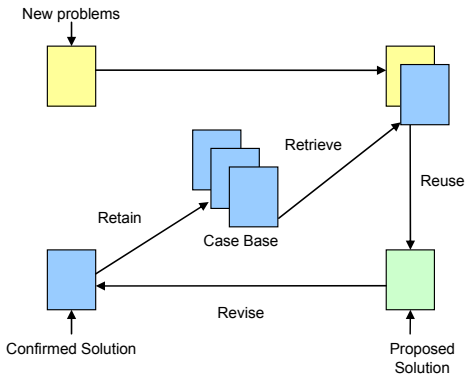


Figure 4-15: The CBR Cycle (Adapted from [Watson94])

An explanation of each step is as follows:

Retrieve: When a new problem occurs, the CBR system retrieves similar cases from the Case Base. Usually, a case consists of a problem, its solution and comments about how the solution was derived. For example, Alice wants to use the oven to roast a small chicken. The problem is that Alice has never cooked such a small chicken before. However, luckily, she has previously cooked some bigger ones. Therefore, Alice tries to remember how long it took, so that she can set the cooking time for the small chicken correctly.

Reuse: After retrieving the similar cases, the CBR system compares the new problem with the past cases, reuses the past solutions and proposes a new solution based on the past ones for the new problem. In Alice's example, Alice remembers that it took 1 hour for the bigger ones so she reuses this solution and tries to set the roasting time as 30 minutes for the small chicken.

Revise: After proposing a new solution, the CBR system tests the new solution. If the new solution does not meet the requirement, the CBR system revises the solution. In Alice's example, Alice checks the chicken after 30 minutes. However, she finds that the chicken is not done yet. Therefore, she revises the solution and adds another 5 minutes for the chicken.

Retain: After solving the problem, the CBR system retains the useful information as a new

case and stores it in the Case Base for further use. In Alice's example, Alice finally gets the chicken roasted. She now retains the experience of cooking a small chicken so that this experience can be reused in future.

There are a number of advantages of using CBR, especially in a domain where there is a lack of strong theory. In such a domain, a rule induction reasoner is not practical. "When the relationship between the case attributes and the solution or outcome is not understood well enough to represent it in rules, or when the ratio of cases that are 'exceptions to the rule' is high, rule based systems become impractical. CBR is especially useful in such situations because it models the exceptions and novel cases." [Morris95]

Procedure of the CBR Migration Algorithm

In general, the CBR is different from any other job-scheduling and job migration algorithm proposed above as the adopted TDE prediction method is not used in this algorithm. Instead of using *Job Execution Availability* to make predictions for job-scheduling or job migration, this algorithm use *CPU Availability* to trigger job migration.

The basic idea of this migration algorithm is to observe the *CPU Availability* of each resource and trigger job migration procedures if the current value of *CPU Availability* is below a threshold (it is called *CBR Migration Threshold*). *CPU Migration Threshold* is a value between 0% and 100%. Here, whether or not to trigger the job migration procedure can be considered as a new problem in CBR and getting the value of *CBR Migration Threshold* can be considered as the step of retrieving the solution to the past cases. As discussed in Section 4.2.2, *CPU Availability* is the current CPU speed which indicating the current number of idle CPU cycles delivered to the Grid per second. In the meanwhile, a term *CPU Availability Percentage* can be used to describe the percentage of maximum CPU speed available at the moment. Let $C_{percent}$, $C_{current}$ and C_{max} denote the *CPU Availability Percentage*, current value of *CPU Availability* and the maximum value of *CPU Availability* separately; the value of *CPU Availability Percentage* can be calculated by the following equation:

$$C_{percent} = C_{current} / C_{max} * 100\% \qquad \text{(Equation 4.10)}$$

For example, if a resource's maximum CPU speed is 1000 CPU cycles per second and now the number of CPU cycles delivered to the Grid (*CPU Availability*) is 700, then $C_{percent}$ = 70% ((700 / 1000) * 100%). The same as the PSPP migration algorithm described above, CBR migration algorithm scans each resource at regular intervals. If the CBR migration algorithm finds a resource's *CPU Availability Percentage* is below the *CBR Migration Threshold*, the resource is considered likely to become unavailable soon. Therefore, CBR migration algorithm will trigger the job migration procedure and try to migrate the job to another resource by using a job-scheduling algorithm. Here, triggering the job migration procedure by using the CBR Migration Threshold can be considered as reusing the solutions of past cases in CBR.

After making job migration decisions, CBR Migration algorithm will revise the proposed solution. This step is carried out just before observing each resource at regular intervals. If a

resource considered to be *unqualified* turns out to be unavailable after triggering the job migration procedure, the solution is considered to be correct and no revision is needed (the value of *CBR Migration Threshold* will not change). However, if a resource considered as *unqualified* turns out to stay in the state of *Available to Grid* until now, then the solution is considered to be incorrect and revision to the CBR *Migration Threshold* is needed.

After the revision, the new value of *CBR Migration Threshold* is considered to be a confirmed solution and it will be retained.

Specifically, the procedure of CBR migration algorithm can be described as follows:

First, at the end of a predefined regular interval, the CBR migration algorithm checks the migration decisions made at the end of last interval. If the total number of incorrect solutions is over the total number of correct solutions, the value of *CBR Migration Threshold* will be increased or reduced by x percent. Here, x percent is called *Adjustment Percentage* and it is a random value and the range of value is (0%, Max%]. After adjustment, the value of *CBR Migration Threshold* should be always within the range of (0%, 100%).

Second, the job scheduler picks the first resource that currently has guest job(s). As in PSPP migration algorithm, this interval is also called *Migration Prediction Interval*. The reason why the job scheduler only picks the resource that currently has guest job(s) is that only this kind of resource has the potential to experience job failures.

Third, the job scheduler checks resource *CPU Availability Percentage*.

Fourth, if the resource's *CPU Availability Percentage* is below the *CBR Migration Threshold*, the resource will be considered as *unqualified*. A resource marked as *unqualified* means it is considered as to becoming *unavailable* to the Grid soon. Therefore, all the job(s) on the resource will need migration to avoid process failures.

Fifth, the job scheduler uses a job-scheduling algorithm to make job allocation decisions for job(s) on the *unqualified* resource one by one.

Sixth, if a job scheduler finds a suitable resource for a job, it will migrate the job to the new resource at once. If not, the job will stay on the resource.

Seventh, the job scheduler will pick the next resource that currently has guest job(s) and repeat steps one to five until all resources have been scanned. After all resources have been scanned, the job scheduler waits for a predefined interval and then starts this procedure again.

Figure 4.16 provides a flowchart to show this procedure.

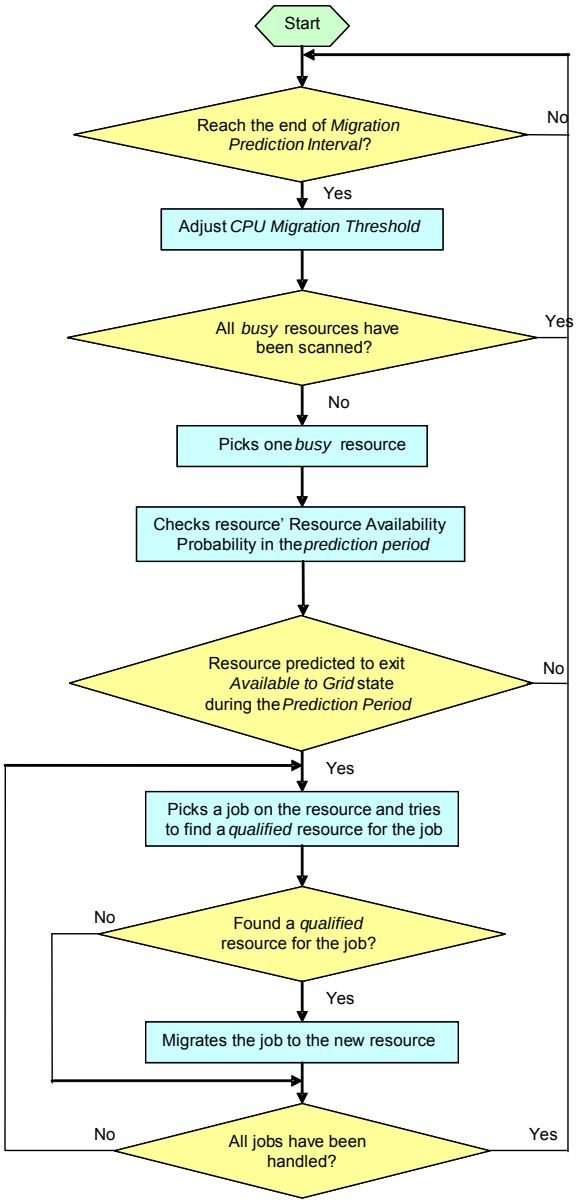


Figure 4-16: Main Procedure of CBR Migration Algorithm

The *CPU Migration Threshold* adjustment procedure is shown in Figure 4.17:

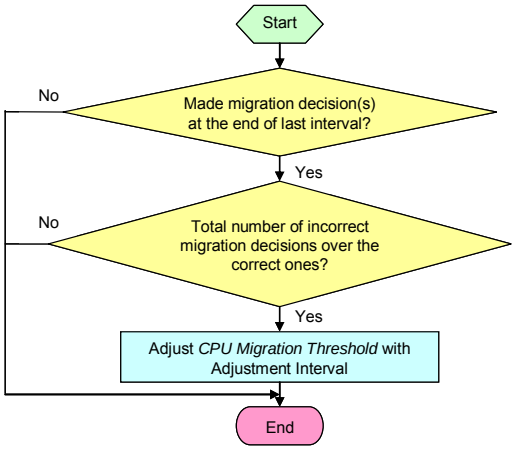


Figure 4-17: Procedure of CPU Migration Threshold Adjustment

Chapter 5 Analysis of Proposed Algorithms

In this chapter, these algorithms will be analysed before providing simulation and evaluation results in Chapter 7.

5.1 Analysis of the Adopted TDE Prediction Method

If the goal of a job-scheduling algorithm is to improve *speed* (such as improving job throughput or to shorten job *Makespan*), *CPU Availability* will be the primary concern as CPU performance affects *speed* most among the three levels of "availability" described in Section 4.2.2. However, if a job-scheduling algorithm is to improve *reliability* (such as reducing the number of failed jobs or the ratio of failed jobs to total jobs), *Job Execution Availability* will be the primary concern as *Job Execution Availability* shows whether jobs will be able to keep running on the resources or not. A running job will be failed if a resource's *Job Execution Availability* changes from true to false. In this thesis, this state change event is called an *Unavailability Event*.

As a resource's *Job Execution Availability* is either true of false, the pattern of a resource's *Job Execution Availability* in a Grid looks like an on-off pattern. When *Job Execution Availability* is true or false, then it is on or off respectively. This is illustrated in Figure 5.1.

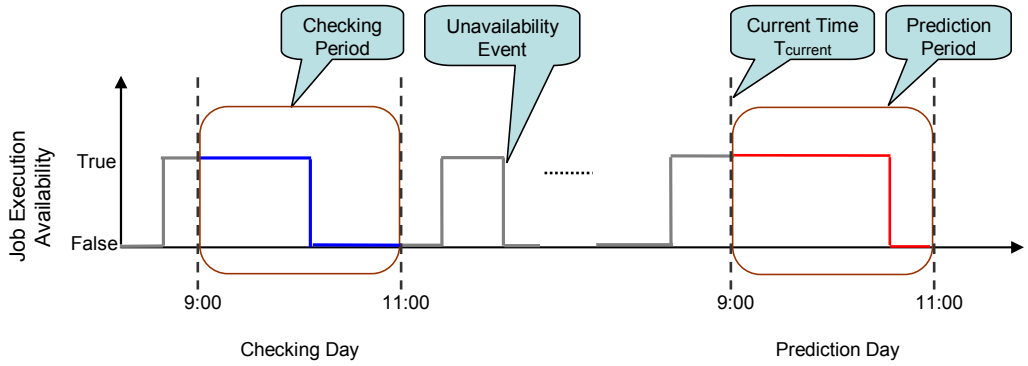


Figure 5-1: Example Resource *Job Execution Availability* Pattern

If *Job Execution Availability* is true (the value of *Job Execution Availability* is above 0 in Figure 5.1), the resource is in the state of *Available to Grid* (described in Section 4.2). If the *Job Execution Availability* is false (the value of *Job Execution Availability* equals 0 in Figure 5.1), the resource is one of the following states: *User Present*, *CPU Threshold Exceeded*, *Job Eviction* or *Graceful Shutdown* or *Unavailable*. The equation for calculating *Resource Availability Probability* described in Section 4.2.1 (Equation 4.1) shows that the number of times of *Available to Grid* to *Available to Grid* is the numerator of the equation. As a result, the adopted TDE prediction method is in fact checking a resource's *Job Execution Availability* history in the *Checking Period* in the *Checking Day* and then calculating the resource's *Resource Availability Probability* in the *Prediction Period* in the *Prediction Day*.

If the *Job Execution Availability* pattern in the *Checking Period* is exactly the same as *Job Execution Availability* pattern in the *Prediction Period*, the result of *Resource Availability*

Probability will be completely accurate. Here, "accurate" means the result of *Resource Availability Probability* in the *Checking Period* is exactly the same as the result of *Resource Availability Probability* in the *Prediction Period*. However, if the *Job Execution Availability* pattern in the *Checking Period* is not exactly the same as the *Job Execution Availability* pattern in the *Prediction Period*, it is not straightforward to judge whether the result of *Resource Availability Probability* will be accurate or not. In addition, the accuracy of prediction is influenced by the values of some important parameters used in the TDE prediction method, such as *Number of Checking Days* and *Multiply Factor*. The FCFSPP algorithm is based on the TDE prediction method, so more detailed analysis about the TDE prediction method and the influences of prediction results on the FCFSPP algorithm will be introduced in Section 5.2.

5.2 Analysis of the FCFSPP Algorithm

5.2.1 Features of the FCFSPP Algorithm

As described in Section 4.2.3, the FCFSPP algorithm is based on FCFS with the added TDE predictor (the predictor is described in Section 4.2.1 and analysed in Section 5.1). Therefore, this algorithm will have three distinct features:

1. Only *idle* resources are possible candidates when the FCFSPP algorithm tries to make job allocation decisions. As discussed in Section 4.2.2, an *idle* resource is a resource that is not only *available* to the Grid but also not *busy* (does not have any guest job from the Grid) at the moment. Therefore, a *busy* (*available* but not *idle*) resource will not be considered as a candidate. This is because one resource having more than one job at a time is usually difficult to provide benefits in terms of *speed*. In addition, if resource reliability is unknown, one resource having more than one job at a time is difficult to provide benefits in terms of *reliability*.

   In terms of *speed*, as the number of CPU cycles provided by a resource is fixed and as all guest jobs are assumed to have the same priority, all guest jobs on a resource have to share the CPU cycles at the same time. Therefore, if there is only one guest job on a resource at a time, the job's *Makespan* will be the shortest. If there is more than one guest job on a resource at a time, each job's *Makespan* will become longer and job throughput will be influenced as well.

   If resource availability is unknown, a resource may become *unavailable* at any time. When a resource becomes *unavailable* to the Grid, all guest jobs running on the resource will be lost. Therefore, if there is only one guest job on a resource at a time, only one job will be lost. Though the FCFSPP algorithm uses TDE prediction, the resources may still become *unavailable* at any time as the resource owners control them. Therefore, only *idle* resources are possible to become a candidate when the FCFSPP algorithm tries to make job allocation decisions.
2. There is only one candidate resource at a time when the FCFSPP algorithm tries to make a

job allocation decision. The FCFSPP algorithm always tries to allocate a new job to the next *idle* resource. This is a feature inherited from FCFS algorithm and this is different from some other job-scheduling algorithms. For example, with Matchmaker[Thain05] and MTTF [Ren07], all *available* (not necessarily to be *idle*) resources will be candidates and the job-scheduling algorithms will try to find out the "best" resource from all candidates (let's call this approach as *Finding the Best*). Here, "best" can have different meanings. It could be the resource that has the highest CPU speed or the resource that has the highest number of completed jobs, and so forth.

Unlike these algorithms, only one resource is the candidate in algorithms like FCFS and FCFSPP, in which the algorithm always has one candidate at a time and the algorithm tries to find out whether the candidate resource is *qualified* or not. This approach will be referred to as *Checking if Qualified*. Generally speaking, *Checking if Qualified* is quicker in terms of making job allocation decisions when comparing with *Finding the Best* approach. Imagine a scenario that hundreds of thousands of resources are idle in the Grid. *Finding the Best* approach may be very time consuming while *Checking if Qualified* is not. Furthermore, as many resources are volatile in a volunteered resource based environment, the "best" one found by *Finding the Best* approach may not still be the "best" one when the job allocation decision is made. However, using more powerful resources can reduce this difference.

In a type of scenario, *Finding the Best* will become the same as FCFS. This is the scenario in which *Finding the Best* looks for *idle* resources but only one resource is *idle* at a time. This scenario is common when the number of jobs is much higher than the number of resources. One resource idle occurs when a resource has just finished a job. In the meantime other resources are still busy with jobs. Therefore, it will be common that only one resource is idle at a time. In such a case, the approach of *Finding the Best* will have to allocate the new job to the next and the only idle resource. As a result, *Finding the Best* cannot provide any more benefits than just using a simple FCFS algorithm. This scenario is not uncommon in practical world. For example, for a High Throughput Computing environment [HTC10], the number of jobs is far more than the number of resources and the one important objective of this environment is to get as high as possible job throughput. In addition, in some volunteered resources based environments, such as BOINC [Anderson05], all resources will be *busy* all the time.

3. The FCFSPP algorithm will only allocate a new job to the next *idle* resource if the resource is considered as *qualified*. In the FCFSPP algorithm, a *qualified* resource means the resource's *Resource Availability Probability* is over the *Resource Availability Probability Threshold*. This is different from the FCFS algorithm, in which no threshold is defined for candidate resources. This measure in the FCFSPP algorithm is for ensuring *reliability* as the primary objective of the FCFSPP algorithm is to ensure jobs being processed successfully as much as possible. Therefore, the FCFSPP algorithm will only allocate the new job to the

candidate resource if it is *qualified*. As discussed in Chapter 1, there is a trade-off between *reliability* and *speed*. In FCFSPP, if no resource is considered as *qualified*, the Grid Job Scheduler will wait halt for a while and then start to find a *qualified* resource again. This measure may affect the time to complete the job as a result. In addition, some idle CPU cycles on the *idle* but *unqualified* candidate resource(s) will be wasted.

5.2.2 Influences on the FCFSPP Algorithm

In addition to the features described above, the FCFSPP algorithm is also influenced by some factors and parameters. Therefore, in this subsection, important influences will be considered.

i. System Case

If the *Checking Period* and the *Prediction Period* in Figure 5.1 are extracted and combined together, then what they look like is shown in Figure 5.2.

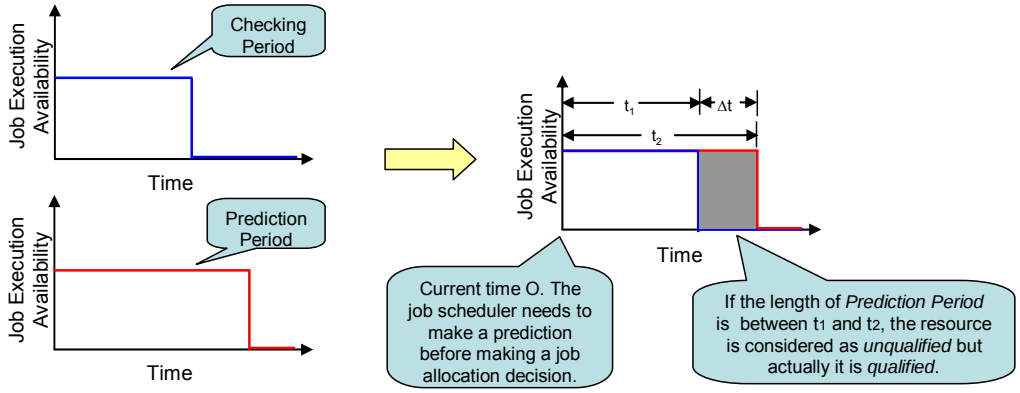


Figure 5-2: Extracted and Combined *Job Execution Availability* Pattern (Type 1)

In Figure 5.2, diagrams of *Job Execution Availability* in both the *Checking Period* and *Prediction Period* on the left hand side are put in the same diagram on the right hand side. t_1 is the time interval between the start of checking time (the same is current time $T_{current}$) and the time that the first *Unavailability Event* occurs in the *Checking Day*. In the example shown in Figure 5.2, t_1 is shorter than the length of *Checking Period* (or Prediction Period). However, note t_1 may also be longer or shorter than the *Checking Period* and this depends on the length of *Checking Period*, which is further depends on the length of *Job Execution Time* and *Multiply Factor*. t_2 is the time interval between the start of checking time and the time that the first *Unavailability Event* occurs in the *Prediction Day*. The same as t_1, t_2 may also be longer or shorter than the *Checking Period* and it also depends on the length of *Checking Period*, which is further depends on the length of *Job Execution Time* and *Multiply Factor*. Δt is the time difference between the length of t_1 and t_2.

In Figure 5.2, assume a resource is available for t_1 in the *Checking Period*, available for t_2 in the *Prediction Period* and Δt is the difference between the length of t_1 and t_2. The current time is $T_{current}$ and the first job in the job queue lasts for time L. So the Grid job scheduler needs to use TDE prediction to make a job allocation decision now and the prediction method will check resource *Job Execution Availability* history in the *Checking Period* (assume the length of *Job*

Execution Time is L and *Multiply Factor* is 1 and the length of *Checking Period* is L*1=L) to predict the *Resource Availability Probability* of the resource in the *Prediction Period* (assume the *Prediction Period*'s length is also L). In addition, in the following analysis of Section 5.2.2, if there is no further notice, the parameter *Number of Checking Days* is assumed to be 1 and *Resource Availability Probability Threshold T* is assumed to be 100%.

Based on these assumptions and with the length of *Checking Period* L (or *Prediction Period*), several cases would occur in the system:

Case 1: If L is shorter than t_1, then the resource is always available in the *Checking Period*. As a result, the *Resource Availability Probability* of the resource will be 100% and it is not lower than the *Resource Availability Probability Threshold*. So the resource will be considered as *qualified* and the job will be allocated to the resource. As resource is still available at time $T_{current}$ + L, the prediction result will turn out to be correct and the job will processed successfully by the resource.

Case 2: If L is longer than t_1 but shorter than t_2, then the resource is not always available in the *Checking Period*, so the *Resource Availability Probability* of the resource will be lower than 100%. It is lower than the *Resource Availability Probability Threshold* so the resource is considered as *unqualified* and the job will not be allocated to the resource. However, the resource turns out to be still available at time $T_{current}$ + L, then the prediction is incorrect and resource's idle CPU cycles will be wasted until another job being allocated to the resource.

Case 3: If L is longer than t_2, then the resource is not always available in the *Checking Period*, so the *Resource Availability Probability* of the resource will be lower than 100%. It is lower than the *Resource Availability Probability Threshold* so the resource is considered as *unqualified* and the job will not be allocated to the resource. The resource turns out to become unavailable before time $T_{current}$ + L, so the prediction is correct. Some idle CPU cycles of the resource will be wasted until another job being allocated to the resource. However, more importantly, a job failure is successfully avoided.

If t_2 is shorter than t_1, then the situation will be different. Figure 5.3 illustrates this situation.

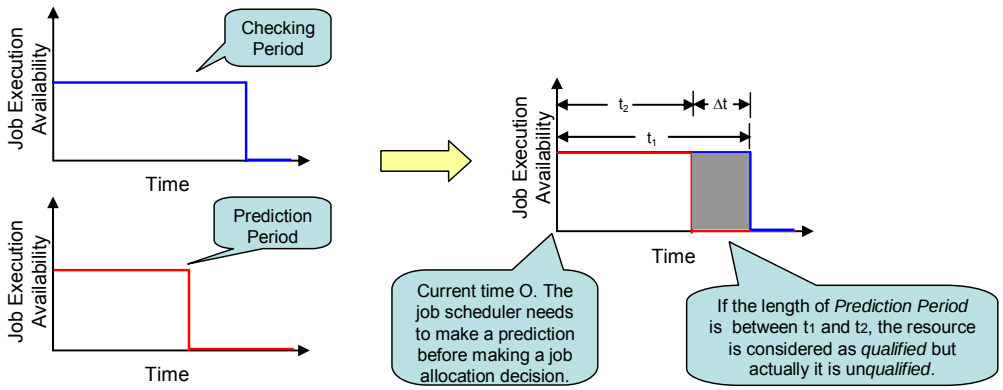


Figure 5-3: Extracted and Combined *Job Execution Availability* Pattern (Type 2)

Case 4: If L is longer than t_2 but shorter than t_1, resource is always available in the *Checking Period*, so the *Resource Availability Probability* of the resource will be 100% and it is not lower

than the *Resource Availability Probability Threshold*. So the resource will be considered as *qualified* and the job will be allocated to the resource. However, the resource turns out to be become unavailable before time $T_{current}$ + L, then the prediction is incorrect and job will be failed to be processed.

Case 5: If L is longer than t_1, then the resource is not always available in the *Checking Period*, so the *Resource Availability Probability* of the resource will be lower than 100%. So the resource will be considered as un*qualified* and the job will not be allocated to the resource. The resource turns out to become unavailable before time $T_{current}$ + L, so the prediction is correct. Resource's CPU cycles will be wasted until another job being allocated to the resource, but a job failure is successfully avoided.

Figure 5.4 shows an example of *Job Execution Availability* in the above 5 cases.

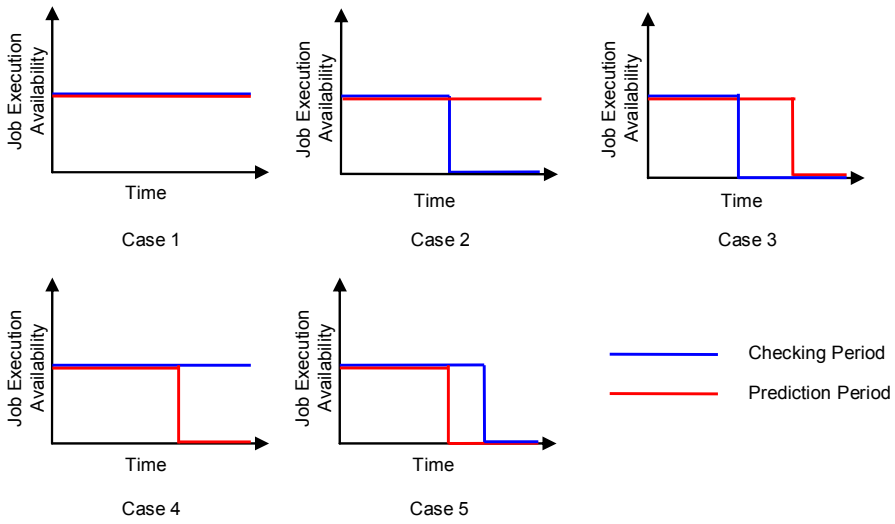


Figure 5-4: Examples of *Job Execution Availability* in 6 Cases

It should be noted that in all of these cases, the *Checking Period* and the *Prediction Period* may change between *available* and *unavailable* a number of times. However, as one *Unavailability Event* in the *Checking Period* will result in a *Resource Availability Probability* below 100%. As one *Unavailability Event* in the *Prediction Period* will cause the job to fail to complete, only the first *Unavailability Event* in both *Checking Period* and *Prediction Period* is considered.

According to the above analysis, the TDE prediction will affect the job-scheduling algorithm's performance in both terms of *speed* and *reliability*. Next, the influences brought by the TDE prediction technique will be discussed by comparing the performance of the FCFS and FCFSPP algorithms in different cases:

In case 1, both the FCFS and the FCFSPP algorithm will allocate the job to the resource and the resource will complete the job successfully. Therefore, both the FCFS and the FCFSPP algorithm perform the same in terms of *speed* and *reliability* in these two cases.

In case 2, the FCFS algorithm will allocate the job to the resource and the resource will complete the job successfully. The FCFSPP algorithm will not allocate the job to the resource and

the idle CPU cycles of the resource will be wasted. Therefore, in terms of *speed*, FCFS is better than the FCFSPP algorithm as the job throughput in FCFS algorithm is higher than the job throughput in the FCFSPP algorithm. In terms of *reliability*, two types of algorithm perform the same and the job will not be failed.

In case 3 and 5, FCFS algorithm will allocate the job to the resource and the resource will not complete the job successfully. The FCFSPP algorithm will not allocate the job to the resource and a potential job failure is avoided. Therefore, in terms of *speed*, both types of algorithm perform the same and the job will not complete. In terms of *reliability*, the FCFSPP algorithm is better than the FCFS algorithm as the FCFSPP algorithm does not allocate any job to the resource, and so the potential job failure is avoided.

In case 4, both FCFS and the FCFSPP algorithm will allocate the job to the resource but the resource will not complete the job successfully. Therefore, both FCFS and the FCFSPP algorithm perform the same in terms of *speed* and *reliability* in these two cases.

Assume job 1's length L_1 is shorter than both t_1 and t_2, so it is facing case 1. After processing this job, the Grid job scheduler starts to handle the next job in the job queue. Therefore, if the job 2's length L is shorter than t_1 and t_2, then it faces 1 again. If L_2 is longer than t_1 but shorter than t_2, then it faces 2. If L_2 is longer than t_1 and t_2, then it faces 3. Figure 5.5 illustrates all the possible system cases transitions.

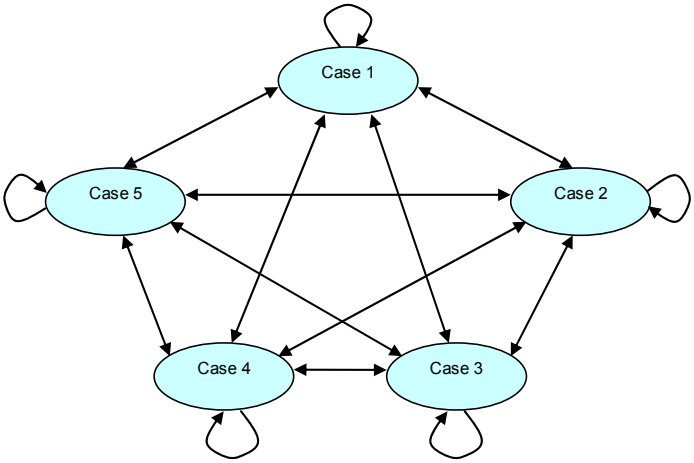


Figure 5-5: Transitions among all cases

If the system transits among all the cases between 1 and 5 uniformly, the FCFSPP algorithm is supposed to be better than FCFS algorithm in 2/5 = 40% cases in terms of *reliability* as the FCFSPP algorithm performs better than FCFS algorithm in case 3 and 5. In terms of *speed*, the FCFSPP algorithm performs worse than FCFS algorithm in case 2. Therefore, the FCFSPP algorithm is worse than FCFS algorithm in 1/5 = 20% cases.

If the system only transits within case 3 or case 5, then the FCFSPP algorithm is supposed to provide the best performance in terms of *reliability*. In this situation, all jobs will be failed if FCFS algorithm is used and all of these can be avoided if the FCFSPP algorithm is used.

However, note in such an extreme case, though the FCFSPP algorithm can avoid job failures, no job can be processed successfully as the resource(s) are so volatile.

If case 3 or 5 is included in the system transitions, then the FCFSPP algorithm will be better than FCFS algorithm in terms of *reliability*. In this situation, all jobs will be failed if FCFS algorithm is used and all of these failures can be avoided if the FCFSPP algorithm is used.

If the system only transits within case 2, then the FCFSPP algorithm provides worse performance in terms of *speed*. In this situation, all jobs will be allocated to the resource(s) and processed successfully if FCFS algorithm is used while no job will be allocated to the resource(s) if the FCFSPP algorithm is used.

Overall, In terms of *speed*, FCFS is better than FCFSPP in case 2 and both algorithms have the same results in other cases. In terms of *reliability*, the FCFSPP algorithm is better than FCFS in case 3 and 5 and both algorithms have the same results in other cases. Therefore, FCFS will not be worse than the FCFSPP algorithm in terms of *speed* while the FCFSPP algorithm will not be worse than FCFS algorithm in terms of *reliability*.

ii. The Influence of Δt between *Checking Day* and *Prediction Day*

According to the definition above, Δt is the length difference between t_1 and t_2. In general, if the size of Δt changes, the accuracy of the adopted prediction method and the performance of the proposed job-scheduling algorithm will be influenced directly.

Suppose t_1 is much smaller than t_2 (the situation shown in Figure 5.2), then, if Δt becomes larger (t_1 becomes smaller or t_2 become larger), the length of *Checking Period* is more likely to be longer than t_1 while shorter than t_2. Therefore, the occurrence probability of case 2 will increase. According to the analysis above, the FCFSPP algorithm is not as good as FCFS algorithm in terms of *speed* in such a case while both algorithms have the same results in terms of *reliability*.

On the other hand, if Δt becomes smaller (t_1 becomes larger or t_2 become smaller), the length of *Checking Period* is more likely to be shorter than t_1 or longer than t_2. If the length of *Checking Period* is shorter than t_1, case 1 will occur and both FCFS and the FCFSPP algorithm will have the same results in both terms of *speed* and *reliability*. If the length of *Checking Period* is longer than t_2, case 3 will occur and both FCFS and the FCFSPP algorithm will have the same results in terms of *speed* while the FCFSPP algorithm will have better results than FCFS algorithm in terms of *reliability*.

Suppose t_1 is much larger than t_2 (the situation shown in Figure 5.3), then if Δt becomes larger, the length of *Checking Period* is more likely to longer than t_2 while shorter than t_1. Therefore, the occurrence probability of case 5 will increase. According to the analysis above, jobs will fail to be processed in both FCFS and the FCFSPP algorithm if case 4 occurs, so the more times the system entering case 4, the worse performance in terms of *reliability* both algorithms will have.

On the other hand, if Δt becomes smaller (t_1 becomes smaller or t_2 become larger), the length

of *Checking Period* is more likely to be shorter than t_2 or longer than t_1. If the length of *Checking Period* is shorter than t_2, case 1 will occur and both FCFS and the FCFSPP algorithm will have the same results in both terms of *speed* and *reliability*. If the length of *Checking Period* is longer than t_1, case 5 will occur and both FCFS and the FCFSPP algorithm will have the same results in terms of *speed* while the FCFSPP algorithm will have better results than FCFS algorithm in terms of *reliability*.

iii. Influence of Similarity of *Job Execution Availability* between *Checking Period* and *Prediction Period*

The adopted prediction method checks a resource's *Job Execution Availability* history in the *Checking Period* and predicts the resource's *Job Execution Availability* in the *Prediction Period* (e.g. a period of time in the next few days/hours). So if the prediction results will be completely accurate when *Job Execution Availability* in *Checking Period* and *Prediction Period* are exactly the same.

Here, "exactly the same" could be presented as two cases. The first case is resource's *Job Execution Availability* does not change throughout the *Checking Period* and *Prediction Period*. Figure 5.6 shows two examples of this case.

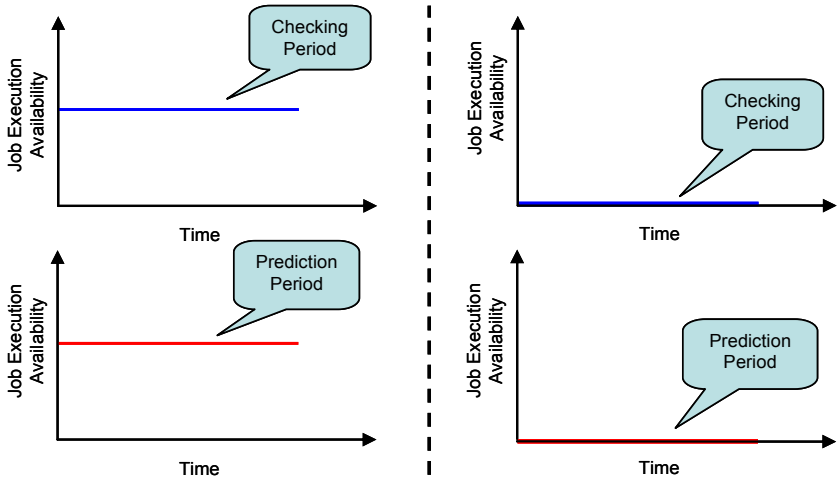


Figure 5-6: Two Examples of "Exactly the Same" Checking and Prediction Period

The second case is resource's *Job Execution Availability* changes in the *Checking Period* and *Prediction Period*, but the change pattern in the *Checking Period* and *Prediction Period* are exactly the same. Figure 5.7 shows an example of this case.

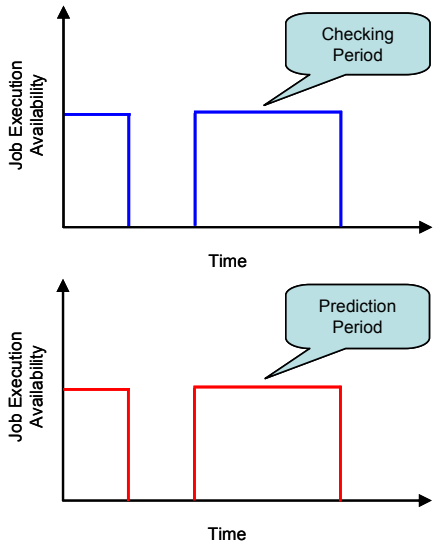


Figure 5-7: Third Example of "Exactly the Same" Checking and Prediction Period

In statistics, the Pearson Product-Moment Correlation Coefficient (PMCC) "is a measure of the correlation (linear dependence) between two variables *X* and *Y*, giving a value between -1 and +1 inclusive" [Rodgers88][Stephen89]. According to [PMCCwiki], the PMCC result ρ can be calculated by the following equations:

$$\rho = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{X_i-\mu_X}{\sigma_X}\right)\left(\frac{Y_i-\mu_Y}{\sigma_Y}\right) \qquad \text{(Equation 5.1)}$$

where $\frac{X_i-\mu_X}{\sigma_X}$, μ_X and σ_X are the standard score, population mean, and population standard deviation. The standard deviation is calculated as:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(X_i-\overline{X})^2} \qquad \text{(Equation 5.2)}$$

where $\overline{x}$ is the arithmetic mean of the values x_i, defined as:

$$\overline{X} = \frac{X_1+X_2+\ldots+X_N}{N} = \frac{1}{N}\sum_{i=1}^{n} X_i \qquad \text{(Equation 5.3)}$$

PMCC can be used to describe the similarity between two variables/time-series in some cases. In the two cases shown in Figure 5.6, PMCC of the *Checking Period* and *Prediction Period* is not calculable as each series' standard deviation σ is 0. However, in this case shown in Figure 5.7, the PMCC of the *Checking Period* and *Prediction Period* is calculable and the PMCC ρ is 1 in such case.

If a resource's *Job Execution Availability* behaves like the first and the second examples, then the prediction results will be perfect. However, in the mean while, prediction will also become unnecessary as *Job Execution Availability* does not change at all. Therefore, TDE prediction is useful in the cases that *Job Execution Availability* changes (cases like the third case) and similarity (represented by PMCC ρ) between *Checking Period* and *Prediction Period* is important factor that influences prediction results in such cases.

As resources' *Job Execution Availability* patterns is on-off pattern (it is either true or false), PMCC between *Checking Period* and *Prediction Period* is mainly influenced by state changes between on and off. Let's take an example to see how the state changes affect the PMCC and prediction results.

In Figure 5.2 and 5.3, if both t_1 and t_2 are shorter than the length of *Checking Period* and the value of Δt equals 0, then the *Checking Period* and the *Prediction Period* are exactly the same so that ρ will be 1. In such a case, the prediction results will be always accurate and the FCFSPP algorithm will perform the best in both terms of *speed* (it has similar results of job throughput as FCFS algorithm) and *reliability* (it can filter out the unreliable resources correctly and avoid allocating jobs to these unreliable resources so few jobs will be failed).

If t_1 and t_2 are shorter than the length of *Checking Period* but the absolute value of Δt becomes larger (the value of Δt far lower or far higher than 0), then the *Checking Period* and the *Prediction Period* becomes less similar so that ρ will become smaller accordingly. If the absolute value of Δt is close to the maximum value, the value of ρ will be close to -1. According to analysis above, if the absolute value Δt becomes larger, the prediction results will more likely to be inaccurate and the FCFSPP algorithm tends to perform worse in terms of either *speed* or *reliability*. Therefore, if ρ becomes smaller, prediction results will more likely to be inaccurate and the FCFSPP algorithm will tend to perform worse in terms of either *speed* or *reliability* as well.

iv. Influence of *Number of Checking Days N*

If *Number of Checking Days* (abbreviated as *N*) becomes larger (which mean the prediction method checks more days for prediction) and if the *Resource Availability Probability Threshold* is 100%, then the day with shortest *Availability Interval* (the time between two consecutive periods of unavailability) in *Checking Period* will be the most important day if each day's weight is equal. Figure 5.8 shows the case when *Number of Checking Days* is 2 (abbreviated as *N*=2):

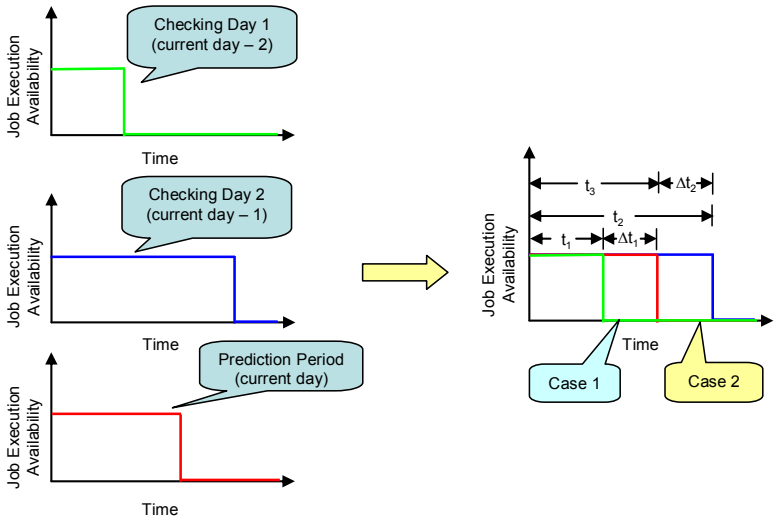


Figure 5-8: Checking Past 2 Days for Prediction

In this example, the prediction method checks past two days. The resource has to be available throughout the whole *Checking Period* in both days to get the result of 100% in terms of *Resource Availability Probability*. A day with a shorter *Availability Interval* will be more important. In this example, Checking Day 1 has shorter *Availability Interval*, so the result of *Resource Availability Probability* will be 100% if the length of *Checking Period* is shorter than t_1 and the resource will be considered as a *qualified* resource. If the length of *Checking Period* is longer than t_1 (no matter the length of *Checking Period* is longer than t_2 or not), the result of *Resource Availability Probability* will be lower than 100% and the resource will be considered as an *unqualified* resource.

In general, the job allocation decisions tend to be more *conservative* (tend to NOT allocate the job to the resource) when *N* becomes larger. When *N* becomes larger, more *Checking Days* will be checked and it is more likely to find a *Checking Day* with shorter *Availability Interval*. Therefore, if higher *reliability* via the approach of conservative job allocation decisions is desired, then a larger *N* is required. On the other hand, though larger *N* can help for improving *reliability*, the performance of the FCFSPP algorithm in term of *speed* will tend to decrease at the same time.

If *N* is larger than 1, if the FCFSPP algorithm tries to make a job allocation, it has to face extra cases than it has to face when *N* equals 1. If *N* equals 2, the FCFSPP algorithm has to face four extra cases (shown in Figure 5.8). However, the differences between *N*=1 and *N*=2 will only occur in cases 1 and 2 shown in Figure 5.8:

Case 1: If the job length L is longer than t_1 but shorter than t_3, the FCFSPP algorithm will face case 1. In such a case, the FCFSPP algorithm will allocate the job to the resource if *N*=1 while it will NOT allocate the job to the resource if *N*=2. *N*=1 is better as it allocates the job to the resource and the job will be able to complete before the resource becomes *unavailable*.

Case 2: If the job length L is longer than t_3 but shorter than t_2, the FCFSPP algorithm will face case 3. As for case 2, in such a case, the FCFSPP algorithm will allocate the job to the resource if *N*=1 while it will NOT allocate the job to the resource if *N*=2. Different from case 1, *N*=2 is better in such a case as it does NOT allocate the job to the resource and avoid a job failure (the job is not be able to be completed before the resource becomes *unavailable*).

v. Influence of *Resource Availability Probability Threshold T*

If the value of *Resource Availability Probability Threshold T* (abbreviated as *T*) becomes smaller (approaching to 0%), the FCFSPP algorithm becomes looser and achieves the loosest situation (becomes a completely non-prediction based FCFS algorithm) when *T* is 0%. On the other hand, if the value of *T* becomes larger (approaching to 100%), it becomes more *conservative* and achieves the most *conservative* situation when *T* is 100%.

Therefore, if *T* is between 0% and 100%, in both terms of *speed* and *reliability*, the result should be between the results achieved when *T* is 0% and 100%.

vi. Influence of Job Arrival Process

Similarly to the influence of the job sorting algorithm, each job's "destiny" might be different if jobs arrived with different process and the FCFSPP algorithm's performance in terms of *speed* might be affected as well.

For environments like High Throughput Computing environment or some volunteered resources based environments (e.g. BOINC), the number of jobs is far more than the number of resources. Therefore, if a Grid system is at the initial stage (in which many resources are *available*), the job arrival process will influence the result of *speed* at the initial stage for a period of time as most (or even all) resources are available. If the job arrival interval is large/small, the initial stage will tend to be long/short and the job throughput will tend to be low/high in the initial stage. However, once the Grid system enters a steady state (in which all resources are kept *busy* all the time), the result of *speed* is not influenced by the job arrival process any longer. In steady state, all new jobs will have to wait in the job queue first. In such a case, job throughput or job *Makespan* depends on the resources' CPU speed.

For environments in which the number of jobs is lower than the number of resources, the job arrival process will influence the result of *speed* at both the initial and steady stage. If few/many jobs arrive in a period of time, job throughput will tend to be low/high. However, the job arrival process will not affect each job's *Makespan* in the FCFSPP algorithm as one resource will always process one job at a time.

vii. Influence of Resource Availability Interval

The resource *Total Availability Interval* means the time during which the resource's *Job Execution Availability* is true in a give period of time and the resource *Total Unavailability Interval* means the time during which the resource's *Job Execution Availability* is false in a given period of time. The resource *Average Availability Interval* means the average length of the resource's *Availability Intervals* and the resource *Average Unavailability Interval* means the average length of the resource's *Unavailability Intervals.*

A given period of time is composed of *Total Availability Interval* and *Total Unavailability Interval*. Therefore, if *Total Availability Interval* is longer, job throughput in both FCFS and the FCFSPP algorithm will tend to be higher (more jobs will be finished). However, the *Average Availability Interval* will influence the job throughput if the *Total Availability Interval* is fixed.

For a given resource with a given period of time and a given *Total Availability Interval*, if *Average Availability Interval* is small, then the resource tends to be very unreliable (become *unavailable* frequently). In such a case, the job throughput tends to be lower than the case in which the *Average Availability Interval* is small. On the other hand, as the resource becomes *unavailable* frequently, many jobs will fail with the FCFS algorithm. This is not true for the FCFSPP algorithm, if TDE prediction can provide accurate prediction results. Potential job failures will be avoided, as the FCFSPP algorithm will not allocate a job to the resource if the

resource is predicted to become *unavailable* before the job is finished.

For the same resource with the same given period of time and the same given *Total Availability Interval*, if *Average Availability Interval* is large, then the resource tends to be very reliable (become *unavailable* infrequently). In such a case, the job throughput tends to be higher than the case in which the *Average Availability Interval* is small. On the other hand, as the resource becomes *unavailable* infrequently, few jobs will be failed as a result in FCFS algorithm. For the FCFSPP algorithm, if TDE prediction can provide accurate prediction results, the job throughput result in the FCFSPP algorithm will be the same (or very closed) to the job throughput result obtained by FCFS algorithm as many jobs will be allocated to the resource in the FCFSPP algorithm. However, if TDE prediction cannot provide accurate results, the job throughput result in the FCFSPP algorithm will be fewer than the job throughput obtained by FCFS algorithm as few jobs will be allocated to the resource as the FCFSPP algorithm will consider the resource as unreliable resource (but in fact the resource is very reliable).

viii. Influence of *Average Job Size* and Resource *Average Availability Interval*

Here, a job's *Job Size* means the job's *Job Execution Time* and *Average Job Size* means the average *Job Execution Time*. In general, if the *Average Job Size* is small (compared with the resources' *Average Availability Interval*), then the system will tend to enter case 1 and 4 frequently. In such a case, the FCFSPP algorithm performs more or less the same as the FCFS algorithm both in terms of *speed* (represented by the number of processed jobs) and *reliability* (represented by the number of failed jobs).

If the *Average Job Size* is medium (also compared with resources' *Average Availability Interval*), then the system will tend to enter case 2 and 4 frequently. In such a case, the FCFSPP algorithm performs more or less the same as the FCFS algorithm in terms of *reliability* while it tends to perform worse than FCFS algorithm in terms of *speed*.

If the *Average Job Size* is large (also compared with resources' *Average Availability Interval*), then the system will tend to enter case 3 and 5 frequently. In such a case, the FCFSPP algorithm performs more or less the same as the FCFS algorithm in terms of *speed* while it tends to perform better than FCFS algorithm in terms of *reliability*.

In contrast to job size, resource's *Average Availability Interval* has the opposite influence to the FCFSPP algorithm.

If resources' *Average Availability Interval* is large (compared with *Average Job Size*), the system will tend to enter case 1 and 4 frequently. In such a case, the FCFSPP algorithm performs more or less the same as FCFS algorithm in both terms of *speed* and *reliability*.

If resources' *Average Availability Interval* is medium (also compared with *Average Job Size*), then the system will tend to enter case 2 and 5 frequently. In such a case, the FCFSPP algorithm performs more or less the same as FCFS algorithm in terms of reliability while it tends to perform worse than FCFS algorithm in terms of *speed*.

If the resources' *Average Availability Interval* is large (also compared with *Average Job Size*), then the system will tend to enter case 3 and 6 frequently. In such a case, the FCFSPP algorithm performs more or less the same as the FCFS algorithm in terms of *speed* while it tends to perform better than the FCFS algorithm in terms of *reliability*.

According to previous research work in [Lazarevic06] [Li04][Iosup06][Medernach05], job size varies significantly in different Grid systems, but normally job size is between a couple of seconds to 24 hours. According to previous research work in [Kondo05][TUDelft10], each resource' *Availability Interval* in a volunteered resources based Grid system varies significantly (more about this will be discussed in Chapter 6), even mean *Availability Interval* of each Grid also varies, but normally the *Average Job Size* of all resources in a Grid is between 10 minutes to a couple of hours. Therefore, the possible highest results in terms of *speed* and *reliability* vary from one Grid system to another.

ix. **Influence of the *Multiply Factor M***

If the value of M is 1, then the length of *Checking Period* and *Prediction Period* equal the length of the job execution time X. If the parameter of *Number of Checking Days* is 1, the result of *Resource Availability Probability* will be either 0% or 100%. This is because of the equation that used to calculate the result of *Resource Availability Probability*. As mentioned in Equation 4.1, the result of *Resource Availability Probability* is calculated by Equation 4.1. In this equation, the numerator is the *Times of Available to Grid* to *Available to Grid*. If the resource stays in the state of *Available to Grid* throughout the *Checking Period*, *Times of Available to Grid* to *Available to Grid* will be 1 and Times of *Available to Grid* to other states will be all 0. Therefore, the result of *Resource Availability Probability* will be 100%. On the other hand, if resource exits the state of *Available to Grid* during the *Checking Period*, *Times of Available to Grid* to *Available to Grid* will be 0 and Times of *Available to Grid* to some other states will be above 0. Therefore, the result of *Resource Availability Probability* will be 0%.

If the value of M becomes larger, the result of *Resource Availability Probability* can be any value between 0% and 100%. However, a larger value of M does not mean it will definitely bring more accurate prediction results and it also brings difficulty for analysis and evaluation of the performance of the FCFSPP algorithm. Therefore, without any further notice, the default value of *Multiply Factor* will be 1 in this thesis.

5.3 Explanation of the FLP Algorithm

5.3.1 Features of the FLP Algorithm

As described in Section 4.2.4, the FLP algorithm is based on the FCFSPP algorithm (described in Section 4.2.3) and it adds Fuzzy inference system. In general, the FLP algorithm inherits all the important features of the FCFSPP algorithm described in Section 5.2. In addition, the factors and parameters influence FLP in the same way as it is in the FCFSPP algorithm.

However, as FLP has an extra Fuzzy inference system, this will bring some one important new feature to the FLP algorithm:

Different from FCFSPP, the *Resource Availability Probability Threshold* will change over time according to the trend of overall resource reliability in FLP. Here, the overall resource reliability is represented as "*Disposed Jobs Dot*" (described in Section 4.2.4) and FLP takes reactive actions to this change. As discussed in Section 4.2, FLP tries to achieve a better balance between *speed* and *reliability* by replacing the fix setting of *Resource Availability Probability Threshold* with a dynamic and artificial intelligent algorithm controlled setting.

If the number of "*Disposed Jobs Dot*" is above 0, it means resources become more volatile and tend to dispose more jobs now. As a result, FLP will raise *Resource Availability Probability Threshold* to get better results in terms of *reliability*. If the number of "*Disposed Jobs Dot*" is not above 0, it means resources becomes more reliable and tend to dispose fewer jobs now. As a result, FLP will lower *Resource Availability Probability Threshold* to get better results in terms of *speed*.

If the pattern(s) of all resources' *Job Execution Availability* is similar, *Job Execution Availability* on some resources will provide good indications for all resources. In such a case, FLP will make good balance between *speed* and *reliability*. Imagine a scenario in which a Grid is composed of personal computers within a company, resources availability patterns will be similar as these personal computers are typically utilised during the office hours. As a result, all resources tend to be very volatile during office hours (e.g. 9am to 5pm) but tend to be rather reliable during non-office hours. When office hour starts, "*Disposed Jobs Dot*" will increase and *Resource Availability Probability Threshold* will increase as a result. Therefore, high value of *Resource Availability Probability Threshold* will ensure jobs not being allocated to volatile resources and better results of *reliability* can be achieved. When non-office hour starts, "*Disposed Jobs Dot*" will decrease and *Resource Availability Probability Threshold* will decrease as a result. Therefore, low value of *Resource Availability Probability Threshold* will ensure idle CPU cycles on many resources can be utilised efficiently and better results of *speed* can be achieved.

If the pattern(s) of all resources' *Job Execution Availability* is dissimilar, *Job Execution Availability* on some resources will NOT provide good indications for all resources. In such a case, FLP is difficult to make good balance between *speed* and *reliability*. Imagine a scenario in which a Grid is composed of computers from individual users, resources availability patterns can be dissimilar as people have different life styles. As a result, some resources will be very volatile while some others are not. Therefore, "*Disposed Jobs Dot*" will increase/decrease over time. However, the increase/decrease of "*Disposed Jobs Dot*" may not be a good indication, and it may even be misleading. In such a scenario, changing *Resource Availability Probability Threshold* may not be able to provide good balance between *speed* and *reliability*.

As the value of *Resource Availability Probability Threshold* varies between 0% and 100%,

FLP's performance in terms of *speed* and *reliability* should generally (an exceptional case will be described in Section 7.3) between the FCFSPP algorithm with *Resource Availability Probability Threshold* 100% (equals the FCFS algorithm) and the FCFSPP algorithm with *Resource Availability Probability Threshold* 100%. In addition, the FLP algorithm is also influenced by some factors and parameters. Therefore, in next subsection, some important influences will be analysed.

5.3.2 Influences on the FLP Algorithm

i. Influence of *Resource Availability Probability Threshold Adjustment Interval*

In the FLP algorithm, the Grid job scheduler checks whether the time has reached the end of the current time interval or not. Here, the time interval is called *Resource Availability Probability Threshold Adjustment Interval*, which is a predefined value, such as 1 minute, 10 minutes and so on.

The length of *Resource Availability Probability Threshold Adjustment Interval* will affect the sensitivity of:

If the value of *Resource Availability Probability Threshold Adjustment Interval* is large, the FLP algorithm will rarely trigger the procedure of *Resource Availability Probability Threshold Adjustment*. As a result, *Resource Availability Probability Threshold* will be relatively stable and the FLP algorithm becomes more like the FCFSPP algorithm in such a case.

If the value of *Resource Availability Probability Threshold Adjustment Interval* is small, the FLP algorithm will trigger the procedure of *Resource Availability Probability Threshold Adjustment* frequently. As a result, *Resource Availability Probability Threshold* tends to be changed frequently in such a case.

ii. Influence of Parameter λ

In the FLP algorithm, a fuzzy inference system is used and λ and – λ are two important thresholds to define a truth value of *Disposed Jobs Dot*:

If the absolute value of λ is small, *Disposed Jobs Dot*'s truth value of Zero tends to be low while Negative and Positive tends to high. As a result, *Resource Availability Probability Threshold* tends to be changed sharply in such a case.

If the absolute value of λ is large, *Disposed Jobs Dot*'s truth value of Zero tends to be high while Negative and Positive tends to low. As a result, *Resource Availability Probability Threshold* tends to be changed smoothly in such a case.

iii. Influence of Initial *Resource Availability Probability Threshold*

At the initial stage, the value of *Resource Availability Probability Threshold* is defined manually and this initial value will influence the performance of the FLP algorithm at the initial stage(s). For example, if the value of *Resource Availability Probability Threshold* is very small (a value close to 0%), the FLP algorithm will be loose and tend to allocate jobs to unreliable

resources at first. On the other hand, if the value of *Resource Availability Probability Threshold* is very large (a value close to 0%), the FLP algorithm will be strict and NOT tend to allocate jobs to unreliable resources at first. However, this influence may be limited as the value of *Resource Availability Probability Threshold* could later change.

5.4 Analysis of the PSOPP Algorithm

5.4.1 Features of the PSOPP algorithm

As described in Section 4.2.5, the PSOPP algorithm uses a completely different job-scheduling algorithm when comparing with the FCFSPP and the FLP algorithm. This new algorithm has some important features:

1. The PSOPP algorithm inherits features of the PSO algorithm. It uses a number of particles to search the best solution in the search space. In this algorithm, the best solution is the resource which has the highest fitness value after a number of iterations and the search space is all available resources. After each iteration, each particle's position will be updated with the personal best value and global best value.

   PSOPP's fitness function is based on the considerations of both speed and reliability of each resource. The fitness function of the PSOPP algorithm is shown by Equation 4.7. It uses TDE prediction to calculate the *reliability* of the resource in the first part of the equation and calculates the *speed* that the job can get from the resource the second part of the equation.

2. All *available* resources are candidates when the PSOPP algorithm tries to make job allocations. In other words, the PSOPP algorithm does not keep any coming job waiting in the job queue if *available* resource(s) exists. If *available* resource(s) exists, all these *available* resources will be candidate for the new coming jobs and the PSOPP algorithm will try to allocate these new coming jobs to one or some of the *available* resource(s).

   As discussed in Section 5.2.1 and 5.3.1, the FCFSPP and the FLP algorithm always try to allocate a new job to the next *idle* resource (which is called *Checking If Qualified*). Different from these *Checking If Qualified* algorithms, PSOPP is a type of *Finding the Best* (the job-scheduling algorithms will try to find out the "best" resource from some candidates) algorithm and all *available* resources (not necessarily *idle*) will be candidates when the PSOPP algorithm tries to make job allocation.

   As discussed in Section 5.2.1, *Checking if Qualified* is quicker in terms of making job allocation decisions when comparing with *Finding the Best* approach. If the number of jobs is far more than the number of resources, *Finding the Best* approach will become the same as *Checking if Qualified* if there is always one candidate at a time. However, as the PSOPP algorithm uses *available* resources rather than *idle* resources as candidates, it is less likely that the whole Grid has only one candidate at the time when the PSOPP algorithm tries to make a job allocation decision. If there is more than one candidate when the PSOPP algorithm tries to make a job allocation decision, the PSOPP algorithm is possible to

perform better than *Checking if Qualified* algorithms as multiple choices are available at the moment. However, on the other hand, multiple choices also make PSOPP possible to perform worse than *Checking if Qualified* algorithms.

5.4.2 Influence of *Workload*

Here, *Workload* means the total number of jobs in the Grids at a certain moment. As the PSOPP algorithm will consider all *available* resources when it tries to make job allocation decisions, it is possible to allocate jobs to a *busy* resource if *Workload* is high. If all resources are *busy* at the moment, then the PSOPP algorithm will have to allocate new jobs to *busy* resources. Therefore, *Workload* influences the PSOPP algorithm in terms of *speed* (including both job *Makespan* and job throughput) greatly.

If the *Workload* is high (the number of jobs is usually above the number of *idle* resources), the PSO algorithm will have to allocate jobs to *busy* resources and keep one resource have more than one job at a time. As discussed in Section 5.2.1, keeping one resource have more than one job at a time is usually difficult to provide benefit in terms of *speed*. In terms of *speed*, as the number of CPU cycles provided by a resource is fixed and as all guest jobs are assumed to have the same priority, all guest jobs on a resource have to share the CPU cycles at a time. Therefore, if there is only one guest job on a resource at a time, each job's *Makespan* will be the shortest. If there is more than one guest job on a resource at a time, each job's *Makespan* will become longer and job throughput will tend to be low as well.

If the *Workload* is low (the number of jobs is usually below the number of *idle* resources), the PSOPP algorithm will not have to allocate new jobs to *busy* resources. Therefore, the influence to each job's *Makespan* will tend to be smaller. However, the result of job throughput will tend to be low as resources tend to have not enough jobs to process in such a case.

5.4.3 Influence of Fitness Function

In the PSOPP algorithm, the fitness function is used to calculate the fitness value of each solution. As discussed in Section 4.2.5, the fitness value of a resource is calculated by the Equation 4.7. It is based on the considerations of both *speed* (represented by resource's current *CPU Availability* and the number of jobs running on the resource) and *reliability* (represented by the *Resource Availability Probability*) of each resource. The multiply factor x and y are used to adjust the proportion of speed and reliability. Therefore, if a resource currently has a high *CPU Availability*, low number of jobs and high reliability, it is likely to be chosen by the PSOPP algorithm when the PSOPP algorithm makes job allocations.

5.4.4 Influence of Resource Reliability

Resource reliability means the *Availability Interval* lasts for a very long time. As discussed in Section 5.2.1, if resource reliability is unknown or uncertain, each resource may become *unavailable* to the Grid at any time. When a resource becomes unavailable to the Grid, all guest

jobs running on the resource will be lost. Therefore, if there is only one guest job on a resource at a time, only one job will be lost. If there is more than one guest job on a resource at a time, more than one guest job will be lost. As a result, a job-scheduling algorithm's performance in terms of *reliability* will be influenced. As PSOPP may keep a resource have more than one guest job at a time, the PSOPP algorithm's performance in terms of reliability will be influenced if resources reliability is unknown or uncertain.

However, if resource reliability is known in advance, allocating jobs to *busy* resources (keeping resources have more than one job at a time) may bring benefits to the PSOPP algorithm in terms of *reliability*. Imagine a scenario that two groups of resources are in the Grid, one group is very reliable (*Job Execution Availability* is true for a very long time) and the other group is volatile (*Job Execution Availability* changes between true and false frequently). In such a scenario, if allocating jobs to the second group of resources, jobs have to face potential job failures. So if allocating jobs to the first group of resources only, this will mitigate the problem of job failure brought by the second group of resources though the first group of resources may be have more than one job at a time.

5.4.5 Influences of the PSO Algorithm

In the PSO algorithm, various parameters will influence the performance of the PSO algorithm and the PSOPP algorithm will be influenced as a result. These parameters include:

- The number of particles: Number of particles is a parameter to specify the total number of particles used for searching the best solution in the PSO algorithm. If a search space is small, all positions are likely to be covered (even by a small number of particles) so that number of particles does not influence the performance too much in such cases. However, if the search space is very large and the number of particles is small, it is difficult for particles to cover all possible positions and find the best solution. A typical range is 20 to 40 and, for most problems, 10 particles are enough to obtain good results [Hu06].
- V_{max}: V_{max} is a parameter to specify the maximum velocity a particle algorithm. If a search space is small, particles can find the best solution easily so that V_{max} does not influence the performance of the PSO algorithm too much. However, if the search space is very large and the V_{max} is very small, it is difficult for particles to cover all possible positions. According to [Hu06], the typical value of V_{max} is set as the same as the search space. For example, if the search space is [1, 20], V_{max} is typically set as 20.
- The stop condition: In the PSO algorithm, the procedure will stop once if meets the predefined goal(s) or reaches the maximum number of iterations. In terms of maximum number of iterations, if the search space is small and all positions can be covered by particles easily, the maximum number of iteration does not influence the performance of the PSO algorithm too much. However, if the search space is large and the maximum number of iteration is small, it will be difficult for particles to find the best solution before reaching the

maximum number of iterations.

5.5 Analysis of the PSPP Migration Algorithm

5.5.1 Features of the PSPP Algorithm

As described in Section 4.3.2, the PSPP migration algorithm is a proactive job migration algorithm used for migrate jobs proactively for avoiding potential job failures. Therefore, this algorithm will have the following distinct features:

Firstly, the PSPP migration algorithm uses TDE prediction to carry out predictions and then determine whether to trigger proactive job migrations. Therefore, the performance of the PSPP migration algorithm is directly influenced by the performance of the TDE prediction method (more details about the influences will be introduced in the following subsection).

Secondly, the PSPP migration algorithm only checks resources that are *busy* (running guest job from the Grid) at the moment. For a resource that is *idle* (in the state of *Available to Grid* but does not have guest job) or *unavailable* (not in the state of *Available to Grid*), there is no need to worry about job migration. Therefore, only *busy* resource will be checked.

Thirdly, the PSPP migration algorithm carries out the checking procedure regularly. The algorithm carries out the checking procedure regularly. At the end of each *Migration Prediction Interval*, the PSPP migration algorithm will check each *busy* resource and determine whether the job on the resource needs migration or not. In addition, the PSPP migration algorithm is also influenced by some factors and parameters. Therefore, in next subsection, some important influences will be analysed.

5.5.2 Influences on the PSPP Algorithm

i. Influence of TDE Prediction

According to [Rood08], "A correct prediction is one for which the machine is predicted to exit on a certain non-available state and it does, or for which the machine is predicted to remain available throughout the interval, and it does." Here, "exit on a certain non-available state" means the result of *Resource Availability Probability* in the *Prediction Period* is below 100%. "Remain available throughout the interval" means the result of *Resource Availability Probability* in the *Prediction Period* is 100%. Based on this definition, there are three terms to describe the accuracy of a prediction result in different scenarios:

Correct Prediction: If a resource is predicted to exit or not exit *Available to Grid* state and the resource turns out to exit or not exit *Available to Grid* state some time during the *Prediction Period*, then the prediction result is *Correct Prediction*. Furthermore, the *Correction Prediction* about resource exit *Available to Grid* State is *Correction Prediction Type 1* and the *Correction Prediction* about resource stay in *Available to Grid* State is *Correction Prediction Type 2*.

False Alarm: If a resource is predicted to exit *Available to Grid* state but it turns out NOT to exit *Available to Grid* state throughout the *Prediction Period*, then the prediction result is *False*

Alarm.

Missed Detection: If a resource is predicted to NOT to exit *Available to Grid* state but it turns out to exit *Available to Grid* state some time during the *Prediction Period*, then the prediction result is *Missed Detection*.

Table 5-2 shows the prediction accuracy in different scenarios.

Checking Period	Prediction Period	Prediction Accuracy
exit *Available to Grid*	exit *Available to Grid*	*Correct Prediction Type 1*
exit *Available to Grid*	stays in *Available to Grid*	*False Alarm*
stays in *Available to Grid*	exit *Available to Grid*	*Missed Detection*
stays in *Available to Grid*	stays in *Available to Grid*	*Correct Prediction Type 2*

Table 5-1: Prediction Accuracy

Based on these terms, the TDE prediction method influences the PSPP migration algorithm in the following ways:

1. The performance of the PSPP migration algorithm is directly influenced by the accuracy of results from the TDE prediction scheme.

   If the TDE prediction scheme predicts that a resource will stay in the state of *Available to Grid* throughout the *Prediction Period* and it is a *Correct Prediction Type 2*, then the job migration procedure will not be triggered and the job migration also turns out to be not necessary. This type of *Correction Prediction* will avoid PSPP algorithm wasting time on unnecessary job migration. However, note the effect of this type of prediction is actually the same as no prediction.

   If the TDE prediction method predicts that a resource will exit *Available to Grid* state during the *Prediction Period* and it is a *Correct Prediction Type 1*, then the job migration procedure will be triggered and the job migration turns out to be necessary. This type of *Correct Prediction* will protect jobs on the resource from job failures because of the resource becoming *unavailable*.

   If the prediction algorithm predicts that a resource will exit *Available to Grid* state during the *Prediction Period* but it is a *False Alarm*, then the job migration procedure will be triggered and the job migration turns out to be unnecessary. The reasons why it is unnecessary are:

   - Firstly, it will be a waste of idle CPU cycles of the resources if all jobs on the resource are migrated and the resource will leave idle until being allocated new job(s). However, this influence might be trivial as a new job may be allocated to this resource after a short period time (such as a couple of seconds).
   - Secondly, it may lengthen job(s)'s *Makespan* reduce job throughput as it will take some time to accomplish the procedure of job migration. However, this influence might be trivial, as the job migration procedure will usually take a couple of seconds.

   If the prediction algorithm predicts that a resource will stay in *Available to Grid* state

during the *Prediction Period* but it is a *Missed Detection*, then the job migration procedure will not be triggered and the job migration turns out to be very necessary. In such a case, all job(s) on the resource will be lost and the original job(s) needs to be allocated to a new resource and process from the beginning, which may lengthen job(s)'s *Makespan* and reduce job throughput.

2. As the PSPP migration algorithm and the FCFSPP job-scheduling algorithm are both based on TDE prediction, some factors and parameters influencing the FCFSPP algorithm will also influence the PSPP migration algorithm.

   Firstly, the PSPP migration algorithm encounters the same system states (described in Section 5.2.2) as the FCFSPP algorithm. The TDE prediction method will make a correct prediction when facing all cases except cases 2 and 4.

   In case 1, the resource is predicted to stay in the state of *Available to Grid* throughout the *Prediction Period* and the resource turns out to stay in the state of *Available to Grid* throughout the *Prediction Period*. Therefore, the prediction will be a *Correct Prediction* when facing these two cases.

   In case 3 and 5, the resource is predicted to exit the state of *Available to Grid* during the *Prediction Period* and the resource turns out to exit the state of *Available to Grid* during the *Prediction Period*. Therefore, the prediction will be a *Correct Prediction* when facing these two cases as well.

   In case 2, the resource is predicted to exit the state of *Available to Grid* during the *Prediction Period* but the resource turns out to stay in the state of *Available to Grid* during the *Prediction Period*. Therefore, the prediction will be a *False Alarm* when facing these two cases as well.

   In case 4, the resource is predicted to stay in the state of *Available to Grid* throughout the *Prediction Period* but the resource turns out to exit the state of *Available to Grid* during the *Prediction Period*. Therefore, the prediction will be a *Missed Detection* when facing these two cases.

   Second, the factor "Δt between *Checking Day* and *Prediction Day*" (described in Section 5.2.2) will also influence the performance of PSPP algorithm. If the value of Δt is small, a prediction result will tends to be a *Correct Prediction*. On the other hand, if the value of Δt is large, the prediction result will tends to be either a *False Alarm* or *Missed Detection*.

   Third, the factor "Similarity of *Job Execution Availability* between *Checking Period* and *Prediction Period*" (described in Section 5.2.2) will also influence the performance of PSPP algorithm. If the pattern of *Job Execution Availability* in *Checking Period* is similar to pattern of *Job Execution Availability* in *Prediction period*, the prediction result will tend to be a *Correct Prediction*. On the other hand, if the pattern of *Job Execution Availability* in *Checking Period* is dissimilar to pattern of *Job Execution Availability* in *Prediction period*, the prediction result will tends to be a *False Alarm* or *Missed Detection*.

Fourth, the parameter *Number of Checking Days* will also influence the performance of PSPP algorithm. According to the analysis about *Number of Checking Days* in Section 5.2.2, when the value of *Number of Checking Days* becomes larger, the total length of *Checking Period* becomes longer so it is more likely to have an *Unavailability Events* in the *Checking Period*. Therefore, the result of *Resource Availability Probability* tends to below 100% more likely and the TDE prediction method tends to predict many resources to exit the state of *Available to Grid* during the *Prediction Period*.

ii. Influence of *Checking Period* (or *Prediction Period*)

In the PSPP algorithm, if the length of *Checking Period* (or *Prediction Period*) becomes longer, it is more likely to have *Unavailability Events* in the *Checking Period* for each resource. Therefore, the result of *Resource Availability Probability* tends to below 100% and the TDE prediction method tends to predict each resource to exit the state of *Available to Grid* during the *Prediction Period*. On the other hand, if the length of *Checking Period* (or *Prediction Period*) becomes shorter, it is more likely to have *Unavailability Events* in the *Checking Period* for each resource. Therefore, the result of *Resource Availability Probability* tends to be 100% and the TDE prediction method tends to predict many resources to stay in the state of *Available to Grid* during the *Prediction Period*.

The length of *Checking Period* will only influence the prediction results, but it will not influence the accuracy of the prediction results.

5.6 Analysis of the CBR Migration Algorithm

5.6.1 Features of the CBR Migration Algorithm

The proposed CBR migration algorithm has the following features:

Firstly, it uses each resource's *CPU Availability Percentage* and a *CPU Migration Threshold* to predict whether a resource will exit the state of *Available to Grid* or not. Different from other job-scheduling or migration algorithm proposed in this research, CBR migration algorithm does not used the adopted the TDE prediction method to make predictions.

Secondly, it uses CBR to refine the job migration decisions. As discussed in Section 4.3.2, CBR migration algorithm uses *CPU Migration Threshold* as solutions to solve the problem of whether to trigger job migration procedure or not. More importantly, CBR migration algorithm revises the value of *CPU Migration Threshold* according to the accuracy level of previous migration decisions.

Thirdly, the same as the PSPP migration algorithm, CBR migration algorithm only checks resources that are *busy* (running guest job from the Grid) at the moment. For a resource that is *idle* (in the state of *Available to Grid* but does not have guest job) or *unavailable* (not in the state of *Available to Grid*), there is no need to worry about job migration. Therefore, only busy resource will be checked.

Thirdly, as with the PSPP migration algorithm, CBR carries out the checking procedure regularly. The algorithm carries out the checking procedure regularly. At the end of each *Migration Prediction Interval*, CBR migration algorithm will check each *busy* resource and determine whether the job on the resource needs migration or not.

In addition, CBR migration algorithm is also influenced by some factors and parameters; these will be considered in Section 5.6.2.

5.6.2 Influence of *CPU Availability Percentage*

CBR migration algorithm observes the change of *CPU Availability Percentage* and then makes prediction based on the current value of *CPU Availability Percentage*.

Therefore, the performance of the proposed CBR migration algorithm depends on whether the *CPU Availability Percentage* can provide any useful information. If all resources' *CPU Availability Percentage* will become low before they become *unavailable*, then using CBR migration algorithm will be possible to observe this change and then make correct job migration decisions. However, on the other hand, if all resources' *CPU Availability Percentage* does not become low before they become *unavailable*, then using CBR migration it is difficult to observe the change of *CPU Availability Percentage* and make correct job migration decisions. It requires all resource have similar behaviour when they become unavailable.

5.6.3 Influence of *CPU Migration Threshold*

In CBR migration algorithm, the *CPU Migration Threshold* is a very important value as whether to trigger job migration decisions are based on this value. If resources' *CPU Availability Percentage* always get lower than the *CPU Migration Threshold* before become *unavailable*, using the *CPU Migration Threshold* to make predictions will be a good solution.

If resources have *CPU Availability Percentage* higher than the *CPU Migration Threshold* will also exit the state of *Available to Grid* soon, then using the *CPU Migration Threshold* to make predictions will probably still be good as CBR migration algorithm will probably find a suitable *CPU Migration Threshold* by adjusting the value of *CPU Availability Percentage* regularly.

In addition to the value of the *CPU Migration Threshold*, the range of the value of *CPU Migration Threshold* is also important. As *CPU Availability Percentage* is always between 0% and 100%, the value of the *CPU Migration Threshold* should at most ranges from 0% to 100%. However, if a resource has a *CPU Availability Percentage* of 0%, then the resource is already left the state of *Available to Grid* so CBR migration algorithm will not make prediction for such a resource. As a result, the value of the *CPU Migration Threshold* should not be 0% at any time. On the other hand, if the value of the *CPU Migration Threshold* is too high (e.g. 100%), then all resources are *unqualified* and they will all need migration in such a case. Therefore, the value of the *CPU Migration Threshold* should not be 100% at any time. So the maximum range of the *CPU Migration Threshold* is (0%, 100%)

Only defining the range of the *CPU Migration Threshold* as (0%, 100%) might not be enough.

If the value of the *CPU Migration Threshold* is close to 0%, a small proportion of resources will tend to be considered as *unqualified*. On the other hand, if the value of the *CPU Migration Threshold* is close to 0%, a large proportion of resources will be considered as *unqualified*. So the range of the value of the *CPU Migration Threshold* can be further restricted to a smaller range, such as (0%, 30%).

5.6.4 Influence of *Migration Prediction Interval*

The *Migration Prediction Interval* is a parameter that influences the behaviour of CBR migration algorithm. In brief, if the value of the *Migration Prediction Interval* is small, then CBR migration algorithm tends to review the value of the *CPU Migration Threshold* frequently and the value of the *CPU Migration Threshold* might be adjusted frequently. On the other hand, if the value of the *Migration Prediction Interval* is large, then CBR migration algorithm tends to review the value of the *CPU Migration Threshold* infrequently and the *CPU Migration Threshold* will be adjusted infrequently as a result.

5.6.5 Influence of *Adjustment Percentage*

In addition to *Migration Prediction Interval*, *Adjustment Percentage* is another parameter that influences the behaviour of CBR migration algorithm.

When the *CPU Migration Threshold* needs an adjustment, the value of *Adjustment Percentage* will be generated over the range (0, Max%]. Here "(0, Max%]" means the value of *Adjustment Percentage* should be above 0 and not larger than the predefined maximum value Max. Therefore, if the maximum value of *Adjustment Percentage* is small, the value of *CPU Migration Threshold* will tend to be adjusted a small value at a time. So if the ideal value of *CPU Migration Threshold* is not far away from the current value of *CPU Migration Threshold*, then making small adjustments at a time tends to get to this ideal value quickly. However, if the ideal value of *CPU Migration Threshold* is far away from the current value of *CPU Migration Threshold*, adjusting a small amount at a time tends to delay the time to get to this ideal value.

Chapter 6 Characteristics of Real Resources

Resource availability data traces record different resource availability data over time (e.g. an hour, a day, etc). As mentioned in Section 4.2.2, three levels of availability can be used to describe a resource's availability (*Resource Availability*, *Job Execution Availability* and *CPU Availability*). *Host Availability* indicates whether a resource is in the Grid or not but it does not mean the Grid can utilise the resource's idle CPU cycles or not. *CPU Availability* and *Job Execution Availability* are more important terms as they indicate whether the resource allows guest jobs to run and how many CPU cycles the resource contributes to the Grid in a certain period of time. In [Kondo09], four sets of resource availability data traces that record *CPU Availability* in different real volunteered resources based Grids are provided. Therefore, some important characteristics of the resources (especially *Job Execution Availability*) in these four data sets will be described and discussed in this chapter.

6.1 Data Traces Overview

In terms of collecting resource availability data traces, some research has been done before. In [Brevik03], the authors designed an "up-time sensor" sensor to record each machine's uptime from the /proc file system. In [Long95], the authors used Remote Procedure Call (RPC) to get a response from each resource. If a resource replies, then the resource is marked as up, otherwise, the resource will be considered as down. In [Bhagwan03], the authors designed a "prober" in peer-to-peer networks. At regular intervals, the "prober" performs a lookup for a certain resource. If the resource responds to the prober, then the resource is considered as up, otherwise, the resource is down. In [Saroiu02], the authors used ping/pong messages for peers to discover other nodes. In [Dinda99][Dinda02], the authors use the kernel to collect and record resources' CPU load at regular intervals. In [Kondo09], the authors run their CPU-bound, fixed-time length tasks on different resources to record *CPU availability*. At regular intervals, the information of *CPU availability* is written into a trace file.

A number of data traces from the same Grid compose a data set and there are few resource availability data sets available in the Internet. On the website of [Dinda00], the authors published two sets of data traces collected from Pittsburgh Supercomputing Center (PSC) and Computers, Media, and Communication Laboratory (CMCL) of Carnegie Mellon University in two periods (August 1997 and March 1998). The kernel records the resource CPU load once a second. So there are two columns in each trace file, one for the timestamp and the other for the value of measured CPU load. Though the CPU load and CPU cycles delivered to the Grid (*CPU availability*) are correlated, CPU load does not necessarily provide accurate information about *CPU availability*, this is because of the priority issue in Unix systems. In Unix, processes share CPU time according to their priority. Therefore, the value of the CPU load does not mean all the spare CPU time will be available to the Grid.

On the website of [Kondo09], the author published four data sets collected from different volunteered resources based Grids. All these data sets were collected with the approach they proposed in [Kondo05][Kondo07]. The publisher claims that the *CPU Availability* are accurately recorded by the fixed-time length tasks on the resource, so these four data sets were used for analysis and simulation evaluations in this research.

6.1.1 Analysed Data Sets

There are four data sets provided on the website of [Kondo09]: UCB, SDSC, LRI and DEUG. The first data set is called UCB, which was originally obtained from [Arpaci95]. As mentioned above, the authors of [Kondo09] collected these data by using a daemon program to log CPU and keyboard/mouse activity every second of the resources. Note here the daemon program only records whether a resource is up and whether any local activity occurs. As a result, *Resource Availability* and *Job Execution Availability* can be derived from the log file while *CPU availability* cannot be derived. However, according to [Kondo05], a resource is considered to be *Available to the Grid* when the Resource load is below 5% and no local activity occurs during that time, so the publishers believe the result of post-processing is most likely to be accurate. Therefore, [Kondo05] post-processed the UCB trace files and published the processed data traces on their website. The traces record 80 resources' *CPU Availability* in 10 consecutive working days from 28th of February to 9th of March in 1994. According to their description, the resources are quite stable during off-peak (non-business) hours. So the traces only show *CPU Availability* once a second during peak (business) hours of each day. In this data set, the business hours are from 10AM to 5PM. As the data set is more than ten years old, the publisher also mentioned that this might be a potential weakness with this data set.

The second data set is called SDSC, collecting from the Entropia DCGridTM desktop grid software system that was deployed at San Diego Super Computing Centre (SDSC). The traces record 244 resources' *CPU Availability* in 7 consecutive working days from 3rd of September to 12th of September in 2003. As for data set SDSC, the traces show the *CPU Availability* once a second during peak hours in each day. The business hour in this data set is 9AM to 5PM. According to [Kondo05], "30 are used by secretaries, 20 are public Resources that are available in SDSC's conference rooms, 12 are used by system administrators, and the remaining are used by SDSC staff scientists and researchers."

The third data set is called LRI, collecting from the XtremWeb desktop grid software system deployed at University of Paris-Sud. The traces record resources' *CPU Availability* in consecutive working days from 5th of January to 26th of January in 2005. Each day the trace records the resources' *CPU Availability* once a second. For the purpose of simulation [Kondo05], the traces of resources on different days were pooled together to increase the number of the resources in the platform. After post-processing, 275 resources' data traces in 7 consecutive working days were created. According to [Kondo05], all the resources in this data set are a

cluster computer.

The fourth data set is called DEUG, also collecting from the XtremWeb desktop grid software system deployed at University of Paris-Sud. The traces record 136 resources' *CPU availability* in consecutive working days from 5th of January to 26th of January in 2005. In each day, the trace records the *CPU availability* once a second during business hours – 6AM to 6PM. As with DEUG, for the purpose of simulation [Kondo05], the traces of resources on different days were pooled together to increase the number of the resources in the platform. After post-processing, 680 resources' data traces in 7 consecutive working days were created. All the resources in this data set are computers in different classrooms [Kondo05].

6.1.2 Data Trace Formats

For each data set, a file named "Resourceinfo.dat" contains some general information about each resource. The information includes each resource's name, clocks rates and maximum number of CPU cycles delivered to the Grid per second. Here is a part of the "Resourceinfo.dat" of data set SDSC:

```
Resource Name   Clock Rates      Maximum Number of CPU Cycles
MWAN-2K         179              6851.7776566835
RITKE-PC        198              7706.3301532329
OUYAR           198              7738.4406779661
LAGRANGE-2K     297              14628.53780466
CENON-2K        298              12438.263009999
BASEBALL-2K     330              13775.014899006
LPW8            331              13778.188795437
CRBPUB-4        331              14226.06326689
POTOROO-2K      333              16529.418732542
JCZECH-2K       397              19508.218134882
```

A single data trace file records a single resource's *CPU Availability* in the business hours of a single day. The business hour is defined as 10am to 5pm, 9am to 5pm, 6am to 6pm in data set UCB, SDSC and DEUG. For data set LRI, 24 hours of a day are considered as the business hours and recorded.

In each trace file, there are three columns in a line. The first and the second column shows the epoch start and finish time respectively. The third column shows the CPU cycles delivered to the Grid between the epoch start time and epoch finish time. For example, here is a part of extracted from a trace file:

```
Start time  End time   CPU cycles Delivered to Grid
 0             1             110653
 1             2             110395
 2             3             110650
 3             4             110653
 4             5             110653
 5             6             110653
 6             7             -1
 7             8             -1
 8             9             -1
 9             10            -1
```

"0 1 110653" means resource's *CPU Availability* (CPU cycles delivered to the Grid) from

time 0 to 1 is 110653 and it also indicates resource's *Job Execution Availability* is true in that period. In "6 7 -1", *CPU Availability* is 0 from time 6 to 7 and it indicates resource's *Job Execution Availability* is false in that period.

6.1.3 *Job Execution Availability* Characterisation

In terms of resource *CPU Availability*, the publisher provides detailed information in his PhD dissertation [Kondo05]. So here only some important points will be summarised:

First, in terms of *Job Execution Availability*, resources are volatile during the peak hours in every working day except the LRI trace. That means the number of available resources vary a lot during the peak hours. According to their statistical results, the mean length of all platforms is about 2.6 hours. Even in the most volatile platform – UCB, the availability intervals tend to be 10 minutes or greater.

Second, job failure rates on each platform are correlated with the job size and it can be approximated as a linear function of *Job Size*. Whether a job can be completed or not is directly influenced by the length of *Availability Intervals*. If an *Availability Interval* is long enough, then the job can be completed. Otherwise, the job will be failed before completion.

In [Kondo05], the authors chose hundreds of thousands of random points in the data traces and then allocated random size jobs to the resources at these points. If the allocated jobs can finish successfully, then it was counted as a success. Otherwise, it was counted as a failure. After producing statistics, the author found that task failure rate has a strong relationship with the job size on all platforms and the lowest correlation coefficient is 0.98.

Third, on all platforms, *Job Execution Availability* tends to be independent between resources used by separate users. The authors studied the correlation of *Job Execution Availability* between paired resources. They used a method proposed in [Bolosky00] to study the correlations between resources. Specifically, this method compares the availability for each paired resources and see if both resources are available or unavailable at the same time. After study, the authors found resources on all four platforms show significant correlation relative to random if separate users use the resources.

If the same user uses multiple resources, e.g. resources with wake-on-LAN enabled Ethernet adapters that are controlled by a single network administrator, or resources used to run batch jobs, then these resources show strong correlation in terms of *Job Execution Availability*.

Fourth, the length of *Availability Interval* is not correlated with resources' clock rates. This means a resource with high clock rates does not necessarily used more often than a resource with a low clock rates.

Fifth, the length of *Availability Interval* is not correlated with the percentage of time a resource is unavailable. This means resources with high percentage of available time do not necessarily have longer availability intervals than resources have low percentage of available time.

Though plenty of information has been provided in [Kondo05], there is still some important characteristics information not provided, especially in terms of *Job Execution Availability* correlations within each single resource. In addition, such kind of information is very important to the job-scheduling algorithms proposed in this thesis. Therefore, some detailed information about these terms is discussed next.

There are 80 resources in data set UCB and 10 day's data are provided in [Arpaci95]. Figure 6.1 shows the number of available resource over time in 10 days and Figure 6.2 shows the average number of available resource over time in this data set. Overall, the number of available resources varies between 56 and 78 over time and the average number of available resources varies between 65 and 73.

The variance of the average number of available resources over time is 2.719. According to [Kondo05], resources' *Average Availability Interval* in UCB is only 0.166 hours and the *Average Unavailability Interval* in UCB is 0.119 hours, which indicates the resources are quite volatile (resource's *Job Execution Availability* changes between true and false frequently). Overall, the total number of *Unavailability Events* is 6076, which means all resources have 607.6 *Unavailability Events* in a day on average and each resource has 7.595 *Unavailability Events* in a day on average.

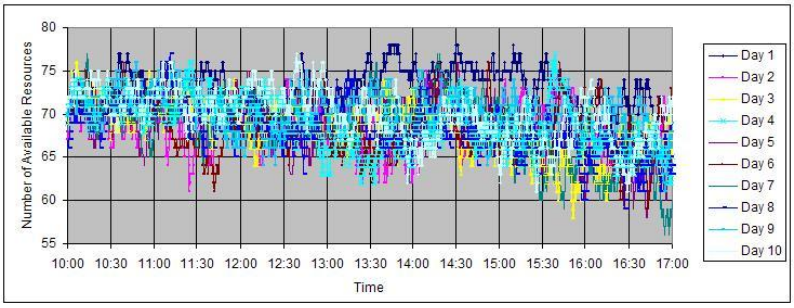


Figure 6-1: Number of Available Resources for Different Days (UCB)

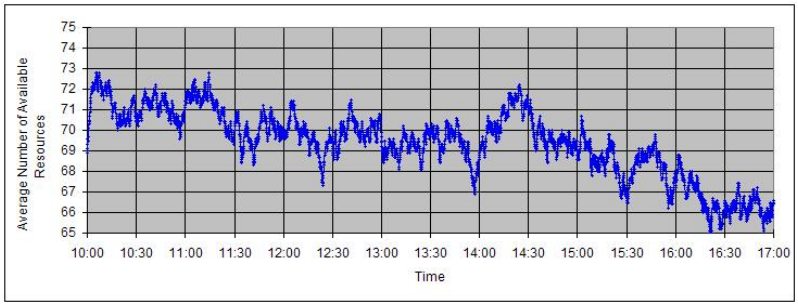


Figure 6-2: Average Number of Available Resources over Time (UCB)

There are 244 resources in data set SDSC and 7 day's data are provided in [Kondo05]. Figure 6.3 shows the number of available resource over time in 7 days and Figure 6.4 shows the average number of available resource over time in this data set. Overall, the number of available resources varies between 0 and 155 over time and the average number of available resources varies between 74.7 and 115.

The variance value of average number of available resources over time is 54.375, which

indicates the average number of available resource over time is quite unstable. According to [Kondo05], resources' *Average Availability Interval* in SDSC is 2.034 hours, which indicates that the number of *Unavailability Events* will be lower than UCB. On the other hand, resource's average *Unavailability Interval* in SDSC is 1.256 hours. Overall, the total number of *Unavailability Events* is 2370, which means all resources have about 339 *Unavailability Events* on average in a day, and each resource has about 1.388 *Unavailability Events* on average in a day, which is lower than the number of *Unavailability Events* in UCB.

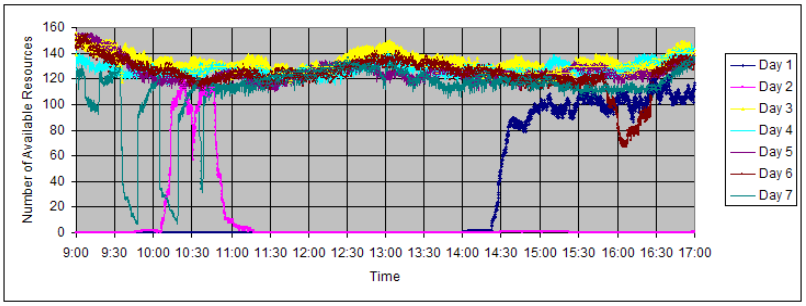


Figure 6-3: Number of Available Resources for Different Days (SDSC)

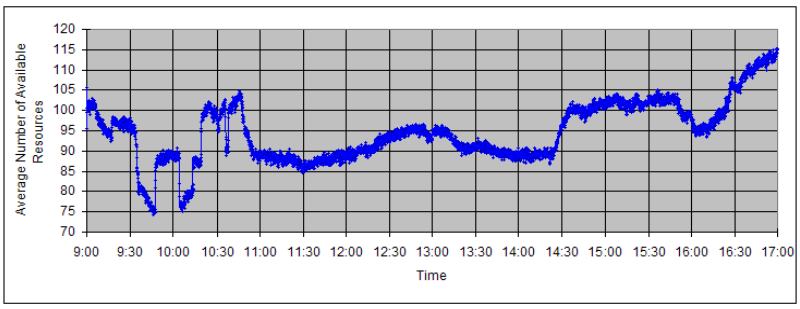


Figure 6-4: Average Number of Available Resources over Time (SDSC)

There are 275 resources in data set LRI and 7 day's data are provided in [Kondo05]. Figure 6.5 shows the number of available resource over time in 7 days and Figure 6.6 shows the average number of available resource over time in this data set. Overall, the number of available resources varies between 40 and 109 over time and the average number of available resources varies between 73.2 and 95.

The variance value of the average number of available resources over time is 12.801, which indicates the average number of available resources over time is quite unstable. According to [Kondo05], resources' *Average Availability Interval* in LRI is 23.535 hours, which indicates the number of *Unavailability Events* will be lower than UCB and SDSC. On the other hand, the resource's average *Unavailability Interval* in LRI is 3.756 hours. Overall, the total number of *Unavailability Events* is 390, which means all resources have about 55.714 *Unavailability Events* on average in a day and each resource has about 0.203 *Unavailability Events* on average in a day, which is much lower than the number of *Unavailability Events* in UCB and SDSC.

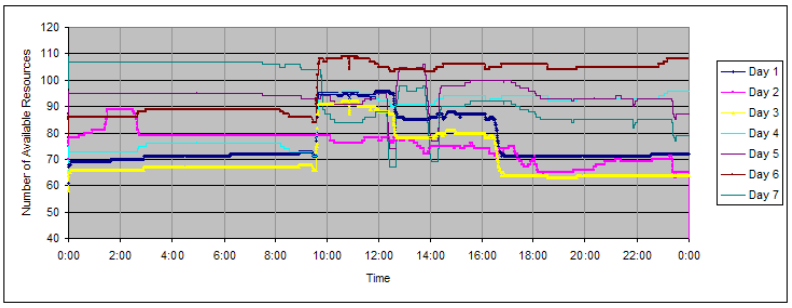


Figure 6-5: Number of Available Resources for Different Days (LRI)

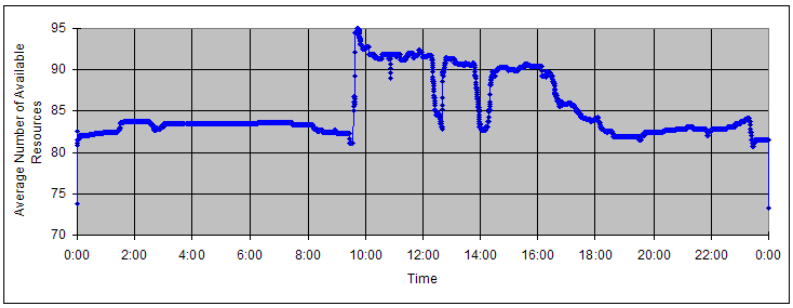


Figure 6-6: Average Number of Available Resources over Time (LRI)

There are 680 resources in data set DEUG and 7 day's data are provided in [Kondo05]. Figure 6.7 shows the number of available resource over time in 7 days and Figure 6.8 shows the average number of available resource over time in this data set. Overall, the number of available resources varies between 57 and 184 over time and the average number of available resources varies between 89.3 and 149.

The variance value of average number of available resources over time is 254.822, which indicates the average number of available resource over time is very unstable. According to [Kondo05], resources' *Average Availability Interval* in DEUG is 0.477 hours, which indicates the number of *Unavailability Events* is lower than UCB. On the other hand, resource's average *Unavailability Interval* in DEUG is 0.357 hours. Overall, the total number of *Unavailability Events* is 10764, which means all resources have about 1537.714 *Unavailability Events* on average in a day and each resource has about 2.261 *Unavailability Events* in a day on average, which is lower than the number of *Unavailability Events* in UCB but higher than SDSC and LRI.

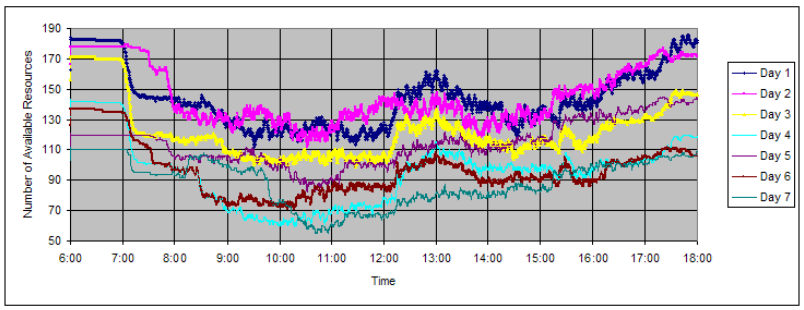


Figure 6-7: Number of Available Resources for Different Days (DEUG)

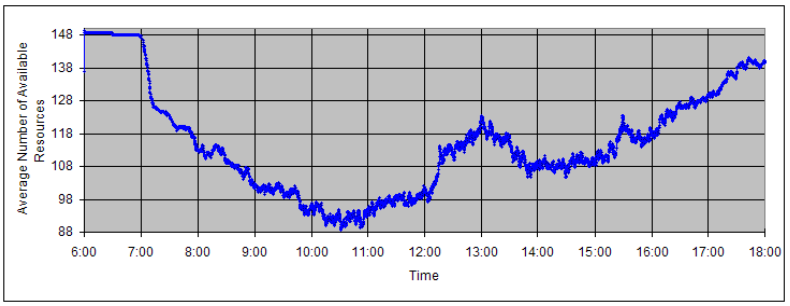


Figure 6-8: Average Number of Available Resources over Time (DEUG)

According to the above, each data set has its own characteristics. For example, they have quite different *Average Availability Interval* and *Average Unavailability Interval*. For comparison, some important results are summarised in Table 6-1.

Name of data set	Total number of resources	Number of days analysed	Length of a data trace (hours)	*Average Availability Interval* (hours)	*Average Unavailability Interval* (hours)	Variance of available resources over time	Average number of *Unavailability Events* for per day for each resource
UCB	80	10	7 (10am to 5pm)	0.166	0.119	2.719	7.595
SDSC	244	7	8 (9am to 5pm)	2.034	1.256	54.375	1.388
LRI	275	7	24	23.535	3.756	12.801	0.203
DEUG	680	7	12 (6am to 6pm)	0.477	0.356	254.822	2.261

Table 6-1: UCB, SDSC, LRI and DEUG Data Sets

6.1.4 *Job Execution Availability* Correlations

Some job-scheduling and job migration algorithms proposed in Chapter 4 are based on the TDE prediction method (described in Section 4.2.1). As discussed in Chapter 5, the accuracy of the TDE prediction method is highly reliant on the similarity of *Job Execution Availability* between the *Checking Period* (the period of availability history used for prediction) and *Prediction Period* (the period to predict). If resources' *Job Execution Availability* in the *Prediction Period* is similar to the availability in the *Checking Period*, then the prediction will be accurate and helpful for job scheduling. Therefore, to check if each single resource has such predictable pattern is important.

As mentioned in Section 5.2.2, Pearson product-moment correlation coefficient (PMCC) [Rodgers88] [Stigler89] is a useful tool to find out the relationships between different days' availability pattern for each single resource. If we have a series of n measurements of X and Y written as x_i and y_i where $i = 1, 2, ..., n$, then the Pearson product-moment correlation coefficient can be used to estimate the correlation of X and Y" [PMCC10]. So in the case of prediction, X can be considered as the *Checking Period* and Y can be considered as the *Prediction Period*. If the series values of x_i and y_i have strong relationship, then the results of PMCC will close to +1 or -1. On the other way round, if PMCC is close to +1 or -1, then X and Y will have strong linear relationship. In other word, If PMCC is close to +1 or -1, then resources' availability is more predictable, and prediction results are more useful for job scheduling. As discussed in Section 6.1.3, the downloaded data trace files not only records resources' *CPU Availability*.

Therefore, to calculate the *Job Execution Availability*, the trace files have been added in a new column to show the value of *Job Execution Availability* clearly. Here is an example:

Start time	End time	CPU cycles Delivered to Grid	Job Execution Availability
0	1	110653	1
1	2	110653	1
2	3	110650	1
3	4	110653	1
4	5	110653	1
5	6	110653	1
6	7	-1	0
7	8	-1	0
8	9	-1	0
9	10	-1	0

If the value of *CPU Availability* is above 0, then *Job Execution Availability* is true and it is represented by the value of 1. If the value of *CPU Availability* is below 0, then *Job Execution Availability* is false and the value of *Job Execution Availability* is 0. A resource's *Job Execution Availability* in a period (e.g. 1 hour, 1 day) can be considered as a series S_i. Therefore, the correlation between different periods can be calculated. Based on this idea, each the correlation coefficient values of each resource's *Job Execution Availability* in different days and different hours were calculated.

In brief, the correlation coefficient values between different days were calculated using the following method: Firstly, a single resource's availability traces are selected. As mentioned above, each resource has n days' traces and each day's trace is in a separate file. Next, each day's trace is considered as a *Daily Series* and then the relationships between a pair of series are calculated. For example, assume resource i's availability traces a picked, S_i^{day1}, S_i^{day2} and so on. Here, S_i^{day1} means resource i's *Daily Series* in day 1, S_i^{day2} means resource i's *Daily Series* in day 2 and so on. So PMCC of paired series are calculated, e.g. (S_i^{day1}, S_i^{day3}), (S_i^{day2}, S_i^{day5}) and so on. PMCC results of different days are calculated by the Equation 5.1.

The correlation coefficient values between different hours were calculated with the following method: firstly, a single resource's availability traces are picked at a time. Each resource has n days' traces and each day's trace is in a separate file, which is called *Daily Series*. A *Daily Series* is composed of a couple of hours' data (e.g. 9am to 5pm) and each hour's data is considered as an *Hourly Sub-Series*. Then the relationship between a pair of *Hourly Sub-Series* is calculated. For example, assume the resource i's availability traces picked are S_i^{day1}, S_i^{day2} and so on. S_i^{day1} is composed of $S_i^{hour1\ of\ day1}$, $S_i^{hour2\ of\ day1}$, etc and S_i^{day2} is composed of $S_i^{hour1\ of\ day2}$, $S_i^{hour2\ of\ day2}$, etc. So PMCC of paired *Hourly Sub-Series* are calculated, e.g. ($S_i^{hour1\ of\ day1}$, $S_i^{hour1\ of\ day2}$), ($S_i^{hour3\ of\ day2}$, $S_i^{hour3\ of\ day5}$) and so on. PMCC results of different hours are calculated via Equations 5.1 to 5.3.

With the PMCC equations, some results about resource availability correlations between different days and different hours of a resource have been calculated for the downloaded data

sets of UCB, SDSC, LRI and DEUG and important results are summarised. The aim of these correlation calculations is to find out if there is any strong correlation between different days/hours for each single resource.

Note here the results of PMCC calculations only include correlations of different data traces within a certain resource, not including correlations of different data traces between different resources. The reason is because the latter type of calculation has been done by [Kondo05], which is briefly described in Section 6.1.3.

Due to the limitations of PMCC calculation method, it is impossible to calculate PMCC results for some *Daily Series* or *Hourly Sub-Series* and these *Daily Series* or *Hourly Sub-Series* are excluded from the PMCC calculation. For a *Daily Series* or an *Hourly Sub-Series*, its standard deviation can be 0 (such a *Daily Series* is called *Zero Standard Deviation Daily Series* and such an *Hourly Sub-Series* is called *Zero Standard Deviation Hourly Sub-Series*) as the value of *Job Execution Availability* does not change throughout the whole series. For example, a resource is not available in a certain day so that *Job Execution Availability* is in that day will be always 0. Such a *Zero Standard Deviation Daily Series* is called *Zero Standard Deviation Unavailable Daily Series*. Similarly, a resource is not available in a certain hour so that *Job Execution Availability* is in that hour will be always 0. Such a *Zero Standard Deviation Hourly Sub-Series* is called *Zero Standard Deviation Unavailable Hourly Sub-Series*. On the other hand, if a resource is available in a certain day so that *Job Execution Availability* is in that day will be always 1. Such a *Zero Standard Deviation Daily Series* is called *Zero Standard Deviation Available Daily Series*. Similarly, a resource is available in a certain hour so that *Job Execution Availability* is in that hour will be always 1. Such a *Zero Standard Deviation Hourly Sub-Series* is called *Zero Standard Deviation Available Hourly Sub-Series*.

Zero Standard Deviation Daily Series and *Zero Standard Deviation Hourly Sub-Series* make it impossible to calculate PMCC result with Equation 5.1 as the result of standard deviation in Equation 5.2 is the denominator in Equation 5.1. Therefore, these *Daily Series* and *Hourly Sub-Series* are excluded from the PMCC calculation.

In UCB, each resource has 10 days' data traces so there are 10 data traces to be calculated for each resource. Each data trace is called a *Daily Series*. There are 80 resources in the data set of UCB. Overall, there are 800 (80 * 10) series for all 80 resources in the UCB data set. In these 800 *Daily Series*, 193 *Daily Series* are *Zero Standard Deviation Daily Series*. Therefore, these 193 *Zero Standard Deviation Daily Series* are not valid and excluded from the PMCC calculations while other 607 series (such a *Daily Series* is called *Non-zero Standard Deviation Daily Series*) were used.

Overall, the PMCC results of valid paired *Non-zero Standard Deviation Daily Series* range from -0.59742 to 0.84117 with the mean value 0.06878. Figure 6.9 shows the range of PMCC results.

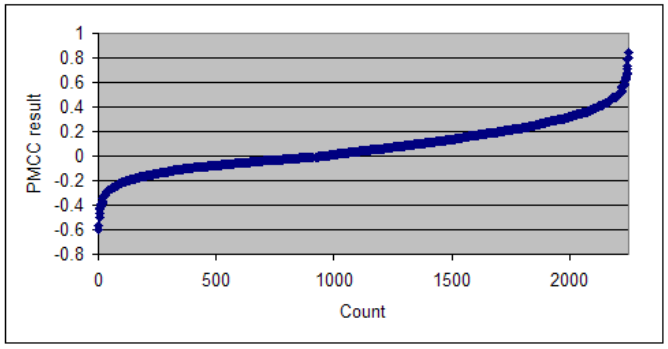


Figure 6-9: *Daily Series* PMCC Range of Results in UCB

There are a number of ways to interpret the PMCC results in [Kumar06][Simon05] [Correlation10][VSS10]. [Cohen88] argues that all criteria are in some ways arbitrary and should not be observed too strictly. However, according to [VSS10], "As a rule of thumb, correlation coefficients between .00 and .30 are considered weak, those between 0.30 and 0.70 are moderate and coefficients between .70 and 1.00 are considered high". Therefore, the PMCC interpretation below will use these criteria: if absolute value of PMCC is below 0.3, then the correlation is considered as low. If the absolute value of PMCC is between 0.3 and 0.7, then the correlation is considered medium. If the absolute value of PMCC is between 0.7 and 1, then the correlation is considered high.

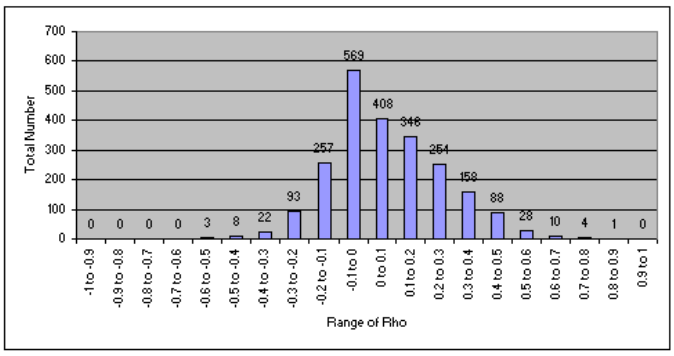


Figure 6-10: *Daily Series* PMCC Distribution in UCB

Overall, according to Figure 6.10, 85.68% (1927 PMCC results out of 2249) of the absolute values of PMCC results are small (below 0.3). 14.10% (317 PMCC results out of 2249) is medium (not smaller than 0.3 and below 0.7) and only 0.22% (5 PMCC results out of 2249) is large (between 0.7 and 1). Therefore, in data set UCB, most correlations between different *Non-zero Standard Deviation Daily Series* are low for each resource, which means the resources' *Non-Zero Standard Deviation Daily Series* tend to be independent on different days.

In UCB, 7 hours' data (from 10am to 5pm) were recorded for each resource in each day. Therefore, a *Daily Series* can be divided into 7 *Hourly Sub-Series*. Each single resource is available for 10 days so each resource has 70 (10*7) *Hourly Sub-Series* and all 80 resources have 5600 (80*10*7). In these 5600 *Hourly Sub-Series*, 3057 *Hourly Sub-Series*' are *Zero Standard Deviation Hourly Sub-Series* and these 3057 sub-series were excluded from the PMCC calculations and 2543 *Hourly Sub-Series* (such an *Hourly Sub-Series* is called *Non-zero Standard Deviation Hourly Sub-Series*) were used.

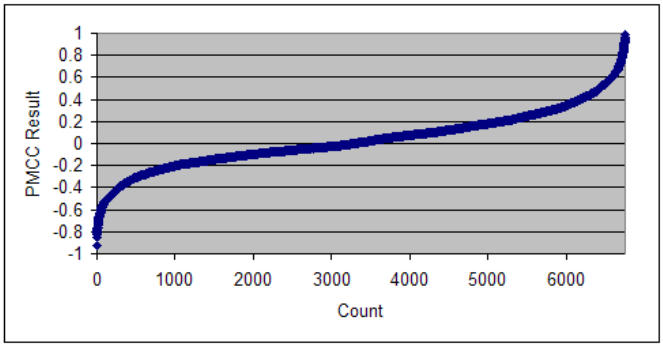


Figure 6-11: *Hourly Sub-Series* PMCC Range of Results in UCB

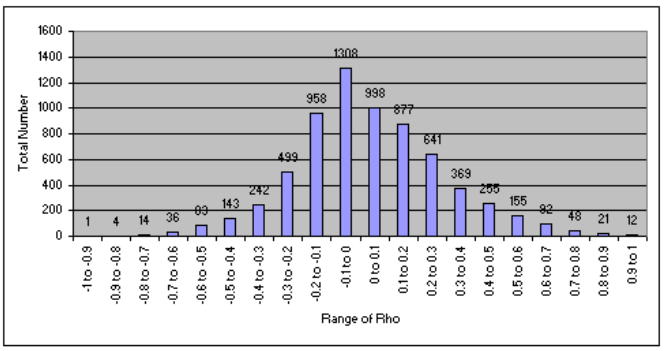


Figure 6-12: *Hourly Sub-Series* PMCC Distribution in UCB

According to Figure 6.12, in UCB, 78.17% (5281 PMCC results out of 6756) of all PMCC results' absolute value is below 0.3 (which are considered as low correlations), 20.35% (1375 PMCC results out of 6756) is below 0.7 (which are considered as medium correlations) and 1.48% (100 PMCC results out of 6756) is between 0.7 and 1 (which as considered as high correlations). Therefore, in data set UCB, most correlations between different *Non-zero Standard Deviation Hourly Sub-Series* are low for each resource, which means the resources' *Non-Zero Standard Deviation Hourly Sub-Series* tend to be independent in different hours.

There are 244 resources in data set of SDSC and each resource has 7 days' data traces. For a resource, Not every single resource is available for 7 days and there are 1708 data traces (*Daily Series*) overall for 244 resources. In these 1708 *Daily Series*, 755 *Daily Series* are *Zero Standard Deviation Series*. Therefore, all these 755 *Daily Series* were excluded in the PMCC calculations while the other 953 *Non-zero Standard Deviation Series* were included.

Overall, the PMCC results of *Non-zero Standard Deviation Daily Series* range from -0.84933 to 0.98897 with the mean value 0.03628. Similar to UCB, the mean is positive but close to 0. Therefore, the correlation between different days' availability pattern is low and positive on average in SDSC. Figure 6.13 shows the range of PMCC results.

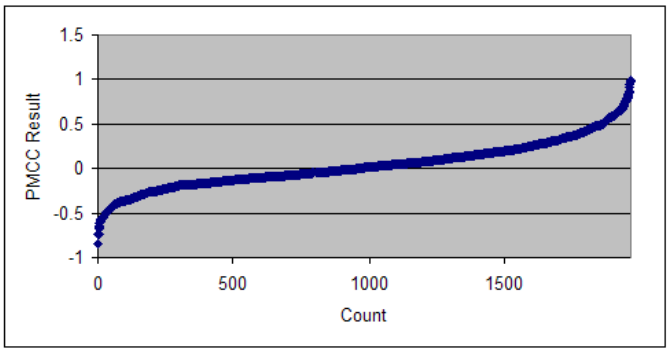


Figure 6-13: *Daily Series* PMCC Range of Results in SDSC

According to Figure 6.14, in SDSC, 76.79% (1509 PMCC results out of 1965) of the absolute value of PMCC is below 0.3. 21.68% (426 PMCC results out of 1965) is not smaller than 0.3 and below 0.7 and only 1.53% (30 PMCC results out of 1965) is between 0.7 and 1. Therefore, in data set SDSC, most correlations between different *Non-zero Standard Deviation Daily Series* are low for each resource, which means the resources' *Non-zero Standard Deviation Daily Series* tends to be independent in different days.

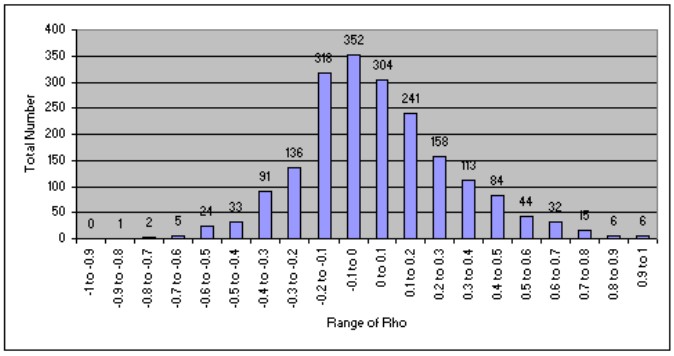


Figure 6-14: *Daily Series* PMCC Distribution in SDSC

In data set SDSC, 8 hours' data (from 9am to 5pm) were recorded for each resource in each day. Therefore, a *Daily Series* can be divided into 8 *Hourly Sub-Series*. As mentioned above, not every single resource is available for 7 days and there are 1708 data traces (*Daily Series*). Therefore, there are 13664 (1708*8) *Hourly Sub-Series* for all 244 resources. In these 13664 *Hourly Sub-Series*, 11253 *Hourly Sub-Series*' are *Zero Standard Deviation Hourly Sub-Series*. Therefore, these 11253 *Hourly Sub-Series* were excluded from the PMCC calculations and other 2411 *Non-zero Standard Deviation Hourly Sub-Series* were used.

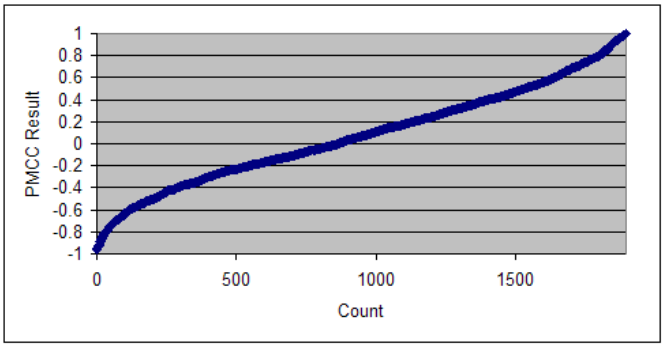


Figure 6-15: *Hourly Sub-Series* PMCC Range of Results in SDSC

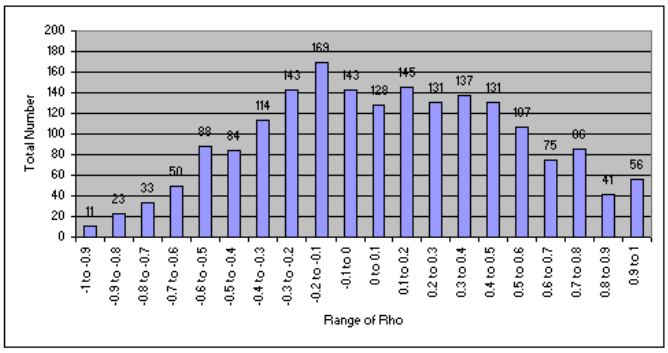


Figure 6-16: *Hourly Sub-Series* PMCC Distribution in SDSC

According to Figure 6.16, in SDSC, 45.33% (859 PMCC results out of 1895) of all PMCC results' absolute value is not larger than 0.3 (which are considered as low correlations), 41.48% (786 PMCC results out of 1895) is not larger than 0.7 (which are considered as medium correlations) and 13.19% (250 PMCC results out of 1895) is between 0.7 and 1 (which is considered as high correlations). Therefore, in SDSC, most correlations between different *Hourly Sub-Series* are not high (86.81% results are below 0.7), for each resource, which means the resources' *Non-zero Standard Deviation Hourly Sub-Series* tend to low small or medium correlations in different hours.

There are 275 resources in the data set of LRI and each resource has 7 days' data traces. Therefore, there are 1925 (275*7) data traces (*Daily Series*) overall for 275 resources. In these 1925 *Daily Series*, 1444 *Daily Series* are *Zero Standard Deviation Series*. Therefore, these 1444 Daily Series were excluded from the PMCC calculations and other 481 *Daily Series* were included in the PMCC calculations.

Overall, the PMCC results of these *Non-Zero Standard Deviation Daily Series* are ranged from -0.98692 to 0.69623 with the mean value -0.19927. Different from UCB and SDSC, the mean of PMCC in LRI is negative so this means the correlation between different days' availability pattern is a decreasing linear relationships. In addition, the result is also close to 0. So this means the correlation between different days' availability pattern tend to be small and negative on average in LRI. Figure 6.13 shows the range of PMCC results:

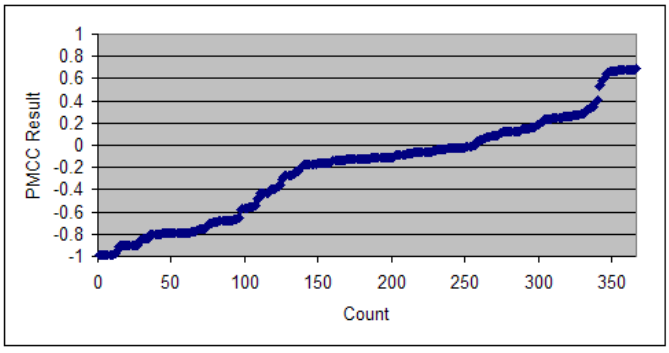


Figure 6-17: *Daily Series* PMCC Range of Results in LRI

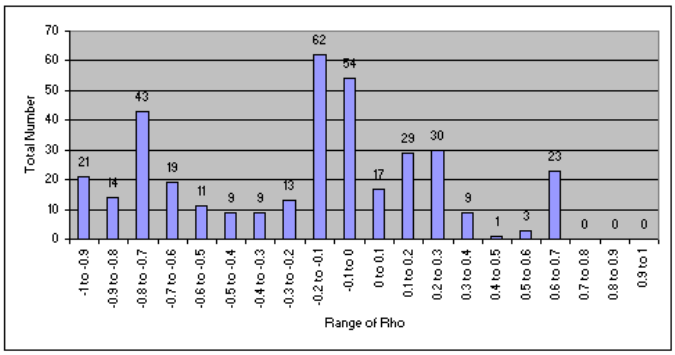


Figure 6-18: *Daily Series* PMCC Distribution in LRI

Accordingly Figure 6.18, in LRI, 55.86% (205 PMCC results out of 367) of the absolute value of PMCC is below 0.3. 22.89% (84 PMCC results out of 367) is not smaller than 0.3 and below 0.7 while 21.25% (78 PMCC results out of 367) is between 0.7 and 1. Therefore, in data set LRI, most correlations between different *Daily Series* are low for each resource, which means the resources' *Non-zero Standard Deviation Daily Series* tends to have low or medium correlations in different days.

In data set LRI, 24 hours' data were recorded for each resource in each day. Therefore, a *Daily Series* can be divided into 24 *Hourly Sub-Series*. Therefore, there are 46200 (1925*24) *Hourly Sub-Series* for all 275 resources. In these 46032 *Hourly Sub-Series*, 45498 *Hourly Sub-Series*' are *Zero Standard Deviation Hourly Sub-Series*. Therefore, these 45498 sub-series were excluded from the PMCC calculations and other 702 sub-series were used.

Figure 6.19 shows the range of PMCC results:

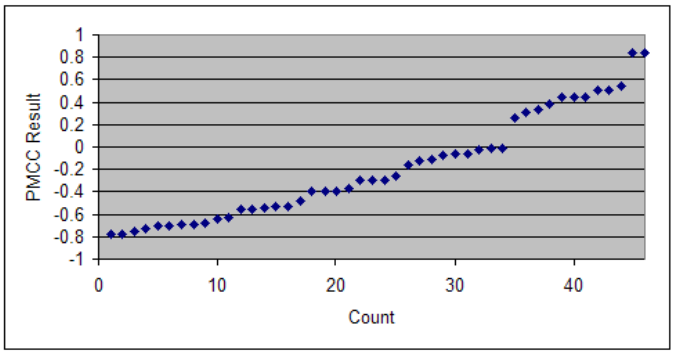


Figure 6-19: *Hourly Sub-Series* PMCC Range of Results in LRI

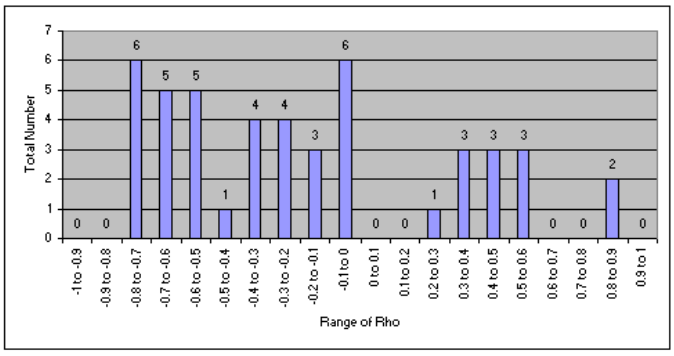


Figure 6-20: *Hourly Sub-Series* PMCC Results Distribution in LRI

According to Figure 6.20, in LRI, 30.43% (14 PMCC results out of 46) of all PMCC results'

absolute value is not larger than 0.3 (which are considered low correlations), 52.17% (24 PMCC results out of 46) is not larger than 0.7 (which are considered as medium correlations) and 17.39% (8 PMCC results out of 46) is between 0.7 and 1 (which is considered as high correlations). Therefore, in data set LRI, most correlations between different *Hourly Sub-Series* are not high (82.60% results are below 0.7) for each resource, which means the resource *Non-Zero Standard Deviation Hourly Sub-Series* tends to have low or medium correlations in different hours.

There are 680 resources in the data set of DEUG and each resource has at most 7 days' data traces. Therefore there are 4760 (680*7) data traces (*Daily Series*) overall for 680 resources. In these 4760 *Daily Series*, 3041 Daily Series are *Zero Standard Deviation Series*. Therefore, these 3041 Daily Series are excluded from the PMCC calculation and other 1719 *Daily Series* were included in the PMCC calculations.

Overall, the PMCC results of these *Non-zero Standard Deviation Daily Series* range from -0.99302 to 0.99305 with the mean value -0.06560. In DEUG, the mean of PMCC in DEUG is negative so this means the correlation between different days' availability pattern is a decreasing linear relationships. It is a low correlation on average as it is close to 0. Figure 6.21 shows the range of PMCC results.

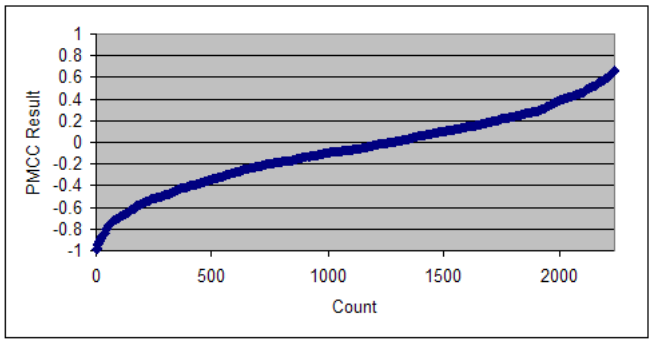


Figure 6-21: *Daily Series* PMCC Range of Results in DEUG

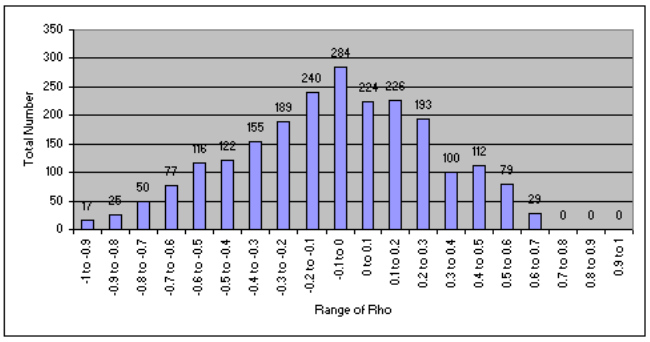


Figure 6-22: *Daily Series* PMCC Results Distribution in DEUG

According to Figure 6.22, in DEUG, 60.59% (1356 PMCC results out of 2238) of the absolute value of PMCC is below 0.3. 35.30% (790 PMCC results out of 2238) is between 0.3 and 0.7 and only 4.11% (92 PMCC results out of 2238) is between 0.7 and 1.

In data set DEUG, 12 hours' data were recorded (6am to 6pm) for each resource in each day. Therefore, a *Daily Series* can be divided into 12 *Hourly Sub-Series*. Therefore, there are 57120

(4760*12) *Hourly Sub-Series* for all 680 resources. In these 57120 *Hourly Sub-Series*, 52244 *Hourly Sub-Series*' are *Zero Standard Deviation Hourly Sub-Series*. Therefore, these 52244 sub-series were excluded from the PMCC calculations and other 4876 *Non-Zero Standard Deviation Sub-Series* were used.

Figure 6.23 and Figure 6.4 show the range of PMCC results and their distribution, respectively.

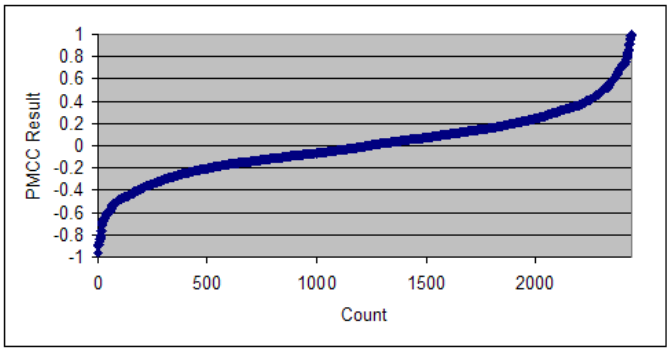


Figure 6-23: *Hourly Sub-Series* PMCC Range of Results in DEUG

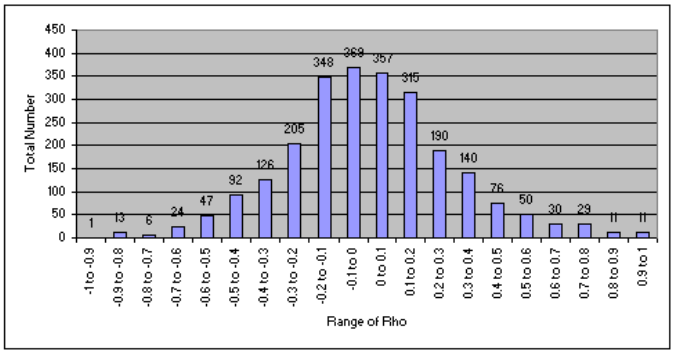


Figure 6-24: Hourly Sub-Series PMCC Results Distribution in DEUG

According to the PMCC results shown above, the key points are as follows:

Firstly, the results show that each data set has its distinct features and the PMCC results have different distributions.

Secondly, though each data set has different features and PMCC results, the PMCC results indicate that the correlations between different *Non-zero Standard Deviation Daily Series* and *Non-zero Standard Deviation Hourly Sub-Series* are generally not high (over 0.7). This means the resources' *Non-zero Standard Deviation Daily Series* and *Non-zero Standard Deviation Hourly Sub-Series* tends to have low (or even no) correlations in different days. In comparison, some important results are summarised in Table 6-2 and 6-3.

Name of data set	Total number of *Non-zero Standard Deviation Daily Series*	Minimum PMCC	Maximum PMCC	PMCC mean	Percentage of low correlation (|PMCC| < 0.3)	Percentage of medium correlation (|PMCC| > 0.3|PMCC| < 0.7)	Percentage of high correlation (|PMCC| >= 0.7)
UCB	607	-0.59742	0.84117	0.06878	85.68%	14.10%	0.22%
SDSC	953	-0.84933	0.98897	0.03628	76.79%	21.68%	1.53%
LRI	481	-0.98692	0.69623	-0.19927	55.86%	22.89%	21.25%
DEUG	1719	-0.99302	0.99305	-0.06560	60.59%	35.30%	4.11%

Table 6-2: *Non-zero Standard Deviation Daily Series* PMCC Results

Name of data set	Total number of *Non-zero Standard Deviation Hourly Sub-Series*	Minimum PMCC	Maximum PMCC	PMCC result mean	Percentage of low correlation (|PMCC| < 0.3)	Percentage of medium correlation (|PMCC| > 0.3|PMCC| < 0.7)	Percentage of high correlation (|PMCC| >= 0.7)
UCB	2543	-0.92733	0.99070	0.03264	78.17%	20.35%	1.48%
SDSC	2411	-0.96346	0.99827	0.07701	45.33%	41.48%	13.19%
LRI	702	-0.77787	0.84494	-0.18391	30.43%	52.17%	17.39%
DEUG	4876	-0.96379	0.99760	0.00781	73.11%	23.98%	2.91%

Table 6-3: *Non-zero Standard Deviation Hourly Sub-Series* PMCC Results

Day Interval means the intervals of days between two *Daily Series*. For example, for (S_i^{day2}, S_i^{day5}), *Day Interval* is 5–2 =3. For (S_i^{day1}, S_i^{day3}), *Day Interval* is 3-1=2. *Rho Mean* calculates the average value of the PMCC results which having the same *Day Interval*. For example, for *Rho Mean* of *Day Interval* 8, it aggregates PMCC results of (S_i^{day1}, S_i^{day9}) and $(S_i^{day2}, S_i^{day10})$ and then calculates the average.

To find out if there is any strong correlation between particular days (e.g. if two *Daily Series* have strong correlation when *Day Interval* is 5 as they are the same day in different weeks), *Rho Mean* of different *Day Interval* were calculated and summarised in Table 6-4.

Name of data set	Rho Mean of *Day Interval* X								
	X=1	**X=2**	**X=3**	**X=4**	**X=5**	**X=6**	**X=7**	**X=8**	**X=9**
UCB	0.05030	0.08312	0.06331	0.06440	0.12774	0.03914	0.08232	0.01462	0.07289
SDSC	0.04604	0.07648	0.06260	0.05324	-0.13735	0.06893	N/A	N/A	N/A
LRI	-0.01148	-0.37804	-0.32262	0.11189	-0.26969	-0.75934	N/A	N/A	N/A
DEUG	-0.08830	-0.02766	-0.02788	0.097392	-0.02233	0.05184	N/A	N/A	N/A

Table 6-4: Rho Mean of Different *Day Intervals*

According to the *Day Interval* results shown above, the key points are as follows:

Firstly, the results show that the correlations between particular days are different in each data set.

Secondly, though the correlations are different, the results indicate the correlations are generally low (below 0.3) no matter what value *Day Interval* it is. This indicates that for a resource, each day's *Non-zero Standard Deviation Daily Series* tends to be independent from any other days.

To find out if there is any regular pattern in *Zero Standard Deviation Daily Series* and *Zero Standard Deviation Hourly Sub-Series*, another approach was taken. Assuming a given type of *Zero Standard Deviation Daily Series* (either *Zero Standard Deviation Available Daily Series* or *Zero Standard Deviation Unavailable Daily Series*), two questions are raised. The first one is "is this type of *Zero Standard Deviation Daily Series* or *Zero Standard Deviation Hourly Sub-Series* going to occur again in the following days?" and the second one is "if the answer to the first question is yes, then what is the probability that it will occur again tomorrow (the day after tomorrow, the same day next week, etc)?" For example, assuming a resource has a *Zero Standard Deviation Available Daily Series* today (the resource is always available so its *Job Execution Availability* is always 1 today), is it possible that the resource is going to have another *Zero Standard Deviation Available Daily Series* (the same type of series) in the following days? If it is possible, then what is the probability (called *Same Type Series Occurrence Probability*) that this type of *Zero Standard Deviation Available Daily Series* will occur again after X days (e.g. after 1 day, 7 days, etc)?" If these two questions can be answered, then some relationships between *Zero Standard Deviation Daily Series* and *Zero Standard Deviation Hourly Sub-Series* can be found and this will be helpful for job-scheduling algorithms. For example, if a resource has a *Zero Standard Deviation Available Daily Series* today, and the *Same Type Series Occurrence Probability* after 1 day is 100%, then the job-scheduling algorithm will know this resource will be very reliable tomorrow based on today's observation.

Therefore, to answer these two questions, some statistical results from each data set are collected for *Zero Standard Deviation Daily Series* and *Zero Standard Deviation Hourly*

Sub-Series. Let Pr(X) denotes the result of *Same Type Series Occurrence Probability*; Pr(X) can be calculated by the following equation:

$$Pr(X) = S(X) / T(X) \quad \text{(Equation 6.1)}$$

where S(X) denotes total occurrence times of the same type series after X days and T(X) denotes total occurrence times of all types' series after X days.

Here, for *Daily Series*, all types' series includes *Zero Standard Deviation Available Daily Series*, *Zero Standard Deviation Unavailable Daily Series* and *Non-Zero Standard Deviation Daily Series*. For *Hourly Sub-Series*, all types' series includes *Zero Standard Deviation Available Hourly Sub-Series*, *Zero Standard Deviation Unavailable Hourly Sub-Series* and *Non-Zero Standard Deviation Hourly Sub-Series*. Table 6-5 through to Table 6-8 summarise some statistical results relating to the *Same Type Series Occurrence Probability*.

Name of data set	Total number of *Non-zero Standard Deviation Available Daily Series*	Same Type Series Occurrence Probability after X day(s)								
		X=1	X=2	X=3	X=4	X=5	X=6	X=7	X=8	X=9
UCB	193	52.87%	41.72%	36.24%	29.32%	33.04%	36.00%	30.86%	27.27%	37.21%
SDSC	267	50.40%	30.20%	15.45%	0.00%	0.00%	0.00%	N/A	N/A	N/A
LRI	315	37.96%	38.03%	35.85%	45.71%	31.91%	43.48%	N/A	N/A	N/A
DEUG	91	18.30%	29.62%	8.33%	0.00%	0.00%	0.00%	N/A	N/A	N/A

Table 6-5: *Zero Standard Deviation Available Daily Series* Results

Name of data set	Total number of *Non-zero Standard Deviation Unavailable Daily Series*	Same Type Series Occurrence Probability after X day(s)								
		X=1	X=2	X=3	X=4	X=5	X=6	X=7	X=8	X=9
UCB	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
SDSC	488	78.05%	68.58%	61.42%	57.65%	48.91%	54.43%	N/A	N/A	N/A
LRI	1129	83.84%	77.00%	67.29%	62.50%	63.23%	45.00%	N/A	N/A	N/A
DEUG	2950	81.36%	72.64%	69.60%	71.06%	71.06%	77.29%	N/A	N/A	N/A

Table 6-6: *Zero Standard Deviation Unavailable Daily Series* Results

Name of data set	Total number of *Non-zero Standard Deviation Available Hourly Sub-Series*	Same Type Series Occurrence Probability after X day(s)								
		X=1	X=2	X=3	X=4	X=5	X=6	X=7	X=8	X=9
UCB	3052	66.61%	65.11%	63.00%	61.70%	65.10%	60.31%	63.51%	59.57%	64.57%
SDSC	4247	79.86%	77.02%	71.39%	62.38%	77.19%	54.90%	N/A	N/A	N/A
LRI	14030	95.73%	94.69%	94.67%	94.78%	93.90%	94.80%	N/A	N/A	N/A
DEUG	6755	74.31%	74.14%	75.07%	75.08%	75.43%	76.47%	N/A	N/A	N/A

Table 6-7: *Zero Standard Deviation Available Hourly Sub-Series* Results

Name of data set	Total number of *Non-zero Standard Deviation Unavailable Hourly Sub-Series*	Same Type Series Occurrence Probability after X day(s)								
		X=1	X=2	X=3	X=4	X=5	X=6	X=7	X=8	X=9
UCB	5	0.00%	0.00%	0.00%	0.00%	0.00%	N/A	0.00%	N/A	N/A
SDSC	7006	82.56%	75.95%	73.58%	71.16%	65.77%	59.70%	N/A	N/A	N/A
LRI	31468	99.03%	98.40%	97.74%	97.20%	97.56%	96.12%	N/A	N/A	N/A
DEUG	45489	93.84%	93.47%	93.22%	93.87%	94.07%	95.59%	N/A	N/A	N/A

Table 6-8: *Zero Standard Deviation Unavailable Hourly Sub-Series* Results

According to the results shown above *Same Type Series Occurrence Probability*, the key points can be summarised as follows:

Firstly, in general, the statistical results show that the *Same Type Series Occurrence Probability* varies from one data set to another.

Secondly, in some data sets, the results of *Same Type Series Occurrence Probability* are low whilst other data sets have high results of *Same Type Series Occurrence Probability*. In some data sets, the result of *Same Type Series Occurrence Probability* after 1 day tend to have a higher probability than the results *Same Type Series Occurrence Probability* after any other day in some data sets while other data sets have different results. Therefore, it is not straightforward to conclude that is *Zero Standard Deviation Daily Series* and *Zero Standard Deviation Hourly Sub-Series* are predictable or not in general and it is also not straightforward to conclude that which day will have a higher result of *Same Type Series Occurrence Probability* consistently.

Chapter 7 Simulation and Evaluation

This chapter provides details of the simulation set up together with evaluation results for the proposed job-scheduling and job migration algorithms in different scenarios.

7.1 Simulation Environment

The evaluations were carried out in a discrete-event simulation environment [DES10]. "In discrete-event simulation, the operation of a system is represented as a chronological sequence of events" and "each event occurs at an instant in time and marks a change of state in the system" [Robinson04]. It was developed by the author of this thesis and the source code was written in Delphi (Object-Pascal) programming language [Delphi10].

7.1.1 Components

There are five components in the simulation environment: *User*, *Grid Job Scheduler*, *Resource*, *Job* and *Event*. Figure 7.1 shows the structure of the simulation environment.

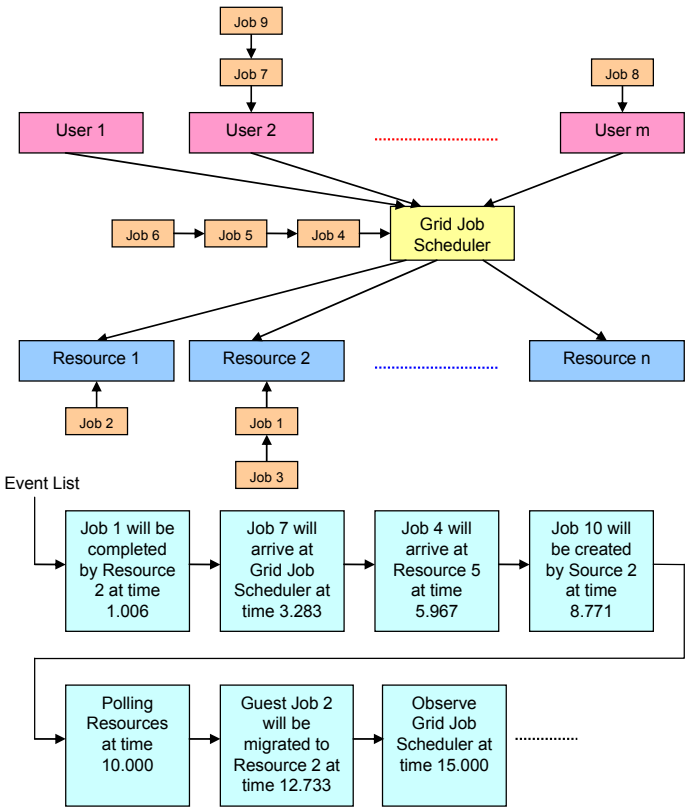


Figure 7-1: Structure of the Simulation Environment

- ***User***

The *User* is responsible for generating *Jobs* and sending them to the *Grid Job Scheduler*. It is defined as a pointer type data in the simulation code. The interval between generating two *Jobs* for a certain *User* can follow a certain probability distribution, e.g. uniform distribution, negative exponential distribution, normal distribution, etc. When a *User* generates a *Job*, it firstly initialises the parameters for the *Job*, like *Job* number, *Job* size and so on. Then the *User* will

send the generated *Job* to the *Grid Job Scheduler*. Later, an *Event* will be inserted into the *Event* list, which indicates what time the *Job* will arrive at that *Grid Job Scheduler*.

In the simulation code, the *User* has the following important attributes:

- *User* number: This index number identifies each *User*.
- *User* Probability Density Function (PDF) type: This attribute specifies the interval time between a *User* generating two *Jobs* follows. The PDF type may be uniform, exponential or any other PDF type.

- ***Grid Job Scheduler***

The *Grid Job Scheduler* is the core component in the simulation environment. It is defined as a pointer type data in the simulation code. It has the following functions:

1) It receives *Jobs* sent from the *User* and allocates the *Jobs* to *Resource* according to a certain algorithm. For example, the *Grid Job Scheduler* may assign *Jobs* to each *Resource* in turn with FCFS algorithm or it may assign *Jobs* to the *Resource* according to an AI mechanism based algorithm such as FLP. When the *Grid Job Scheduler* decides to allocate a *Job* to a certain *User*, it will insert an *Event* to the *Event* list, which indicates which *Resource* the *Job* will be sent to and what time the *Job* will arrive at the *Grid Job Scheduler*.

2) It observes the performance of *Resource* at regular interval and decides whether to migrate a *Job* from one *Resource* to another. Details about why and how to migrate a *Job* is described in the Chapter 5. The intervals between two observing time may be uniform or follow a probability like Normal distribution. The migration decision can be based on an algorithm introduced in Chapter 4. When the *Grid Job Scheduler* decides to migrate a *Job*, it will insert an *Event* to the *Event* list, which indicates which *Resource* the *Job* will be migrated to and what time the *Job* will arrive at that *Resource*.

In the simulation code, the *Grid Job Scheduler* has the following important attributes:

- *Resource* List: This is a single linked list specifying the *Resource* which the *Grid Job Scheduler* is connected with. Take Figure 7.1 for example, Grid job scheduler 1 is connected with *Resource* 1 to *Resource* n, so the linked list has n pointers specifies that it is connected with *Resource* 1 to *Resource* n.

- *Job* List: This is a single linked list specifying the *Jobs* which have arrived at the *Grid Job Scheduler* but not allocated to *Resource* yet. Take Figure 7.1 for example, *Job* 4, *Job* 5 and *Job* 6 have arrived at the *Grid Job Scheduler* and they are waiting for allocation.

- ***Resource***

The *Resource* is responsible for processing *Jobs* allocated by the *Grid Job Scheduler*. It is defined as a pointer type data in the simulation code. The *Resource* can handle multiple tasks at the same time and all the *Jobs* will be treated equally. Take Figure 7.1 for example, for *Resource*

1, there is only one *Job* running on it at the moment so the *Job* will occupy the CPU cycles until a new *Job* comes. For *Resource* 2, there are 2 *Jobs* running on it at the moment so these two *Jobs* will share the number of CPU cycles equally.

In the simulation code, the *Resource* has the following important attributes:

- *Resource* number: This index number identifies each *Resource*.
- *Resource* capability: This attribute specifies the CPU speed of a *Resource*. It is the *Resource*'s CPU cycles delivered to the Grid at a certain time.
- *Job* list: This is a single linked list that lists all the *Job*(s) being processed by the *Resource*.

- ***Event***

An *Event* is a discrete time during which something occurs within the simulation and an *Event* list is a single link-list that stores all known *Events*. The *Events* are listed in order of *Event* time. When the simulation is running, the execution engine will keep on picking the first *Event* in the *Event* list and invoking the corresponding *Event*. For example, if the first *Event* in the list is a *Job Creation Event*, the execution engine will invoke the procedure of *Job Creation* to generate a *Job* and the *Job* generator will be marked as the *User* that is specified in the *Event*. When the corresponding procedure is finished, the *Event* will be disposed of and the *Event* list header will point to the next *Event*.

As the simulation environment is discrete, the current time is the same as the time specified in the first *Event*. For example, the first *Event* in the *Event* list of Figure 7.1 is "*Job* 1 will be completed by *Resource* 2 at time 1.006", so when the execution engine picks this *Event*, the current time in the simulation is 1.006. Later, when this *Event* is finished and the second *Event* "*Job* 7 will arrive at *Grid Job Scheduler* at time 3.283", the current time in the simulation will become 3.823.

The *Event* list is changed along with the progress of simulation. During the simulation, some *Event*s will be executed and disposed of and some other *Event*s will be generated and added to the *Event* list as well. For example, when the first *Event* "*Job* 1 will be completed by *Resource* 2 at time 1.006" in Figure 7.1 finishes, *Job* 1 will be completed by *Resource* 2. As *Job* 1 is finished, *Job* 3 will occupy *Resource* 2 solely. Therefore, it is necessary to recalculate the completion time of *Job* 3 and insert an *Event* to the *Event* list to specify the time when *Job* 3 is going to be completed. Assume the completion time of *Job* 3 is 5.509, so the new *Event* "*Job* 3 will be completed by *Resource* 2 at time 5.509" will be created and inserted to the *Event* list after the *Event* "*Job* 7 will arrive at *Grid Job Scheduler* at time 3.283" and before the *Event* "*Job* 4 will arrive at *Resource* 5 at time 5.967".

There is a number of *Event* types used in the simulations as follows:

1) *Job Creation*: This *Event* is used to generate a *Job* at the specified time given in the *Event*. For example, when the *Event* "*Job* 10 will be created by *User* 1 at time 8.771" in Figure 7.1

becomes the first *Event* in the *Event* list, the execution engine will invoke the corresponding procedure(s) to generate a *Job* and the generator of the *Job* will be marked as *User* 1.

2) *Job Arrive at Grid Job Scheduler*: This *Event* is used to specify the time when a *Job* arrives at the *Grid Job Scheduler*. For example, when an *Event* "*Job* 7 will arrive at *Grid Job Scheduler* at 3.823" becomes the first *Event* in the *Event* list, the execution engine will invoke the corresponding procedure(s) to move *Job* 7 to *Grid Job Scheduler*'s *Job* list.

3) *Job Arrive at Resource*: This *Event* is used to specify the time when a *Job* arrives at a *Resource*. For example, when an *Event* "*Job* 4 will arrive at *Resource* 5 at time 5.967" becomes the first *Event* in the *Event* list, the execution engine will invoke the corresponding procedure(s) to move *Job* 4 to *Resource* 5's *Job* list.

4) *Job Completion*: This *Event* is used to specify the completion time for a *Job*. For example, the "*Job* 1 will be finished by *Resource* 2 at time 1.006" in Figure 7.1 is the first *Event* in the *Event* list, so the execution engine will invoke the corresponding procedures(s) to dispose the *Job* and update the stat information on *Resource* 1.

5) *Poll Resources*: This *Event* is used to specify the time to poll all *Resources*. The work includes updating some important information of each *Resource* (such as *Resource*'s current CPU speed, each job's completion time), collecting some statistical results (such as the number of *Jobs* completed/disposed since last observation) and check if the *Jobs* on the *Resource* need migration or not. When the *Event* "Polling *Resources* at time 10.000" becomes the first *Event* in the *Event* list, the execution invokes the procedure(s) to poll all *Resources* and make *Job* migration decisions if necessary. This kind of observation is typically carried out at regular interval.

6) *Job Migrated to Resource*: This *Event* is used to specify the time when a *Job* will be migrated and arrived at a new *Resource*. For example, when the *Event* "*Job* 2 will be migrated to *Resource* 2 at time 12.733" becomes the first *Event* in the *Event* list, the execution engine will invoke the corresponding procedure(s) to remove the *Job* 12 from *Resource* 2's *Job* list and add it to the *Resource* 1's *Job* list.

7) *Observe Grid Job Scheduler*: This *Event* is used to specify the time to observe the *Grid Job Scheduler*. The work includes triggering a *Job* allocation procedure for the first *Job* in the *Job* queue, collecting some statistical results (such as the number of *Jobs* allocated to *Resources* since last observation). For example, when the *Event* "Observe *Grid Job Scheduler* at time 15.000" becomes the first *Event* in the *Event* list, the execution engine will invoke the procedure(s) to observe the *Grid Job Scheduler*. Similar to *Poll Resources*, this kind of observation is carried at regular interval as well.

Each *Event* has the following attributes:

- *Event* time: This attribute specifies when the *Event* is going to happen.
- *Event* type: This attribute specifies the type of the *Event*. It is one of the seven *Event* types that are described earlier in this section.
- *Event* place: This attribute specifies where the *Event* is going to happen, including *User*, *Grid Job Scheduler* and *Resource*.
- *User* pointer: This pointer points to a *User* if the *Event* is related to that *User*.
- *Grid Job Scheduler* pointer: This pointer points to the *Grid Job Scheduler* if the *Event* is related to the *Grid Job Scheduler*.
- *Resource* pointer: This pointer points to a *Resource* if the *Event* is related to that *Resource*.
- *Job* pointer: This attribute specifies the *Job* pointer if the *Event* is related to that *Job*.

- ***Job***

The *Job* is generated by *User* and processed by the *Resource*. It is created dynamically and referenced via a pointer in the simulation code. It has the following important attributes:

- *Job* number: Each *Job* has a unique number.
- Current *Job* size: This attribute records the current *Job* size of the *Job*. It is represented by a numerical value (the number of CPU cycles to complete the job).
- *Job* Completion time: This attribute records the completion time of the *Job*.
- *Job User* number: This attribute records the *User* number that generated the *Job*.
- *Job Resource* number: This attribute records the *Resource* number if the *Job* has already allocated to a *Resource*.

7.1.2 General Evaluation Approach

In order to carry out evaluations of the proposed job-scheduling and job migration algorithms, a series of simulation experiments were carried out based on the simulation environment described in Section 7.1.1. To carry out simulation experiments, availability data traces recording resource *CPU Availability* are required. As discussed in Section 4.2.2, *CPU Availability* indicates current CPU speed (number of CPU cycles delivered to the Grid per second) and *Job Execution Availability* indicates whether jobs are allowed to run on the resource or not in that period. In generally, there are two approaches to obtain such data traces:

The first approach is to create synthetic data traces. In this approach, resources' availability data traces are created artificially. Data can be created quickly and any kind of simulation scenarios can be designed accordingly. For example, to simulate a reliable resource, the

resource's *CPU Availability* can be created to be always above 0. To simulate a powerful resource, the resource's *CPU Availability* can have very high values. For example, if a Grid is composed of reliable/powerful resources, then many reliable/powerful resources can be created. However, as the data traces are artificial, they may not be very realistic. Note, some simulations with synthetic data traces are provided in appendix B.

The second approach is to gather availability data traces from real Grid system(s). In this approach, the resource availability data traces are collected rather than created. Unlike the first approach, it is not straightforward to get data traces and to design simulation scenarios in this approach. To collect data traces, the resources have to be monitored and the *CPU Availability* data need to be recorded. In addition, it will be time consuming if a long trace is desired. As they are real data traces, to design simulation scenarios, they may have to be processed first. Without processing, the resource *CPU Availability* may not be readily available. For example, whether a resource is reliable or powerful is unknown before analysing the collected data traces. Therefore, compared with the first approach, it is a more complex and time-consuming. However, the data traces represent realistic cases. As each approach has advantages and disadvantages, both kinds of data were used in the simulation experiments.

In terms of synthetic data traces, they are easy to create but not easy to ensure they are sensible. Real data traces, are neither straightforward to collect nor easy to ensure that they are useful. Therefore, it is important to ensure the collected data traces are collected and processed in an appropriate way. Ideal resource data traces were discussed in [Kondo05][Kondo07]. In summary, the ideal trace should have the following characteristics:

Firstly, the trace should accurately record *CPU Availability* information throughout the observation period. Accurate information permits replaying the behaviour of the resource accurately, which is especially useful for debugging.

Secondly, the trace should not just record the *CPU Availability* when failures occur but also the reason for the failure. When a failure occurs, *CPU Availability* changes to -1 but it does not indicate what caused this failure. It could be caused by *Event*s like a resource shutting down, network connection failure and user reclaiming the resource. Therefore, if reasons for each failure are recorded, it facilitates analysis in terms of statistics and prediction.

Some assumptions of the simulation environments are also important:

First, the purpose of simulating with synthetic data is to validate the simulator or to explore influences brought about by different parameter settings. To achieve this, typically only a few resources are used in such simulations. The main reason for this is to keep the system as simple as possible and more clearly observe how different parameter settings influence the system performance. For example, if many resources are used and each of them has different patterns in terms of *CPU Availability*, then the system becomes complicated.

Secondly, the purpose of simulations with real data is to examine the influence of different

parameter settings and compare the performance of various job-scheduling algorithms in practical scenarios, both for non-prediction and prediction-based algorithms. Therefore, many resources are used in such practical scenarios.

Third, in the simulations with synthetic data, the simulations last the minimum number of days. The purpose of this is also to keep the system as simple as possible and clearly show how different parameters influence a single resource. In the first day, the TDE prediction method (used by the proposed job-scheduling algorithm) does not have any historical data of the resource so that it cannot make any prediction based on historical data. Therefore, the TDE prediction based algorithms (e.g. the proposed FCFSPP, FLP and PSOPP job-scheduling schemes) behave the same as non-prediction based methods (e.g. FCFS job-scheduling algorithm). However, in the second day of the simulation, the TDE prediction method now has access to historical data. With this data, it can make predictions for the *Prediction Period* by reviewing the *Checking Period* before job allocations in the second day. Therefore, the difference between TDE prediction based algorithms and the FCFS algorithm can be clearly observed.

Fourth, in simulations with both synthetic data and real data, the FCFS algorithm is used for comparison. This is mainly because of some of the proposed job-scheduling algorithms (FCFSPP and FLP) are based on it. Therefore, it is easy to observe the impact of TDE prediction. In addition, previous research [Kondo05][Kondo07] proposes a number of resource prioritisation/exclusion methods for job-scheduling in a Grid computing context, especially in the Grid context, where utilising volunteered and unreliable resource. According to their results, FCFS works relatively well in many scenarios, especially ones where the number of jobs are more than the number of resources.

Fifth, simulations are mainly focused on results not from the first simulation day. This is due to *Initial Bias*. *Initial Bias* means the system may be in a transition stage at the beginning stage of a simulation and the results gathered from this transition stage of simulation may distort the results gathered from steady stage. At the beginning stage of a simulation, the job queue of the Grid job scheduler is empty and many *available* resources are *idle*. Later, as new jobs join the job queue of Grid job scheduler and then being allocated to *available* resources, the simulation will gradually enter a steady state in which the job queue of Grid job scheduler may not be empty and many *available* resources may not be *idle*. Therefore, results gathered from the transition stage may be different from the results gathered from the steady stage. For example, if the simulation simulates a HTC environment, the job throughput at the transition stage may be lower than steady stage as there are not enough jobs for resources to process in the transition stage.

Sixth, in a simulation, each job is assumed to have the same job size (*Job Execution Time*) and the job scheduler when making a job allocation knows the size. This is for the purpose of

showing the influence brought by different parameters and factors clearly. If the job size is not fixed, the results gathered from a simulation may be affected. In such a case, the influence brought by different parameters and factors may be distorted by the influence of unfixed job size. However, the job size may vary from one simulation to another.

Seventh, in the simulation, jobs arrive at the Grid job scheduler at regular and fixed intervals. This is for the sake of simplicity and based on the following considerations:

If a simulation simulates a scenario in which *Workload* is high (the number of jobs are far more than resources), the main focus is on the number of jobs processed and failed in a given period. As resources will always have jobs to process, job arrival process will not influence these results.

If a simulation simulates a scenario where the *Workload* is low, the main focus is on the *Makespan* of each job and the proportion of processed and failed jobs. As the *Workload* is low and the job will not wait at the job queue, each job's *Makespan* is hardly influenced by job arrival process. In terms of processed and failed jobs, the main focus is on the proportion rather than the absolute values of processed and failed jobs. Here, the job arrival process does not influence the proportion either.

Eight, in each simulation, the time required to transfer a job from one component to another (e.g. from a user to the Grid job scheduler or from the Grid job scheduler to a resource) is assumed to be 0. In practise, it takes some time to transfer a job from one component to another in a Grid system and it may affect the results such as job throughput or each job's *Makespan*. Therefore, the job transmission time in these scenarios are assumed to be 0 to ensure the results are not affected by various transmission times.

Finally, there is some commonality to the experimental setup across the simulations. In the following sections, unless stated, the simulations use the setup shown in Table 7-1:

Name	Setting
Number of User	The value is 1
Number of Grid Job Scheduler	The value is 1
Job Sorting Algorithm	First-Come-First-Served
Number of Checking Days	The value is 1 day
Resource Availability Probability Threshold	The value is 100%
Job Size	The value is 10 minutes
Job Arrival Interval	The value is 1 minute
Length of *Checking Period*	The same as *Job Size*
Length of *Prediction Period*	The same as *Job Size*
Multiply Factor	The value is 1
Resource Availability Probability Threshold Adjustment Interval in FLP	The value is 10 minutes
Parameter λ *in FLP*	The value is 1
Number of Particles in PSOPP	The value is 10
The Predefined Goal in PSOPP	The fitness value is above 1
V_{max} in PSOPP	The value is 10
Parameter x in PSOPP algorithm	The value is 1
Parameter y in PSOPP algorithm	The value is 1

Table 7-1: Common Experimental Setup for Simulations

7.2 Evaluation of FCFSPP Algorithms

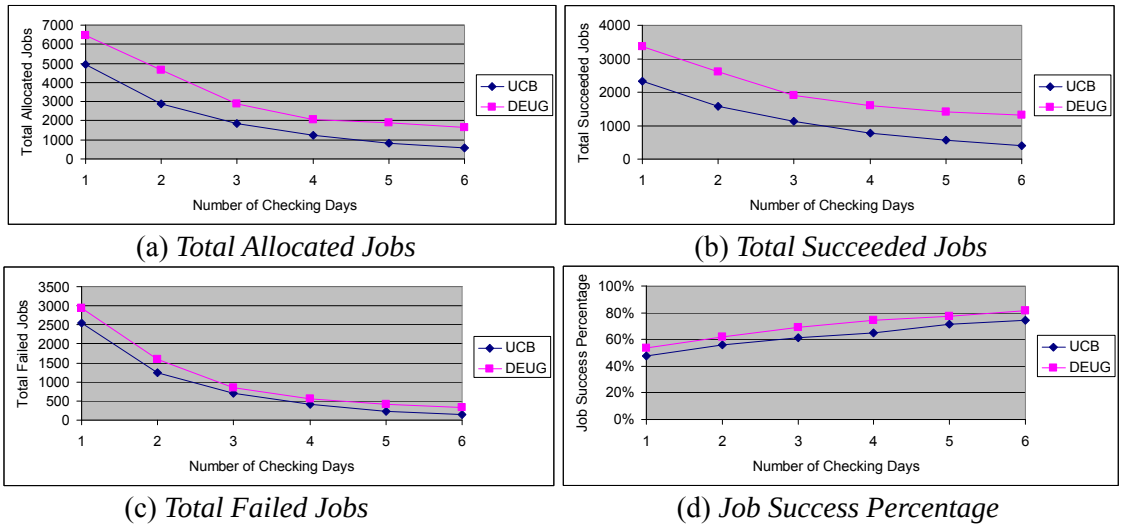
7.2.1 Evaluation of the *Number of Checking Days*

To verify the analysis of the *Number of Checking Days* (abbreviated as N) and examine the performance of the FCFSPP algorithm with different *Number of Checking Days* in practical scenarios, a set of simulations are presented in this subsection. In this set of simulations real data sets UCB and DEUG are used. As well as the setup shown in Table 7-1, these simulations have the experimental setup shown in Table 7-2:

Name	Setting
Number of Resources	80 in UCB and 680 in DEUG (depends on the available data in the downloaded data sets)
Job-scheduling Algorithm	FCFS, FCFSPP
Number of Checking Days	1, 2, 3, 4, 5 and 6
Resource Availability Probability Threshold	The value is 100%
Length of Simulation	10 days in UCB and 7 days in DEUG (depends on the available data in the downloaded data sets)

Table 7-2: Experimental Setup for Simulations of *N*

In the first simulation day, as no historical data is available, the FCFSPP algorithm behaves the same as FCFS (for more details about this, please refer to appendix B.I). Therefore, the following results are composed of the remaining simulation days.



(a) *Total Allocated Jobs* (b) *Total Succeeded Jobs*

(c) *Total Failed Jobs* (d) *Job Success Percentage*

Figure 7-2: *Number of Checking Days* in UCB and DEUG

The *Total Allocated Jobs* shows the total number of job allocated to the resource in all simulation days except the first day. According to the Figure 7.2, when *N* is larger, the algorithm becomes more "conservative" and allocates fewer jobs to resources. In data set UCB and DEUG, the results of *Total Allocated Jobs* are 4934 and 6481 respectively when *N* equals 1. *Total Allocated Jobs* gradually decreases as *N* increases confirming the analysis.

The *Total Succeeded Jobs* shows the total number of jobs processed by the resources in all simulation days except the first day. Similar to *Total Allocated Jobs*, when *N* increases, the resources process fewer jobs as fewer jobs are allocated to them. In data set UCB and DEUG,

the results of *Total Succeeded Jobs* are 2329 and 3359 respectively when *N* equals 1. *Total Succeeded Jobs* gradually decrease as *N* increases. These results are directly influenced by the result of *Total Allocated Jobs*.

Total Failed Jobs shows the total number of jobs failed by the resource in all simulation days except the first day. Similar to *Total Allocated Jobs* and *Total Succeeded Jobs*, fewer jobs fail to finish when *N* becomes larger. In data set UCB and DEUG, the results of *Total Failed Jobs* are 2546 and 2942 respectively when *N* equals 1. *Total Failed Jobs* gradually decrease as *N* increases. Again, these results are also directly influenced by the result of *Total Allocated Jobs* and fewer jobs tend to be failed when the FCFSPP algorithm becomes more *conservative* as *N* becomes larger.

The *Job Success Percentage* shows the percentage of job processed successfully among all the jobs being processed. Let S_p denotes *Job Success Percentage*, the result of $S_{percent}$ can be calculated by the following equation:

$$S_{percent} = S_{total} / F_{total} * 100\% \qquad \text{(Equation 7.1)}$$

where S_{total} is *Total Succeeded Jobs* and F_{total} is *Total Failed Jobs*. As opposed to *Total Allocated Jobs, Total Succeeded Jobs*, *Total Succeeded Jobs* and *Job Success Percentage* all increase when *N* becomes larger. In data set UCB and DEUG, the results of *Job Success Percentage* are 47.77% and 44.78% respectively when *N* equals 1. Then the result of *Total Failed Jobs* gradually increases to 74.17% and 81.34% when *N* equals 6. This means though *Total Succeeded Jobs* and *Total Failed Jobs* decrease when *N* becomes larger; the decreasing rate of *Total Failed Jobs* is much faster than *Total Succeeded Jobs*. These results show though the FCFSPP algorithm becomes more conservative with the increase of *N*, providing more reliable job allocation decisions.

The results in this section show that in the tested data sets, the cost in terms of *Total Allocated Jobs* and *Total Succeeded Jobs* can bring benefits in terms of *Total Failed Jobs* and Increase of *Job Success Percentage*. If more reliable job allocation decisions are desired, then a larger value of *N* is required.

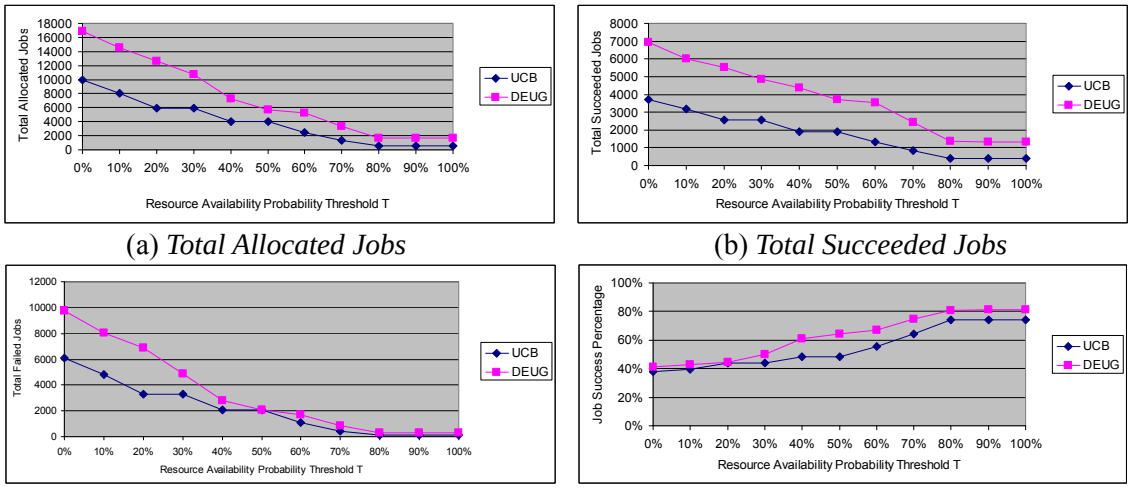
7.2.2 Evaluation of *Resource Availability Probability Threshold T*

To check the analysis about Parameter *Resource Availability Probability Threshold T* (abbreviated as *T*) and to examine the performance of FCFSPP with different setting of T in practical scenarios, a set of simulations scenarios are presented using real data sets UCB and DEUG. Besides the setup shown in Table 7-1, these simulations have the experimental setup shown in Table 7-3:

Name	Setting
Number of Resources	80 in UCB and 680 in DEUG (depends on the available data in the downloaded data sets)
Job-scheduling Algorithm	FCFS, FCFSPP
Number of Checking Days	The value is 6
Resource Availability Probability Threshold	The value varies from 0% to 100%
Length of Simulation	10 days in UCB and 7 days in DEUG (depends on the available data in the downloaded data sets)

Table 7-3: Experimental Setup for Simulations of *T*

Note the parameter *Number of Checking Days* here is 6. According to the Figure 7.3, when the value of *Resource Availability Probability T* is 0%, it behaves the same FCFS algorithm as the every resource will be qualified as the *Resource Availability Probability* of each resource cannot below 0%. In both data set UCB and DEUG, the allocation decisions are more "conservative" as the qualification standard (*Resource Availability Probability Threshold*) for job allocations becomes higher when the value of *T* increases. Therefore, the result of *Total Allocated Jobs* tends to decrease when the value of *T* increases.



(a) *Total Allocated Jobs* (b) *Total Succeeded Jobs*

(c) *Total Failed Jobs* (d) *Job Success Percentage*

Figure 7-3: *Resource Availability Probability Threshold* in UCB and DEUG

Interestingly, the result of *Total Allocated Jobs* does not always decrease smoothly when the value of *T* becomes larger. For example, in data set UCB, the result of *Total Allocated Jobs* only decreases from 5883 to 5882 when the value of *T* increases from 20% to 30%. This is because of very few resources' *Resource Availability Probability* is between 20% and 30%. Setting the value of *T* as 20% or 30% therefore has very little influence to the result of *Total Allocated Jobs*. On the other hand, if many resources' value of *Resource Availability Probability* is within a small range (e.g. 0% to 10% in Figure 7.3), increasing or decreasing the value of *T* will have a large influence to result of *Total Allocated Jobs* (the result of the result of *Total Allocated Jobs* decreases from 9897 to 8092 when the value of *T* increases from 0% to 10% in data set UCB).

As the number of *Total Allocated Jobs* tends to drop when *T* increases, the number of *Total Succeeded Jobs* tends to drop along with the increase of *T*. In general, *Total Succeeded Jobs* has the similar trend as *Total Allocated Jobs* and the result of *Total Succeeded Jobs* decreases when

the value of *T* increases. In addition, the trend does not always smooth due to the same reason discussed in the result of *Total Allocated Jobs*. Again, directly influenced by the result *Total Allocated Jobs*, it has the similar trend as *Total Allocated Jobs* and *Total Succeeded Jobs* and the result of *Total Failed Jobs* tend to drop when *T* increases. In addition, the trend does not always smooth due to the same reason discussed in the result of *Total Allocated Jobs*.

As opposed to the previous trends, the results of *Job Success Percentage* become higher when *T* increases. This is because when *T* increases the rate of *Total Failed Jobs* reduces slower than the rate of *Total Succeeded Jobs*. This shows the job allocation decisions are more reliable when qualification standard is higher (i.e. the value of *T* becomes higher).

According to the results in this subsection, different settings of *Resource Availability Threshold T* show that the cost in terms of *Total Allocated Jobs* and *Total Succeeded Jobs* can bring benefits in terms of *Total Failed Jobs* and *Job Success Percentage*. If more reliable job allocation decisions are desired, then a larger value of *T* is required.

7.2.3 Evaluation of Different Weights on TDE Prediction

According to *Job Execution Availability* correlation results shown in Section 6.2, the resources' *Job Execution Availability* in a certain day tends to be independent if the series are *Non-zero Standard Deviation Daily Series*. For *Zero Standard Deviation Daily Series*, it is not straightforward to consistently judge that which day(s) will have a higher result of *Same Type Series Occurrence Probability*. Therefore, this indicates none of the *Checking Days* should have higher weights than any other *Checking Days*. However, it is still interesting to confirm whether this indication is true. Therefore, a set of simulations with real data is used to check this.

Two non-equal weight schemes are used as a comparison with the equal weight scheme. Each scheme checks the resource's past 3 days' *Job Execution Availability* in the *Prediction Period* to make job allocation decisions. In such a case, the equal weight scheme (Equation 4.2) employs the following equation:

$$P(r) = 0.33*P_{day1}(r) + 0.33*P_{day2}(r) + 0.33*P_{day3}(r) \quad \text{(Equation 7.2)}$$

The two non-equal weight schemes are described as follows:

Non-equal weight scheme 1:

$$P(r) = 0.7*P_{day1}(r) + 0.2*P_{day2}(r) + 0.1*P_{day3}(r) \quad \text{(Equation 7.3)}$$

Where day 1 is three days before, day 2 is the day before yesterday and day 3 is yesterday. In this scheme, the result of *Resource Availability Probability* in day 3(yesterday) has the heaviest weight while the result of *Resource Availability Probability* in day 1(two days before) has the lightest weight.

Non-equal weight scheme 2:

$$P(r) = 0.1*P_{day1}(r) + 0.2*P_{day2}(r) + 0.7*P_{day3}(r) \quad \text{(Equation 7.4)}$$

Where day 1 is three days before, day 2 is the day before yesterday and day 3 is yesterday. In

this scheme, the result of *Resource Availability Probability* in day 1(two days before) has the heaviest weight while the result of *Resource Availability Probability* in day 3(yesterday) has the lightest weight.

In this set of simulations, each simulation lasts for 4 days and the fourth day will be the most important day. In the first three days, the prediction method will not have 3 days' historical data to check and this cannot clearly show the differences between the three schemes. In the fourth day, the prediction method will have 3 days' historical data to check and this can show the differences between the three schemes. Data taken from data set SDSC and LRI were used for the simulations. 10 simulation runs were carried out. Besides this and the setup shown in Table 7-1, these simulations have the experimental setup shown in Table 7-4:

Name	Setting
Number of Resources	The value is 20
Job-scheduling Algorithm	FCFSPP with 3 different weight schemes
Number of Checking Days	The value is 3
Resource Availability Probability Threshold	The value is 50%
Length of Simulation	4 simulation days
Multiply Factor	The value is 3

Table 7-4: Experimental Setup for Simulations of Different Weight Schemes

In these simulations, the first three simulation days are used to provide historical data and the following results are from the fourth simulation days. Three types of results are compared among these three schemes in the simulations, including the results of *Total Allocated Jobs Mean*, *Total Succeeded Jobs Mean* and *Total Failed Jobs Mean*. *Total Allocated/Succeeded/Failed Jobs Mean* is the average number of *Total Allocated/Succeeded/Failed Jobs* from a number of times simulations.

According to Figure 7.4, n the simulations with data set SDSC, the three schemes tend to have very similar results in terms of *Total Allocated Jobs Mean*, *Total Succeeded Jobs Mean* and *Total Failed Jobs Mean*. For example, the results of *Total Allocated Jobs Mean* are 182.0 (with margin of error 9.6), 180.3 (with margin of error (with margin of error 10.8) and 181.3 (with margin of error 9.1) in the three schemes respectively, showing the difference of in terms of *Total Allocated Jobs Mean* tends to be small (within 1%). In terms of *Total Succeeded Jobs Mean* and *Total Failed Jobs Mean*, the differences between these three schemes' results are small as well. The difference between these three schemes is within 2% in both terms of *Total Succeeded Jobs Mean* and *Total Failed Jobs Mean*). Here the margin of error is calculated by the following equation:

$$\text{Margin of error} = Z * S / \sqrt{n} \qquad \text{(Equation 7.5)}$$

Where Z-value is 1.96 as 95% confident is required, S is the sample standard deviation and n is the sample size.

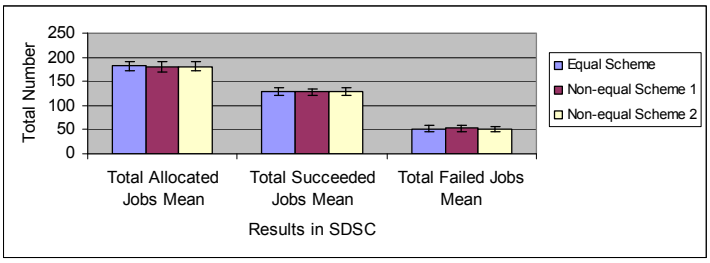


Figure 7-4: Fourth Day Simulation Performance with SDSC

From Figure 7.5, with data set LRI, the three schemes tend to have very similar results in terms of *Total Allocated Jobs Mean*, *Total Succeeded Jobs Mean* and *Total Failed Jobs Mean* as well. For example, the results of *Total Allocated Jobs Mean* are 324.4 (with margin of error 22.3), 326.8 (with margin of error (with margin of error 21.2) and 323.9 (with margin of error 21.1) in the three schemes respectively, showing the difference of in terms of *Total Allocated Jobs Mean* tends to be small (within 1%). In terms of *Total Succeeded Jobs Mean* and *Total Failed Jobs Mean*, the differences between these three schemes' results are small as well. The difference between these three schemes is within 1.5% in terms of *Total Succeeded Jobs Mean* and within 5% in terms of *Total Failed Jobs Mean*).

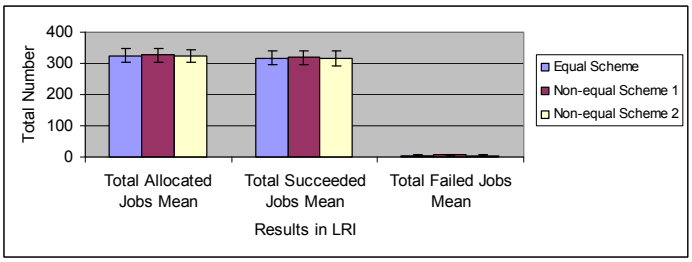


Figure 7-5: Fourth Day Simulation Performance with LRI

There are two key observations from the results:

- The results of each scheme tend to be very similar (the biggest difference in the simulations is within 5%).
- There is no strong indication showing which scheme will perform well. For example, the equal scheme has a slightly higher *Total Allocated Jobs Mean* than the two non-equal schemes with data set SDSC, but "non-equal scheme 1" has a higher result than the "equal scheme" with data set LRI.

Therefore, it is not straightforward to judge whether a non-equal weight scheme will be consistently better than the equal weight scheme or not.

7.2.4 Influence of Similarity of *Job Execution Availability* between Days

In the analysis of influence of similarity between *Checking Period* and *Prediction Period* in Section 5.2.2, the similarity level between *Checking Period* and *Prediction Period* can be represented by PMCC result ρ if they belong to the case shown in Figure 5.7. ρ is calculated by *Checking Period* and *Prediction Period* and it shows the similarity of *Job Execution Availability*

between *Checking Period* and *Prediction Period*. The *Checking Period* and the *Prediction Period* are known if only the time to a specific job's *Job Execution Time* and the time to make job allocation decision for the job are known.

However, for simulations, it is difficult to know the length of *Checking Period* and *Prediction Period* for each specific job beforehand. One compromise solution is to use the value of ρ between *Checking Day* and *Prediction Day* instead. As mentioned in Section 4.2.2, *Checking Period* is a period of time in the *Checking Day* and *Prediction Period* is a period of time in the *Prediction Day*. In general, if the *Job Execution Availability* in the *Prediction Day* is strongly correlated with the *Job Execution Availability* in the *Checking Day* (the value of ρ is close to +1 or -1), the *Job Execution Availability* in the *Prediction Period* will tend to be strongly correlated with the *Job Execution Availability* in the *Checking Period*, especially when the *Job Execution Availability* in the *Prediction Day* is very strongly correlated with the *Job Execution Availability* in the *Checking Day* (the value of ρ is very close to 1). On the other hand, if the *Job Execution Availability* in the *Prediction Day* is not correlated with to the *Job Execution Availability* in the *Checking Day* (the value of ρ is close to 0), the *Job Execution Availability* in the *Prediction Period* will not tend to be strongly correlated with the *Job Execution Availability* in the *Checking Period*, especially when the *Job Execution Availability* in the *Prediction Day* is uncorrelated with the *Job Execution Availability* in the *Checking Day* (the value of ρ is very close to 0). Therefore, the evaluation of similarity of *Job Execution Availability* between *Checking Day* and *Prediction Day* in this section is an approximation to similarity of *Job Execution Availability* between *Checking Period* and *Prediction Period.*

A set of simulations with real data were designed and carried out. The purpose of real data is to show how ρ influences the performance of FCFS and the FCFSPP algorithm in real cases, which cannot be completely provided by analysis or synthetic data.

To carry out these simulations, pairs of data traces were extracted from the downloaded data set DEUG (discussed in Chapter 6) and categorised by the range of ρ. As mentioned in Chapter 6, any paired traces of each resource were calculated and a paired trace has a result of ρ, showing the similarity level between these two traces. There are 20 categories in the simulations and the range of each category is defined by the value ρ. The ranges of these categories are: -1 to -0.9, -0.9 to -0.8 … 0.8 to 0.9 and 0.9 to 1 for each category, 20 paired traces were randomly picked except the categories of -1 to -0.9 and 0.9 to 1 as the total number of qualified (the value of ρ is within the range) paired traces is less than 20, all available 17 paired traces were used for the category of -1 to -0.9 and 18 paired traces were used for the category of 0.9 to 1. For example, if the value of a paired trace (S_i^{day2}, S_i^{day5}) is 0.55, it could picked and categorised in the category of "0.5 to 0.6". Here, S_i^{day2} means resource i's data trace in day 2 and S_i^{day5} means resource i's data trace in day 5.

If the absolute value of ρ is large (close to +1 or -1), it means the two *Daily Series* have strong positive/negative relationship. If the absolute value of ρ is small (close to 0), it means the two *Daily Series* have weak positive/negative relationship. In these simulations, one *Daily Series* is considered as the resource's *Job Execution Availability* data in *Checking Day* while the other one is considered as the resource's *Job Execution Availability* data *Prediction Day*. In the FCFSPP algorithm, it checks a resource's *Job Execution Availability* in *Checking Period* (a part of *Checking Day*) and predicts the resource's *Job Execution Availability* in the *Prediction Period* (a part of *Checking Day*) when making job allocation decisions.

Therefore, these simulations with real data are used to assist the previous analysis (in Section 5.2.2) and simulations with synthetic data (in Appendix B.I) and to provide more information in the cases that synthetic data cannot provide. Besides the setup shown in Table 7-1, these simulations have the experimental setup shown in Table 7-5:

Name	Setting
Number of Resources	17 to 20 (depends on the number of available paired traces)
Job-scheduling Algorithm	FCFS and FCFSPP
Length of Simulation	2 simulation days

Table 7-5: Experimental Setup for Simulations of ρ

In the first simulation day, as no historical data is available, the proposed algorithm behaves the same as FCFS (for more details about this, please refer to appendix B.I). Therefore, the following results are from the second simulation day:

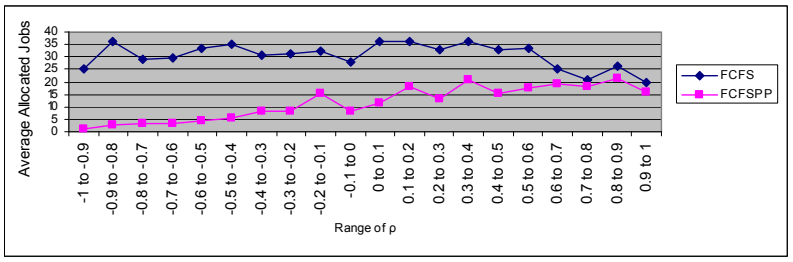


Figure 7-6: *Average Allocated Jobs* in the Second Simulation Day

Average Allocated Jobs shows the average number of jobs allocated to each resource in the second simulation day. Let $A_{average}$ denotes the number *Average Allocated Jobs*; the result of $A_{average}$ is calculated by the following equation:

$$A_{average} = A_{total} / n \qquad \text{(Equation 7.6)}$$

where A_{total} is the total number of jobs allocated to all resources in the second simulation day and n is the number of resources. According to Figure 7.6, the results of *Average Allocated Jobs* show the FCFSPP algorithm allocates fewer jobs than FCFS in all simulated cases, especially when the absolute value of ρ is small (close to -1). When the absolute value of ρ is small, the gap between FCFS and FCFSPP is large. This is because resources tend to be considered as *unqualified* after making prediction method in the FCFSPP algorithm. As a result, to ensure *reliability*, the FCFSPP algorithm does not allocate many jobs to resources in such cases. When the absolute value of ρ becomes larger (especially when it is close to 1), the gap between the

results of *Average Allocated Jobs* of FCFSPP and FCFS algorithms tends to be shortened. This is because resources tend to be considered as *qualified* after making prediction method in the FCFSPP algorithm. As a result, the FCFSPP algorithm will allocate many jobs to resources in such cases.

Fluctuations occur in both FCFS and the FCFSPP algorithms. However, the reasons for these fluctuations are slightly different. For FCFS algorithm, it is because that resources' *available* time varies in each data trace. For each paired data traces, the value of ρ can only show the similarity level of the two traces but ρ cannot show how long the resource is available. However, in addition to the value of ρ, resource available time also influences results of *Average Allocated Jobs*. For FCFS algorithm, the longer available period a resource has, the more jobs will be allocated to the resources. Therefore, the results of *Average Allocated Jobs* fluctuate in FCFS algorithm.

For the FCFSPP algorithm, in addition to different resource available time, the fluctuations are also caused the following reasons:

- The method to calculate the value of ρ. As mentioned earlier, the values of ρ are calculated for *Checking Day* and *Prediction Day* rather than *Checking Period* and *Prediction Period*. *Checking Period* and Prediction Period are subseries of *Checking Day* and *Prediction Day* so the value of ρ between *Checking Period* and *Prediction Period* may be different from the value of ρ between *Checking Day* and *Prediction Day*.
- The method to categorise the data traces. As mentioned earlier, the paired data traces are categorised by the range of ρ (e.g. 0.5 to 0.6) rather than the specific value of ρ (e.g. 0.5). This means the paired data traces in the same category has similar value of ρ but may not the same value of ρ. For each resource in the same category, different value of ρ may also bring different results of *Total Allocated Jobs*. Therefore, the aggregated results of *Total Allocated Jobs* are influenced.
- Therefore, this set of simulations mainly focuses on showing the general trends and differences between FCFS and the FCFSPP algorithm rather than providing accurate results.

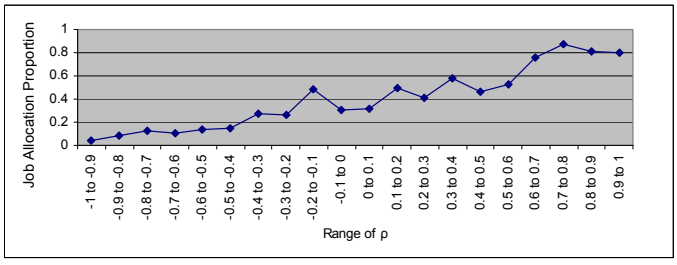


Figure 7-7: *Job Allocation Proportion* in the Second Simulation Day

Job Allocation Proportion is calculated by *Total Allocated Jobs* in FCFSPP divided by *Total Allocated Jobs* in FCFS algorithm for each category. Let $A_{proportion}$ denotes *Job Allocation Proportion*; It is calculated by the following equation:

$$A_{proportion} = A_{FCFSPP} / A_{FCFS} \quad \text{(Equation 7.7)}$$

where A_{FCFSPP} is the *Total Allocated Jobs* in FCFSPP and A_{FCFSPP} is the *Total Allocated Jobs* in FCFSPP. For example, in category "0.9 to 1", the number of *Total Allocated Jobs* in the FCFSPP algorithm is 49 and the number of *Total Allocated Jobs* in FCFS is 70, so *Job Allocation Proportion* is 49/70 = 0.7.

According to Figure 7.7, *Job Allocation Proportion* is high when the value of ρ is large (above 0.7). In addition, the value of *Job Allocation Proportion* tends to become larger when ρ increases. The trend of *Job Allocation Proportion* indicates that the FCFSPP algorithm's performance in terms of allocating jobs tends to be similar to FCFS algorithm when the value of ρ becomes larger.

Average Succeeded Jobs shows the average number of jobs processed by each resource in the second simulation day. Let $S_{average}$ denotes *Average Succeeded Jobs*; it is calculated by the following equation:

$$S_{average} = S_{total} / N \quad \text{(Equation 7.8)}$$

Where S_{total} is *Total Succeeded Jobs* in the second simulation day and N is the total number of resources. According to the analysis in Section 5.2.2, the FCFSPP algorithm does not processing higher number of jobs than FCFS algorithm under the same condition. The result in Figure 7.8 also confirms this. In terms of *Average Succeeded Jobs*, the FCFSPP algorithm has lower values than FCFS in all simulated cases.

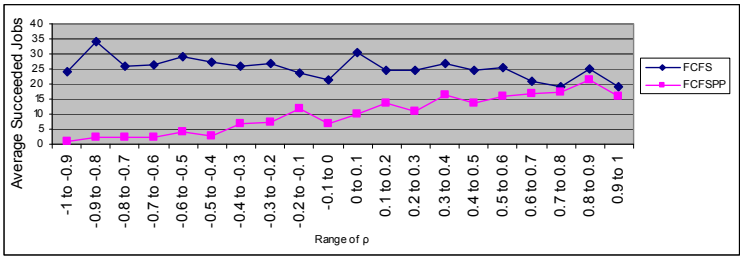


Figure 7-8: *Average Succeeded Jobs* in the Second Simulation Day

In addition, Figure 7.8 shows a trend that the FCFSPP algorithm tends to process more jobs when ρ becomes larger. These results also show that the FCFSPP algorithm does not perform better than FCFS algorithm in terms of getting jobs processed quickly. The fluctuations can be also explained by the three reasons discussed above: different available time in each trace and the method to calculate the value of ρ.

Job Success Proportion is calculated by *Total Succeeded Jobs* in the FCFSPP algorithm divided by *Total Succeeded Jobs* in FCFS algorithm for each category. Let $S_{proportion}$ denotes *Job Success Proportion*; it is calculated by the following equation:

$$S_{proportion} = S_{FCFSPP} / S_{FCFS} \quad \text{(Equation 7.9)}$$

Where S_{FCFSPP} is *Total Succeeded Jobs* in FCFSPP and S_{FCFS} is *Total Succeeded Jobs* in FCFS. For example, in category "0.9 to 1", the number of *Total Succeeded Jobs* in the FCFSPP

algorithm is 288 and the number of *Total Succeeded Jobs* in FCFS algorithm is 342, so *Job Processed Proportion* is 288/342 ≈ 0.842.

As seen in Figure 7.9, when ρ is small, *Job Success Proportion* tends to be low and it has the trend to become higher when ρ becomes larger. When ρ is higher than 0.5, *Job Success Proportion* increases rapidly and it is over 0.8 when ρ is close to 1. These results show the analysis that the FCFSPP algorithm tends to perform as well as FCFS in terms of *Job Success Proportion* when ρ has a high value. The fluctuations can be also explained by the reasons given above: insufficient data and the method to calculate the value of ρ.

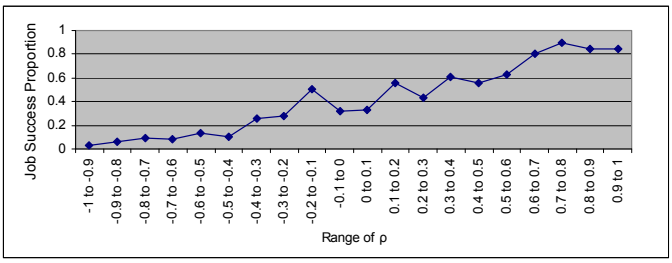


Figure 7-9: *Job Success Proportion* in the Second Simulation Day

Average Failed Jobs shows the average number of jobs disposed by each resource in the second simulation day. Let $F_{average}$ denotes *Average Failed Jobs*; It is calculated by the following equation:

$$F_{average} = F_{total} / N \qquad \text{(Equation 7.10)}$$

Where F_{total} is *Total Failed Jobs* in the second simulation day and N is the total number of resources. In terms of *Average Failed Jobs*, a job will be failed when an *Unavailability Event* occurs. As FCFS will keeps all the resources busy when the resources are available, so when an *Unavailability Event* occurs, a job will be failed in FCFS algorithm. So the number of *Total Failed Jobs* in FCFS is identical as the number of *Unavailability Event* in each simulation. In the mean while, the FCFSPP algorithm does not always keep the resources busy, so when an *Unavailability Event* occurs, a job will not necessarily be failed in the FCFSPP algorithm.

When the absolute value of ρ is large (the original value is close to -1 or 1), the number of *Unavailability Event* is lower than the cases when the absolute value of ρ is small (close to 0). The number of *Average Failed Jobs* in FCFS algorithm represents this. According to Figure 7.10, the number of *Total Failed Jobs* in the FCFSPP algorithm also tends to have a high value when the absolute value of ρ is small. In all simulated scenarios, the results show FCFS algorithm cannot perform better than the FCFSPP algorithm in terms of *Average Failed Jobs*.

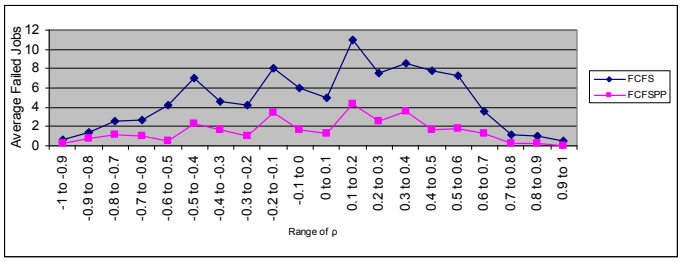


Figure 7-10: *Average Failed Jobs* in the Second Simulation Day

Job Failure Proportion is calculated by *Total Failed Jobs* the FCFSPP algorithm divided by *Total Failed Jobs* in FCFS algorithm for each category. Let $F_{proportion}$ denotes *Job Failure Proportion*; it is calculated by the following equation:

$$F_{proportion} = F_{FCFSPP} / F_{FCFS} \quad \text{(Equation 7.11)}$$

Where F_{FCFSPP} is *Total Failed Jobs* in FCFSPP and F_{FCFS} is *Total Failed Jobs* in FCFS.

For example, in category "0.9 to 1", the number of *Total Failed Jobs* in FCFS algorithm is 1 and the number of *Total Failed Jobs* in FCFS is 10, so *Job Success Proportion* is 1/10 = 0.9.

According to Figure 7.11, *Job Failure Proportion* fluctuates when ρ varies. The reason of this is that *Unavailability Event* does not occur regularly. For example, for a particular resource, assume all *Unavailability Events* can be predicted and avoided within a short period. So within the period, the number of can be varied. It could be a small number and it also could be a very large number. Therefore, *Job Failure Proportion* does not indicate something clearly in such cases.

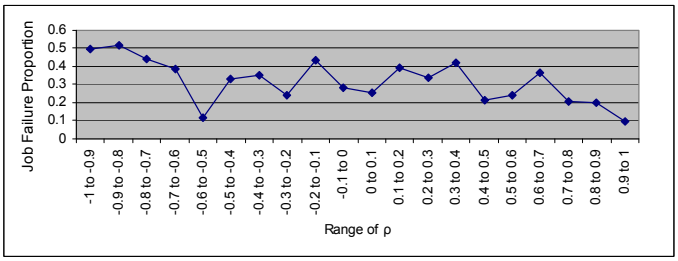


Figure 7-11: *Job Failure Proportion* in the Second Simulation Day

Still according to the Figure 7.11, when ρ is small, *Job Failure Proportion* tends to be high and it has the trend to become lower when ρ becomes larger. These results show that FCFS algorithm cannot perform as well as the FCFSPP algorithm in terms of *Job Failure Proportion*. This difference between FCFS and the FCFSPP algorithm in terms of *Job Failure Proportion* tends to be obvious when ρ has a high value. In addition, the fluctuations can be also explained by the two reasons discussed above: different available time in each data trace and the method to calculate the value of ρ.

As seen in Figure 7.12, in terms of *Job Success Percentage*, the FCFSPP algorithm has higher results than FCFS in most cases. The exceptions occur when the original value of ρ is small (below 0 and close to -1). This shows the FCFSPP algorithm can make reliable decisions – higher percentage of allocated jobs can be completed successfully in most cases, especially when the

value of ρ is high. When the value of ρ is small, the historical data actually become misleading for TDE predictor in the FCFSPP algorithm as the behaviour in the *Prediction Period* will significantly different from the *Checking Period* in such cases. Therefore, job allocation decisions based on misleading information cannot be very reliable.

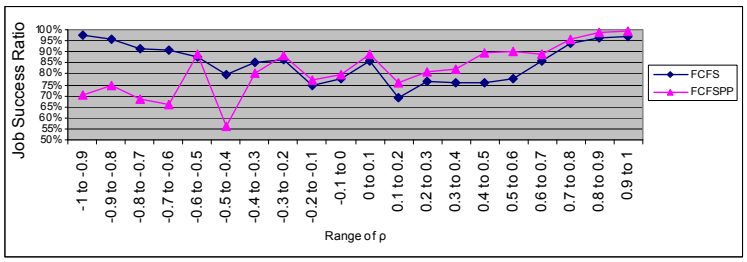


Figure 7-12: *Job Success Percentage* in the Second Simulation Day

As mentioned above, the fluctuations always occur in different results. To see if these fluctuations can be mitigated and clearer trends be obtained by more runs of simulations, some simulation replications for some categories have been taken. In the simulated 20 categories, only 5 of them (-0.2 to -0.1, -0.1 to 0, 0 to 0.1, 0.1 to 0.2 and 0.2 to 0.3) have more than 200-paired data traces (this is shown in Figure 6.22). As one simulation uses 20 paired data traces, 9 replications (so the total number of simulations for one category is 10 times) are carried out for these 5 categories. The result of *Total Allocated Jobs Mean* is added and subtracted by the margin of error. Here, the result collected from each simulation is considered as a sample and the margin of error for the sample average.

According Figure 7.13, the fluctuations in results of *Average Total Allocated Jobs* tend to narrow after replications. However, it does not show a clear trend that the gaps between FCFSPP and FCFS tend to be closer when the absolute value of ρ becomes larger (approaches to 1) in these 5 simulated categories. On the other hand, it indicates the gaps between the results of *Total Allocated Jobs Mean* of FCFSPP and FCFS tend to be unpredictable when the two *Daily Series* have a very weak relationship (in these 5 simulated cases, the absolute value of ρ is below 0.3).

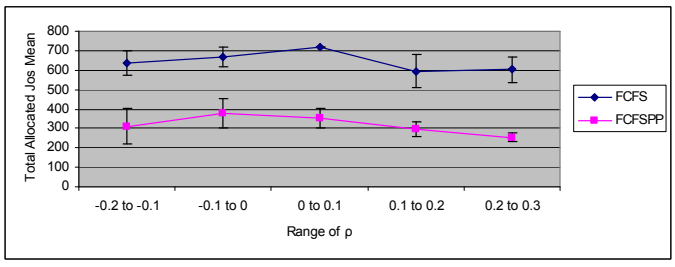


Figure 7-13: *Total Allocated Jobs* with Margin of Error in the Second Simulation Day

According to Figure 7.14, in terms of *Total Succeeded Jobs Mean*, it does not show a clear trend that the gaps between FCFSPP and FCFS tend to be closer when the absolute value of ρ becomes larger (approaches to 1) in these 5 simulated categories either. Similar to the results of *Average Total Allocated Jobs*, it indicates that the gaps between the results of *Average Total*

Succeeded Jobs of FCFSPP and FCFS tend to be unpredictable when the two *Daily Series* have a very weak relationship. In addition, it also indicates FCFSPP tends to perform worse than FCFS algorithm in terms of *Average Total Succeeded Jobs* in all simulated scenarios.

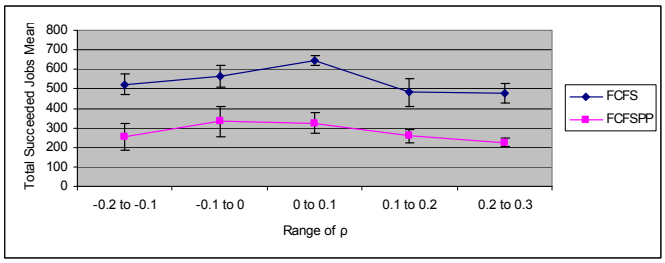


Figure 7-14: *Total Succeeded Jobs* with Margin of Error in the Second Simulation Day

As seen in Figure 7.15, in terms of *Total Failed Jobs Mean*, it does not show a clear trend that the gaps between FCFSPP and FCFS tend to be closer when the absolute value of ρ becomes larger (approaches to 1) in these 5 simulated categories either. As with the results of *Total Allocated Jobs Mean* and *Average Total Succeeded Jobs*, it indicates the gaps between the results of *Total Failed Jobs Mean* of FCFSPP and FCFS tend to be unpredictable when the two *Daily Series* have very weak relationship. In addition, it also indicates FCFS tends to perform worse than the FCFSPP algorithm in terms of *Total Failed Jobs Mean* in all simulated scenarios.

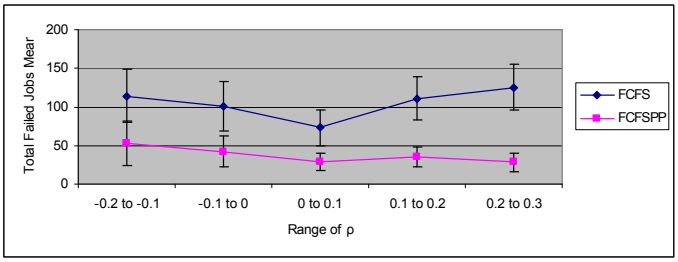


Figure 7-15: *Total Succeeded Jobs* with Margin of Error in the Second Simulation Day

7.3 Evaluation of FLP Algorithm

To show the performance of the FLP algorithm and check the difference between FCFS, FLP and the FCFSPP algorithm, a set of simulations with real data were carried out. Besides the setup shown in Table 7-1, these simulations have the experimental setup shown in Table 7-6:

Name	Setting
Number of Resources	17 to 20 (depends on the number of available paired traces)
Job-scheduling algorithm	FCFS, FCFSPP and FLP
Length of Simulation	2 simulation days

Table 7-6: Experimental Setup for Simulations of FLP

According to Figure 7.16, in terms of *Total Allocated Jobs*, the result of the FLP algorithm is always between FCFS and the FCFSPP algorithm. Furthermore, the result of *Total Allocated Jobs* is closer to FCFS algorithm in all simulation scenarios and this indicates the value of *Resource Availability Probability Threshold* tend to be low (so it behaves more like FCFS algorithm) in all simulation scenarios.

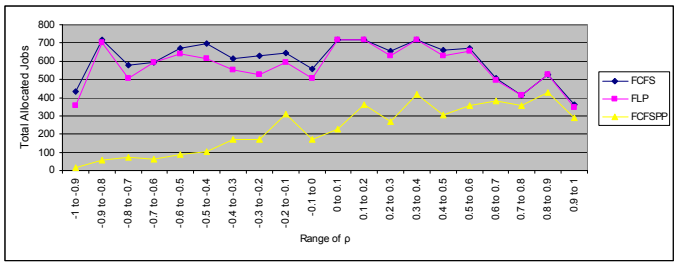


Figure 7-16: *Total Allocated Jobs* in the Second Simulation Day

As can be seen from Figure 7.17, the result of *Total Succeeded Jobs* in the FLP algorithm is always between FCFS and the FCFSPP algorithm and it is closer to FCFS algorithm in all simulation scenarios.

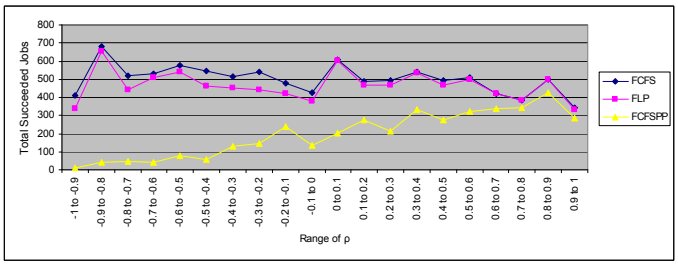


Figure 7-17: *Total Succeeded Jobs* in the Second Simulation Day

As seen in Figure 7.18, the result of *Total Succeeded Jobs* in the FLP algorithm is always between FCFS and the FCFSPP algorithm in most simulation scenarios. According to the analysis in Section 5.3, the results of the FLP algorithm in terms of *speed* and *reliability* are generally supposed to between FCFS and the FCFSPP algorithm. However, interestingly, the result of *Total Failed Jobs* is even higher than FCFS, the FLP algorithm performs even worse than FCFS algorithm in some scenarios (e.g. in the simulation scenario "0.1 to 0.2" and "-0.7 to -0.6").

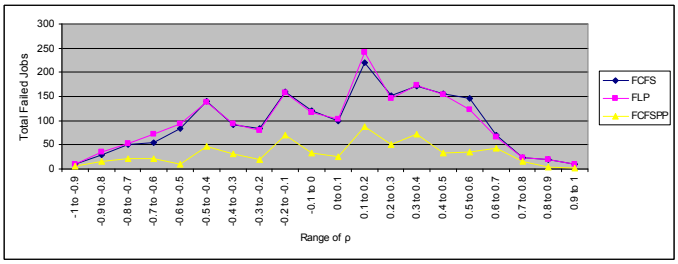


Figure 7-18: *Total Failed Jobs* in the Second Simulation Day

This could be explained by making job allocations at different times. For example, assume a resource is available for 60 minutes in the *Prediction Period* and 30 minutes in the *Checking Period* and a job waiting for allocation also lasts for 60 minutes. For FCFS algorithm, it always allocates the job to the resource. As the resource will available for 60 minutes, the job will be completed successfully. For the FCFSPP algorithm, as the resource's *Resource Availability Probability* is 50% (30/60 = 50%), it will not allocate the job to the resource during the hour. Therefore, the job will not be failed either. However, for the FLP algorithm, it may also refuse

to allocate the job to the resource for a while (e.g. 10 minutes) as the *Resource Availability Probability Threshold* is high in this period (e.g. the *Resource Availability Probability Threshold* can be 60% for the first 10 minutes). However, after this period, the *Resource Availability Probability Threshold* might be reduced (e.g. reduced to 30%). Now if the FLP algorithm tries to allocate the job to this resource again, as the resource's *Resource Availability Probability* is 33.3% ((30-10)/60 = 33.3%), it will allocate the job to the resource now. However, this job will be failed as the resource will not available for one hour now (60-10=50 minutes), the job will be failed.

This exceptional case can be called as *Failure of Delayed Job Allocation Decision*, which means job failures are caused by the delayed job allocation decision. When the resource is considered as *unqualified* and the job will not be allocated to the resource, the resource may be actually *qualified* at that moment. When the resource is considered as *qualified* later and the job is allocated to the resource, the resource may have already become *unqualified*. However, this case does not occur frequently and only arises due to the following conditions:

- The resource is considered as *unqualified* firstly. This will only occur if system is facing case 2 and 3 (the cases described in section 5.2.2).
- After being considered as *unqualified*, the resource is considered as *qualified* soon after (before the job is allocated to another resource). This requires the *Resource Availability Probability Threshold* reduces quickly enough to let this resource becomes *qualified* soon.
- Normal job allocation decisions will not result in a job failure while *Delayed Job Allocation Decision results* in a job failure.
- The resource is being considered as the resource candidate for the same job again after the *Resource Availability Probability Threshold* has been lowered. This requires the job to have not been allocated to another resource yet.

In addition, the results show this case does not occur frequently as the result of *Total Failed Jobs* in the FLP algorithm is below the performance of the FCFS algorithm in most cases.

Figure 7.19 shows that FLP is usually between FCFS and FCFSPP in terms of *Job Success Percentage*. However, the FLP algorithm has the lowest *Job Success Percentage* some simulation scenarios. This is due to *Failure of Delayed Job Allocation Decisions*.

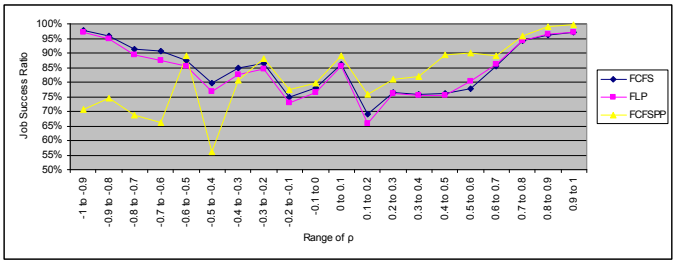


Figure 7-19: *Job Success Percentage* Comparison

According to the simulation scenarios with real data, the FLP algorithm's *Total Succeeded*

Jobs, *Total Failed Jobs* and *Job Success Percentage* are typically between FCFS and FCFSPP. However, in these simulation scenarios, it does not manage to obtain a better performance than FCFS or FCFSPP, as FLP cannot achieve a higher *Job Success Percentage* than other two algorithms in these scenarios. The reason for this is that the patterns of all resources' *Job Execution Availability* tend to be dissimilar (as shown in Chapter 6), that is employing the *Job Execution Availability* from some resources does not necessarily provide a good indication for all resources.

7.4 Evaluation of the PSOPP Algorithm

To evaluate the performance of the PSOPP algorithm and check the analysis (discussed in Section 5.4), some simulations have been designed and carried out.

7.4.1 Influence of *Workload*

As discussed in Section 5.4.2, the PSOPP algorithm will be influenced by different level of *Workload*. Therefore, a set of simulations with synthetic data is used to check the influences. Besides the setup shown in Table 7-1, these simulations have the experimental setup shown in Table 7-7:

Name	Setting
Number of Resources	The value is 1
Job-scheduling algorithm	FCFS, FCFSPP, FLP and PSOPP
CPU Availability of the resource	1GHz
t_1	24 hours (resource is available throughout the first simulation day)
t_2	24 hours (resource is available throughout the second simulation day)
Job Size	12 hours
Job Arrival Interval in simulation scenario 1	The value is 16 hours
Job Arrival Interval in simulation scenario 2	The value is 12 hours
Job Arrival Interval in simulation scenario 3	The value is 8 hours
Resource Availability Probability Threshold Adjustment Interval	The value is 1 hour
Number of Iteration in PSOPP	The value is 1
Length of simulation	2 simulation days

Table 7-7: Experimental Setup for Simulations of Different *Workload* in PSOPP

These three simple but representative scenarios show the influence of *Workload*. In these simulation scenarios, there is only one resource and the resource is always available in the two simulation days. As there is only one resource, the PSOPP algorithm will always allocate the new jobs to the resource. The only difference between these two scenarios is that the job creation interval (the same as job arrival interval). A job arrives at the Grid job scheduler every 16 hours, 12 and 8 hours in these three scenarios respectively. This represents three levels of *Workload*: low, medium and high. For comparisons, FCFS, FCFSPP and the FLP algorithms are used in the simulations. Note no job is created in the first simulation day in these simulation scenarios. Therefore, the resource will be *idle* at the beginning of the second simulation day.

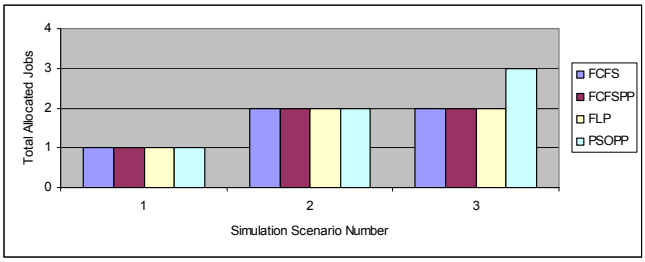


Figure 7-20: *Total Allocated Jobs* in the Second Simulation Day

Figure 7.20 shows the results of *Total Allocated Jobs* of each algorithm in the second simulation day. In terms of *Total Allocated Jobs*, the possible highest result is 3. In simulation scenario 3, a job arrives at the Grid job scheduler every 8 hours. Therefore, 3 jobs (excluding the last one arrives just at the end of the second simulation day) will arrive at the Grid job scheduler in the second simulation day.

For FCFS algorithm, it only allocates a new job to the resource when the resource is *idle*. As each job lasts for 12 hours in these simulation scenarios, the resource will be *idle* for 2 times (including the last time just at the end of the second simulation day) in the second simulation day, the result of *Total Allocated Jobs* is 2 in simulation scenario 2 and 3. In simulation scenario 1, the number of *Total Allocated Jobs* is 1 as the *Workload* is low (job arrives every 16 hours).

For the FCFSPP and the FLP algorithms, they only allocate a new job to the resource when the resource is *idle* and the resource's *Resource Availability Probability* is not below the predefined *Resource Availability Probability Threshold*. In these simulation scenarios, the resource is always available so the *Resource Availability Probability* is not below the predefined *Resource Availability Probability Threshold* all the time. Therefore, FCFSPP and FLP always allocate a new job to the resource when the resource becomes *idle* in these simulation scenarios. As each job lasts for 12 hours in these simulation scenarios, the resource will be *idle* for 2 times (including the last time just at the end of the second simulation day) in the second simulation day, the result of *Total Allocated Jobs* is 2 in simulation scenario 2 and 3. In simulation scenario 1, the number of *Total Allocated Jobs* is 1 as the *Workload* is low (job arrives every 16 hours).

For the PSOPP algorithm, it allocates a new job to the resource when the resource is *available*. As the resource is always *available* in three simulation scenarios, PSOPP always allocates new jobs to the resource. As a result, PSOPP always have the possible highest result of *Total Allocated Jobs* in these three simulation scenarios.

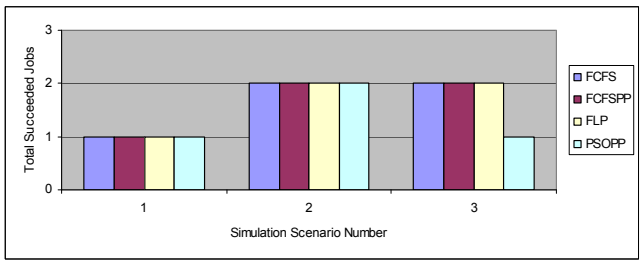


Figure 7-21: *Total Succeeded Jobs* in the Second Simulation Day

Figure 7.21 show the results of *Total Succeeded Jobs* of each algorithm in the second simulation day. In terms of *Total Succeeded Jobs*, the possible highest result is 2 as a job last for 12 hours.

For FCFS, FCFSPP and the FLP algorithm, they have the highest result in terms of *Total Succeed Jobs* in simulation scenario 2 and 3 as it always has a job to process at a time. In simulation scenario 1, they do not have the highest result in terms of *Total Succeed Jobs* as the *Workload* is low (the resource is left *idle* for 4 hours).

For the PSOPP algorithm, it has the highest results in terms of *Total Succeed Jobs* in simulation scenario 2. In this scenario, the *Workload* is medium and the PSOPP algorithm keeps the resource has job and only one job to process at a time. In simulation scenario 1, as for FCFS, it does not have the highest result in terms of *Total Succeed Jobs* as the *Workload* is low (the resource is left *idle* for 4 hours). In simulation scenario 3, it does not have the highest result as the *Workloa*d is high and the PSOPP algorithm keeps the resource have more than 1 job for 16 hours.

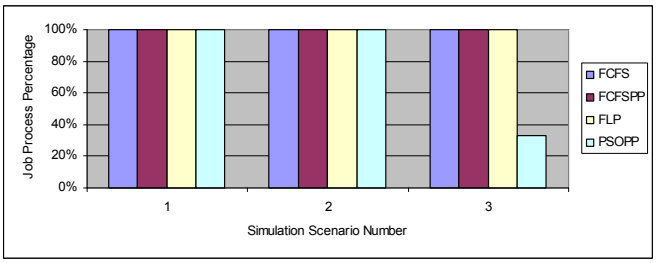


Figure 7-22: *Job Process Percentage* in the second simulation day

Figure 7.22 show the results of *Job Process Percentage* of each algorithm in the second simulation day. *Job Process Percentage* shows the percentage of jobs processed by all resources in a given period. Let $P_{percentage}$ denotes *Job Process Percentage*; calculated by the following equation:

$$P_{percentage} = (S_{total} + F_{total}) / A_{total} * 100\% \qquad \text{(Equation 7.12)}$$

Where S_{total} is *Total Succeeded Jobs*, F_{total} is *Total Failed Jobs* and A_{total} is *Total Allocated Jobs*. For FCFS, FCFSPP and the FLP algorithms, they have the high results (it is 100% in these three simulation scenarios) in terms of *Job Process Percentage* as it always keep the resource has at most one job at a time rather than multiple jobs at a time.

For the PSOPP algorithm, it has the highest results in terms of *Job Process Percentage* in simulation scenario 1 and 2. In these scenarios, the *Workload* is not high and the PSOPP algorithm keeps the resource has job and only one job to process at a time. In simulation scenario 3, it does not have the highest result as the *Workload* is high and the PSOPP algorithm keeps the resource have more than 1 job for 16 hours.

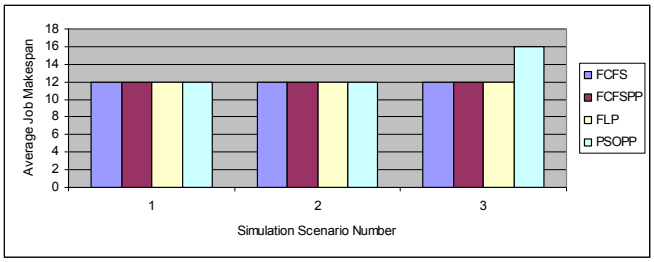


Figure 7-23: *Average Job Makespan* in the Second Simulation Day

Figure 7.23 show the results of *Average Job Makespan* of each algorithm in the second simulation day. *Average Job Makespan* describes the average time to complete a job. Let $M_{average}$ denotes *Average Job Makespan*; It is calculated by the following equation:

$$M_{average} = \frac{\sum_{j=1}^{S} M_j}{S} \qquad \text{(Equation 7.13)}$$

where M_j is the *Makespan* of job j and S is *Total Succeeded Jobs* so far.

For FCFS, FCFSPP and the FLP algorithms, the result of *Average Job Makespan* is always 12 hours in these three simulation scenarios as each job will occupy the CPU cycles of the resource solely if it is allocated to the resource.

For the PSOPP algorithm, the result of *Average Job Makespan* is 12 hours simulation scenario 1 and 2 as each job is occupy the CPU cycles of the resource solely if it is allocated to the resource in these two scenarios. In simulation scenario 3, the result of *Average Job Makespan* is 16 hours as multiple jobs have to share the CPU cycles of the resource for some time.

According to the above results, the performances of the PSOPP algorithm under different *Workload* levels are presented. In brief, the PSOPP algorithm' result of job throughput tends to be low if the *Workload* is low or high. The PSOPP algorithm's result of *Average Job Makespan* tends to be long if the *Workload* is high.

In addition, these results also show some differences between the algorithm using the approach of allocating jobs to *idle* resources and the algorithm using the approach of allocating jobs to *available* resources, especially when *Workload* is high. If *Workload* is high, the algorithm using the approach of allocating jobs to *available* resources is influenced greatly in terms of *speed*.

7.4.2 Influence of PSO Fitness Function

As discussed in Section 5.4.3, the PSOPP algorithm will be influenced by the correctness level of fitness function. Therefore, a set of simulations with synthetic data and representative scenarios are used to check the influence of the PSO fitness function. Besides the setup shown in Table 7-1, these simulations have the experimental setup shown in Table 7-8:

Name	Setting
Number of Resources	The value is 2
Job-scheduling algorithm	FCFS, FCFSPP, FLP and PSOPP.
CPU Availability of resource 1	In simulation scenario 1, it is always 2GHz. In simulation scenario 2, it is 2GHz at first but becomes 1GHz after 12 hours in the second simulation day
CPU Availability of resource 2	In simulation scenario 1, it is always 1GHz. In simulation scenario 2, it is 1GHz at first but becomes 2GHz after 12 hours in the second simulation day
t_1	24 hours (resource is available throughout the first simulation day)
t_2	24 hours (resource is available throughout the second simulation day)
Job Size	12 hours if *CPU Availability* is always 2GHz and 24 hours if *CPU Availability* is always 1GHz
Job Arrival Interval	The value is 12 hours
Resource Availability Probability Threshold Adjustment Interval	The value is 1 hour
Number of Iteration in PSOPP	The value is 1
Length of simulation	The value is 48 hours

Table 7-8: Experimental Setup for Simulations of Fitness Function of PSOPP

These are two simple but representative scenarios to show how the correctness level of the fitness function influences the PSOPP algorithm. In the first simulation scenario, both resources' *CPU Availability* does not change in the two simulation days. In such a case, the fitness function is supposed to represent the fitness value of both resources correctly. In the second simulation scenario, both resources' *CPU Availability* changes in the middle of second simulation days. In such a case, the fitness function is supposed to represent the fitness value of both resources correctly.

For comparisons, FCFS, FCFSPP and the FLP algorithms are used in the simulations. The results from the second simulation day are focused here as FCFSPP, FLP and the PSOPP algorithm use the TDE prediction method (TDE prediction method needs historical data and the first simulation day's data is considered as the historical data here).

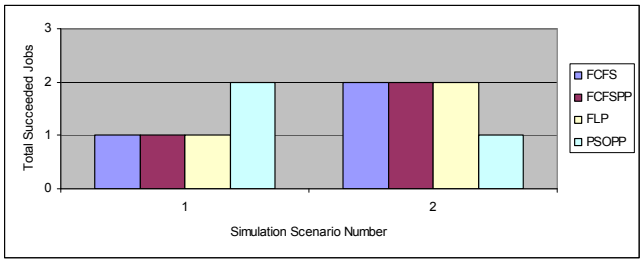


Figure 7-24: *Total Succeeded Jobs* in the Second Simulation Day

Figure 7.24 show the results of *Total Succeeded Jobs* of each algorithm in the second simulation day. In terms of *Total Succeeded Jobs*, the possible highest result is 2 as a job last for 12 hours.

For FCFS, FCFSPP and the FLP algorithms, they do not have the highest result in terms of *Total Succeeded Jobs* in simulation scenario 1 as the second job is not allocated to the more powerful resource. At the beginning of second simulation day, the first job will be allocated to resource 1 as resource 1 is the first resource in the resource list. The first job will be finished by resource 1 after 12 hours. After 12 hours, the second job arrives and it will be allocated to resource 2 as resource 2 is the first resource in the resource list. The second job will not be

finished by resource 2 after 12 hours as it has low *CPU Availability* (the *CPU availability* of resource 2 is 1GHz and it requires 24 hour to finish a job).

The FCFS, FCFSPP and FLP algorithms have the highest result in terms of *Total Succeed Jobs* in simulation scenario 2 as the first two jobs are allocated to the more powerful resource. At the beginning of second simulation day, the first job will be allocated to resource 1 as resource 1 is the first resource in the resource list. The first job will be finished by resource 1 after 12 hours. After 12 hours, the second job arrives and it will be allocated to resource 2 as resource 2 is the first resource in the resource list.

For the PSOPP algorithm, it has the opposite results to FCFS algorithm in terms of *Total Succeeded Jobs* in these two scenarios. PSOPP has the highest result in terms of *Total Succeeded Jobs* in simulation scenario 1 as both jobs are allocated to the more powerful resource. At the beginning of the second simulation day, the first job will be allocated to resource 1 as resource 1's fitness value is higher than resource 2's. The first job will be finished by resource 1 after 12 hours. After 12 hours, the second job arrives and it will still be allocated to resource 1 as resource 1 has a higher fitness value than resource 2. The second job will be finished by resource 2 after 12 hours as well.

PSOPP has the highest result in terms of *Total Succeeded Jobs* in simulation scenario 1 as the second job is allocated to the more powerful resource. At the beginning of the second simulation day, the first job will be allocated to resource 1 as resource 1's fitness value is higher than resource 2's. The first job will be finished by resource 1 after 12 hours. After 12 hours, the second job arrives and it will still be allocated to resource 1 as resource 1 has a higher fitness value than resource 2. However, the *CPU Availability* of resource 1 changes to 1GHz after the job is allocated to resource 1. Therefore, the second job will not be finished by resource 1 after 12 hours.

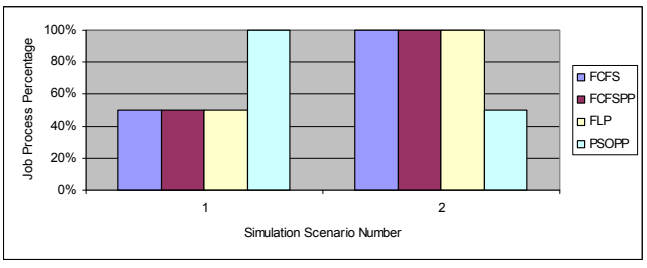


Figure 7-25: *Job Process Percentage* in the Second Simulation Day

Figure 7.25 show the results of *Job Process Percentage* of each algorithm in the second simulation day. Similar to the result of *Total Succeeded Jobs*, FCFS, FCFSPP, FLP have the highest result in terms of *Job Process Percentage* in simulation scenario 2 as the first two jobs are allocated to the more powerful resource. At the beginning of second simulation day, the first job will be allocated to resource 1 as resource 1 is the first resource in the resource list. The first job will be finished by resource 1 after 12 hours. After 12 hours, the second job arrives and it

will be allocated to resource 2 as resource 2 is the first resource in the resource list.

The PSOPP algorithm has the highest result in terms of *Total Succeeded Jobs* in simulation scenario 1 as the second job is allocated to the more powerful resource. At the beginning of second simulation day, the first job will be allocated to resource 1 as resource 1's fitness value is higher than resource 2's. The first job will be finished by resource 1 after 12 hours. After 12 hours, the second job arrives and it will still be allocated to resource 1 as resource 1 has a higher fitness value than resource 2. However, the *CPU Availability* of resource 1 changes to 1GHz after the job is allocated to resource 1. Therefore, the second job will not be finished by resource 1 after 12 hours.

According to the above results, the correctness level of the fitness value will influence the performance of the PSOPP algorithm in terms of *Total Succeeded Jobs* and *Job Process Percentage* here). If the fitness function can represent fitness values of each solution correctly, the job allocation decisions based on the correct fitness value will perform well in terms of *speed* and it will better than a job-scheduling algorithm with *Checking if Qualified* approach (represented by FCFS, FCFSPP and the FLP algorithm), such as simulation scenario 1.

But on the other hand, if the fitness function cannot represent fitness values of each solution correctly, the job allocation decisions based on the correct fitness value will not be able to perform well in terms of *speed* and/or *reliability* and it will worse than job-scheduling algorithms with *Checking if Qualified* approach, such as simulation scenario 2.

7.5 Evaluation of PSPP Migration Algorithm

To examine the performances of the Periodical Scanning with Predictor Migration Algorithm (PSPP), especially in terms of predicting *Unavailability Events* of resources, a set of simulations have been designed and carried out.

The examination method works as follow: At regular intervals, the Grid job scheduler uses PSPP algorithm to check each currently available resource's availability history and predict the *Resource Availability Probability* of each resource in the *Prediction Period*. Note here the Grid job scheduler focus on the resources which are currently available because only those currently available resources may have jobs and the jobs might need proactive migration as a result of potential *Unavailability Events*. For those currently unavailable resources, no job is running on them so it is no need to worry about job migration consequently.

After making a prediction for a particular resource, the prediction result shows the *Resource Availability Probability* of the resource in the *Prediction Period*. If a resource's *Resource Availability Probability* is lower than 100%, then it means the resource is predicted to have an *Unavailability Event* (the resource will become unavailable) at some time during the *Prediction Period*. In such a case, a job migration is considered to be necessary. If a resource's *Resource Availability Probability* is 100%, then it means the resource is predicted to be staying available

throughout the *Prediction Period*. In such a case, a proactive migration is not considered to be necessary.

With the prediction results, the Grid job scheduler checks if the prediction results are correct after the *Prediction Period*. For example, assume an *Prediction Period* lasts for time P and the PSPP prediction algorithm makes a prediction at time $T_{current}$, then the Grid job scheduler will check the predictions at time $T_{current} + P$.

Three terms describing a prediction result introduced in Section 5.5.2 are used to describe whether a prediction result is correct or not: *Correct Prediction Type 1*, *False Alarm* and *Missed Detection*. Note here *Correct Prediction Type 2* is not used for evaluating the performance of prediction result as the evaluation is mainly focused on whether the prediction method can detect *Unavailability Events* correctly.

A set of simulations with real data were used to show the performance of PSPP algorithm in practical scenarios. In this set of simulation, data sets UCB, SDSC, LRI and DEUG were used.

To test the accuracy of prediction algorithm in with different *Prediction Period P*, the parameter of *Prediction Period* (the same length as *Migration Prediction Interval*) were set as 1, 5 and 10 minutes in different series of simulations. The reason for choosing these values is for the consideration of job migration procedure. According to previous research [Ma00][Bouchenak00b], a job migration is expected to finished between a couple of seconds and a couple of minutes. Therefore, if the *Prediction Period* (abbreviated as *P*) is too short (e.g. 1 second), the Grid job scheduler will not be able to have enough time to finish the procedure of job migration. On the other hand, if *P* is too long (e.g. 1 hour), some idle CPU cycles of the resources may be wasted as a result of migration too early. Therefore, making prediction for the next couple of minutes should be a suitable range of time.

To test the influence brought by the parameter *Number of Checking Days* (abbreviated as *N*), the parameter was varied from 1 to 6 in different simulations.

In this set of simulations, two extra results are used to represent the performance of the prediction algorithm. The first one is *Unavailability Event Detection Percentage*. Let $D_{percent}$ denotes *Unavailability Event Detection Percentage*; it is calculated by the following equation:

$$D_{percent} = C_{type1} / U_{total} * 100\% \qquad \text{(Equation 7.14)}$$

where C_{type1} is total number of *Correct Predictions Type 1* and U_{total} is the total number of *Unavailability Events* (equals Total Number of *Correction Prediction Type 1* + Total Number of *Missed Detection*). According to Equation 7.14, to what extent the prediction algorithm can detect all *Unavailability Events* can be tested.

The second equation is *Correct Prediction Percentage*. Let $C_{percent}$ denotes *Correct Prediction Percentage*; it is calculated by the following equation:

$$C_{percent} = C_{type1} / (C_{type1} + F_{total}) * 100\% \qquad \text{(Equation 7.15)}$$

where C_{type1} is is total number of *Correct Predictions Type 1* and F_{total} is the total number of

False Alarm. According to Equation 7.15, to what extent the prediction algorithm can make correct predictions can be tested.

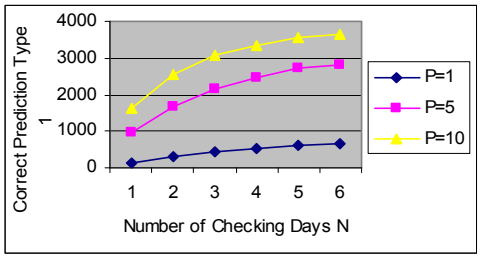
7.5.1 Evaluation with Data Set UCB

UCB contains 10 days' availability data and there are 6076 *Unavailability Events* overall. However, as the prediction algorithm needs past 1 day's availability history to make prediction and the prediction algorithm does not have any availability history in the first day, the prediction algorithm does not make any prediction in the first day consequently. Therefore, 460 *Unavailability Events* which occur in the first day were excluded from the examination. As a result, 5616 *Unavailability Events* which occur in the rest 9 days were used for the examination. In general, there are two trends shown in Figure 7.26: firstly, the results of *Correct Prediction Type 1* and *False Alarm* increase while the result of *Missed Detection* decreases along with the increase of *N*. Secondly, the result of *Correct Prediction Type 1* increases while the results of *Missed Detection* and *False Alarm* decrease along with the increase of *P*.

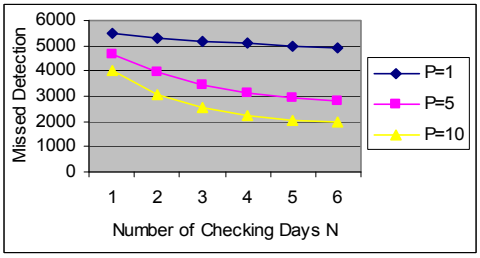
For *Correct Prediction Type 1*, it has the lowest result 149 when *N*=1 (day) and *P*=1 (minute). When *N* increase to 6 (days) and *P* increases to 10 (minutes), the result of *Correct Prediction Type 1* also increases to 3644. According to the analysis in Section 5.4, this can be explained by the reason that the *Checking Period* has been lengthened so it is more likely to have at least one *Unavailability Event* in the *Checking Period*.

For *Missed Detection*, it has the highest result 5927 when *N*=1 (day) and *P*=1 (minute). When *N* increase to 6 (days) and *P* increases to 10 (minutes), the result of *Correct Prediction Type 1* decreases to 2432. This can be explained by the increase of *Correct Prediction Type 1*. As the total number of *Unavailability Event* is fixed (it is 6076 in UCB). Therefore, opposite to *Correction Prediction Type 1*, if the result of *Correction Prediction Type 1* increases, the result of *Missed Detection* will decrease.

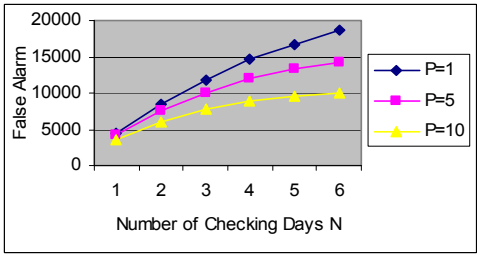
For *False Alarm*, it has the lowest result 3592 when *N*=1 (day) and *P*=10 (minutes). When *N* increase to 6 (days) and *P* decreases to 1 (minute), the result of *False Alarm* increases to 18567. The result of *False Alarm* increasing with the increase of *N* can be explained by the reason that the *Checking Period* has been lengthened so it is more likely to have at least one *Unavailability Event* in the *Checking Period*. The result of *False Alarm* decreasing with the increase of *P* can be explained by the reason that the *Prediction Period* has been lengthened so it is more likely to have at least one *Unavailability Event* in the *Prediction Period*.



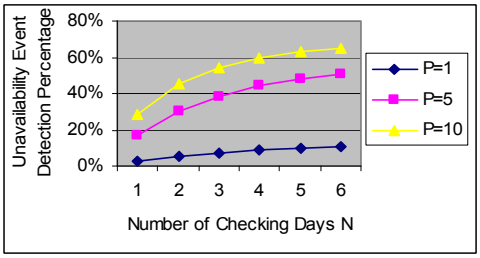
(a) *Correct Prediction Type 1*



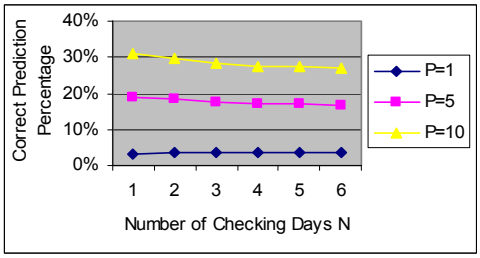
(b) *Missed Detection*



(c) *False Alarm*



(d) *Unavailability Event Detection Percentage*



(e) *Correct Prediction Percentage*

Figure 7-26: Effect of Different Values of *N* and *P* in UCB

The results of *Unavailability Event Detection Percentage* increase along with the increase of *N* and *P*. It has the lowest result 2.58% when *N*=1 (day) and *P*=1 (minute). When *N* increases to 6 (days) and *P* increases to 10 (minutes), the result of *Unavailability Event Detection Percentage* increases to 64.89%. The result of *Unavailability Event Detection Percentage* increasing with the increase of *N* and *P* can be explained by the reason that the increase of *Correction Prediction Type 1*.

For *P*=5 and *P*=10, the result of *Correct Detection Ratio* decreases along with the increase of *N* and decreases of *P*. It has the highest result 30.94% when *N*=1 (day) and *P*=10 (minutes). When *N* increase to 6 (days) and *P* decreases to 5 (minutes), the result of *Correct Prediction Percentage* decreases to 16.57%. The result of *Correct Detection Ratio* decreasing with the increase of *N* can be explained by the reason that the increase of *Correction Prediction Type 1* is slower than the increase of *False Alarm* when *N* increases. The result of *Correct Detection Ratio* increases with the decrease of *P* can be explained by the reason that the increase of *Correction Prediction Type 1* is slower than the increase of *False Alarm* when *N* increases, especially in the case when *P* decreases.

For P=1, the result of *Correct Detection Ratio* increases along with the increase of *N* when *N* changes from 1 to 5. When *N*=1 and *N*=5, the result of *Correct Detection Ratio* is 3.24% and 3.58% respectively. When *N*=6, the result of *Correct Detection Ratio* is 3.49%. The result of

Correct Detection Ratio increases with the decrease of *P* when *N* change from 1 to 5 can be explained by the reason that the increase of *Correction Prediction Type 1* is faster than the increase of *False Alarm* when *N* increases. However, later, when *N* changes from 5 to 6, the increase of *Correction Prediction Type 1* is slower than the increase of *False Alarm* so the result of *Correct Detection Ratio* decreases consequently.

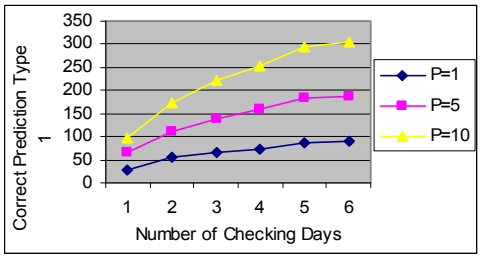
According to the above results, if *P* increases, the performance of PSPP algorithm improves as both results of *Unavailability Event Detection Percentage* and *Correct Detection Ratio* increases. However, for parameter *N*, if the value of *N* increases, it is difficult to say the performance of PSPP algorithm improves as the result of *Unavailability Event Detection Percentage* increases but *Correct Detection Ratio* decreases.

7.5.2 Evaluation with Data Set SDSC

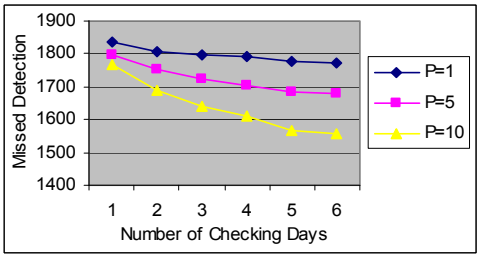
SDSC contains 7 days availability data and there are 2370 *Unavailability Event* overall. However, as for UCB, as the prediction algorithm needs past 1 day's availability history to make prediction and the prediction algorithm does not have any availability history in the first day, the prediction algorithm does not make any prediction in the first day consequently. Therefore, 506 *Unavailability Events*, which occur in the first day, were excluded from the examination. As a result, 1864 *Unavailability events* that occur in the remaining 6 days were used for the examination.

There are two trends shown in Figure 7.27: firstly, the results of *Correct Prediction Type 1* and *False Alarm* (though it is not very obvious in this data set) increase while the result of *Missed Detection* decreases along with the increase of *N*. Secondly, the result of *Correct Prediction Type 1* increases while the results of *Missed Detection* and *False Alarm* decrease along with the increase of *P*.

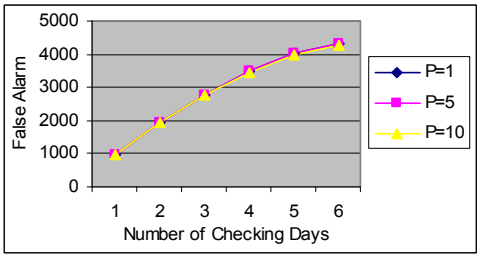
For *Correct Prediction Type 1*, it has the lowest result 29 when *N*=1 (day) and *P*=1 (minute). When *N* increase to 6 (days) and *P* increases to 10 (minutes), the result of *Correct Prediction Type 1* also increases to 306. According to the analysis in Section 5.4, this can be explained by the reason that the *Checking Period* has been lengthened so it is more likely to have at least one *Unavailability Event* in the *Checking Period*.



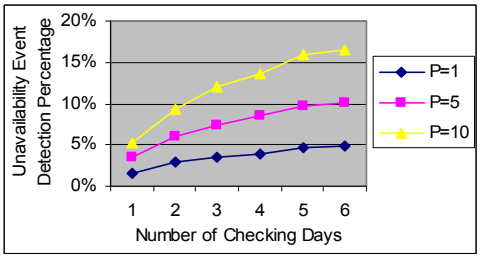
(a) *Correct Prediction Type 1*



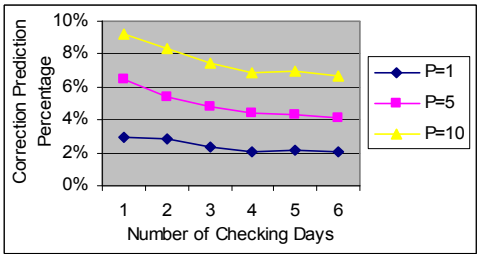
(b) *Missed Detection*



(c) *False Alarm*



(d) *Unavailability Event Detection Percentage*



(e) *Correct Prediction Percentage*

Figure 7-27: Effect of Different Values of *N* and *P* in SDSC

For *Missed Detection*, it has the highest result 2280 when *N*=1 (day) and *P*=1 (minute). When *N* increase to 6 (days) and *P* increases to 10 (minutes), the result of *Correct Prediction Type 1* decreases to 2064. This can be explained by the increase of *Correct Prediction Type 1*. As the total number of *Unavailability Event* is fixed (it is 1864 in SDSC). Therefore, opposite to *Correction Prediction Type 1*, if the result of *Correction Prediction Type 1* increases, the result of *Missed Detection* will decrease.

For *False Alarm*, it has the lowest result 951 when *N*=1 (day) and *P*=10 (minutes). When *N* increase to 6 (days) and *P* decreases to 1 (minute), the result of *False Alarm* increases to 4317. The result of *False Alarm* increasing with the increase of *N* can be explained by the reason that the *Checking Period* has been lengthened so it is more likely to have at least one *Unavailability Event* in the *Checking Period*. The result of *False Alarm* decreasing with the increase of *P* can be explained by the reason that the *Prediction Period* has been lengthened so it is more likely to have at least one *Unavailability Event* in the *Prediction Period*.

The results of *Unavailability Event Detection Percentage* increase along with the increase of *N* and *P*. It has the lowest result 1.56% when *N*=1 (day) and *P*=1 (minute). When *N* increase to 6 (days) and *P* increases to 10 (minutes), the result of *Unavailability Event Detection Percentage* increases to 16.42%. The result of *Unavailability Event Detection Percentage* increasing with the increase of *N* and *P* can be explained by the reason that the increase of

Correction Prediction Type 1.

The result of *Correct Detection Ratio* decreases along with the increase of *N* and decreases of *P*. It has the highest result 9.17% when *N*=1 (day) and *P*=10 (minutes). When *N* increase to 6 (days) and *P* decreases to 1 (minutes), the result of *Correct Prediction Percentage* decreases to 2.04%. The result of *Correct Detection Ratio* decreasing with the increase of *N* can be explained by the reason that the increase of *Correction Prediction Type 1* is slower than the increase of *False Alarm* when *N* increases. The result of *Correct Detection Ratio* increases with the decrease of *P* can be explained by the reason that the increase of *Correction Prediction Type 1* is slower than the increase of *False Alarm* when *N* increases, especially in the case when *P* decreases.

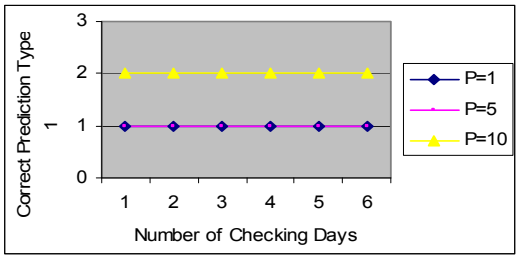
7.5.3 Evaluation with Data Set LRI

LRI contains 7 days' availability data and there are 390 *Unavailability Events* overall. However, as the prediction algorithm needs past *N* days availability history to make a prediction and the prediction algorithm does not have any availability history in the first day, consequently the prediction algorithm does not make any predictions in the first day. Therefore, 45 *Unavailability Events*, which occur in the first day, were excluded from the examination. As a result, 345 *Unavailability Events* that occur in the remaining 6 days were used for the examination.

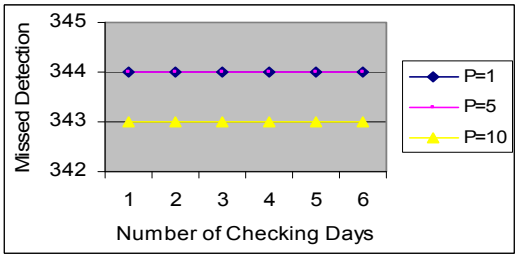
Different from data set UCB and SDSC, there is no obvious trend along with the change of *N* and *P* in some results shown in Figure 7.28.

For *Correct Prediction Type 1*, it has the lowest result 1 when and *P*=1 (minute) and *P*=5 (minutes) no matter how the number *N* changes. When *P*=10 (minutes), the result of *Correct Prediction Type 1* is 2 no matter how the number of *N* changes. This can be explained by the reason that the relationship of *Job Execution Availability* between *Checking Period* and *Prediction Period* is very weak (which is detailed described in Section 6.2.3) so that it is difficult to have *Unavailability Event* both in *Checking Period* and *Prediction Period*.

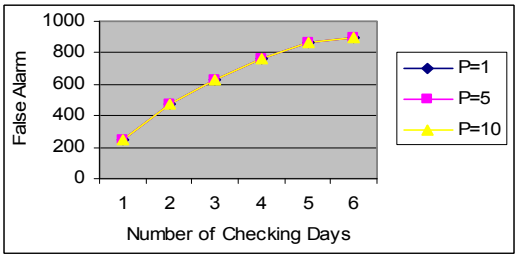
For *Missed Detection*, it has the highest result 344 when and *P*=1 (minute) and P=5 (minutes) no matter how the number *N* changes. When *P*=10 (minutes), the result of *Missed Detection* is 343 no matter how the number of *N* changes. As the total number of *Unavailability Event* is fixed, the change of *Missed Detection* is directly influenced by the change of the result of *Correct Prediction Type 1*.



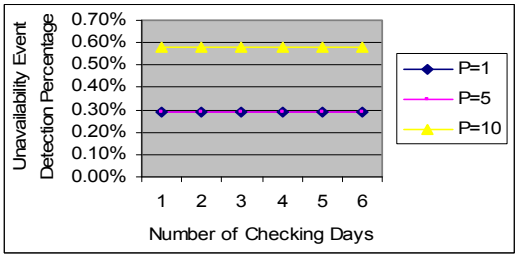
(a) *Correct Prediction Type 1*



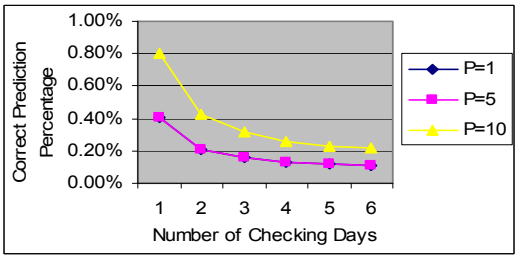
(b) *Missed Detection*



(c) *False Alarm*



(d) *Unavailability Event Detection Percentage*



(e) *Correct Prediction Percentage*

Figure 7-28: Effect of Different Values of *N* and *P* in LRI

For *False Alarm*, it has the lowest result 247 when *N*=1 (day) no matter how the number *P* changes. When *N* increase to 6 (days), the result of *False Alarm* increases to 900 no matter how the number *P* changes. The result of *False Alarm* increasing with the increase of *N* can be still explained by the reason of weak relationship of *Job Execution Availability* between *Checking Period* and *Prediction Period*.

The result of *Unavailability Event Detection Percentage* has no obvious trend along with the change of *N* and *P*. It has the lowest result 0.29% when *P*=1 (minute) and *P*=5 (minutes) no matter how the number *N* changes. When *P*=10 (minutes), the result of *Unavailability Event Detection Percentage* is 0.22%. The result of *Unavailability Event Detection Percentage* can be explained by the reason that the nearly unchanged result of *Correct Prediction Type 1* and *Missed Detection*.

The result of *Correct Detection Ratio* decreases along with the increase of *N* and decreases of *P*. It has the highest result 0.80% when *N*=1 (day) and *P*=10 (minutes). When *N* increase to 6 (days) and *P* decreases to 1 (minutes), the result of *Correct Prediction Percentage* decreases to 0.40%. The result of *Correct Detection Ratio* decreasing with the increase of *N* can be explained by the reason that the increase of *Correction Prediction Type 1* is slower than the increase of *False Alarm* when *N* increases. The result of *Correct Detection Ratio* increases with the decrease of *P* can be explained by the reason that the increase of *Correction Prediction Type 1* is

slower than the increase of *False Alarm* when *N* increases, especially when *P* decreases.

7.5.4 Evaluation with Data Set DEUG

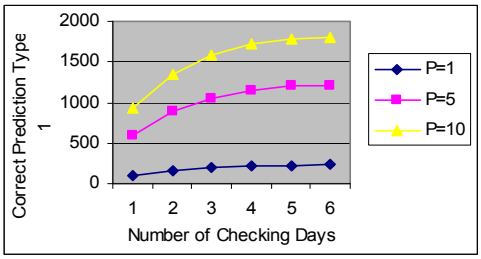
DEUG contains 7 days' availability data and there are 9764 *Unavailability Events* overall. However, as the prediction algorithm needs past *N* days availability history to make predictions and the prediction algorithm does not have any availability history in the first day, consequently the prediction algorithm does not make any predictions in the first day. Therefore, 2235 *Unavailability Events*, which occur in the first day, were excluded from the examination. As a result, 7529 *Unavailability Events* that occur in the remaining 6 days were used for the examination.

As for data set DEUG, there are two trends shown in Figure 7.29: firstly, the results of *Correct Prediction Type 1* and *False Alarm* (though it is not very obvious in this data set) increase while the result of *Missed Detection* decreases along with the increase of *N*. Secondly, the result of *Correct Prediction Type 1* increases while the results of *Missed Detection* and *False Alarm* decrease along with the increase of *P*.

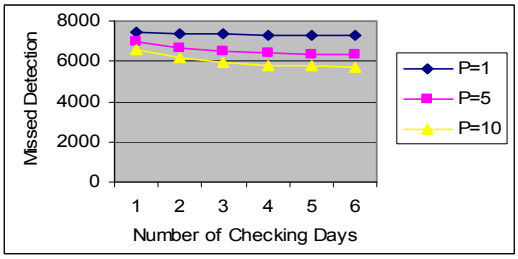
For *Correct Prediction Type 1*, it has the lowest result 104 when *N*=1 (day) and *P*=1 (minute). When *N* increase to 6 (days) and *P* increases to 10 (minutes), the result of *Correct Prediction Type 1* also increases to 1802. According to the analysis in Section 5.4, this can be explained by the reason that the *Checking Period* has been lengthened so it is more likely to have at least one *Unavailability Event* in the *Checking Period*.

For *Missed Detection*, it has the highest result 28392 when *N*=1 (day) and *P*=1 (minute). When *N* increase to 6 (days) and *P* increases to 10 (minutes), the result of *Correct Prediction Type 1* decreases to 4844. This can be explained by the increase of *Correct Prediction Type 1*. As the total number of *Unavailability Event* is fixed (it is 7529 in DEUG). Therefore, opposite to *Correction Prediction Type 1*, if the result of *Correction Prediction Type 1* increases, the result of *Missed Detection* will decrease.

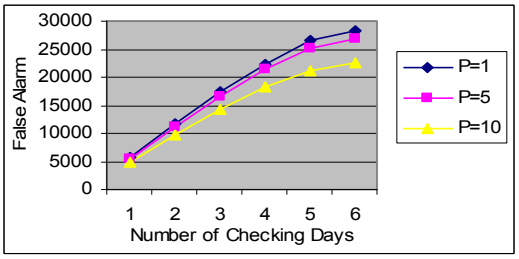
For *False Alarm*, it has the lowest result 9660 when *N*=1 (day) and *P*=10 (minutes). When *N* increase to 6 (days) and *P* decreases to 1 (minute), the result of *False Alarm* increases to 7962. The result of *False Alarm* increasing with the increase of *N* can be explained by the reason that the *Checking Period* has been lengthened so it is more likely to have at least one *Unavailability Event* in the *Checking Period*. The result of *False Alarm* decreasing with the increase of *P* can be explained by the reason that the *Prediction Period* has been lengthened so it is more likely to have at least one *Unavailability Event* in the *Prediction Period*.



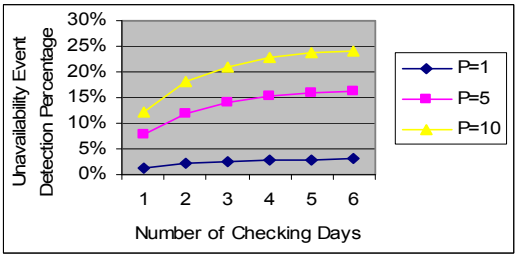
(a) *Correct Prediction Type 1*



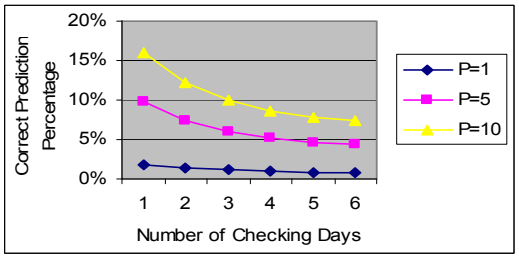
(b) *Missed Detection*



(c) *False Alarm*



(d) *Unavailability Event Detection Percentage*



(e) *Correct Prediction Percentage*

Figure 7-29: Effect of Different Values of *N* and *P* in DEUG

The results of *Unavailability Event Detection Percentage* increase along with the increase of *N* and *P*. It has the lowest result 1.38% when *N*=1 (day) and *P*=1 (minute). When *N* increase to 6 (days) and *P* increases to 10 (minutes), the result of *Unavailability Event Detection Percentage* increases to 23.93%. The result of *Unavailability Event Detection Percentage* increasing with the increase of *N* and *P* can be explained by the reason that the increase of *Correction Prediction Type 1*.

The result of *Correct Detection Ratio* decreases along with the increase of *N* and decreases of *P*. It has the highest result 16.03% when *N*=1 (day) and *P*=10 (minutes). When *N* increase to 6 (days) and *P* decreases to 1 (minutes), the result of *Correct Prediction Percentage* decreases to 0.81%. The result of *Correct Detection Ratio* decreasing with the increase of *N* can be explained by the reason that the increase of *Correction Prediction Type 1* is slower than the increase of *False Alarm* when *N* increases. The result of *Correct Detection Ratio* increases with the decrease of *P* can be explained by the reason that the increase of *Correction Prediction Type 1* is slower than the increase of *False Alarm* when *N* increases, especially when *P* decreases.

7.5.5 Summary

The results of *Unavailability Events Detection Ratio* and *Correct Prediction Percentage* are relative low (lower than 30%) in most cases, especially in the case of LRI. Though some results

of *Unavailability Events Detection Ratio* are over 60% in some cases of UCB, the results of *Correct Prediction Percentage* are still quite low in such cases. For example, in UCB, when the parameter *Checking Days N* equals 6 and *Prediction Period* is 10 minutes, *Unavailability Events Detection Ratio* is 64.89%. This means 64.89% *Unavailability Events* can be detected by the prediction algorithm and the other 35.11% *Unavailability Events* cannot. Furthermore, *Correct Prediction Percentage* is 30.94%, which means only 30.94% predictions are *Correct Prediction* while the other 69.06% predictions are *False Alarms*.

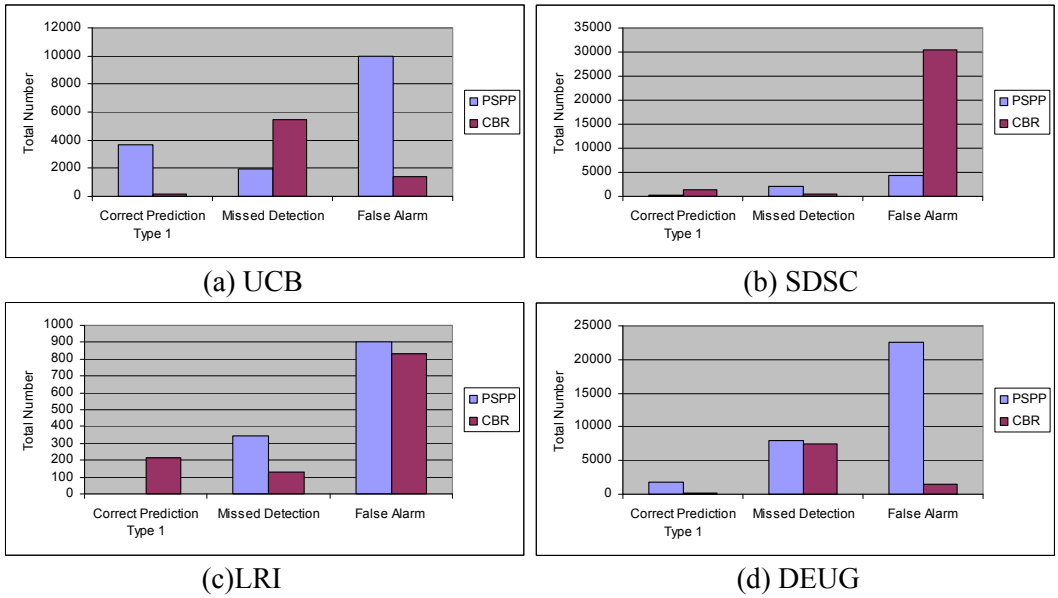
Therefore, in these four data sets, if this prediction algorithm is used to make proactive migration decisions, it can make *Correct Predictions* and trigger proactive migration decisions to help avoid potential job failures to some extent. For example, it can help avoid 64.89% *Unavailability Events* in UCB. However, in the meanwhile, many unnecessary migration decisions will be triggered by *False Alarms* and many necessary migrations cannot be triggered as a result of *Missed Detections*. For example, 69.06% predictions are *False Alarms* in UCB and 35.11% *Unavailability Events* cannot be detected by the prediction algorithm.

7.6 Evaluation of CBR Migration Algorithm

To examine the accuracy of the CBR migration algorithm and to compare CBR migration with the proposed PSPP migration algorithm in real Grid environments, a set of simulations were designed and carried out with the downloaded data sets UCB, SDSC, LRI and DEUG.

The examination method used by CBR migration algorithm can be described as follows: Every 10 minutes, the Grid job scheduler uses CBR migration algorithm to check each available resource's current *CPU Availability Percentage* to predict at regular interval. If a resources' If a resource's *Resource Availability Probability* is lower than the predefined *CPU Migration Threshold*, then it means the resource is predicted to have an *Unavailability Event* at some time during the *Prediction Period*. Here, the length of *Prediction Period* is 10 minutes and the prediction interval is also 10 minutes. After making predictions, CBR migration algorithm will review the accuracy of these predictions after the *Prediction Period* and adjust the *CPU Migration Threshold* with the value of *Adjustment Percentage*. In this set of simulations, the maximum value of *Adjustment Percentage* is $\pm 5\%$ and the initial value of *CPU Migration Threshold* is 25%. As the *Adjustment Percentage* is a random value uniformly distributed between [-5%, 5%], the simulations for each data set have been run for 3 replications and the results shown in this section will be the average results of the 3 replications.

The terms describing a prediction result in Section 5.5.2 are used to describe whether a prediction result is correct or not, including *Correct Prediction Type 1*, *False Alarm* and *Missed Detection*. Here are some important results from the simulations and the results from PSPP algorithm are also put together for comparison:



(a) UCB (b) SDSC

(c)LRI (d) DEUG

Figure 7-30: Comparison of PSPP and CBR Algorithms

According to Figure 7.30, the results *Correct Prediction Type 1*, *Missed Detection* and *False Alarm* gathered from PSPP and CBR vary from one data set to another in general. For example, in data set UCB, PSPP algorithm's result of *Correct Prediction Type 1* is 3644 while CBR algorithm's result of *Correct Prediction Type 1* is only 130.8 (average results from 5 replications), which means PSPP algorithm can predict 3644 *Unavailability Events* out of 5616 in data set UCB while CBR algorithm can only predict 130.8 *Unavailability Events* out of 5616 on average in data set UCB.

But in data set LRI, PSPP algorithm's result of *Correct Prediction Type 1* is 2 while CBR algorithm's result of *Correct Prediction Type 1* is only 213.4 (average results from 5 replications), which means PSPP algorithm can only predict 2 *Unavailability Events* out of 345 in data set UCB while CBR algorithm can predict 213.4 *Unavailability Events* out of 345 on average in data set UCB.

This is because of different characteristics of each data set and different prediction approaches used by the migration method. For example, if a resource's *CPU Availability* in different days is strongly correlated, the PSPP migration algorithm tends to perform well as the prediction method in PSPP algorithm can observe the pattern of *CPU Availability* from past days and then make correct predictions. If a resource's *CPU Availability Percentage* usually decreases to a low level before becoming unavailable, CBR migration tends to perform as well as the prediction methods in the CBR algorithm can observe this change and then make correct predictions. Therefore, it is not straightforward to judge which migration method is better as it is highly reliant not only on the characteristics of the resources but the characteristics of the resources in a data set tend to be different from resources in another data set and even the characteristics of a resource in a data set tends to be different from another resource in the same data set.

Chapter 8 – Discussion, Conclusion and Future Work

8.1 Discussion

In this thesis, job-scheduling and migration algorithms have been proposed, analysed and evaluated. The general performance of each algorithm under different situations is now briefly summarised.

In terms of job-scheduling, FCFSPP, FLP and the PSOPP algorithms have been proposed:

- FCFSPP is a job-scheduling algorithm based on a basic FCFS algorithm that leverages the TDE prediction method for improving *reliability* with little cost in terms of *speed*.

  According to the analysis in Section 5.2, validation in Appendix B.I and evaluation work shown in Section 7.2, this algorithm works well in scenarios where *Workload* is high (the number of jobs exceeds the number of resources) and each resource's *Job Execution Availability* possesses a pattern that can be observed by the TDE prediction scheme. In such cases, the FCFSPP algorithm performs well both in terms of *speed* and *reliability.* In terms of *speed*, job throughput can be maximised by allocating jobs to *qualified* resources. In terms of *reliability*, potential job failures can be avoided by not allocating job to *unqualified* resources.

  If the *Workload* is low, the FCFSPP algorithm's performance in terms of *speed* is affected as it may not be able to find the most suitable resources for jobs[1]. Furthermore, if each resource's *Job Execution Availability* does not have a regular pattern or the predictor cannot observe the pattern effectively, the performance in terms of *reliability* is affected, too.

- FLP is a job-scheduling algorithm based on FCFSPP and a Fuzzy Inference System to adjust setting of *Resource Availability Probability Threshold* to achieve a balance between *speed* and *reliability*.

  According to the analysis in Section 5.3, validation in Appendix B.II and evaluation work given in Section 7.3, this algorithm works well if the pattern(s) of all resources' *Job Execution Availability* is similar as *Job Execution Availability* on some resources will provide good indications of the behaviour of all resources. In such cases, the FLP algorithm can provide a good balance between *speed* and *reliability* as the Fuzzy Inference System can learn[2] from the indications about the change of resources' *Job Execution Availability* pattern and then response to the change quickly and correctly.

---

[1] This is because FCFSPP uses the *Checking If Qualified* approach, checking one idle resource at a time, when trying to make job allocation decisions. Therefore, it may not be able find the most powerful resources for jobs. This can be improved by using the *Finding the Best* approach, checking multiple resources at a time when trying to make job allocation decisions. However, *Finding the Best* approach only helps in cases where the *Workload* is low.

[2] The fuzzy inference system makes adaptations in response to observations. The fuzzy inference system is based on fuzzy logic as described in Section 4.2.4. In the proposed algorithm, the system adjusts the *Resource Availability Probability Threshold* based on the value of *Disposed Jobs Dots*.

If the pattern(s) of all resources' *Job Execution Availability* is dissimilar, the FLP algorithm finds it difficult to provide a good balance between *speed* and *reliability* as Fuzzy Inference System may not be able to learn something correctly.

- PSOPP is a job-scheduling algorithm based on the PSOPP algorithm and uses TDE prediction. As discussed in Section 5.2.1 and 5.3.1, FCFSPP and FLP always try to allocate a new job to the next *idle* resource (which is called *Checking If Qualified*). Different from these *Checking If Qualified* algorithms, PSOPP is a type of *Finding the Best* algorithm (i.e. it will try to find out the "best" resource from some candidates) and all *available* resources, not necessarily *idle* ones, will be candidates when the job allocation decision is being made.

  Therefore, if the *Workload* is high, the PSOPP algorithm may allocate jobs to *busy* resources and cause a resource to have more than one job at a time. As discussed in Section 5.2.1, allowing a resource to have more than one job at a time may cause the PSOPP algorithm to perform poorly in terms of *speed* and *reliability*. In terms of speed, placing multiple jobs on a single resource increases each job's *Makespan* and the job throughput becomes lower. In terms of *reliability*, if the resource becomes unavailable before the jobs complete, then multiple jobs will be failed at a time.

  If the *Workload* is low, the PSOPP algorithm will not have to allocate new jobs to *busy* resources. In such cases, if PSOPP algorithm can identify the resource with most powerful CPU and adequate reliability with its fitness function, then the algorithm can perform well in both terms of *speed* and *reliability*.

Table 8-1 compare these three proposed job-scheduling algorithms.

Name	Advantages	Disadvantages
FCFSPP	If the *Workload* is high and each resource has its own regular pattern(s) in terms of *Job Execution Availability*, FCFSPP can make reliable job allocation decisions and have a high job throughput. Secondly, the increase of *Resource Availability Probability Threshold* and *Number of Checking Days* makes FCFSPP more conservative, but this conservativeness permits more reliable job allocation decisions to be made.	If the *Workload* is not high or each resource does not have its own regular pattern(s) in terms of *Job Execution Availability*, FCFSPP can make reliable job allocation decisions and have a high job throughput at the same time.
FLP	If the pattern(s) of all resources' *Job Execution Availability* is similar, the Fuzzy Inference System can observe the changes of some resources' *Job Execution Availability* and achieve a higher job throughput than FCFSPP whilst still making reliable decisions.	If the pattern(s) of all resources' *Job Execution Availability* is dissimilar, the Fuzzy Inference System cannot observe the changes of some resources' *Job Execution Availability* and achieve a higher job throughput than FCFSPP whilst still making reliable decisions.
PSOPP	If the *Workload* is not high and if PSOPP can identify the resource with powerful CPU and adequate reliability, then PSOPP can get shorter job *makespan* than FCFSPP and FLP whilst still making reliable job allocation decisions.	If the *Workload* is high or if PSOPP cannot identify the resource with powerful CPU and adequate reliability, then PSOPP cannot get shorter job *makespan* than FCFSPP and FLP whilst still making reliable job allocation decisions.

Table 8-1: Comparison of Proposed Job-Scheduling Algorithms

In terms of job migration algorithms, especially proactive migration, Periodical Scanning with Predictor (PSPP) and Case Based Reasoning (CBR) algorithms have been proposed:

- The PSPP algorithm is based on scanning resources periodically and judging whether job(s) on each resource need to be migrated using TDE prediction. The objective of this algorithm is to help job-scheduling in terms of improving *reliability* – reducing the number of job failures caused by resource unavailability.

  According to the analysis in Section 5.5, validation in Appendix B.III and evaluation work shown in shown in Section 7.5, this migration algorithm works well if each resource's *Job Execution Availability* possesses a regular pattern that can be observed by the predictor. This is similar to FCFSPP and FLP job-scheduling schemes as they all use the TDE prediction method. If each resource's *Job Execution Availability* is irregular the PSPP algorithm's performance deteriorates, as the TDE prediction method does not work well in such cases.
- CBR migration is a job migration algorithm that observes the *CPU Availability* of each resource and triggers a job migration procedure if the current value of *CPU Availability* is below the *CBR Migration Threshold*. This differs from PSPP, which observes *Job Execution Availability* to make predictions.

  According to the analysis in Section 5.6, validation in Appendix B.IV and evaluation in Section 7.6, this algorithm works well if the all the resources' *CPU Availability Percentage* is lower than the *CPU Migration Threshold* before they become completely *unavailable.* The CBR migration algorithm can observe this (with or without learning from recent decisions) and make correct job migration decisions.

  However, if all resources' *CPU Availability Percentage* do not become low before they become completely *unavailable*, it is difficult for CBR migration algorithm to offer better performance, as there is nothing valuable that can be learnt from the past cases. The CBR migration algorithm is unable to observe the correct threshold after learning or make correct job migration decisions.

  For example, assuming all resources' *CPU Availability Percentage* follow a regular pattern that is always lower than a certain value (e.g. 80%) before they become completely *unavailable*. If the current value of *CPU Migration Threshold* is 90%, the job migration algorithm will trigger job migrations correctly before resources become *unavailable*. Conversely, if the current value of *CPU Migration Threshold* is 50%, the job migration algorithm will not be able to trigger job migrations correctly. Nevertheless, CBR will learn from past decisions and the value of *CPU Migration Threshold* will be raised (e.g. to 90%). Subsequently *CPU Migration Threshold* will be able to capture the change of *CPU Availability Percentage* and the job migration algorithm will be able to trigger job migrations at the appropriate times.

  However, if the resources' *CPU Availability Percentage* does not have a regular pattern before they become completely *unavailable* (e.g. one resource may become *unavailable*

from 100% while another one becomes *unavailable* from 20%), it is difficult for the job migration algorithm to determine a suitable value of *CPU Migration Threshold*. Here is a table to compare these proactive job migration algorithms.

Name	Advantages	Disadvantages
PSPP	If each resource's *Job Execution Availability* has its own regular pattern, PSPP can make correct predictions and trigger proactive migrations to avoid potential job failures.	If each resource's *Job Execution Availability* does not have regular pattern(s), PSPP cannot make correct predictions and trigger proactive migrations to avoid potential job failures.
CBR	If resources' *CPU Availability Percentage* becomes low before becoming *unavailable*, CBR can observe this (with or without learning from recent decisions) and trigger job migrations correctly.	If resources' *CPU Availability Percentage* do not become low before becoming *unavailable*, CBR cannot observe this (with or without learning from recent decisions) and trigger job migrations correctly.

Table 8-2: Comparison of Proposed Proactive Job Migration Algorithms

8.2 Conclusion

This research focuses on three main aspects of volunteered resources based Grids and the research work has been presented and discussed in this thesis. Firstly, this research proposes a new Grid computing system architecture to utilise idle CPU cycles from volunteered resources. The proposed system architecture supports heterogeneous resources, enables resources to support live and automatic job migration and ensures resource owner's local activities are not affected by the Grid jobs.

Secondly, this research proposes some new job-scheduling and migration algorithms aimed at providing reliable job allocation and reallocation decisions whilst maintaining acceptable job throughput. In these job algorithms, a prediction method TDE and AI techniques (including Fuzzy Logic, Particle Swarm Optimisation and Case Based Reasoning) have been utilised. After proposing these algorithms, this research also critically analyses, validates and evaluates all the algorithms in the various scenarios with both synthetic and real data. According to the analysis and simulation results, each algorithm has its own advantages and disadvantages. In general, a certain algorithm performs well if certain specific conditions are met, e.g. each resource behaves with a regular pattern(s) or showing indications before state change.

Thirdly, this research analyses the characteristics of resources in real volunteered resources based Grids. The analysis shows that each Grid has its own characteristics and, perhaps surprisingly, each volunteered resource's *Job Execution Availability* tends to possess weak correlations across different days and times-of-day.

8.3 Future Work

There are a number of ways in which this research work can be extended to further studies both in terms of refining a Grid system architecture for utilising idle CPU cycles on volunteered resources and also job scheduling/migration algorithms it can employ.

Firstly, further developments in terms of the proposed architecture include:

- The system could provide more advanced functionalities to users and resources. This would provide benefits to all components in this system. For users, the system can try to provide quality of service. This could include allowing users to specify the type of resource they would like to use. For example, a user may only wish to use a resource with sufficient CPU speed to get the job completed within a specified time. For resources, the system can try to give rewards to all resources for their contributions or support policies so owners can choose gracefully the times when the resources will be donated to the grid.
- In terms of Java application migration technology, some techniques proposed by other researchers have been adopted in this research. However, as mentioned in Section 3.4, each technique has its own advantage(s) and disadvantage(s). Therefore, more work could be done to provide more efficient and reliable job migration.

Secondly, further studies could be undertaken to improve the performance of the user/resource management components, especially providing better jobs servicing. These include:

- Combining the proposed job scheduling algorithms with regular job checkpointing. This is an option to provide more reliable service to jobs, especially in a Grid system with volunteered and reliable resources. With job checkpointing, the latest job execution states with data can be recorded and stored. They might be stored on the local resources or even by sending back information to the user/resource management in cases where the job needs to be rescheduling if it fails. With the recorded job execution state and data, the job can be executed from the checkpoint onwards using another resource rather than starting from the scratch.
- Combining the proposed job scheduling algorithms with job replication. This is another option to provide more reliable service to jobs, especially in a Grid system with volunteered and reliable resources. With job replication, different resources can execute a single job at the same time for extra *reliability*. If one of the resources remains available until the job completes, the job will finish successfully. However, this extra *reliability* "costs" more resources and provides no *speed* benefit.
- Combining the proposed job scheduling algorithms with job prioritisation. This is an option to provide better service to users. It can be achieved by letting users to specify their jobs' priority when submitting jobs. In job prioritisation, jobs with high priority can be put at the front of the job queue or a separate queue when they arrive at the Grid job scheduler. As a result, jobs with a high priority can be completed more quickly than those jobs with low priority and the users' requirement can be fulfilled.
- The use of additional real data. This could be collected and more characteristics could be obtained, especially in terms of determining characteristics of specific resources or between

different resources in terms of *CPU Availability*, *Job Execution Availability* or *Resource Availability*. According to the results shown in [Kondo05] and this research, the results so far appear to suggest that this type of relationship is weak. However, more data needs to be collected and analysed to confirm this. In addition, even if a weak relationship is confirmed for "today's" networks, it is not easy to say whether this will still be the case in future as network technologies are still developing and human behaviour in terms of computer usage patterns may change as well.

Thirdly, though this research work was aimed at solving some challenges in Grid computing environments, especially volunteered resources based Grid computing environments, the results from this research (including the proposed system architecture, job scheduling and migration algorithms) are not only applicable in Grid computing context but it is also possible to apply it to other areas. One of the application areas is Cloud computing context.

Cloud computing is a new terminology which has been proposed in recent years. The basic idea of Cloud computing indicates that in the future, people will compute in centralised facilities (somewhere on the "cloud") operated by third-party providers, there are some significant differences between Cloud computing and Grid computing.

In a Cloud computing environment, although resources are deemed to be reliable and fully controlled by the service provider, resource management is still an important issue as the environment is dynamic. For example, different resources may have to face different workloads at different times and need to cope with different applications that have different latency requirements. Therefore, the job-scheduling and migration algorithms proposed in this research could be used or adapted to the scheduler and to reschedule different applications.

In the core of the Cloud, it is likely to be composed of various software and hardware components and the CPU processor will be an important component. Therefore, it is also possible to apply the system architecture proposed in this research to utilise idle CPU cycles from different CPU processors. For each CPU processor in the Cloud, it may not be busy all the time. Therefore, the idle CPU cycles on each resource can be utilised to process computational jobs (this could be jobs submitted by users of the Cloud) by using a job-scheduling algorithm proposed in this research. If the resources are going to be busy again, the computational jobs can be migrated to another idle resource for completion with the job migration algorithm proposed by this research. As a result, this can be considered as opportunistic computing in a Cloud computing context and the idle CPU cycles of different CPU processors can be utilised efficiently.

References

[Aamodt94] Agnar Aamodt, Enric Plaza, Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches, AICom – Artificial Intelligence Communications, IOS Press, Vol.7:1, pp. 39-59, 1994

[Acharya97] A. Acharya, G. Edjlali, and J. Saltz. The Utility of Exploiting Idle Workstations for Parallel Computation. In Proceedings of the 1997 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pages 225–234, 1997.

[AMD10] Quad-Core AMD Processors, Advanced Micro Devices, URL: http://multicore.amd.com/us-en/AMD-Multi-Core.aspx, Date accessed: 10th Feb, 2010

[Anderson05] Anderson, D.P, Korpela, E. Walton, R, High-performance task distribution for volunteer computing, e-Science and Grid Computing, 2005, 1-1 July 2005, page(s): 8 pp.-203, Melbourne, Vic., ISBN: 0-7695-2448-6

[Anderson07] Anderson, D.P, Local Scheduling for Volunteer Computing, Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, 26-30 March 2007, page(s): 1-8, ISBN: 1-4244-0910-1

[Arpaci95] The Interaction of Parallel and Sequential Workloads on a Network of Workstations Arpaci, R.H. and Dusseau, A.C. and Vahdat, A.M. and Liu, L.T. and Anderson, T.E. and Patterson, D.A. Proceedings of SIGMETRICS'95, May, 1995, pp. 267-278

[Arnold05] Ken Arnold, James Gosling, David Holmes, The Java Programming Language (Fourth Edition), URL: http://java.sun.com/docs/books/javaprog/, Date accessed: 10th Feb, 2010

[Barak05] Barak A., Shiloh A. and Amar L., An Organizational Grid of Federated MOSIX Clusters, Proc. 5-th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05), pp. 350-357, Cardiff, May 2005.

[Barak08] Maoz T., Barak A. and Amar L., Combining Virtual Machine Migration with Process Migration for HPC on Multi-Clusters and Grids, IEEE Cluster 2008, Tsukuba, Sept. 2008

[Beelucid09] Beelucid Software LLC, A New Approach to Migrating VB.Net Applications to Java, URL: http://www.beelucid.com/products/do_download_pdf, Date accessed: 10th Feb, 2010

[Bhagwan03] R. Bhagwan, S. Savage, and G. Voelker. Understanding Availability. In In Proceedings of IPTPS'03, 2003.

[Black08] Paul E. Black, "NP-complete", in Dictionary of Algorithms and Data Structures [online], Paul E. Black, ed., U.S. National Institute of Standards and Technology, URL:http://www.itl.nist.gov/div897/sqg/dads/HTML/npcomplete.html, Date accessed: 10th Feb, 2010

[BOINC10a] Volunteer computing, URL: http://boinc.berkeley.edu/trac/wiki/VolunteerComputing, Date accessed: 10th Feb, 2010

[BOINC10b] BOINC Project, URL: http://boinc.berkeley.edu/, Date accessed: 10th Feb, 2010

[Bolosky00] W. Bolosky, J. Douceur, D. Ely, and M. Theimer. Feasibility of a Serverless Distributed file System Deployed on an Existing Set of Desktop PCs. In *Proceedings of SIGMETRICS*, 2000.

[Bouchenak00a] S. Bouchenak, D. Hagimont, Approaches to Capturing Java Threads State", In Middleware 2000

[Bouchenak00b] S. Bouchenak, D. Hagimont, Pickling threads in the Java System, Technology of Object-Oriented Languages, 2000. TOOLS 33. Proceedings. 33rd International Conference

[Boyd02] Tom Boyd and Partha Dasgupta, "Process Migration: A Generalized Approach Using a Virtualizing Operating System", Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on 2-5 July 2002 Page(s):385 - 392 Digital Object Identifier 10.1109/ICDCS.2002.1022276

[Brevik03] J. Brevik, D. Nurmi, and R. Wolski. Quantifying Machine Availability in Networked and Desktop Grid Systems, Technical Report CS2003-37, Dept. of Computer Science and Engineering, University of California at Santa Barbara, November 2003.

[Bridgeport01] Bridgeport University, CPU Scheduling, URL: http://www1bpt.bridgeport.edu/sed/projects/cs503/Spring_2001/kode/os/scheduling.htm#sjf, Date accessed: 08 Apr, 2010

[Brule05] James F. Brule, Fuzzy System – A Tutorial, URL: http://www.austinlinks.com/Fuzzy/tutorial.html, Date accessed: 10th Feb, 2010

[C2J01] C2J Converter, URL: http://tech.novosoft-us.com/product_c2j.jsp, Date accessed: 10th Feb, 2010

[Casavant88] Thomas L. Casavant, Jon G. Kuhl, A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems, IEEE Transaction on software engineering, Vol. 14, No. 2, February 1988

[CBRwiki10] Case Based Reasoning, URL: http://en.wikipedia.org/wiki/Case-based_reasoning, Date accessed: 10th Feb, 2010

[Climate10] Climateprediction.net, URL: http://climateprediction.net/, Date accessed: 10th Feb, 2010

[Cohen88] Cohen, J. (1988). Statistical power analysis for the behavioral sciences (2nd ed.)

[Condor10a] Condor High Throughput Computing, URL: http://www.cs.wisc.edu/condor/, Date accessed: 10th Feb, 2010

[Condor10b] An Overview of the Condor System on High-Throughput Computing, URL: http://www.cs.wisc.edu/condor/overview/, Date accessed: 10th Feb, 2010

[Condor10c] Condor: A Distributed Job Scheduler, URL: http://www.cs.wisc.edu/condor/doc/beowulf-chapter-rev1.ps, Date accessed: 10th Feb, 2010

[Corbatto00] M.Corbatto, An introduction to PORTABLE BATCH SYSTEM (PBS), URL: http://hpc.sissa.it/pbs/pbs.html, Date accessed: 10th Feb, 2010

[Correlation10] Correlation and dependence, URL: http://en.wikipedia.org/wiki/Correlation_and_dependence, Date accessed: 10th, Feb, 2010

[Czajkowski10] Karl Czajkowski, Globus GRAM, URL: http://www.globusconsortium.org/journal/20060109/czajkowski.html, Date accessed: 10th Feb, 2010

[Delphi10] Delphi Basics, URL: http://www.delphi.co.nr/, date accessed: 10th, Feb, 2010.

[DES10] Discrete-event simulation, URL: http://en.wikipedia.org/wiki/Discrete_event_simulation, Date accessed: 10th Feb, 2010

[DGRID03] Desktop Grids: Critical Systems and Applications Research (DGRID 2003), Phoenix, Arizona, 17 November 2003, URL: http://www-csag.ucsd.edu/DGRID03/, Date accessed: 10th Feb, 2010

[Dinda99] P. Dinda and D. O'Hallaron. An extensive toolkit for resource prediction in distributed systems. Technical Report CMU-CS-99-138, Carnegie Mellon University, 1999.

[Dinda00] Load Trace Archive, URL: http://www.cs.northwestern.edu/~pdinda/LoadTraces/, Date accessed: 10th Feb, 2010

[Dinda02] P. Dinda. A Prediction-Based Real-Time Scheduling Advisor. In Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02), April 2002.

[Dogan02] A, Dogan. F, Ozguner., Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems, 13(3):308–323, 2002.

[Dusseau09] Remzi Arpaci Dusseau, Scheduling: Introduction, URL: http://pages.cs.wisc.edu/~remzi/Classes/537/Spring2009/Notes/cpu-sched.pdf, Date accessed: 08 Apr, 2010

[Esklcloglu01] Esklcloglu and Marsland, Scheduling, URL: http://webdocs.cs.ualberta.ca/~tony/C379/Notes/PDF/05.4.pdf, Date accessed: 10th Feb, 2010

[Feller07] M.Feller, I.Foster and S. Martin, GT4 GRAM: A Functionality and Performance Study, TeraGrid Conference, 2007, Madison, WI, June 2007

[Foster02] Foster I. What is the grid? A three points checklist. Grid Today, 2002,1(6). URL: http://www.gridtoday.com/02/0722/100136.html, Date accessed: 10th Feb, 2010

[Foster06] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems, IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2006.

[Foster09] Ian Foster, Carl Kesselman's, The Grid 2: Blueprint for a New Computing Infrastructure, ISBN: 9780080521534, parent-ISBN: 9781558609334, Publisher: Morgan Kaufmann Publishers, Jan, 2009

[Frey01] Frey, J.; Tannenbaum, T.; Livny, M.; Foster, I.; Tuecke, S.; Condor-G: a computation management agent for multi-institutional grids, High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on 7-9 Aug. 2001 Page(s):55 – 63

[Garey79] Garey, M.R.; Johnson, D.S., Computers and Intractability: A Guide to the Theory of NP-Completeness. New York: W.H. Freeman. ISBN 0-7167-1045-5. 1979

[Garritano03] Tom Garritano, Globus: An Infrastructure for Resource Sharing, ClusterWorld's On the Grid Column, December 2003

[GIMPS10] Great Internet Mersenne Prime Search GIMPS, URL: http://www.mersenne.org/freesoft/, Date accessed: 10th Feb, 2010

[Globus10a] The Globus Alliance, URL: http://www.globus.org/, Date accessed: 10th Feb, 2010

[Globus10b] Globus Team, URL: http://www.globus.org/alliance/team/, Date accessed: 10th Feb, 2010

[Goebel03] Greg Goebel, An Introduction to Fuzzy Control Systems, URL: http://www.faqs.org/docs/fuzzy/ v1.0.4, 01 June 2003, Date accessed: 10th Feb, 2010

[Gosling96] James Gosling, Henry McGilton, White Paper – The Java Language Environment, URL: http://java.sun.com/docs/white/langenv/, Date accessed: 10th Feb, 2010

[Haas10] Juergen Haas, Compiled Language, URL: http://www.tldp.org/LDP/Linux-Dictionary/html/index.html, Date accessed: 10th Feb, 2010

[Huang02] Zhisheng Huang, Anton Eliëns, and Cees Visser, 3D Agent-based Virtual Communities, Proceedings of the 2002 Web3D Conference, Tempe, Arizona, USA, 2002

[Hellmann01] Martin Hellmann, Fuzzy Logic Introduction, URL: http://epsilon.nought.de/tutorials/fuzzy/fuzzy.pdf, Date accessed: 10th Feb, 2010

[HTC10] High Throughput Computing, URL: http://www.cs.wisc.edu/condor/htc.html, Date accessed: 10th Feb, 2010

[Hu06] Xiaohui Hu, PSO Tutorial, URL: http://www.swarmintelligence.org/tutorials.php, 2006, Date accessed: 10th Feb, 2010

[IBM08] Compiled versus Interpreted Language, IBM, URL: http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp?topic=/com.ibm.zos.zappldev/zappldev_85.htm, Date accessed: 10th Feb, 2010

[Illmann00] Torsten Illmann, Frank Kargl, Migration of Mobile Agent in Java - Problems,Classification and Solutions, Torsten Illmann University, Torsten Illmann, Michael Weber, In Proc. of the Int'l ICSC Symposium on Multi-Agents and Mobile Agents in Virtual Organizationa and E-Commerce, MAMA 2000

[Intel10] Intel® Multi-Core Technology, Intel Corporation, URL: http://www.intel.com/multi-core/, Date accessed: 10th Feb, 2010

[Jacob02] Bart Jacob, Viktors Berstis, Fundamental of Grid Computing, URL: http://www.redbooks.ibm.com/abstracts/redp3613.html, Date accessed: 10th Feb, 2010

[JPC10] The Pure Java x86 PC Emulator, Oxford University, URL: http://www-jpc.physics.ox.ac.uk/download_faq.html, Date accessed: 10th Feb, 2010

[Kenndy95] James Kenndy, Russell Eberhart, Particle Swarm Optimization, Proc. IEEE Int'l. Conf. on Neural Networks, 1995

[Kondo05] Derrick Kondo, Scheduling Task Parallel Aplications For Rapid Turnaround on Desktop Grids, Dissertation, University of California, San Diego, La Jolla, CA 92093, 2005.

[Kondo07] Derrick Kondo, Gilles Fedak, Franck Cappello, Andrew A. Chien, Henri Casanova, Resource Availability in Enterprise Desktop Grids, appear in the Journal of Future Generation Computer Systems, 2007

[Kondo09] Desktop Grid Trace Archive, URL: http://xw01.lri.fr:4320/dg/, Date accessed: 10th Feb, 2010

[Kumar06] J.I Nirmal Kumar, Hiren Soni and Rita N.Kumar, Biomonitoring of selected freshwater macrophytes to access lake trace element contamination: a case study of Nal Sarovar Bird Sanctuary, Gujarat, Indiat, J. Limnol, 65(1): 9-16, 2006

[Lazarevic06] A. Lazarevi′c and L. Sacks, "Managing Uncertainty - A Case for Probabilistic Grid Scheduling", Proceedings of The Seventh International Meeting on High Performance for Computational Science - VECPAR 2006, Rio de Janeiro, Brazil, July 2006.

[Leake96] David B. Leake, CBR in Context: The Present and Future, Case-Based Reasoning: Experience, lessons, and Future Directions, AAAI Press/MIT Press, 1996

[Lo84] V. M. Lo, Heuristic algorithms for task assignment in distributed systems, in Proc. 4th Internal Conference Distributed Computing Systems, May 1984, pp. 30-39

[Long95] D. Long, A. Muir, and R. Golding. A Longitudinal Survey of Internet Host Reliability, In 14th Symposium on Reliable Distributed Systems, pages 2–9, 1995.

[Ma00] Matchy J.M. Ma, Cho-Li Wang and Francis C.M. Lau, "Delta Execution: A preemptive Java thread migration mechanism", Cluster Computing, Springer Netherlands ISSN1386-7857 (Print) 1573-7543 (Online), Volume 3, Number 2 / 2000 DOI10.1023/A:1019071902255 Page 83-94

[Ma02] Ma, R.K.K., Cho-Li. Wang, Lau, F.C.M, "M-JavaMPI: A Java-MPI Binding with Process Migration Support", Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium

[Mickens05] J. Mickens and B. Noble. Predicting node availability in peer-to-peer networks. In International Conference on Measurement and Modeling of Computer Systems, 2005.

[Mickens06] J. Mickens and B. Noble. Exploiting availability prediction in distributed systems. In Network Systems Design and Implementation, pages 73–86, 2006.

[Mickens07] J. Mickens and B. Noble. Improving distributed system performance using machine availability prediction. International Conference on Measurement and Modeling of Computer Systems Performance Evaluation Review, 34(2), 2006.

[Microsoft10] MSDN, Checkpoint Restart, Microsoft, URL: http://msdn.microsoft.com/en-us/library/ms894386.aspx, Date accessed: 10th Feb, 2010

[Morris95] Bonnie Morris, "Case-Based Reasoning", URL: http://accounting.rutgers.edu/raw/aies/www.bus.orst.edu/faculty/brownc/aies/news-let/fall95/casebase.htm, Date accessed: 10th Feb, 2010

[Mu'alem01] Mu'alem, A.W., Feitelson. D.G., Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling, Parallel and Distributed Systems, IEEE Transactions on June 2001, Volume: 12, Issue: 6, On page(s): 529-543 ISSN: 1045-9219

[Mutka91] M. Mutka and M. Livny. The available capacity of a privately owned workstation environment . *Performance Evaluation*, 4(12), July 1991.

[Neri00] C. Germain, V. Neri, G. Fedak and F. Cappello, XtremWeb: Building an Experimental Platform for Global Computingin Proceedings of Grid2000 (Worshop in HIPC2000), Bngalore India, 2000

[NPCWiki10] NP-Complete, URL: http://en.wikipedia.org/wiki/NP-complete, Date accessed: 10th Feb, 2010

[Physorg10] Grid computing, URL: http://www.physorg.com/tags/grid+computing/, Date accessed: 10th Feb, 2010

[PMCC10] Pearson product-moment correlation coefficient, URL: http://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient, Date accessed: 10th Feb, 2010

[Ren06a] X. Ren, S. Lee, R. Eigenmann, and S. Bagchi. Resource failure prediction in fine-grained cycle sharing system. In International Conference on High Performance Distributed Computing, 2006.

[Ren06b] Xiaojuan Ren; Eigenmann, R., Empirical Studies on the Behavior of Resource Availability in Fine-Grained Cycle Sharing Systems, Parallel Processing, 2006. ICPP 2006. 14-18 Aug. 2006 Page(s):3 – 11, Digital Object Identifier 10.1109/ICPP.2006.39

[Ren07a] Xiaojuan Ren, Rudolf Eigenmann, Saurabh Bagchi, Failure-aware checkpointing in fine-grained cycle sharing systems, High Performance Distributed Computing archive, Proceedings of the 16th international symposium on High performance distributed computing table of contents, Pages: 33 – 42, Year of Publication: 2007, ISBN:978-1-59593-673-8

[Ren07b] Xiaojuan Ren, Seyong Lee, Rudolf Eigenmann, Saurabh Bagchi, Prediction of Resource Availability in Fine-Grained Cycle Sharing Systems Empirical Evaluation. J. Grid Comput. 5(2): 173-195, 2007

[Richter06] Michael M. Richter, Agnar Aamoodt, Case-based reasoning foundations, The Knowledge Engineering Review, Vol. 20:3, 203-207, 2006, Cambridge University Press

[Robinson04] Stewart Robinson, Simulation - The practice of model development and use. Wiley, 2004.

[Rodgers88] J. L. Rodgers and W. A. Nicewander. Thirteen ways to look at the correlation coefficient. The American Statistician, 42(1):59–66, Feb 1988.

[Rood07] Rood, B. Lewis, M.J., Multi-state Grid Resource Availability Characterization, Proceedings of Grid Computing, 2007 8th IEEE/ACM International Conference, ISBN: 978-1-4244-1560-1

[Rood08] Rood, B. Lewis, M.J., Resource Availability Prediction for Improved Grid Scheduling, Proceedings of eScience, 2008. eScience '08. IEEE Fourth International Conference, ISBN: 978-1-4244-3380-3

[Saroiu02] S. Saroiu, P.K. Gummadi, and S.D. Gribble. A measurement study of peer-to-peer file sharing systems. In Proceedinsg of MMCN, January 2002.

[SETI10] SETI@Home, URL: http://setiathome.berkeley.edu/, Date accessed: 10th Feb, 2010

[Simon05] Steve Simon, Stats: What is a correlation? (Pearson correlation), URL: http://childrens-mercy.org/stats/definitions/correlation.htm, Date accessed: 10th Feb, 2010

[Slade91] Stephen Slade, Case-Based Reasoning: A Research Paradigm, AI Magazine Volume 12 Number 1, 1991

[Smith] Roger Smith, Grid Computing: A Brief Technology Analysis, URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.109.943, Date accessed: 10th Feb, 2010

[Sun04] Java™ Virtual Machine Debug Interface Reference, Sun Microsystems, URL: http://java.sun.com/j2se/1.5.0/docs/guide/jpda/jvmdi-spec.html, Date accessed: 10th Feb, 2010

[Sun06] Java™ Virtual Machine Tool Interface (JVMTI), Sun Microsystems, URL: http://java.sun.com/javase/6/docs/platform/jvmti/jvmti.html, Date accessed: 10th Feb, 2010

[Sun10a] Java Technology Homepage, Sun Developer Network, URL: http://java.sun.com, Date accessed: 10th Feb, 2010

[Sun10b] The Java Tutorials, Sun Developer Network, URL: http://java.sun.com/docs/books/tutorial/, Date accessed: 10th Feb, 2010

[Sun10c] About the Java Technology, Sun Developer Network, URL: http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html, Date accessed: 10th Feb, 2010

[Sun10d] Java Technology Reference, Sun Developer Network, URL: http://java.sun.com/reference/index.jsp, Date accessed: 10th Feb, 2010

[StanKovic98] John A. StanKovic, Macro Spuri, Krithi, Ramamritham, Giorgio C. Buttazzo, Deadline Scheduling for Real-Time Systems – EDF and Related Algorithms (The Springer International Series in Engineering and Computer Science), Springer, 1998, ISBM:0792382692

[Stigler89] Stigler, Stephen M. "Francis Galton's Account of the Invention of Correlation". *Statistical Science* **4** (2), 1989

[Stroustrup04] Bjarne Stroustrup, The C++ Programming Language (Third Edition and Special Edition), Addison-Wesley, ISBN 0-201-88954-4 and 021-70073-5

[Synaptic06] Fuzzy Logic, URL: http://en.wikipedia.org/wiki/Fuzzy_logic, Date accessed: 10th Feb, 2010

[Thain05] Douglas Thain and Todd Tannenbaum and Miron Livny, Distributed Computing in Practice: The Condor Experience, Concurrency and Computation: Practice and Experience, 2005, volume 17, pages = 2-4

[Thomas56] Thomas E. Phipps Jr. and W.R. Van Voorhis, Machine Repair as a PriorityWaiting-Line Problem, Operations Research, Vol. 4, No. 1 (Feb. 1956), pp. 76-86.

[Truyen00] Eddy Truyen, Bert Robben, Bart Vanhaute, Tim Coninx, Wouter Joosen and Pierre Verbaeten, Portable Support for Transparent Thread Migration in Java, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, ISSN 0302-9743 (Print) 1611-3349 (Online), Volume 1882/2000, Agent Systems, Mobile Agents, and Applications

[TUDelft10] The Grid Workloads Archive, URL: http://gwa.ewi.tudelft.nl/pmwiki/pmwiki.php?n=Workloads.Overview, Date accessed: 10th Feb, 2010

[UNC09] LSF (Load Sharing Facility) Overview, URL: http://help.unc.edu/4484, Date accessed: 10th Feb, 2010

[VCSC10] Create a Virtual Campus Supercomputing Center (VCSC), URL: http://boinc.berkeley.edu/trac/wiki/VirtualCampusSupercomputerCenter, Date accessed: 10th Feb, 2010

[Volwiki10] Volunteer computing, URL: http://en.wikipedia.org/wiki/Volunteer_computing, Date accessed: 10th Feb, 2010

[VSS10] Correlation: Interpretations, Visual Statistics Studio, URL: http://www.visualstatistics.net/Visual%20Statistics%20Multimedia/correlation_interpretation.htm, Date accessed: 10th, Feb, 2010

[Watson94] Ian Watson, Farhi Marir, "Case-Based Reasoning: A Review", The Knowledge Engineering Review, 1994

[Xgrid10a] Xgrid Overview, URL: http://www.apple.com/server/macosx/technology/, Date accessed: 10th Feb, 2010

[Xgrid10b] Xgrid, URL: http://en.wikipedia.org/wiki/Xgrid, Date accessed: 10th Feb, 2010

[Xtremweb08a] Introduction to Xtremweb, URL: http://www.xtremweb.net/introduction.html, Date accessed: 10th Feb, 2010

[Xtremweb08b] Computing on Large Scale Distributed Systems, URL: http://www.xtremweb.net/desktopgrids.html, Date accessed: 10th Feb, 2010

[Zadeh65] L.A. Zadeh, Fuzzy Sets, URL: http://www-bisc.cs.berkeley.edu/zadeh/papers/Fuzzy%20Sets-1965.pdf, Date accessed: 10th Feb, 2010

[Zadeh73] L.A. Zadeh, Outline of a new approach to the analysis of complex systems and decision processes, IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-3 (1973), pp. 28-44

Appendix A

In order to make the communication among users, Grid job scheduler and resources efficiently, some types of messages are designed to use in different scenarios.

I. Registration Message

In order to join the Grid, both users and resources need to register at the Grid firstly. Therefore, the *Registration Message* is needed. With this message, the user/resource manager will create a new entry and record the information provided by the resource in its user/resource database for the resource. Figure A.1 shows the format of this message:

Message Type
Message Sequence Number
User/Resource IP Address
User/Resource Name

Figure A.1: *Registration Message*

Each field in the *Registration Message* is described as follow:

Message Type: This field will be filled as "Registration Message" when a resource sends this message to the user/resource manager.

Message Sequence Number: This field records the message's sequence number.

User/Resource IP Address: This field records the user/resource's IP address.

User/Resource Name: This field records the name of the resource.

II. Registration Acknowledgement Message

In order to tell to the resource that the resource is already registered at the user/resource manager, a *Registration Acknowledgement Message* is needed. From this message, the resource will know that it is already registered at the user/resource manager and get the user/resource ID assigned by the user/resource manager. Figure A.2 shows the format of this message:

Message Type
User/Resource ID

Figure A.2: *Registration Acknowledgement Message*

Each field in the *Registration Acknowledgement Message* is described as follow:

Message Type: This field will be filled as "Registration Acknowledgement Message" when a resource sends this message to the user/resource manager.

User/Resource ID: This field specifies the resource ID assigned by the user/resource manager.

Figure A.3 shows the procedure for a resource to register at the user/resource manager:

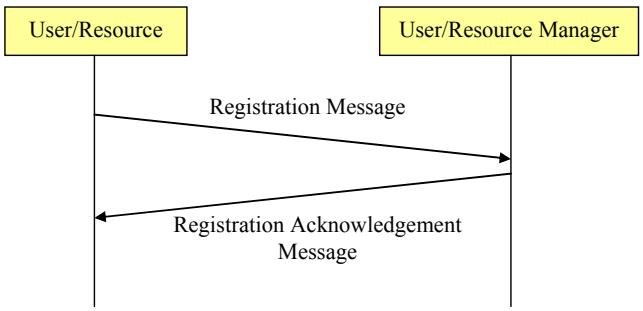


Figure A.3: Procedure of Registration

If the *Registration Message* gets lost on its way to the user/resource manager, the user/resource will resend the *Registration Message* after timeout. If the *Registration Acknowledgement Message* gets lost on its way to the user/resource, the user/resource will resend the *Registration Message* again after time out as well. When the user/resource manager receives this duplicated *Registration Message*, it will find out the user/resource ID from its resource database and send the *Registration Acknowledgement Message* back to the resource again. Figure A.4 shows this scenario:

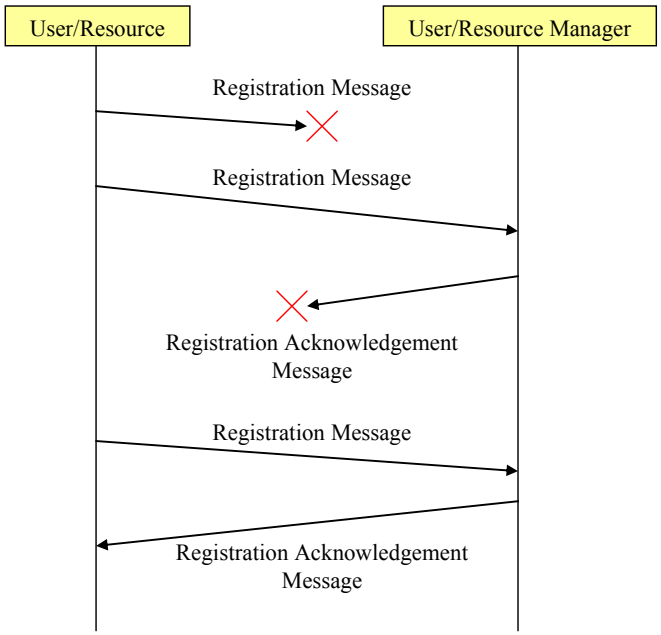


Figure A.4: Procedure of resending *Registration and Registration Acknowledgement Message*

III. Resource Information Message

In order to let the resource manager realise the updated system information, each resource will send its updated system information to the resource manager at regular interval. In addition, the resource manager will request the resource's information whenever necessary. After receiving this message, the resource manager will update the resource's information in the resource database. With this information, the resource manager could also make stats of the resource and analyse the resource's performance. Figure A.5 shows the format of this message.

Message Type
Message Time Stamp
Resource ID
CPU Speed
CPU Usage

Figure A.5: *Resource Information Message*

Each field in the *Resource Information Message* is described as follow:

Message Type: This field will be filled as "Resource Information Message" when a resource sends this message to a resource manager.

Resource ID: Each resource has a unique resource ID.

Message Time Stamp: Shows message's sending time.

CPU Speed: Records the CPU speed of the resource.

CPU Usage: Records the current usage percentage of the CPU.

IV. Request Resource Information Message

Though the resource sends *Resource Information Message* to the resource manager at regular interval, the resource manager may not always receive the message because the message may get lost on its way to the resource manager or the resource is no longer available. Therefore, the resource manager can use *Request Resource Information Message* to request the resource's information initiatively. Figure A.6 shows the format of this message:

Message Type

Figure A.6: *Request Resource Information Message*

Each field in the *Resource Information Message* is described as follow:

Message Type: This field will be filled as "Request resource Information Message" the resource manager sends this message to a resource.

V. Job Submission/Allocation Message

When a user decides to submit a job to the Grid job scheduler, the user will inform the Grid job scheduler firstly. When the Grid job scheduler decides to allocate a job to the resource, the Grid job scheduler will inform the resource firstly. Therefore, *Job Submission/Allocation Message* will be used. Figure A.7 shows the format of this message:

Message Type
Job ID

Figure A.7: *Job Submission/Allocation Message*

Each field in the *Job Submission/Allocation Message* is described as follow:

Message Type: This field will be filled as "Job Submission/Allocation Message" when a user sends this message to the Grid job scheduler or the Grid job scheduler sends this message to a resource.

Job ID: This field specifies the ID of the job which is going to be sent.

VI. Job Submission/Allocation Acknowledgement Message

When the Grid job scheduler receives the *Job Submission/Allocation Message* from a user, the Grid job scheduler needs to tell the user that it is ready to receive the job. When the resource receives the *Job Submission/Allocation Message* from the Grid job scheduler, the resource needs to tell the Grid job scheduler that it is ready to receive the job. Therefore, the *Job Submission/Allocation Acknowledgement Message* will be used. Figure A.8 shows the format of this message:

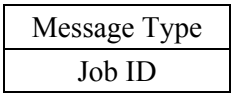


Figure A.8: *Job Submission/Allocation Acknowledgement Message*

Each field in the *Job Submission/Allocation Acknowledgement Message* is described as follow:

Message Type: This field will be filled as "Job Submission/Allocation Acknowledgement Message" when the Grid job scheduler sends this message to a user or a resource sends this message to the Grid job scheduler.

Job ID: This field specifies the ID of the job which is going to be received.

VII. Job Submission/Allocation Completion Message

After sending the whole job, the resource needs to tell the Grid job scheduler or the Grid job scheduler needs to tell the user it has already finished sending the job. Therefore, the Grid job scheduler will send a "Job Submission/Allocation Completion Message" to the resource: Figure A.9 shows the format of this message:

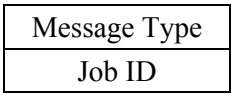


Figure A.9: *Job Submission/Allocation Completion Message*

Each field in the *Job Submission/Allocation Completion Message* is described as follow:

Message Type: This field will be filled as "Job Submission/Allocation Completion Message" when the Grid job scheduler sends this message to a user or a resource sends this message to a Grid job scheduler.

Job ID: This field specifies the ID of the job which has been already sent.

VIII. Job Submission/Allocation Completion Acknowledgement Message

When the Grid job scheduler/resource receives a *Job Submission/Allocation Completion Message*, it should reply to the user/Grid job scheduler to tell the user/Grid job scheduler that it has already received the *Job Allocation Completion Message.* So the resource will send a "*Job Submission/Allocation Completion Acknowledgement Message*" back to the Grid job scheduler. Figure A.10 shows the format of this message:

Message Type
Job ID

Figure A.10: *Job Submission/Allocation Completion Acknowledgement Message*

Each field in the *Job Submission/Allocation Completion Message* is described as follow:

Message Type: This field will be filled as "Job Submission/Allocation Completion Acknowledgement Message" when a resource sends this message to the Grid job scheduler or the Grid job scheduler sends this message to a resource.

Job ID: This field specifies the ID of the job which has been already received.

Figure A.11 shows the whole procedure of allocating and transmitting a job from the user/Grid job scheduler to the Grid job scheduler/resource:

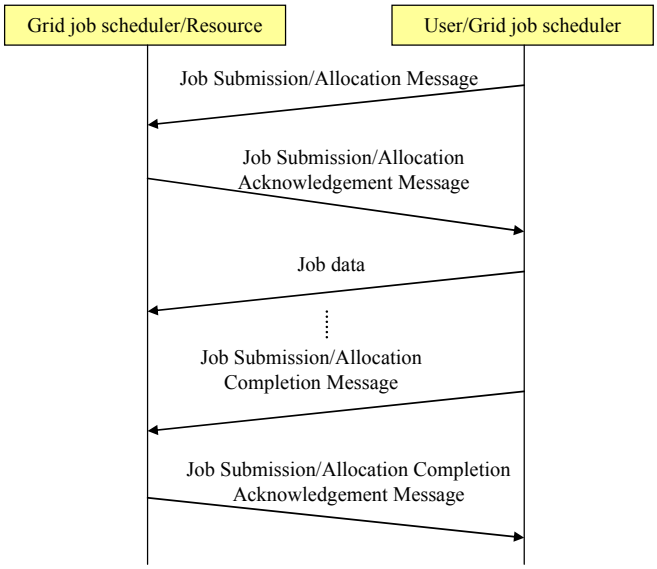


Figure A.11: Procedure of Job Submission/Allocation and Transmission

IX. Job Information Message

During the execution of the job, the resource software will monitor the execution progress and report to the job manager when important events occur. Therefore, *Job Information Message* will be used. Figure A.12 shows the format of this message:

Message Type
Job ID
State

Figure A.12: *Job Submission/Allocation Message*

Each field in the *Job Submission/Allocation Message* is described as follow:

Message Type: This field will be filled as "Job Submission/Allocation Message" when a user sends this message to the Grid job scheduler or the Grid job scheduler sends this message to a resource.

Job ID: This field specifies the ID of the job which is going to be sent.

State: This field specifies the job's current state, e.g. job started, job finished, job delayed, etc.

X. Migration Notification Message

In order to let a resource migrate a job to another resource, the Grid job scheduler should notify each resource firstly. Therefore, *Migration Notification Message* is used when the Grid job scheduler decides to ask a resource to migrate a job to another resource. Figure A.13 shows the format of this message:

Message Type
Original Resource ID
Original Resource IP Address
Destination Resource ID
Destination Resource IP Address
Job ID

Figure A.13: *Migration Notification Message*

Each field in the *Migration Notification Message* is described as follow:

Message Type: This field will be filled as "Migration Notification Message" when the Grid job scheduler sends this message to a resource.

Original Resource ID: This field specifies the ID of the resource which the job should be emigrated from.

Original Resource IP Address: This field specifies the IP address of the original resource.

Destination Resource ID: This field specifies the ID of the resource which the job should be emigrated to.

Destination Resource IP Address: This field specifies the IP address of the destination resource.

Job ID: This field specifies the ID of the job which is going to be migrated.

XI. Migration Notification Acknowledgement Message

In order to let the Grid job scheduler know both the original and the destination resource have received the *Migration Notification Message*, the resource should send a *Migration Notification Acknowledgement Message* back to the Grid job scheduler. Figure A.14 shows the format of this message:

Message Type
Resource ID
Job ID

Figure A.14: *Migration Notification Acknowledgement Message*

Each field in the *Migration Notification Acknowledgement Message* is described as follow:

Message Type: This field will be filled as "Migration Notification Acknowledgement Message" when a resource sends this message to a Grid job scheduler.

Resource ID: This field specifies the ID of the resource which sends this message.

Job ID: This field specifies the ID of the job which is going to be migrated.

XII. Migration Connection Request Message

When the resource receives a *Migration Notification Message* from a Grid job scheduler, it should start to migrate the specified job to the destination resource. Before the job migration, it should firstly set up a connection with the destination resource. Therefore, the *Migration Connection Request Message* is used for the resource to send a connection request to the destination resource. Figure A.15 shows the format of this message.

Message Type
Resource ID
Resource IP Address
Job ID

Figure A.15: *Migration Connection Request Message*

Each field in the *Migration Connection Request Message* is described as follow:

Message Type: This field will be filled as "Migration Connection Request Message" when the resource sends this message to the destination resource.

Resource ID: This field shows the ID of the resource.

Resource IP Address: This field shows the IP address of the resource.

Job ID: This field specifies the ID of the job which is going to be migrated.

XIII. Migration Connection Acknowledgement Message

After the destination resource receives the *Migration Connection Request Message* from the resource, it should send a *Migration Connection Acknowledgement Message* back to the resource to accept this connection. Figure A.16 shows the format of this message:

Message Type
Destination Resource ID
Destination Resource IP Address
Job ID

Figure A.16: *Migration Connection Acknowledgement Message*

Each field in the *Migration Connection Request Message* is described as follow:

Message Type: This field will be filled as "Migration Connection Acknowledgement Message" when the destination resource sends this message to the resource.

Destination Resource ID: This field shows the ID of the destination resource.

Destination Resource IP Address: This field shows the IP address of the destination resource.

Job ID: This field specifies the ID of the job which is going to be migrated.

XIV. Migration Completion Message

After transmitting the whole job, the resource should send a *Migration Completion Message* to tell the destination resource that the job has been completely transmitted. Figure A.17 shows the format for this message:

Message Type
Job ID

Figure A.17: *Migration Completion Message*

Each field in the *Migration Completion Message* is described as follow:

Message Type: This field will be filled as "Migration Completion Message" when the original resource sends this message to the destination resource.

Job ID: This field specifies the ID of the job which has been already migrated.

XV. Migration Completion Acknowledgement Message

After receiving the *Migration Completion Message* sent from the original resource, the destination resource should send back a *Migration Completion Acknowledgement Message* to tell the original resource that it has received the *Migration Completion Message*. In addition, the destination resource should notify the Grid job scheduler so that the Grid job scheduler could update the information stored in its databases. After the Grid job scheduler receives a *Migration Completion Acknowledgement Message*, it needs to tell the destination resource that it has received the *Migration Completion Acknowledgement Message* by sending back another *Migration Completion Acknowledgement Message*. This message is identical as the one it received from the destination resource. Figure A.18 shows the format for this message:

Message Type
Original Resource ID
Destination Resource ID
Job ID

Figure A.**18: *Migration Notification Message***

Each field in the *Migration Completion Acknowledgement Message* is described as follow:

Message Type: This field will be filled as "Migration Completion Acknowledgement Message" when the Grid job scheduler sends this message to a resource.

Original Resource ID: This field specifies the ID of the original resource.

Destination Resource ID: This field specifies the ID of the resource which the job should be emigrated to.

Job ID: This field specifies the ID of the job which has been migrated.

Figure A.19 shows the whole procedure of migrating a job from a resource to another:

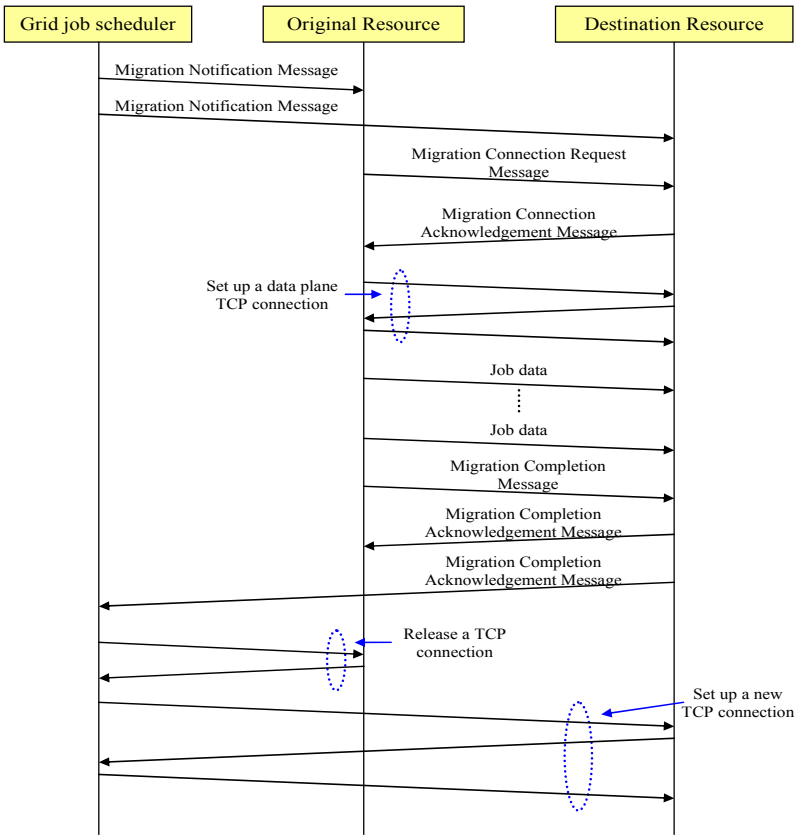


Figure A.19: Procedure of a Job Migration

XVI. Resource Unavailable Message

In order to tell the Grid job scheduler that it is going to be unavailable shortly, the resource can send a *Resource Unavailable Message* to notify the Grid job scheduler. Figure A.20 shows the format:

Message Type
Resource IP Address
Resource Name

Figure A.20: *Resource Unavailable Message*

Each field in the *Resource Unavailable Message* is described as follow:

Message Type: This field will be filled as "Resource Unavailable Message" when a resource sends this message to a Grid job scheduler.

Resource IP Address: This field records the resource's IP address.

Resource Name: This field records the name of the resource.

Appendix B

I. FCFSPP Algorithm with Synthetic Data

Validation of Simulator and the Influence of Different System States

To check the performance of the FCFSPP algorithm in different system states, a set of simulations with synthetic data is carried out. This set of simulations has two objectives: the first objective is to validate whether the simulator works correctly. The second objective is to check if the analysis in Section 5.2.2 is correct. In Section 5.2.2, 5 cases were presented. Therefore, five simulation scenarios with synthetic data were used to represent each case that the Grid job scheduler will have to face. Besides the setup shown in Table 7-1, these simulations have the experimental setup shown in Table B-1:

Name	Setting
Number of Resources	1
Job-scheduling Algorithm	FCFS, FCFSPP
Job Size	The value is 24 hours
t_1	24 hours in scenario 1 and 2; 12 hours in scenario 3 and 4; 10 hours in scenario 5
t_2	24 hours in scenario 1 and 4; 12 hours in scenario 2 and 5; 10 hours in scenario 3
Length of simulation	The value is 48 hours

Table B-1: Experimental Setup in Simulations

There is only one user, one Grid job scheduler and one resource in the simulation scenarios with synthetic data to clearly show how the simulator works. Each simulation is used to simulate one system case described in Section 5.2.2. As the job size is 24 hours, the Grid job scheduler is in fact making one allocation decision in each simulation day. In addition, the Grid job scheduler is assumed to know all resources' *CPU Availability*.

According to Figure B.1, both FCFS and the FCFSPP algorithm have identical results in terms of *Total Allocated Jobs* in the first simulation day. This is because it is the first simulation day and there is no historical data of resource availability. Therefore, the FCFSPP algorithm cannot make any prediction based on resource's historical data and it behaves the same a non-prediction base algorithm. As FCFSPP is based on FCFS algorithm, it behaves the same FCFS in such a case and both algorithms have identical results in terms of *Total Allocated Jobs*.

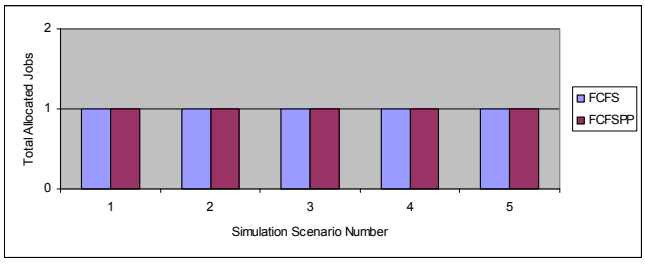


Figure B.1: *Total Allocated Jobs* in the First Simulation Day

With either FCFS or FCFSPP, the Grid job scheduler only allocates a new job to the resource when the resource is idle. As there is only one resource in all these simulation scenarios and each job lasts for 24 hours, the Grid job scheduler only allocates a new job to the only resource in the first day. Therefore, the result of *Total Allocated Jobs* is 1 in all simulation scenarios.

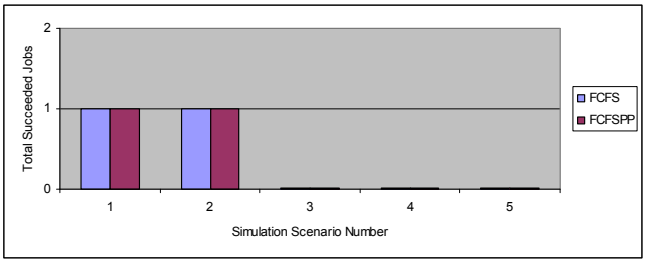


Figure B.2: *Total Succeeded Jobs* in the First Simulation Day

According to Figure B.2, both algorithms have identical results in terms of *Total Succeeded Jobs* in the first simulation day of all simulation scenarios. The same as *Total Allocated Jobs*, this is also explained by the lack of historical resource availability data.

As there is only one resource in all these simulation scenarios and each job lasts for 24 hours, the possible highest result of *Total Succeeded Jobs* is 1 in all simulation scenarios. In simulation scenario 1 and 2, as the resource's *Job Execution Availability* is true throughout the first simulation day, the result of *Total Succeeded Jobs* is 1 in these scenarios. In simulation scenario 3, 4 and 5, as the resource's *Job Execution Availability* is not always true in the first simulation day, the result of *Total Succeeded Jobs* is 0 in these scenarios.

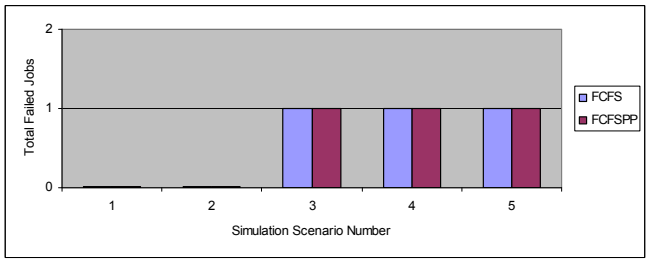


Figure B.3: *Total Failed Jobs* in the First Simulation Day

According to Figure B.3, both algorithms have identical results in terms of *Total Failed Jobs* in the first simulation day. Again, lack of historical resource availability data causes both FCFS and the FCFSPP to have the same results in terms of *Total Failed Jobs*.

The reason why job get lost is because the resource does not available throughout the first day of some scenario. Given the resource has one and only one *Unavailability Event* in the first day in all simulation scenarios, the possible highest results of *Total Failed Jobs* are 1 in all simulation scenarios. In simulation scenario 1 and 2, as the resource's *Job Execution Availability* is true throughout the first simulation day, the result of *Total Failed Jobs* is 0 in these scenarios. In simulation scenario 3, 4 and 5, as the resource's *Job Execution Availability* is not always true in the first simulation day, the result of *Total Succeeded Jobs* is 1 in these scenarios.

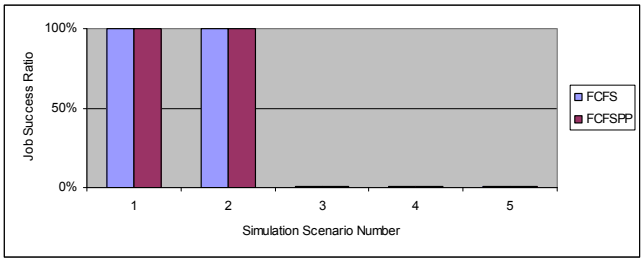


Figure B.4: *Job Success Percentage* in the First Simulation Day

After a job is processed, the result is either completed successfully or failed because of resource's *Unavailability Events*. Therefore, the *Job Success Percentage* is always between 0% and 100%. As both FCFS and FCFSPP have the identical results of *Total Succeeded Jobs* and *Total Failed Jobs* in the simulated scenarios, the result of *Job Success Percentage* in FCFS and the FCFSPP algorithm are identical in Figure B.4.

With reference to Figures B.1 through to B.4, FCFSPP behaves the same as FCFS if no historical data is available for making predictions. In addition, the results confirm that the simulator is working correctly with FCFS and FCFSPP in the cases where historical data is absent.

The results of the second day are more interesting because the FCFSPP algorithm will have historical data (the first simulation day's data) and it should behave differently to FCFS in some scenarios. As shown in Figure B.5, in terms of *Total Allocated Jobs*, the highest possible result is 1 in the second simulation day as each job lasts for 24 hours. In FCFS, it always allocates a new job to the resource when the resource becomes idle. Therefore, the result of *Total Allocated Jobs* is 1 in all simulation scenarios.

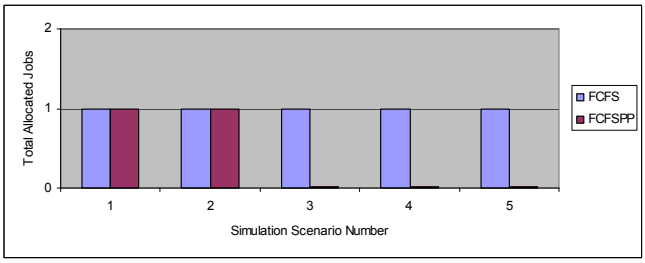


Figure B.5: *Total Allocated Jobs* in the Second Simulation Day

In the FCFSPP algorithm, the value of t_1 will influence the result of *Total Allocated Jobs*. FCFSPP uses the *Job Execution Availability* history data from day one to predict the *Job Execution Availability* in day two. Therefore, if t_1 is shorter than 24 hours, the resource's *Resource Availability Probability* tends to be low when the FCFSPP algorithm makes predictions. As a result, the FCFSPP *Resource Availability Probability* will be lower than 100% and FCFSPP will not allocate jobs to resources in such cases. On the other hand, if t_1 is not shorter 24 hours, the resource's *Resource Availability Probability* will be 100%. As a result, the FCFSPP algorithm will allocate a job to the resource when the resource is idle. For FCFSPP in simulation scenario 1 and 2, the result of *Total Allocated Jobs* is 1 as *Resource Availability*

Probability is 100%. In simulation scenario 3, 4 and 5, the result of *Total Allocated Jobs* is 0 as *Resource Availability Probability* is 100%.

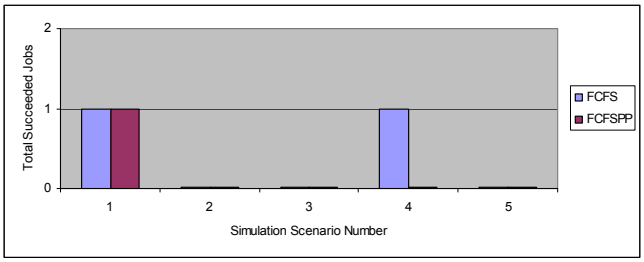


Figure B.6: *Total Succeeded Jobs* in the Second Simulation Day

In terms of *Total Succeeded Jobs*, the possible highest result is 1 in the second simulation day as each job lasts for 24 hours. According to Figure B.6, in FCFS, the result of *Total Succeeded Jobs* is 1 in simulation scenario 1 and 4 as the resource stays available for 24 hours in the second day of these simulation scenarios. The result of *Total Succeeded Jobs* is 0 in simulation scenario 2, 3 and 5 as the resource is not always available for 24 hours in the second day of these simulation scenarios.

In the FCFSPP algorithm, the result of *Total Succeeded Jobs* is 1 in simulation scenario 1 and the result of *Total Succeeded Jobs* is 0 in other simulation scenarios. The results also indicate FCFSPP will not perform better than FCFS in terms of *Total Succeeded Jobs* (this is considered in Section 5.2.2). If the system in case 4, the performance of FCFSPP is worse than FCFS in terms of *Total Succeeded Jobs*. If the Grid job scheduler faces other cases, the performance of the FCFSPP algorithm is the same as FCFS in terms of *Total Succeeded Jobs.*

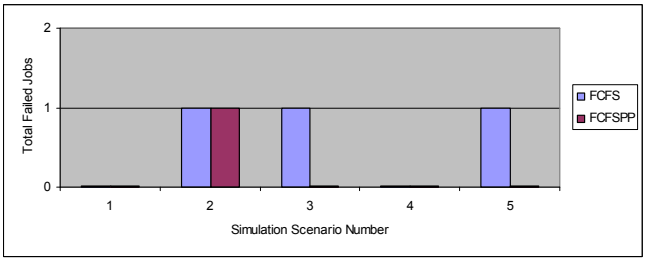


Figure B.7: *Total Failed Jobs* in the Second Simulation Day

In terms of *Total Failed Jobs*, the highest result is 1 as the resource has one *Unavailability Event* (becomes unavailable once) after 12 hours. According to Figure B.7, in FCFS, the result of *Total Failed Jobs* is 0 in simulation scenario 1 and 4 as the resource stays available for 24 hours in the second day of these simulation scenarios. The result of *Total Failed Jobs* is 1 in simulation scenario 2, 3 and 5 as the resource is not always available for 24 hours in the second day of each simulation scenario.

In FCFSPP, the result of *Total Failed Jobs* is 1 in simulation scenario 2 and the result of *Total Failed Jobs* is 0 in other simulation scenarios. The results also indicate FCFS algorithm will not perform better than the FCFSPP algorithm in terms of *Total Failed Jobs* in any case (this is analysed in Section 5.2.2). If the system in case 3 and 5, the performance of FCFS algorithm is

worse than FCFS in terms of *Total Failed Jobs*. If the Grid job scheduler faces other cases, the performance of FCFS algorithm is the same as the FCFSPP algorithm in terms of *Total Failed Jobs*.

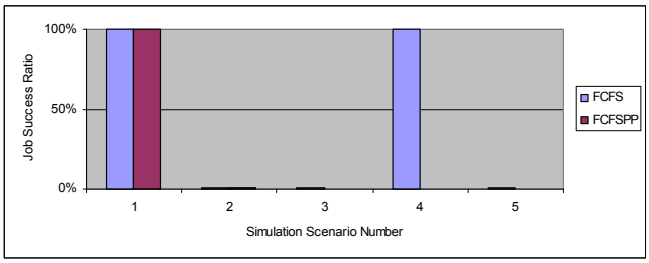


Figure B.8: *Job Success Percentage* in the Second Simulation Day

In terms of *Job Success Percentage*, the possible highest result is 100%. According to Figure B.8, in FCFS, the result of *Total Succeeded Jobs* is 1 and *Total Failed Jobs* 0 in simulation scenario 1 and 4 so that the result of *Job Success Percentage* in these simulation scenarios is 100%. The result is of *Total Succeeded Jobs* is 0 and *Total Failed Jobs* 1 in simulation scenario 2, 3 and 5 so that the result of *Job Success Percentage* in these simulation scenarios is 0%.

In FCFSPP, the result of *Total Succeeded Jobs* is 1 and *Total Failed Jobs* 0 in simulation scenario 1 so that the result of *Job Success Percentage* in this simulation scenario is 100%. The result is of *Total Succeeded Jobs* is 0 and *Total Failed Jobs* 1 in simulation scenario 2so that the result of *Job Success Percentage* in this simulation scenarios is 0%. The result is of *Total Succeeded Jobs* is 0 and *Total Failed Jobs* 0 in simulation scenario 2 so that the result of *Job Success Percentage* in this simulation scenarios is not available.

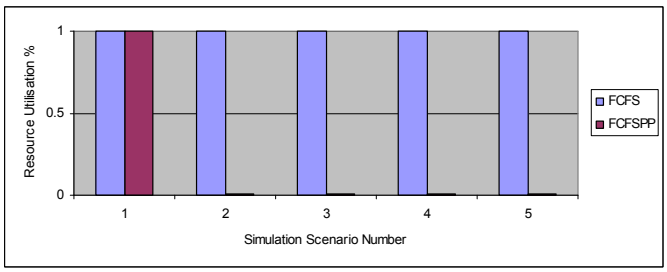


Figure B.9: *Resource Utilisation* in the Second Simulation Day

In terms of *Resource Utilisation*, the possible highest result is 100%, which means the resource's idle CPU cycles are fully utilised. In FCFS, it always sends a new job to the resource when it is become idle. Therefore, *Resource Utilisation* is always 100% in FCFS algorithm. This is shown in Figure B.9. In FCFSPP, as it does not allocate a new job to the resource if the resource's *Resource Availability Probability* is lower than 100%, so *Resource Utilisation* is lower than 100% in such a case. As the resource's *Resource Availability Probability* is 100% in simulation scenario 1, so the result of Resource Utilisation is 100%. As the resource's *Resource Availability Probability* is lower than 100% in other simulation scenarios, so the result of *Resource Utilisation* is lower than 100% (which is 0% here).

According to the simulation results above, the simulator proved to work correctly with FCFS

and FCFSPP in different system cases and the analysis (addressed in Section 5.2.2) regarding the performance of FCFS and FCFSPP under different cases is proved.

Influence of Δt between *Checking Day* and *Prediction Day*

In this subsection, a set of representative simulation scenarios with synthetic data are used to show the influence of a single Δt on a single resource. According to Figure 5.2 and the descriptions in Section 5.2.2, Δt is the time difference between the length of t_1 and t_2 and it is created by a pair of resource *Unavailability Events* (one occurs during the *Checking Period* while the other one occurs during the *Prediction Period*). To show the influence of parameter Δt, the only *Resource* was designed to have a simple *Job Execution Availability* pattern - available for the few hour of a day and then unavailable for the rest hours of that day.

Regarding the influence of Δt, the resource was designed to have similar pattern in terms of *CPU Availability* in both simulation days but the times to become unavailable are different. In the first day of these simulations, the resource is available for the first period of time t_1 and then stays unavailable for the rest time of the day 24 - t_1. In the second day of these simulations, the resource is available for the first period t_2 and then unavailable for the rest time of the day 24 – t_2. Figure 5.2 and 5.3 show two types of this availability pattern.

To facilitate understanding, in these simulations the value of Δt is defined as t_1- t_2. Therefore, the value of Δt will be positive when t_1 is larger than t_2 and it is negative when t_2 is larger than t_1. Besides the setup shown in Table 7-1, these simulations have the experimental setup shown in Table B-2:

Name	Setting
Number of Resources	The value is 1
Job-scheduling Algorithm	FCFS, FCFSPP
Job Size	The value is 6 hours
t_1	3 hours in scenario 1; 6 hours in scenario 2; 9 hours in scenario 3; 12 hours in scenario 4; 15 hours in scenario 5; 18 hours in scenario 6; 21 hours in scenario 7.
t_2	The value is 12 hours.
Length of simulation	The value is 48 hours

Table B-2: Experimental Setup in Simulations

In addition, the Grid job scheduler is assumed to know all resources' *CPU Availability* at all time. Note Δt are decided by the differences between t_1 and t_2. As t_2 is fixed, Δt will change along with the change of t_1 accordingly. If the value of t1 increases/decreases, the value of Δt will increase/decrease accordingly. Figure B.10 shows an example *Job Execution Availability* in these simulation scenarios.

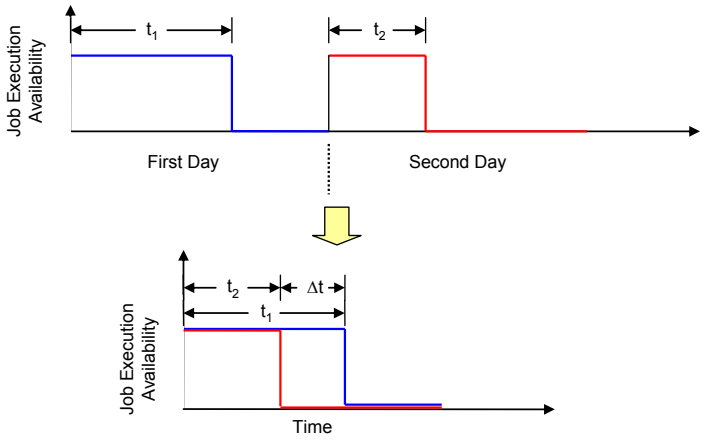


Figure B.10: An example of *Job Execution Availability* in simulation scenarios

According to the analysis in Section 5.2.2 and results earlier in this section, the FCFSPP algorithm performs the same as FCFS algorithm when the FCFSPP algorithm has no historical data of resource *Job Execution Availability* to make predictions. What are more important and interesting are the results in the second simulation day. This is because the FCFSPP algorithm will have historical data (the first simulation day's data) and it may behave different from FCFS any more in the second day. So the results can clearly show the differences between FCFS and the FCFSPP algorithm or the FCFSPP algorithm without historical data and the FCFSPP algorithm with historical data. Therefore, this set of simulations will focus on the simulation results from the second simulation day of each scenario. Here are some results in the second simulation day:

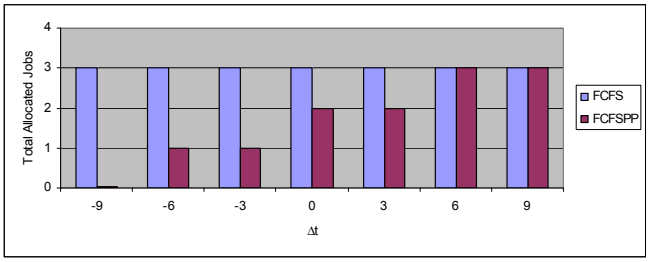


Figure B.11: *Total Allocated Jobs* in the second simulation day

Figure B.11 shows the result of *Total Allocated Jobs* of each algorithm in the second simulation day. In terms of *Total Allocated Jobs*, the possible highest result is 3 as each job lasts for 6 hours and the resource available for 12 hours in the second day of all simulation scenarios. The third job will be failed and disposed of, as the resource will become unavailable before it completes.

The FCFS algorithm allocates the same amount of jobs to the resources in all simulation scenarios as t2 (the resource's *Job Execution Availability* in the second day) does not change and the algorithm allocates a new job to the resource when it becomes idle. In FCFSPP, the value of Δt will influence the result of *Total Allocated Jobs*. FCFSPP uses the *Job Execution Availability* historical data in the first simulation day to predict the *Job Execution Availability* in the second

simulation day.

In the simulation, scenarios where the value of Δt is very low (it is because of large value of t_1), the results of *Total Allocated Jobs* is low in the FCFSPP algorithm is worse than the result of *Total Allocated Jobs* in FCFS algorithm. When the value of Δt increases, the FCFSPP algorithm allocates more jobs to the resource and finally has the same result of *Total Allocated Jobs* as FCFS algorithm.

This phenomenon can be explained by the analysis in Section 5.2.2. If the value of t_1 is low, the resource's *Resource Availability Probability* tends to be low when the FCFSPP algorithm makes predictions. As a result, the FCFSPP algorithm tends to NOT allocate jobs to the resource when the resource is idle. Therefore, when the value of t_1 becomes higher, the resource's *Resource Availability Probability* tends to be higher when the FCFSPP algorithm makes predictions. As a result, the FCFSPP algorithm tends to allocate jobs to the resource when the resource is idle.

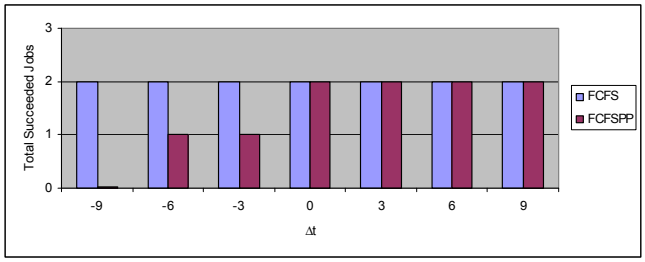


Figure B.12: *Total Succeeded Jobs* in the second simulation day

Figure B.12 shows the result of *Total Succeeded Jobs* of each algorithm in the second simulation day. In terms of *Total Succeeded Jobs*, the possible highest result is 2 as each job lasts for 6 hours and the resource available for 12 hours in the second day.

In FCFS algorithm, the result is always 2 as the algorithm utilises the resource in that 12 hours.

In the FCFSPP algorithm, the result increases along with the increase of Δt and finally achieves 2 when the value of Δt is not below 0. The number of Total Allocated Jobs directly influences this. As FCFS algorithm keeps the resource busy all the time, so it always has the highest result no matter what value of Δt is. However, in the FCFSPP algorithm, which is directly influenced by the result of *Total Allocated Jobs*, *Total Succeeded Jobs* is lower than the possible highest result in some scenarios. These results indicate the FCFSPP algorithm will be no better than FCFS algorithm in terms of *Total Succeeded Jobs* but will tend to the same performance as FCFS algorithm when the value of Δt is not too low.

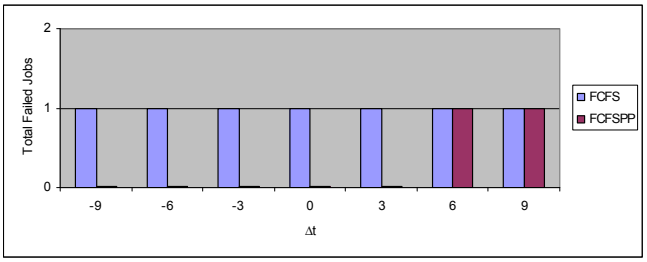


Figure B.13: *Total Failed Jobs* in the second simulation day

Figure B.13 shows the result of *Total Failed Jobs* of each algorithm in the second simulation day. In terms of *Total Failed Jobs*, the highest result is 1 as the resource has one *Unavailability Event* (becomes unavailable once) after 12 hours. In FCFS, as the resource always has a job to process, so a job will be failed in all simulated cases.

In the FCFSPP algorithm, resource's *Resource Availability Probability* is lower than 100% when the value of Δt is lower than the job size (6 hours), so *Total Failed Jobs* is 0 in such cases. When the value of Δt is not lower than the job size (6 hours), *Resource Availability Probability* is 100%, so the third job will allocated to the resource. However, unfortunately, the system will enter case 4 the prediction is incorrect and resource becomes unavailable before the job is completed.

These results also indicate the FCFSPP algorithm tends to perform worse in terms of *Total Failed Jobs* when Δt becomes higher. If Δt becomes higher, the probability that the length of *Checking Period* falls in somewhere between t_1 and t_2 becomes higher as well. Therefore, the prediction results will tend to be less accurate and the occurrence probability of case 4 will increase accordingly. According to the analysis in Section 5.2.2, jobs will fail to be processed with both FCFS and the FCFSPP algorithms in this case. This indicates that FCFSPP will not be worse than FCFS in terms of *Total Failed Jobs* but it will tend perform the same as FCFS in terms of *Total Failed Jobs* when the value of Δt is too high.

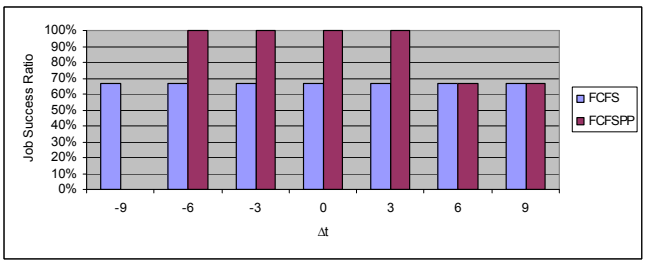


Figure B.14: *Job Success Percentage* in the Second Simulation Day

Figure B.14 shows the result of *Job Success Percentage* of each algorithm in the second simulation day. In terms of *Job Success Percentage*, the highest result is 100%. FCFS always has the same results 66.7%, as *Total Succeeded Jobs* and *Total Failed Jobs* are always the same result in all simulation scenarios.

In the FCFSPP algorithm, when Δt is -9, *Job Success Percentage* is not calculable as either *Total Succeeded Jobs* and *Total Failed Jobs* is 0. In the FCFSPP algorithm, when $-6 \leq \Delta t < 6$,

Job Success Percentage has the highest result 100%. When Δt is not lower than the job size (6 hours), the FCFSPP algorithm has the same result as FCFS algorithm. These results indicate FCFSPP will perform the same as the FCFS algorithm in terms of *Job Success Percentage* if the value of Δt is too low or too high; otherwise FCFSPP is able to perform better than FCFS.

Influence of Similarity of *Job Execution Availability* between *Checking Day* and *Prediction Day*

The influence of Δt between *Checking Day* and *Simulation Day* has been evaluated earlier in this section. When the value of Δt varies, the value of ρ between *Checking Day* and *Simulation Day* varies as well. Some important results from "Influence of Δt between *Checking Day* and *Prediction Day*" are summarised in Table B-3:

Δt (Hours)	**ρ**	***Total Succeeded Jobs* (FCFS)**	***Total Succeeded Jobs* (FCFSPP)**	***Total Failed Jobs* (FCFS)**	***Total Failed Jobs* (FCFSPP)**	***Job Success Percentage* (FCFS)**	***Job Success Percentage* (FCFSPP)**
-9	0.378	2	0	1	0	66.7%	N/A
-6	0.577	2	1	1	0	66.7%	100%
-3	0.775	2	1	1	0	66.7%	100%
0	1	2	2	1	0	66.7%	100%
3	0.775	2	2	1	0	66.7%	100%
6	0.577	2	2	1	1	66.7%	66.7%
9	0.378	2	2	1	1	66.7%	66.7%

Table B-3: Impact of ρ on *Job Success Percentage*

When Δt changes from -9 hours to 9 hours, the value of ρ increases from 0.378 (when Δt is -9 hours) to 1 when Δt is 0. After that, the value of ρ decreases again and reaches the lowest result 0.378 again when Δt is 9 hours. According to the results, the FCFSPP algorithm performs well in both terms of *Total Succeeded Jobs*, *Total Failed Jobs* and *Job Success Percentage* when the value of ρ is 1 (the highest value of ρ). When the value of ρ is 1, the FCFSPP algorithm has the same result as FCFS in terms of *Total Succeeded Jobs* and it has better results than FCFS in terms of *Total Failed Jobs* and *Job Success Percentage*.

If the value of ρ decreases, the FCFSPP algorithm's performance tends to become worse along with the decrease of ρ, either in terms of *Total Succeeded Jobs*, *Total Failed Jobs* or *Job Success Percentage*.

For the same value of ρ, the FCFSPP algorithm's performance might be different. For example, when Δt equals ±3 hours, the value of ρ is 0.775, but the results of *Total Succeeded Jobs* are different: it is 1 when Δt equals -3 hours and it is 2 when Δt equals 3 hours. Therefore, it indicates the value of ρ between *Checking Day* and *Prediction Day* can only used as an approximation to similarity of *Job Execution Availability* between *Checking Period* and *Prediction Period.*

II. FLP Algorithm with Synthetic Data

The simulation scenarios in this set of simulations are representative scenarios with synthetic

data and they are used to clearly show the influence of different parameter(s) and/or factor(s) while keeping the simulation scenarios as simple as possible. Besides the setup shown in Table 7-1, these simulations have the experimental setup shown in Table B-4:

Name	Setting
Number of Resources	1
Job-scheduling algorithm	FCFS, FCFSPP
Job Size	The value is 12 hours
t_1	The value is 20 hours in scenario 1 and 2; 23 hours in scenario 3 and 4
t_2	The value is 23 hours in scenario 1 and 3; 24 hours in scenario 2 and 4
Resource Availability Probability Threshold Adjustment Interval in FLP	The value is 1 hours.
Length of simulation	The value is 48 hours

Table B-4: Experimental Setup in Simulations

Resource Availability Probability Threshold Adjustment Interval equals 1 hour means the FLP algorithm uses a Fuzzy inference system to adjust its *Resource Availability Probability Threshold* once an hour.

The value of parameter λ equals 1 means the *Resource Availability Probability Threshold* will increase 1% if the value of *Disposed Jobs Dot* is equal to or larger than 1 and *Resource Availability Probability Threshold* will decrease 1% if the value of *Disposed Jobs Dot* is equal to or below 0. In other words, if the number of disposed job in the current adjustment interval is larger than the number of disposed job in the last interval, the value of *Resource Availability Probability Threshold* will increase 1%. On the other hand, if the number of disposed job in the current adjustment interval is not larger than the number of disposed job in the last interval, the value of *Resource Availability Probability Threshold* will decrease 1%.

This set of simulations is also focused on the results of the second simulation day in each simulation scenario. Here are some results in the second simulation day:

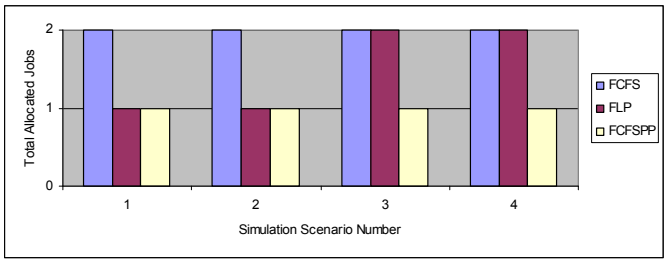


Figure B.15: *Total Allocated Jobs* in the Second Simulation Day

Figure B.15 shows the result of *Total Allocated Jobs* of each algorithm in the second simulation day. In terms of *Total Allocated Jobs*, the possible highest value is 2 as each job last for 12 hours and the resource is available at most 24 hours in the second simulation day.

For FCFS algorithm, it always has the possible highest results as it always keep the resource busy. Therefore, the result of *Total Allocated Jobs* is always 2 in these scenarios.

For the FLP algorithm, the result of *Total Allocated Jobs* tends to change between FCFS and the FCFSPP algorithm as the value of *Resource Availability Probability Threshold* is changed

once an hour.

When the FLP algorithm tries to allocate the second job to the resource after the first one is finished, the value of *Resource Availability Probability Threshold* has decreased from 100% to 88% (reduce 1% each hour). This means the FLP algorithm will allocate the job to the resource if the resource's *Resource Availability Probability* is not lower than 88% in the following 12 hours (the time which job is expected to run on). In simulation scenario 1 and 2, the resource's *Resource Availability Probability* is 75% (lower than 88%), so the FLP algorithm will not allocate the job to the resource in this scenario. In simulation scenario 3 and 4, the resource's *Resource Availability Probability* is 91.7% and 100% respectively, so the FLP algorithm will allocate the job to the resource in these two scenarios. Therefore, the result of *Total Allocated Jobs* is 1 in simulation scenario 1 and 2 and it is 2 in simulation scenario 3 and 4.

FCFSPP is the most *conservative* one among these tested algorithms. It only allocates the second job to the resource if the resource's *Resource Availability Probability* is 100%. However, the *Resource Availability Probability* is always below 100% when FCFSPP tries to make job allocation for the second job. Therefore, the result of *Total Allocated Jobs* is 1 all simulation scenarios.

The results also indicate that the difference between FLP and FCFS will occur in scenarios like 1 and 2 and the difference between FLP and the FCFSPP algorithm will occur in scenarios like 3 and 4.

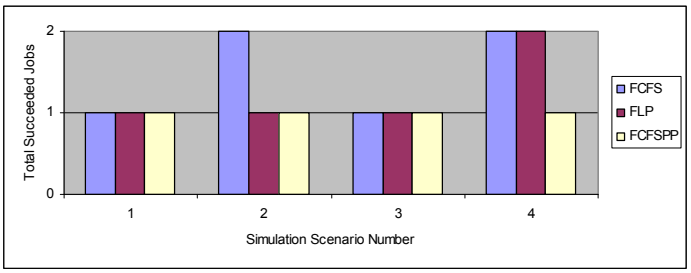


Figure B.16: *Total Succeeded Jobs* in the Second Simulation Day

Figure B.16 shows the result of *Total Succeeded Jobs* of each algorithm in the second simulation day. Directly influenced by the results of *Total Allocated Jobs*, the possible highest result of *Total Succeeded Jobs* is 2. In simulation scenario 1 and 3, as the resource lasts for 23 hours in the second day, the possible highest results of *Total Succeeded Jobs* is 1. In simulation scenario 2 and 4, as the resource lasts for 24 hours in the second day, the possible highest results of *Total Succeeded Jobs* is 2.

For FCFS algorithm, it always has the possible highest result in terms of *Total Succeeded Jobs* as it always keeps the resource busy.

For the FLP algorithm, as it is different from FCFS in simulation scenario 1 and 2, it does not have the possible highest result in simulation scenario 1 and 2. In simulation scenario 3 and 4, FLP will have the possible highest result in terms of *Total Succeeded Jobs* as it behaves the

same as FCFS in such scenarios.

For the FCFSPP algorithm, it cannot achieve the possible highest result in terms of *Total Succeeded Jobs* in all these scenarios due to its conservativeness.

The results also indicate the FLP algorithm's performance is between FCFS and the FCFSPP algorithm in terms of *Total Succeeded Jobs*. Specifically, the FLP algorithm's performance is the same as the FCFSPP algorithm in scenarios like 1 and 2 and the FLP algorithm's performance is the same as FCFS algorithm in scenarios like 3 and 4.

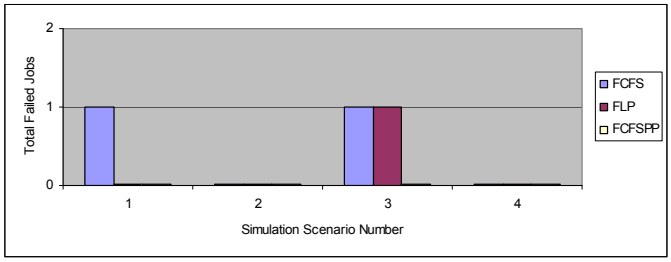


Figure B.17: *Total Failed Jobs* in the Second Simulation Day

Figure B.17 shows the result of *Total Failed Jobs* of each algorithm in the second simulation day. Directly influenced by the results of *Total Allocated Jobs*, the possible highest result of *Total Failed Jobs* is 1. In simulation scenario 1 and 3, as the resource lasts for 23 hours in the second day, the possible highest results of *Total Failed Jobs* is 1. In simulation scenario 2 and 4, as the resource lasts for 24 hours in the second day, the possible highest results of *Total Succeeded Jobs* is 0. For FCFS, it always has the highest result in terms of *Total Failed Jobs* as it always keeps the resource busy.

For the FLP algorithm, as it is different from FCFS in simulation scenario 1 and 2, it does not have the highest result in terms of *Total Succeeded Jobs* simulation in scenario 1 and 2. In simulation scenario 3 and 4, FLP will have the highest result as it behaves the same as FCFS in such scenarios. For the FCFSPP algorithm, it does not achieve the highest result in terms of *Total Failed Jobs* in all these scenarios due to its conservativeness.

The results also indicate the FLP algorithm's performance is between FCFS and the FCFSPP algorithm in terms of *Total Failed Jobs*. Specifically, the FLP algorithm's performance is the same as the FCFSPP algorithm in scenarios like 1 and 2 and the FLP algorithm's performance is the same as FCFS algorithm in scenarios like 3 and 4.

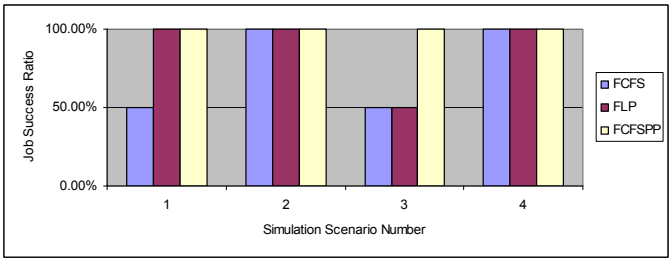


Figure B.18: *Job Success Percentage* in the Second Simulation Day

Figure B.18 shows the result of *Job Success Percentage* of each algorithm in the second

simulation day. In terms of *Job Success Percentage*, the possible highest result is 100%.

For the FCFS algorithm, it only achieves the possible highest result in terms of *Job Success Percentage* in simulation scenario 2 and 4.

For the FLP algorithm, it has the possible highest result in terms of *Job Success Percentage* in simulation scenario 1, 3 and 4.

For the FCFSPP algorithm, it always has the possible highest result in terms of *Job Success Percentage* in all these scenarios due to its conservativeness.

According to the results shown above, the results of the FLP algorithm in both terms of *Total Succeeded Jobs*, *Total Failed Jobs* and *Job Success Percentage* tend to falls in somewhere between the results of the FCFS and the FCFSPP algorithm. In scenarios (e.g. simulation scenario 1 and 4) where *Job Execution Availability* of some resource can provide good indication to all resources, the FLP algorithm can improve *speed* (such as maximising the result of *Total Succeeded Jobs*) while ensuring *reliability* (such as minimising the result of *Total Failed Jobs*). However, on the other hand, if *Job Execution Availability* of some resource cannot provide good indication to all resources, the FLP algorithm cannot improve *speed* while ensuring *reliability* (e.g. simulation scenario 2 and 3).

III. PSPP Algorithm with Synthetic Data

The first set of simulations uses synthetic data and representative scenarios to check if the analysis about the performance PSPP algorithm under different cases is correct. These simulations have the experimental setup shown in Table B-5:

Name	Setting
t_1	1 hour in scenario 1 and 4; 0.5 hours in scenario 2 and 3; 0.7 hours in scenario 5
t_2	1 hour in scenario 1 and 2; 0.7 hours in scenario 3 and 4; 0.5 hours in scenario 5
Length of *Checking Period*	The value is 1 hour
Length of *Prediction Period*	The value is 1 hour
Migration Prediction Interval	The value is 1 hour
Length of simulation	2 simulation days

Table B-5: Experimental Setup in Simulations

Here *Migration Prediction Interval* is a term defined in Section 4.3.2, which means how frequent to carry out the procedure of prediction for each resource. This set of simulations is focused on the results of the second simulation day.

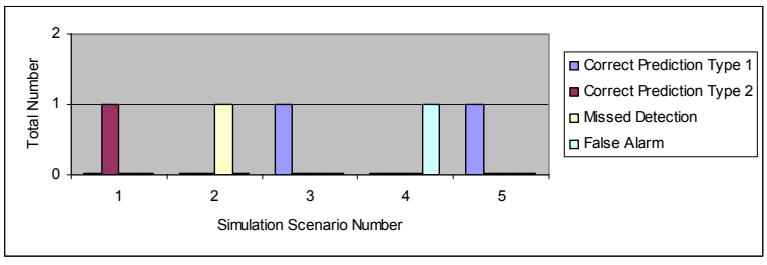


Figure B.19: Prediction Performance in Different Scenarios

Figure B.19 shows the prediction results in each simulation scenario. In each simulation scenario, the total number of prediction is only 1. PSPP algorithm will make prediction at the

beginning of the second simulation day.

In simulation scenario 1, t_1 equals 1 hour and t_2 equals 1 hour, the PSPP algorithm will face case 1 (described in Section 5.2.2) and the resource is predicted to stay in the state of *Available to Grid* in the *Prediction Period* (the length of *Prediction Period* is 1 hour) and the resource turns out to stay in the state of *Available to Grid* throughout the *Prediction Period*. Therefore, the result of *Correct Prediction Type 2* is 1 while other results are 0.

In simulation scenario 2, t_1 equals 0.5 hours and t_2 equals 1 hour, the PSPP algorithm will face case 2 (described in Section 5.2.2) and the resource is predicted to stay *available* in the *Prediction Period* (the length of *Prediction Period* is 1 hour) but the resource turns out to become *unavailable* during the *Prediction Period*. Therefore, the result of *Missed Detection* is 1 while other results are 0.

In simulation scenario 3, t_1 equals 0.5 hours and t_2 equals 0.7 hours, the PSPP algorithm will face case 3 (described in Section 5.2.2) and the resource is predicted to become *unavailable* in the *Prediction Period* (the length of *Prediction Period* is 1 hour) and the resource turns out to become *unavailable* during the *Prediction Period*. Therefore, the result of *Correct Prediction Type 1* is 1 while other results are 0.

In simulation scenario 4, t_1 equals 1 hour and t_2 equals 0.7 hours, the PSPP algorithm will face case 4 (described in Section 5.2.2) and the resource is predicted to become *unavailable* in the *Prediction Period* (the length of *Prediction Period* is 1 hour) but the resource turns out to stay *available* throughout the *Prediction Period*. Therefore, the result of *False Alarm* is 1 while other results are 0.

In simulation scenario 5, t_1 equals 0.7 hours and t_2 equals 0.5 hours, the PSPP algorithm will face case 5 (described in Section 5.2.2) and the resource is predicted to become *unavailable* in the *Prediction Period* (the length of *Prediction Period* is 1 hour) and the resource turns out to become *unavailable* during the *Prediction Period*. Therefore, the result of *Correct Prediction Type 1* is 1 while other results are 0.

IV. CBR Migration Algorithm with Synthetic Data

In order to check performance of CBR migration algorithm and whether the analysis about CBR in Section 5.5 is correct, a set of simulations have been designed and carried out.

This set of simulations use synthetic data and representative scenarios to check if the analysis about the performance CBR algorithm under different patterns of *CPU Availability Percentage* is correct. Figure B.20 shows the patterns of *CPU Availability Percentage* in these scenarios:

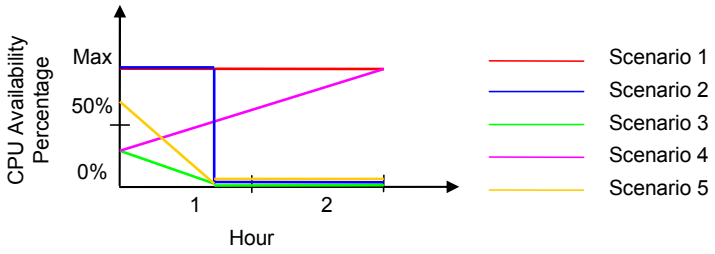


Figure B.20: Change of *CPU Availability Percentage* in Different Scenarios

In addition, the initial value of *CPU Migration Threshold* is 50%. The same as simulations with synthetic data for PSPP algorithm, this simulation also uses the results of *Correct Prediction*, *Missed Detection* and *False Alarm* to describe the performance of CBR migration algorithm. Here are some results after 2-hour simulation of each simulation:

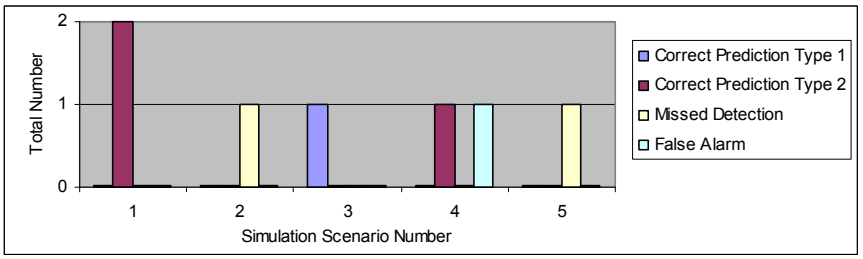


Figure B.21: *Correct Prediction*, *Missed Detection* and *False Alarms* in Different Scenarios

According to Figure B.21, in simulation scenarios 1 and 4, the CBR migration algorithm will make twice predictions for the resource, the first one at the beginning of the simulation and the second one at the beginning of the second hour. Therefore, the possible highest number of each result in these two simulation scenario is 2.

In simulation scenarios 2, 3 and 5, the CBR migration algorithm will make prediction for the resource once at the beginning of the simulation. At the beginning of second hour, the resource is not *available* CBR migration algorithm will not make prediction for the resource at that time. Therefore, the possible highest number of each result in these two simulation scenario is 1.

In simulation scenario 1, as the resource's *CPU Availability Percentage* is always at 100%, the CBR migration algorithm will predict the resource to stay in *available* twice and it turns out that these two predictions are correct. Therefore, the result of *Correct Prediction Type 2* is 2 while other results are 0. This shows CBR migration performs well in the situation where the resource is stable and reliable. However, if resources are always stable and reliable, it is not meaningful to carry out this type of prediction and job migration.

In simulation scenario 2, as the resource's *CPU Availability Percentage* is 100% at the beginning, the CBR migration algorithm will predict the resource to stay in *available* but it turns out that the resource becomes *unavailable* before next prediction. Therefore, the result of *Missed Detection* is 1 while other results are 0. This shows CBR migration algorithm does not work well in the situation where the resource becomes *unavailable* without any or little indication in terms of *CPU Availability Percentage*.

In simulation scenario 3, the resource's *CPU Availability Percentage* is 30% at the beginning. The CBR migration algorithm will predict the resource to become *unavailable* as the value of *CPU Availability Percentage* 30% is below the value of *CBR Migration Threshold* 50% and it turns out that the resource becomes *unavailable* before next prediction. Therefore, the result of *Correct Prediction Type 1* is 1 while other results are 0. This shows CBR migration algorithm work well in the situation where the resource becomes *unavailable* with useful indication in terms of *CPU Availability Percentage*. More importantly, CBR migration algorithm should observe this indication to make correct prediction.

In scenario 4, the resource's *CPU Availability Percentage* is 30% at the beginning. The CBR migration algorithm will predict the resource to become *unavailable* as the value of *CPU Availability Percentage* 30% is below the value of *CBR Migration Threshold* 50% but it turns out that the resource not *unavailable*. After 1 hour, the resource's *CPU Availability Percentage* is 65% and the CBR migration algorithm will predict the resource to stay *available* as the value of *CPU Availability Percentage* 30% is above the value of *CBR Migration Threshold* 50% and it turns out that the resource stay *available* before next prediction.

This shows CBR migration algorithm does not work well in the situation where the resource gives misleading indication in terms of *CPU Availability Percentage* and CBR migration algorithm observes this misleading indication.

In simulation scenario 5, the resource's *CPU Availability Percentage* is 60% at the beginning. The CBR migration algorithm will predict the resource to stay *available* as the value of *CPU Availability Percentage* 60% is above the value of *CBR Migration Threshold* 50% but it turns out that the resource becomes *unavailable* before next prediction. Therefore, the result of *Correct Prediction Type 1* is 1 while other results are 0.

This scenario shows the importance of a suitable *CPU Migration Threshold*. In this scenario, the resource gives useful indication in terms of *CPU Availability Percentage* but CBR migration algorithm cannot observe this as the *CPU Migration Threshold* is too low. CBR migration algorithm should observe this indication to make correct predictions.

In addition, this scenario also shows the importance of a suitable *Migration Prediction Interval*. In the simulation, the value of *Migration Prediction Interval* is set as 1 hour so that CBR migration algorithm cannot observe the change of *CPU Availability Percentage* in the whole hour. However, if the value of *Migration Prediction Interval* is smaller, then the value of *CPU Availability Percentage* will be 15% at that time and CBR migration algorithm will be able to observe this change then make a correct prediction and trigger job migration in due course.

According to the above results, the performance of CBR migration algorithm under different situations and the influences of *CPU Migration Threshold* and *Migration Prediction Interval* are shown and discussed. The results also reflect the analysis of CBR migration algorithm shown in Section 5.5 is correct.