AN ENHANCED DYNAMIC REPLICA CREATION AND EVICTION MECHANISM IN DATA GRID FEDERATION ENVIRONMENT

MUSA SULE ARGUNGU

DOCTOR OF PHILOSOPHY
UNIVERSITI UTARA MALAYSIA
2018

Awang Had Salleh
Graduate School
of Arts And Sciences

Universiti Utara Malaysia

PERAKUAN KERJA TESIS / DISERTASI
(Certification of thesis / dissertation)

Kami, yang bertandatangan, memperakukan bahawa
(We, the undersigned, certify that)

MUSA SULE ARGUNGU

calon untuk Ijazah **PhD**
(candidate for the degree of)

telah mengemukakan tesis / disertasi yang bertajuk:
(has presented his/her thesis / dissertation of the following title):

"AN ENHANCED DYNAMIC REPLICA CREATION AND EVICTION MECHANISM IN DATA GRID FEDERATION ENVIRONMENT"

seperti yang tercatat di muka surat tajuk dan kulit tesis / disertasi.
(as it appears on the title page and front cover of the thesis / dissertation).

Bahawa tesis/disertasi tersebut boleh diterima dari segi bentuk serta kandungan dan meliputi bidang ilmu dengan memuaskan, sebagaimana yang ditunjukkan oleh calon dalam ujian lisan yang diadakan pada : ***24 Mei 2018.***
That the said thesis/dissertation is acceptable in form and content and displays a satisfactory knowledge of the field of study as demonstrated by the candidate through an oral examination held on:
May 24, 2018.

Pengerusi Viva: *(Chairman for VIVA)*	**Assoc. Prof. Dr. Fauziah Baharom**	Tandatangan *(Signature)* [illegible]
Pemeriksa Luar: *(External Examiner)*	**Ir. Prof. Dr.R. Badlishah Ahmad**	Tandatangan *(Signature)* [illegible]
Pemeriksa Dalam: *(Internal Examiner)*	**Dr. Nur Haryani Zakaria**	Tandatangan *(Signature)* [illegible]
Nama Penyelia/Penyelia-penyelia: *(Name of Supervisor/Supervisors)*	**Dr. Ahmad Suki Che Mohamed Arif**	Tandatangan *(Signature)* [illegible]
Nama Penyelia/Penyelia-penyelia: *(Name of Supervisor/Supervisors)*	**Dr. Mohd Hasbullah Omar**	Tandatangan *(Signature)* [illegible]

Tarikh:
(Date) ***May 24, 2018***

Permission to Use

Abstrak

Sistem Grid Data Bersekutu merupakan satu infrastruktur yang menghubungkan beberapa sistem grid, yang memudahkan perkongsian besar yang data, serta sumber penyimpanan dan pengkomputeran. Mekanisme sedia ada bagi replikasi data hanya tertumpu kepada mencari nilai fail berdasarkan jumlah akses fail dalam menentukan fail mana yang akan direplikasi, dan meletakkan replika baru di lokasi yang menyediakan kos bacaan yang minimum. DRCEM melakukan secara berbeza dengan mempertimbangkan kebergantungan logik fail dalam mencari nilai fail, dan menempatkan replika baru di lokasi dengan beban kerja, jarak rangkaian dan kegagalan tapak yang minimum, dengan itu meminimumkan pemindahan data dan kos penyimpanan. Tesis ini memperkenalkan satu penambahbaikan ke atas mekanisme replikasi data yang dikenali sebagai Mekanisme Penciptaan dan Pengeluaran Replika (DRCEM) yang menggunapakai sumber data grid dengan memperuntukkan tapak replika yang sesuai dalam sistem yang bersekutu. Mekanisme yang dicadangkan menggunakan tiga skim: 1) Skim Penilaian dan Pengeluaran Replika Dinamik, 2) Skim Penempatan Replika Dinamik, dan 3) Skim Pengusiran Replika Dinamik. DRCEM telah dinilai menggunakan simulator rangkaian OptorSim berdasarkan empat metrik prestasi: 1) Tempoh Perlengkapan Pekerjaan, 2) Penggunaan Rangkaian yang Berkesan, 3) Penggunaan Elemen Penyimpanan, dan 4) Penggunaan Elemen Pengkomputeran. DRCEM mengatasi mekanisme ELALW dan DRCM sebanyak 30% dan 26% dari segi Tempoh Perlengkapan Pekerjaan. Di samping itu, DRCEM menggunakan ruang penyimpanan yang sedikit berbanding ELALW dan DRCM sebanyak 42% dan 40%. Walau bagaimanapun, DRCEM menunjukkan prestasi yang lebih rendah berbanding dengan mekanisme sedia ada dalam Penggunaan Elemen Pengkomputeran, disebabkan penambahan dalam pengiraan kebergantungan logik fail. Hasil kajian menunjukkan Tempoh Perlengkapan Pekerjaan yang lebih baik dengan penggunaan sumber yang lebih rendah daripada pendekatan sedia ada. Penyelidikan ini menghasilkan tiga skim replikasi yang terkandung dalam satu mekanisme yang dapat menyumbang kepada peningkatan prestasi persekitaran Grid Data Bersekutu, yang mampu membuat keputusan sama ada untuk mencipta atau mengusir lebih daripada satu fail dalam masa yang sama. Tambahan pula, kebergantungan logik fail telah diintegrasikan ke dalam skim penciptaan replika untuk menilai fail data dengan lebih tepat.

Kata kunci: Replikasi Data, Grid Data Bersekutu, Penciptaan Replika, Penempatan Replika, Penggantian Replika

Abstract

Data Grid Federation system is an infrastructure that connects several grid systems, which facilitates sharing of large amount of data, as well as storage and computing resources. The existing mechanisms on data replication focus on finding file values based on the number of files access in deciding which file to replicate, and place new replicas on locations that provide minimum read cost. DRCEM finds file values based on logical dependencies in deciding which file to replicate, and allocates new replicas on locations that provide minimum replica placement cost. This thesis presents an enhanced data replication strategy known as Dynamic Replica Creation and Eviction Mechanism (DRCEM) that utilizes the usage of data grid resources, by allocating appropriate replica sites around the federation. The proposed mechanism uses three schemes: 1) Dynamic Replica Evaluation and Creation Scheme, 2) Replica Placement Scheme, and 3) Dynamic Replica Eviction Scheme. DRCEM was evaluated using OptorSim network simulator based on four performance metrics: 1) Jobs Completion Times, 2) Effective Network Usage, 3) Storage Element Usage, and 4) Computing Element Usage. DRCEM outperforms ELALW and DRCM mechanisms by 30% and 26%, in terms of Jobs Completion Times. In addition, DRCEM consumes less storage compared to ELALW and DRCM by 42% and 40%. However, DRCEM shows lower performance compared to existing mechanisms regarding Computing Element Usage, due to additional computations of files logical dependencies. Results revealed better jobs completion times with lower resource consumption than existing approaches. This research produces three replication schemes embodied in one mechanism that enhances the performance of Data Grid Federation environment. This has contributed to the enhancement of the existing mechanism, which is capable of deciding to either create or evict more than one file during a particular time. Furthermore, files logical dependencies were integrated into the replica creation scheme to evaluate data files more accurately.

Keywords: Data Replication, Data Grid Federation, Replica Creation, Replica Placement, Replica Eviction

Acknowledgements

First, I would like to express my utmost gratitude to Almighty Allah for his creation and making me submissive to Him. I would like to gratefully acknowledge the enthusiastic supervision of my thesis supervisors, for the continuous support of my PhD study and related research, for their patience, motivation and guidance. I could not have imagined having better advisers and mentors for my PhD.

Parts of this work would not have been possible without the active contributions of my colleagues at InterNetWorks Research Group. I thank them for the long discussions we had, the papers they helped write, and their valuable feedback and constructive criticisms.

My profound gratitude goes to the heads of InterNetworks Research Laboratory (IRL) for the great support and for the massive efforts in providing the facilities to conduct this study. I wish to also thank the supporting team of OptorSim simulator for their guidance, and to the Open Grid Forum (OGF), particularly the Working Group on Distributed Computing Infrastructure Federation (WGDCIF), for their persistent updates in this study.

I am also grateful to School of Computing, UUM College of Arts and Sciences, for providing research facilities and related resources that facilitated my study. It was an exciting place to work and study.

Finally, I am immeasurably grateful to my family for their kindness, patience and support. They have always been there and encouraged me; without their selfless love, I know I could not have completed my doctoral studies.

Table of Contents

Permission to Use i

Abstrak ii

Abstract iii

Acknowledgements iv

Table of Contents v

List of Figures x

List of Tables xiii

List of Appendices xiv

List of Abbreviations xv

CHAPTER ONE INTRODUCTION 1

1.1 Research Background 1

1.2 Research Motivation 6

1.3 Problem Statement 9

1.4 Research Questions 12

1.5 Research Objectives 13

1.6 Significance of the Research 14

1.7 Research Contributions 16

1.8 Scope of the Research 17

1.9 Thesis Organisation 18

CHAPTER TWO LITERATURE REVIEW 20

2.1 Introduction 20

2.2 Data Grids 22

2.3 Data Grid Projects 24

2.4 Data Grid Models 26

2.5 Overview of Data Grid Federation Systems 28

2.6 Data Grid Federation Scenarios 29

2.6.1 The Peer-to-Peer or Napster Federation System 33

2.6.2 The CMS Federation System 34

2.6.3 The BIRN Federation System 34

2.6.4 NASA Backup Federation System 35

2.6.5 The BaBar Federation System 36

2.6.6 The Earth System Grid Federation 36
2.6.7 Data Grid Federation Middleware and Frameworks 37
2.7 Data Replication in Data Grid Federation Systems.......................................39
2.7.1 Replica Management System and Replica Eviction..........................39
2.7.2 Stages of Dynamic Replica Creation..42
2.7.3 What Triggers Data Replication in a DGF Environment 43
2.7.4 Replica Optimisation Process 43
2.7.5 Availability of Data and Replica Sites...48
2.7.6 The Concept of Replica Dependency 49
2.7.7 Concept of Grid Jobs and Job Schedules...50
2.8 Related Work on Data Replication...52
2.8.1 Popularity Based Data Replication Mechanisms 56
2.8.2 Availability Based Data Replication Mechanisms............................60
2.8.3 Replica Placement Mechanisms..65
2.8.4 Replica Management Stage..67
2.8.5 Replica Selection..67
2.8.6 Replica Maintenance 68
2.9 Data Grid Federation Simulation Tools...72
2.9.1 Comparisons of Various Grid Simulation Tools..............................73
2.9.2 The OptorSim Simulator...75
2.10 The Existing Data Replication Mechanisms...78
2.10.1 ELALW Data Replication Mechanism..82
2.10.2 DRCM Data Replication Mechanism...83
2.11 Chapter Summary...84
CHAPTER THREE RESEARCH METHODOLOGY 87
3.1 Introduction...87
3.2 Research Design..88
3.3 Research Clarification (RC)...90
3.4 Descriptive Study-I (DS-I)...91
3.4.1 An Overview of the Proposed DRCEM Mechanism 92
3.4.1.1 Conceptual Model of the DRCEM Mechanism..................93
3.4.1.2 Framework for the DRCEM Mechanism 96
3.5 Prescriptive Study (PS)...99

3.5.1 The Procedure for Mechanism Validation 100
3.5.2 Comparison with Manual Computations of the Validation Data 103
3.6 Descriptive Study-II (DS-II) 105
3.6.1 Procedure for Performance Evaluation 106
3.6.1.1 Analytical Modeling 106
3.6.1.2 Evaluation Using Testbed 107
3.6.1.3 Evaluation Using Simulation 107
3.6.2 The Simulation Environment 108
3.6.2.1 The OptorSim Simulator and Simulation Parameters 109
3.6.2.2 The Simulation Parameters used for Data Collection and Analysis 111
3.6.2.3 The Simulation Topology 115
3.6.2.4 Replication and Scheduling Mechanisms in OptorSim Simulator 117
3.6.3 Performance Evaluation Metrics 119
3.6.3.1 Job Completion Time 119
3.6.3.2 Effective Network Usage 120
3.6.3.3 Storage Element Usage 120
3.6.3.4 Computing Element Usage (CE Usage) 121
3.7 Chapter Summary 122
CHAPTER FOUR THE SCHEMES IN DRCEM MECHANISM 124
4.1 Introduction 124
4.2 Design Objectives for the Proposed (DRCEM) Mechanism 124
4.2.1 Access Latency (Tobs Times) 125
4.2.2 Optimising Storage and Computing Element Usage 125
4.2.3 Minimising Bandwidth Consumption 126
4.3 Detailed Schemes Design for the Proposed (DRCEM) Mechanism 127
4.3.1 The Dynamic Replica Evaluation and Creation Scheme 128
4.3.1.1 Determining the Popularity of Data Using Access Frequencies 130
4.3.1.2 Illustrations on How to Determine Access Frequencies 134
4.3.1.3 Framework for Determining the Required Number of Files to Replicate 137

4.3.1.4 Scenarios Used to Determine the Required Number of Files to Replicate 138
4.3.1.5 Illustrations on How the Required Numbers of Files are Computed..139
4.3.2 The Dynamic Replica Placement Scheme 142
4.3.2.1 Determining the Replica Placement Cost (RPC) 144
4.3.2.2 Calculating the File Transfer Time (FTT)........................146
4.3.2.3 Illustrations on RPC computations for Replica Placement.146
4.3.2.4 Determining Workloads for all Regional Sites.................150
4.3.2.5 Computing Site Distance Using Bandwidth Information..151
4.3.2.6 Finding Shortest Paths Using an Iterative Deepening Depth-First Path-Finding Algorithm. 158
4.3.2.7 Mathematical Framework for Replica Site Availability ...159
4.3.2.8 Framework for Determining Files Weights......................164
4.3.2.9 Framework for Determining Files Logical Dependencies 165
4.3.2.10 Mathematical Framework for Determining File Value.....171
4.3.3 The Dynamic Replica Eviction Scheme 173
4.3.4 The Complete Algorithm for the Proposed DRCEM Mechanism ...175
4.3.4.1 Explanations on the Unique Features of for Proposed DRCEM Mechanism 178
4.4 DRCEM Data Replication Mechanism Implementation 181
4.4.1 Diagramatic Representation of DRCEM Integration with OptorSim Simulator 182
4.4.2 Diagramatic Representation of DRCEM Simulation Processes in OptorSim..185
4.4.3 DRCEM Programming and Codes Integration in OptorSim Simulator187
4.5 Chapter Summary 195
CHAPTER FIVE THE DRCEM PERFORMANCE EVALUATION ALONG WITH COMPARISON ON EXISTING MECHANISMS 196
5.1 Introduction 196
5.2 Comparison of DRCEM with ELALW and DRCM Mechanisms 196
5.2.1 Analysis on Number of Jobs and Effects on Replications..................197
5.2.3 Analysis on the Effect of Number of Jobs on Jobs Times..................200

5.2.4 Analysis on the Effect of Number of Jobs on the Effective Network Usage 204
5.2.5 Analysis on the Effect of Number of Jobs on Storage Element Usage 207
5.2.6 Analysis on the Effect of Number of Jobs on Computing Element Usage 208
5.3 Analysis on the Effects of File Dependencies on the Performance Metrics .. 210
5.3.1 Effects of File Logical Dependencies on Jobs Completion Times 213
5.3.2 Effects of File Logical Dependencies on Effective Network Usage ... 215
5.3.3 Effects of File Logical Dependencies on Storage Element Usage 216
5.3.4 Effects of File Dependencies on Computing Element Usage 216
5.4 Effect of Site Availability on Replications 217
5.5 Analysis on Indirect Logical Dependability 218
5.6 Analysis on Access Frequencies and File Weights 220
5.7 Analysis on Access Frequencies and File Values 221
5.8 Chapter Summary 223
CHAPTER SIX CONCLUSION AND FUTURE WORK 225
6.1 Introduction 225
6.2 Revisiting the Research Objectives 228
6.3 Revisiting the Research Contributions 232
6.4 Future Research Works 235
REFERENCES 237

List of Figures

Figure 1.1 Grid system types 3
Figure 1.2 Grid federation data resource sharing system........ 4
Figure 1.3 Scope of the research 17
Figure 2.1 An abstract view of data grid system 22
Figure 2.2 DG models and their characteristic elements 26
Figure 2.3 The Peer-to-Peer federation model 33
Figure 2.4 A Federation middleware for integrating heterogeneous data grids... 37
Figure 2.5 Replica management architecture 40
Figure 2.6 A Concept of grid’s jobs scheduling........ 51
Figure 2.7 A DGF Architecture based on cluster federation topology 61
Figure 2.8 Peer-to-Peer architecture of OptorSim replica optimiser 76
Figure 2.9 Algorithm for ElALW mechanism 82
Figure 2.10 Algorithm for DRCM mechanism 83
Figure 3.1 Research Design 89
Figure 3.2 Main Steps in the Research Clarification Stage 90
Figure 3.3 Conceptual Model of the proposed DRCEM Mechanism 93
Figure 3.4 Relationship between DRCEM, RLS, ISP and other related entities.. 96
Figure 3.5 Framework for the proposed DRCEM Mechanism 97
Figure 3.6 Main Steps in the Prescriptive Study Stage 99
Figure 3.7 Validation process flowchart 101
Figure 3.8 Variation of mean job completion time over 20 simulations runs 105
Figure 3.9 The DRCEM Test bed showing sites connectivity 116
Figure 4.1 Flowchart for dynamic replica evaluation and creation scheme (DRECS) 129
Figure 4.2 Flowchart for dynamic replica placement scheme (DRPS) 144
Figure 4.3 Dijkstra's algorithm for finding distances between replica sites....... 152
Figure 4.4 Graph abstraction for computing distance between sites 156
Figure 4.5 Graph abstraction showing dependencies amongst 16 data files . 169
Figure 4.6 Flowchart for dynamic replica eviction scheme (DRES) 174
Figure 4.7 Algorithm for DRCEM mechanism 175
Figure 4.8 OptorSim UML showing relationships amongst the six packages ... 182
Figure 4.9 OptorSim UML class diagram showing DRCEM implementation .. 184

Figure 4.10 DRCEM integration into OptorSim package 186
Figure 4.11 DRCEMOptimiser implementation into ReplicatingOptimser class 188
Figure 4.12 DRCEMStorageElement implementation into AccessHistoryStorageElement 189
Figure 4.13 DRCEM entire code in the OptorSim simulator 192
Figure 4.14 Implementation of DRCEMOptimiser into the StorageElementFactory class 193
Figure 4.15 Implementation of DRCEMOptimiser into OptimiserFactory class. 194
Figure 5.1 Effect of jobs numbers on replications for 10 GB file size 198
Figure 5.2 Effect of jobs numbers on replications for 5 GB file sizes 198
Figure 5.3 Effect of jobs numbers on replications for 2.5 GB file sizes 198
Figure 5.4 Jobs times for different number of submitted jobs of 10 GB files size 202
Figure 5.5 Jobs times for different number of submitted jobs of 5 GB files size203
Figure 5.6 Jobs times for different number of submitted jobs of 2.5 GB files size 203
Figure 5.7 ENU for different number of submitted jobs of 10 GB files size 205
Figure 5.8 ENU for different number of submitted Jobs of 5 GB files size....... 205
Figure 5.9 ENU for different number of submitted jobs of 2.5 GB files size 205
Figure 5.10 SE Usage of DRCM and existing mechanisms for 10 GB files size. 207
Figure 5.11 SE Usage of DRCM and existing mechanisms for 5 GB files size... 207
Figure 5.12 SE Usage of DRCM and existing mechanisms for 2.5 GB files size 208
Figure 5.13 DRCEM results and existing mechanisms on CE usage for 10 GB files size 208
Figure 5.14 DRCEM results and existing mechanisms on CE usage for 5 GB files size 209
Figure 5.15 DRCEM results and existing mechanisms on CE usage for 2.5 GB files size 209
Figure 5.16 An instance of job configuration file showing logical dependencies 211
Figure 5.17 DRCEM jobs completion times with files dependencies 214
Figure 5.18 DRCEM ENU and existing mechanisms with files dependencies.... 215
Figure 5.19 DRCEM SEU and existing mechanisms with files dependencies 216
Figure 5.20 DRCEM CEU and existing mechanisms with files dependencies.... 217

Figure 5.21 DRCEM and existing mechanisms on number of replications 217
Figure 5.22 DRCEM ILD values computations for 16 case data files................. 219
Figure 5.23 DRCEM, DRCM file dependencies computations for 16 case data files 219
Figure 5.24 DRCEM FW values computations for 16 case data files. 220
Figure 5.25 DRCEM, DRCM FW values computations for 16 case data files. ... 220
Figure 5.26 DRCEM popular file evaluation computations for 16 case data files. 221
Figure 5.27 DRCEM and DRCM on FV computations for 16 case data files. 222

List of Tables

Table 2.1 Summary of strengths and weaknesses of related literature in data replication 69
Table 2.2 Comparisons of various grid simulation tools 74
Table 2.3 Features of DRCM and ELALW on replica management. 78
Table 2.4 Additional features of DRCM and ELALW based on files types and required number of replicas. 79
Table 3.1 Fixed value and internal validity tests data 103
Table 3.2 Statistic data for validation result 104
Table 3.3 Configuration parameters used in the simulations 110
Table 4.1 Computation of popular file using access frequencies 136
Table 4.2 Example of 16 files together with their values and existing number of copies 140
Table 4.3 Example of how DRCEM computes PNoR values of data files 141
Table 4.4 Example of how DRCEM computes RPC values of data files 146
Table 4.5 Sample site status record for availability workload from TeraGrid 162
Table 4.6 Sample availability computation for six sites 163
Table 4.7 Example of file transfer status 164
Table 4.8 Indirect logical dependencies computations for 16 sites 167
Table 4.9 Computed file weight values for 16 data files 170
Table 4.10 Calculating file values for 16 data files 172
Table 5.1 Summary of results from the simulations of 50 to 5000 jobs 197
Table 5.2 DRCEM, DRCM and ELALW on 5000 jobs with 10.0 GB files size 200
Table 5.3 DRCEM, DRCM and ELALW on 5000 jobs with 5.0 GB files size 200
Table 5.4 DRCEM, DRCM and ELALW on 5000 jobs with 2.5 GB files size 201
Table 5.5 Efficiency of DRCEM against ELALW and DRCM mechanisms 202
Table 5.6 Sample types of jobs and number of files they depended on for running the jobs 211
Table 5.7 Performance of DRCEM and the existing mechanisms with file dependencies 212
Table 5.8 DRCEM against ELALW and DRCM on file dependencies 213

List of Appendices

Appendix A Files Access History Sample Workload Data.................................... 248
Appendix B Site Connectivity Data from Gnutella file-sharing Network.............. 249
Appendix C Site Availability Sample Workload Data... 253

List of Abbreviations

APACGrid	Australian Partnership for Advanced Computing Grids
APIs	Application Programming Interfaces
BADC	British Atmospheric Data Centre
BeInGrid	Business Experiments in Grid
BIRN	Bioinformatics Research Network
CE	Computing Element
CERN	European Organization for Nuclear Research
CEU	Computing Element Usage
CG	Compute Grid
CPU	Central Processing Unit
DDG	Data Duplication problem on the Grid
DG	Data Grids
DGF	Data Grid Federation
DIF	Directory Interchange Format
DPSO	Discrete Particle Swarm Optimisation
DRA	Data Replication Agent
DRCEM	Dynamic Replica Creation and Eviction
DRCM	Dynamic Replica Creation Mechanism
DRECS	Dynamic Replica Evaluation and Creation Scheme
DRES	Dynamic Replica Eviction Scheme
DRPS	Dynamic Replica Placement Scheme
EDG	European Data Grid
EGEE	Enabling Grids for E-science
ELALW	Enhanced Largest Access Largest Weight
ENU	Effective Network Usage
ERDA	Efficient Dynamic Replication Algorithm
FV	File Value
FW	File Weight
GCMD	NASA Global Change Master Directory
GF	Grid Federation
GFA	Grid-Federation Agent

GMS	Grid Management Software
GridPP	The UK Grid for Particle Physics
GriPhyN	Grid Physics Network
GRIS	Grid Resource Information Services
HCS	Hierarchical Cluster Scheduling
HDD	Hard Disk Drive
HPC	High Power Computing
HRS	Hierarchical Replication Strategy
HTC	High Throughput Computing
IDE	Integrated Development Environment
ILD	Indirect Logical Dependency
LALW	Largest Access Largest Weight
LAN	Local Area Network
LAS	Live Access Server
LDGF	Large Data Grid Federation
LFU	Least Frequently Used
LHC	Large Hadron Collider
LRMS	Local Resource Management Systems
LRU	Least Recently Used
NASA	National Aeronautics and Space Administration
P2P	Peer-to-Peer
PDB	Protein Data Bank
PDCS	Parallel and Distributed Computing Systems
PH-PSO	Parallel Particle Swarm Optimisation
QoS	Quality of Service
ROP	Replica Optimisation Process
SE	Storage Element
SEU	Storage Element Usage
SGFRS	Sub-Grid-Federation Replication Strategy
SGFS	Sub-Grid-Federation Scheduling
SLAs	Service Level Agreements
TeraGrid	Now Extreme Science and Engineering Digital Environment (XSEDE)

TLC	Total Logical Connections
UML	Universal Mark-up Language
UNICORE	Uniform Interface to Computing Resources
WAN	Wide Area Network

CHAPTER ONE

INTRODUCTION

1.1 Research Background

Data Grid Federation (DGF), as a distributed computing infrastructure, is an interesting area of continued research, which emanates from the popular Grid Computing (GC) paradigm that manages diverse resources from different administrative domains [1]. In this chapter, a brief background on the types of Grid Computing and Data Grid Federation is given as well the types of servies offered by these types of Distributed Computing infrastructure. The chapter, in its subsequent secions, explains the problems that motivated this research together with statements of research questions that need to be addressed. Accordingly, the research objectives are articulated to help address the research questions. In its last three sections, the chapter highlights on the significance of the research, research conributions as well as scope of the research, respectively. Lastly, the chapter maps out how the whole thesis is structured, by highlighting on the contributions of each chapter (One to Six) of this thesis.

The fundamental goal of this research is to develop a dynamic replica creation and eviction mechanism (DRCEM) for improving the performance of DGF systems, interms of jobs completion times, storage element usage (SEU), effective network usage (ENU) and computing element usage (CEU). As mentioned earlier, DGF belongs to Grid Computing paradigm [2] and it is formed by joining more than one Data Grids system [3] or computing clusters together [1]. Furthermore, DGF is a large-scale resource management system consisting of data and computing resources, linked via Peer-to-Peer connections [1]. Other innovative technologies similar to Grid Computing include Pear-to-Pear Computing (P2P) [4], Mobile Computing [5], Cloud Computing [6] and Ubiquitous Computing [7].

Grid Computing paradigm can be categorized into two major parts, namely (i) Computational Grid (CG) and (ii) Data Grid (DG). Computational Grid is an infrastructure that requires massive computational tasks, with little emphasis on extensive data analysis. Data Grid, on the other hand, is deployed for computational tasks that deal with the study, as well as the analysis of a considerably large dataset [1], [8], [9].

Grid Computing makes it possible to share diverse sets of resources such as supercomputers, minicomputers, desktops, laptops, computational clusters, storage facilities, data resources, sensor devices, virtual scientific instruments and various types of applications. Among the services offered on DG platforms include data federations and replication services, that manage a huge number of replica files, hosted on several storage facilities. By analogy, DGF systems incorporate different and heterogeneous computing platforms, thereby enabling fast, up-to-date, reliable and secure access to distributed file storage to users and federating service providers [10], [11].

Some of the applications of Grid Computing are to provide enormous computational capacity to resolve problems that pose challenges to researchers in the fields of sciences as well as engineering, which previously would seem very tasking. These problems exist in the following domains: (i)- High Energy Physics (HEP) [7], (ii)- Earthquake Engineering and Simulations (EES) [7], (iii)- Astrophysics, such as

Climate and Weather Modeling [7], (iv)- Aircraft Engine Diagnostics [7], (v)- Bioinformatics [12], (vi)- Drug Discovery [12], (vii)- Financial Modeling [7], (viii)- Virtual Observatory and Digital Image Analysis [7].

Grid systems come in different flavors, depending on the motive for their development namely; grids for compute-intensive tasks, grids for data-intensive services, grids for application services, grids for utility services, interaction grids and knowledge-based services grids [7]. Thus, when multiple DG platforms are joined they form a federation, which is the focus of this research. The following Figure 1.1 outlines some of the notable grid systems types available globally that could join to form a Grid Federation.

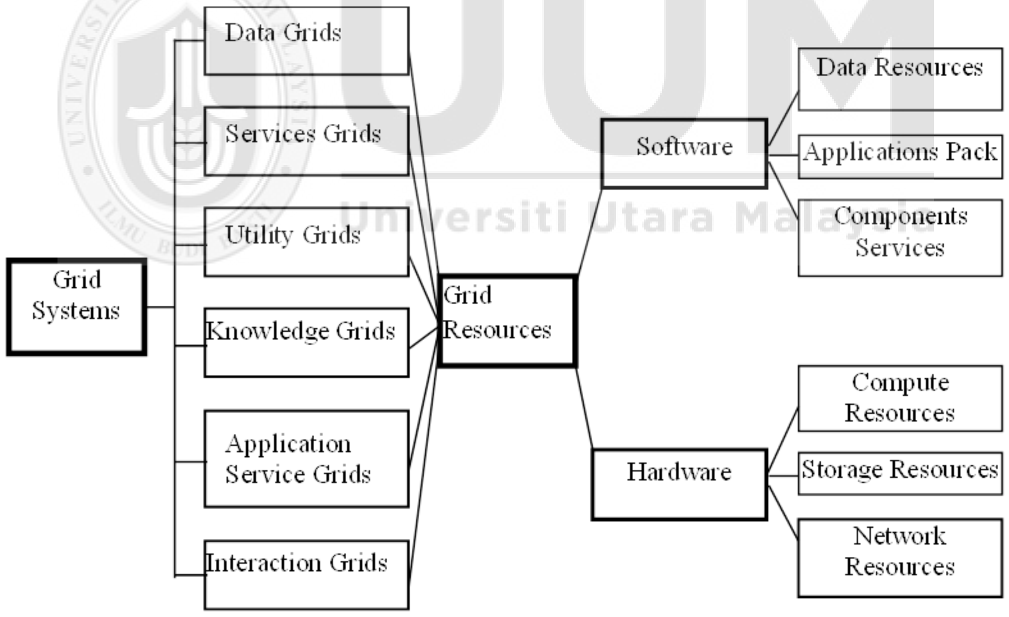


Figure 1.1. Grid system types [7]

From Figure 1.1, grid systems are classified into six sub classes according to the research in [7] namely (i) - Data Grids, (ii) - Services Grids, (iii) – Utility Grids, (iv) – Knowledge Grids, (v) – Application Services Grids and (vi) – Interaction Grids.

All the grids types may have common software and hardware resources. Typical software resources include data resources, application resources and components services. The hardware resources comprise of computing resources, storage resources and network resources. In any type of grid system, the software and hardware support each other, and the components interact to provide integrated grid services. Thus, according to research by researchers in [13], DGF is formed by means of a federation mechanism or software that binds the individual DG systems belonging to different administrative domains, with various computing infrastructure [1]. Figure 1.2 shows an abstract model of DGF data resource sharing system consisting of several institutions globally.

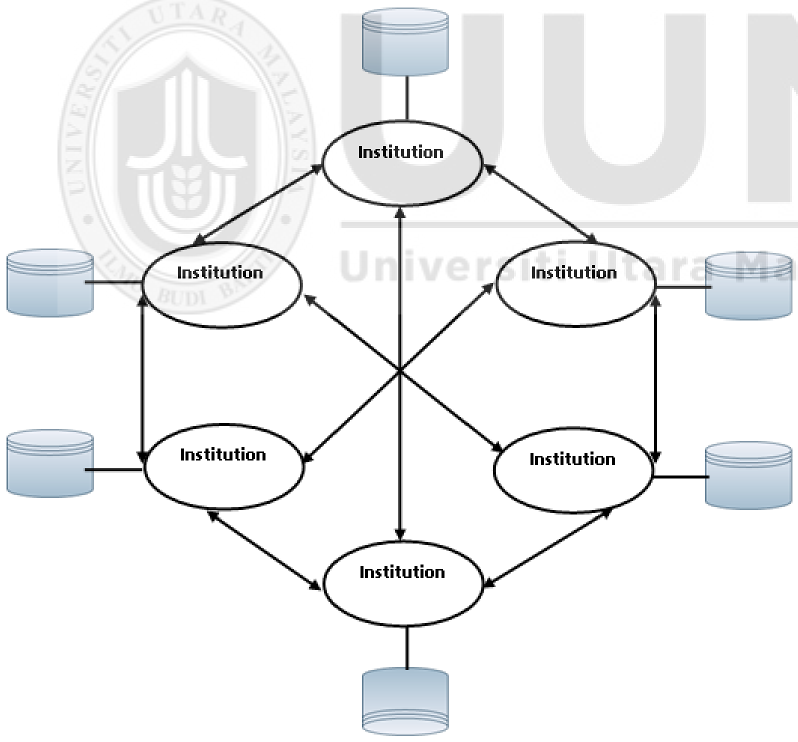


Figure 1.2. Grid federation data resource sharing system [14]

The concept of DGF systems supersedes that of Parallel and Distributed Computing systems (PDCS) because the former incorporates numerous resources, and it spans different administrative domains with varying data management policies [15]. The task of data management over a DGF infrastructure is very complex owing to ambiguous factors such as heterogeneity, dynamicity, organization-specific policies, and various political and socio-economic factors [1]. Each of the grid system type may consist of several hardware and software resources that are available to the numerous users.

Both DG and DGF platforms emanated from the same computing background, except that the latter is formed from the former and consequently, the resulting federation environment inherits some of the characteristics from the traditional DG platforms. The abstract model of DGF data resource sharing system shown on Figure 1.2 consists of various institutions that may span several regions globally, in which each institution serves as both provider as well as requester of contents and services to the federation users.

Thus, DGF systems contain more regions and larger number of sites, compared to the traditional DG system, which affects the way data replication is performed in these systems. In DGF system, resource sharing is necessary to facilitate access to data files or workspaces for collaboration between the federation units or regions. The shared resources could be accessed directly by federated sites, without seamingly passing through intermediate entities. The members of such a federation are thus resource providers as well as resource requestors [1]. Data resources could be made more available via replication service, which makes duplicate copies of available data resources from one region to other regions that may need such data services. However, data replications need to be done with caution, so as not contraints the available storage resources within the system [14]. Also, the federation system consists of several sites that are far apart, and may communicate with one another via WAN connectivity, which may necessitate the need for data replication services within the individual regions of the DGF system. These are some of the motivating factors that call for continued research in this research domain, which will be disussed further in the next section under research motivation.

1.2 Research Motivation

In a Distributed Computing system, such as the DG system, data replication plays a significant role in improving data availability, as well as making the data more accessible to numerous users [16]. Furthermore, data replication helps to improve data availability, which if done with caution could improve the performance of DGF systems. Data availability means to have the required data accessible and obtainable at all times by the users from the nearest possible replica site [17].

A replica of a file is a duplicate copy of that file, which looks exactly as the original file. Also, a file replica may have a link to its original file over the network, by means of some synchronisation mechanisms [18]. Thus, in a DGF system, data replication service is of paramount importance, due to the size of federation. In DGF systems, for data replication to be more efficient, several challenges and constraints have to be encountered. These include Peer-to-Peer sites connectivity issues, file dependencies, availability of replica sites, file transfer time, site workload, the distance between client and host site, and the required number of replica copies to meet up with data demands from the growing number of users [19], [20], [21]. A DGF system could be regarded as a Peer-to-Peer system, if there is resources sharing capability amongst the participants, in terms of hardware facilities such as processing devices, storage devices, network links capacity, printing devices, as well as other related online equipments [22]. In this type of setup, resources failures due to the unpredictable sites' behaviours could present serious issues regarding data replication. This is because, Peer-to-Peer connectivity exhibits specific characteristics, in which the clients behave both as providers and requesters to particular services. In other words, each participant commits a part of its resources to serve other clients directly. Thus, when the provider failed or become inadvertently absent at the time its services are much needed by the requester, this present serious setback in terms of jobs times and bandwidth consumption. Also, due to the loose coupling of the federation architecture, distance between the provider and the requester may increase file transfer time of peer sites that are located far apart, which will consequently affect bandwidth usage and jobs times.

Another major challenge in DGF, similar to conventional DG systems, is the storage issue, and the required number of replica copies to be created. Thus, there is a need for replica creation and placement strategies that consider all the necessities for data replication (availability, reliability, consistency, accessibility, and scalability), without compromising the storage and bandwidth costs within the DGF system. Unfortunately, however, most replication mechanisms in the literature are designed to tackle specific aspects relating to the various cost functions within the grid environment. Some mechanisms concentrate on improving data availability [23], while others seek to improve data reliability and consistency [24], [25]. It is challenging for a single approach to address all the issues in one mechanism, but the ideal situation is to find a mechanism that addresses the most crucial aspects. Usually, a tradeoff is done between the various cost functions involving bandwidth, storage and computational resources [1], [26], [27], [28]. Replication services enable data sharing across regions, which improves data localization to different administrative domains [29]. Numerous issues related to sciences and engineering disciplines stimulate the intensified interest for developing and deploying DG infrastructure [30], [31]. Because of the increasing popularity of the Internet and the globally coordinated virtual approach to conducting scientific and engineering related experiments; for instance the Large Hadron Collider (LHC) project [2], the global community is overwhelmed with huge scores of data worldwide. As these data overflows become wide open, an opportunity has presented itself for real time capturing of scientific data. In addition, the time required to turn the captured data into meaningful information continues to reduce by the day. DGF systems enable research institutes to take advantage of the grid computing facility to address complex computational challenges. Therefore, the needs to buy large and expensive servers for applications that can be split-up and work out to smaller application servers has reduced drastically [32], [33]. Results can then be concatenated and analyzed upon job(s) completion [34], [35], [36].

As seen in Figure 1.2 (page 4), the various institutions or organizations are typically referred to as regions or clusters within the DGF system [37], [38]. In such setup, accessing large datasets from sites across the regions will attract transfer time and consumes high bandwidth. Furthermore, the LAN as well as the WAN communications amongst the regional sites present overhead and may tie down the overall performance of the system. Therefore, if important data files are duplicated in the various regions, it will reduce the number of WAN communication, which will also reduce file transfer time and consequently prevents bandwidth congestion across the regions [38]. Data locality via replication is required to improve access performance, job throughput and decrease WAN communications amongst the sites, and ease the constraints due to cross registrations of users [38]. Within the grid setup, there are job-scheduling mechanisms that convey some particular jobs types onto a given set of sites or destination sites for executions. All grid jobs require certain data items for their execution. These data need to be available to the users' discretion at the time of request. Therefore, data transfer time is a factor, which requires data to be replicated to local sites.

Also, Data replication has the potential to minimise job execution time by decreasing bandwidth usage and time required to transfer data files. The task of data replication is performed by creating several copies of important data files in the individual regions of DGF systems [39], [40]. Thus, data replication has the potential of improving data availability [41], [42], minimizing bandwidth consumption and time required to access data files [38], [43] by creating important replicas of the source data files [44], [45], [46].

1.3 Problem Statement

DGF systems present an interesting and dynamic environment for global data sharing and management [1]. However, issues such as sites communications, file dependencies, sites availability, sites workloads and distance between replica sites present serious concerns, regarding data replication within these systems, which tend to impede in their smooth running. The reason been that there is increase in sites communications and distance between sites due to the federation size, as well as sites failures due to the loosely coupled nature of the federation system, which requires a consistent load sharing amongst the active sites. Thus, sites communications due to data access across regions tends to affect bandwidth usage, which has a consequent effect on the jobs completion times of the users [38], [39].

One of the main problems in DGF systems is how to determine the importance of a data file (file value evaluation), which is the first step in replica creation for determining the desired files that need to be replicated to meet users needs, as well as unwanted files that need to be evicted from the system. The existing works on data replication made several efforts [8], [26], [40] to resolve the issue of file value evaluation. However, their evaluation process had been based on files access frequencies, thereby limiting file importance to users only. Thus, the importance of a file to another file, which is denoted by the inter-dependability of file replicas, is an essential factor in determining the desired file to be replicated or unwanted files to be evicted, and has been ignored by the existing DRCM and ELALWmechanisms.

Another serious concern in DGF system entails selecting sites locations where to place replica files, which could be drastically affeted by the availabiity of the replica site [47], [48]. Availability refers to the situation whereby a site becomes *online* or *offline* at the time it is needed. The existing works made some efforts by cosidering sites distance and workloads, while placing new files replicas. However, they failed to consider the availability status of all the sites where to put the newly created files replicas [8], [26]. These factors combined, are essential for determining a replica placement cost (RPC), which will improve the overall performance of DGF systems [26].

The issue of site workload is a measure of load balancing, which affects the storage usage and the jobs completion times in the DGF systems. The existing works made efforts in addressing the issue of sites workloads. However, their works ignored lightly loaded sites, which ensures highly loaded sites are not considered for replica placement, but only lightly or moderately loaded sites should be considered [46].

Regarding the issue of determining the required number of replica copies, which is a measure of data availability [49], [50] to satisfy users demands within the DGF system, files dependabilities and the size of DGF system affect the way of finding the required number of replica copies, this is a serious issue needed to be addressed. The works of [8], [26] could have resolved this issue, except that their file evaluation process did not adequately addressed the issue of finding the popular files, which is a pre-requisite step to finding the required number of replicas.

A file's popularity [51], [52] can be determined based on access frequencies, but this factor is not enough without due consideration to the possible inter-dependencies amongst the various files within the federation regions. Therefore, the ELALW popularity criteria could be enhanced by associating the access frequencies with file inter-dependency factor. The replica creation decision should select a popular file based on access weight [8], [50], [53], [54], and indirect (clustered) logical dependencies, in addition to direct logical dependencies proposed by the DRCM in [26].

Finding required number of replicas will mean that some replicas are not needed, and thus need to be evicted from the system. This might create serious problems, considering the possible logical inter-dependencies amongst replica files, which may lead to deleting replicas with many connections to other files [8]. Evicting files replicas with high connectivities to other files may lead to erroneous data transfers, which may affect jobs time when dependents files required to complete such jobs are missing. The existing works evicts file replicas based on the least frequently used (LFU) factor, but failed to consider dependability factor.

Further to replica placement decision, ELALW finds direct shortest possible distance using hops counts. This is inadequate considering that hop counts does not always present the shortest possible distance value [53], [55]. In other words, the shortest distance could be via a single or multiple sites. Thus, replica placement cost (RPC) was not adequately addressed by the existing mechanisms.

In addition, site availability impacts on both replica placement and replica eviction decisions, which previously has been treated on the platform of Peer-to-Peer distributed computing [56], [57]. However, despite its potentials in making data replicas more available, site availability factor is yet to be incorporated into data replication mechanisms by existing researches. In this study, an enhanced dynamic replica creation and eviction mechanism (DRCEM), is proposed for improving the performance of Data Grid Federation systems. The proposed DRCEM mechanism strikes a balance between two existing data replication mechanisms DRCM and ELALW, which were also based on the European Data Grid (EDG) infrastructure topology [2].

1.4 Research Questions

The primary research question entails how to adequately address the issues of file values, workloads and site failures, as well as inter-dependencies amongst replica files, while taking decision on replica evaluation (finding popular files and computing required number of replicas), replica creation, replica placement and eviction within a federated data grid environment, for minimising jobs completion times, storage usage, computing element usage and bandwidth usage. The following sub-questions help to answer the main research question.

i. How to determine the highly important files and the required number of replica files needed for replication, based on their access weights and inter-dependability values, within a federated data grid environment ?

ii. How could sites failures, workloads, and distances between replica sites be computed, to determine the most suitable locations to place new replica files, within a data grid federation environment ?

iii. How to determine desired storage space that could accommodate the newly created replica files, as well as avoid deleting files that may be needed in the future ?

iv. How to measure the performance of the proposed mechanism in a numerical simulation environment ?

1.5 Research Objectives

The fundamental goal of this research is to develop a mechanism that creates replicas of important files, allocates the replicas to suitable storage locations based on site distance, site workloads and site availability, and evict less significant replicas based on their sizes, access weights and inter-dependability values. The mechanism aims to improve the performance of Data Grid Federation system in terms of jobs completion times, network bandwidth consumption, storage element usage and computing element usage.

The proposed mechanism incorporates three schemes namely (i) Dynamic Replica Evaluation and Creation Scheme (DRECS), (ii) Dynamic Replica Placement Scheme (DRPS) and (iii) Dynamic Replica Eviction Scheme (DRES). In pursuance of this central goal, the following sub-objectives are formulated.

i. To design a dynamic replica evaluation and creation scheme that selects highly important files and computes the desired number of files replicas required for replication, in a federated data grid environment, based on their access weights and inter-dependability values.

ii. To design a dynamic replica placement scheme based on site failures, sites workloads and sites distance, to determine the most suitable site locations within each region for placing new file replicas.

iii. To design a dynamic replica eviction scheme that frees more space from suitable storage elements based on file sizes, access weights and logical dependencies, to accommodate newly created files replicas, without deleting files that may be needed in the future.

iv. To evaluate the proposed mechanism using jobs completion times, network bandwidth consumption, storage element usage and computing element usage metrics in OptorSim numerical simulation environment.

The replica creation and eviction mechanism seeks to determine the frequency by which individual sites join and leave the federation system, as well as inter-dependency values of the popular files. As a complimentary activity, there is a need to develop an algorithm that arranges the sites according to increasing/decreasing order of failures over a designated period, to achieve the desired goal. The file with highest /frequent failures should not be considered for placing file replicas.

1.6 Significance of the Research

In a DGF system, file replicas could be accessed from different regions for executing a variety of jobs. In addition, if the requested files reside on site locations that are closer to the requesting client, then access time and bandwidth consumption will be minimised to the benefit of the user. This strategy helps to improve the job throughput.

However, if the requested file replicas are not available on the site, which is scheduled to execute the job, then users will have to process their jobs remotely. In such cases, it will take much longer access time for the job to be processed; partly due to bandwidth constraints, and partly due to the size of the replica file, wherein some applications, may scale up to gigabytes in size. However, the constraints of inter-communications bandwidth across DGF sites is avoided via replication of important data files in the various regional sites. One of the significance of this thesis is that data availability is improved via replication of important data items, without constraints on the storage elements usage.

Also, another significance is that data access time is minimised by placing replicas within the regional sites, which improved job completion times in DGF environment. Replica placement helps avoid sites with down times, thereby improves data availability and access times.

In addition, Replica placement performs load balancing by considering lightly and moderately loaded sites, thereby improves jobs times and saves bandwidth consumption, as well as storage usage. Replica eviction considers file dependability, by not deleting an important replica, which may be needed by the users or other files at a later time. Thus, replica eviction creates space for an incoming replica, in case there is no available storage to hold the newly created replica. This also helps to load balance the regions of the DGF system.

1.7 Research Contributions

i. The evaluation of popular files by computing logical dependencies amongst the data files, stands as a major contribution to the research domain, which prviously has been based on files access frequencies.

ii. Another major contribution by this research is its ability to resolve the problem of sites failures, which before now, has been a soring issue in determining the appropriate locations sites for replica placement decision.

iii. Computation of distance between replica sites has been enhanced via the use a modified Dijkstra's algorithm, which previously has been based on hops counts.

iv. The issue of replica eviction has been enhanced in such a way that important replicas that may be needed later, are not caressly evicted from the system.

v. Also, the research has made significant cotribution in making space more available within DGF systems, through the provision of a dedicated dynamic replica eviction mechanism.

vi. The issue of what file to replicate has been significantly addressed by making sure that only important files are replicated; thus avoiding redundant replications, which may consume the much needed storage resources. This could be seen in the fewer number of replications done by the mechanism compared to the existing mechanisms.

1.8 Scope of the Research.

This research focuses on a federation of hierarchical DG systems, similar to tree-like-structured grid model, which reflects the EDG platform structure [58], [59], [60]. The hierarchical DG model is a typical architecture used in various research works [58], [59]. The modality of data that are used in this work is in the form of structured data. The work is limited to a mechanism for data replication. This research focused on replica creation and eviction in a DGF environment. Thus, designing of job scheduling mechanism is not covered by this research. Figure 1.3 shows the scope covered by this research regarding domain, applications, and resources.

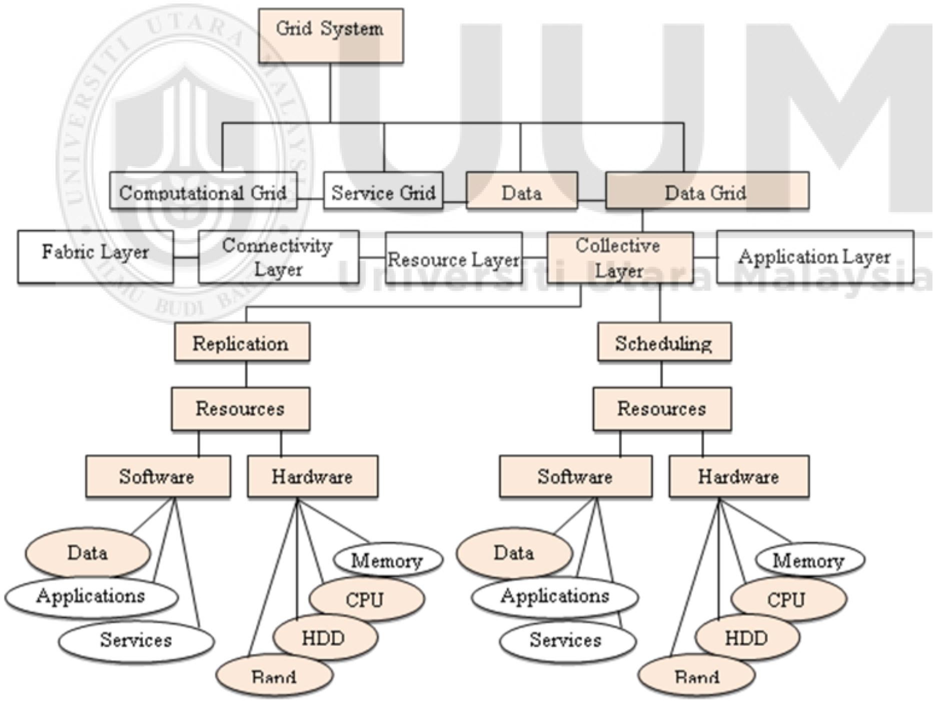


Figure 1.3. Scope of the research

The type of data, applications, topology and resources covered by this research are explained in this section. The data used in this research is of read-only type. Thus, this research has not considered the consistency of write types and costs due to overheads of update propagation. The shaded boxes in the diagram indicate the scope of this research. The resources considered in this work are limited to data, storage facilities, bandwidth, and processor. The proposed mechanism aimed to improve the performance of DGF systems regarding jobs times, storage usage, bandwidth consumption and computing element usage.

1.9 Thesis Organisation

The work in this thesis is organised into six main chapters. The work commences with an overview and explanations of the research domain, which is presented in the introductory Chapter One. The chapter also presents the statement of the research problems, as well as the research questions and objectives, amongst others. The remaining chapters are organized as follows:

Chapter Two presents a broader overview of Data Grids (DG) regarding their models and layered architecture. The chapter discusses the basic conceptions and dissimilarities between the existing DG models, which include Monadic, Hierarchy, Federation, and Hybrid model. Furthermore, the chapter highlights on how Data Grid Federations (DGF) are instituted by joining various DG systems together. Most importantly, the chapter discusses the unique characteristics of DGF systems, which makes them open for continued research, as well as their domain of application. The chapter also discusses the concepts of data availability and locality, which aids to address the research questions adequately, as well as develop a keen understanding of data replication mechanisms. Finally, the chapter reviewed comprehensively, some research gaps from relevant related studies in DG systems with a focus on data replication mechanisms. The outcome of this chapter leads to deciding on the mechanisms to put in play for realizing the aims and objectives of this thesis.

Chapter Three explains the tools and methodology used for carrying out the research. The chapter presents a brief description of the simulator used in this research. Also, the chapter describes the grid federation architecture together with the inputs to and expected outputs from the simulation environment. The measurable metrics that are used as benchmarks, for evaluating the performance of the proposed mechanism, are as well presented in this chapter.

Chapter Four. The chapter presents the detailed design of the proposed mechanism, after a brief overview of the design objectives. The various schemes and algorithmic requirements of the proposed mechanism, as well as the individual components required for the design, are also explained in this chapter. The chapter presented some numerical illustrations aimed at demonstrating how the proposed mechanism works. The implementation processes of the proposed mechanism, which includes integration of the new mechanism into the simulation environment, as well as the necessary codes implementations are detailed in this chapter. The chapter concludes with a summary section.

Chapter Five presents research results of the proposed replica creation and eviction mechanism, for addressing the research problem. It discusses the different scenarios used in the simulation process. The various results output from the simulation process are discussed, as well as compared with other related replication mechanisms for benchmarking.

Chapter Six concludes the research, by summarising the entire research work. The chapter also highlights on the statements of the research contributions, as well as future direction for further works relating to this research domain.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

This chapter presents a general overview of DG models and their layered architecture, together with a brief analysis regarding the different models of DG systems, which include monadic, hierarchical, hybrid and federation models. The chapter also discusses some of the Data Grid Federation models available and their areas of applications. Further, critical review of related researches in specific areas of data replication mechanisms within Data Grids and Data Grid Federations is conducted in this chapter. The central aim of this thesis is improving data availability by proposing an enhanced dynamic replica creation and eviction mechanism (DRCEM) that seeks to improve the performance of DGF environment in terms of jobs completion times, storage usage, bandwidth consumption and computing element usage. Also, this chapter explored in further details, the characteristics features of two selected existing ELALW [8] and DRCM [26] mechanisms, for performance comparison later with the proposed DRCEM mechanism, based on the afore-mentioned performance metrics. Also, the chapter concludes with a summary.

As mentioned earlier, the focus of this thesis is on improving data availability via dynamic replication of valuable data, and eviction of unnecessary files replicas. The proposed mechanism evaluates existing files replicas based on their importance to both users and the DGF system at large and duplicates essential files replicas to attain the desired availability without constraining the storage resources. Thus, the importance of a file to users is indicated by how frequently a file is accessed by the users [42], [53], while file’s importance to the system is indicated by number of direct and indirect logical connections of a file access to other files in the system [61].

For instance, in software engineering, logical dependencies are exhibited by a set of data files that changed simultaneously due to reasons of change in proprietary information, classes refactoring, version changes and other elements that linked the files to the same semantic or logical class [61]. Another instance of logical dependencies is observed on students’ records system in an institution, which typically keeps two or more file records for a single candidate namely personal record and academic record. The academic record is usually split into courses record and exams record. All the three records are linked via a primary key, which may be candidate’s matric number. Changes to the matric number will affect the other files.

In recent times, the information age challenged the global community with an enormous amount of data produced on a daily basis by many works of life. Some of the applications include scientific findings, sensor networks, Internet of things (IoT) and engineering applications, amongst others. The enormously produced data items are used for data sharing and collaboration across a variety of discipline over various WAN-enabled organisations, world over. Consequently, effective administration of these voluminous data resources is regarded as a significant area of research for sciences, engineering as well as other application areas in the global institutions [62].

The Data Grid (DG) tends to resolve this problem by offering a scalable as well as distributed infrastructure that connects several compute and data-intensive resources from diverse places, establishments and varied platforms [63]. The DG environment enables users to access resources transparently, similar to retrieving the resources from their local regional sites. The era of grid computing has witnessed several technological advancements over the years [64], which was fueled by various motives namely computational needs, data requirements, application services requirements, utility services, interaction and knowledge-based services.

2.2 Data Grids

Data Grids enables users to access a significant volume of data, operate and transform the data to suit their needs. The users could also generate and maintain various copies of the data items, as well as store them in distributed storage facilities [18]. Figure 2.1 presents a high-level view of worldwide DG system.

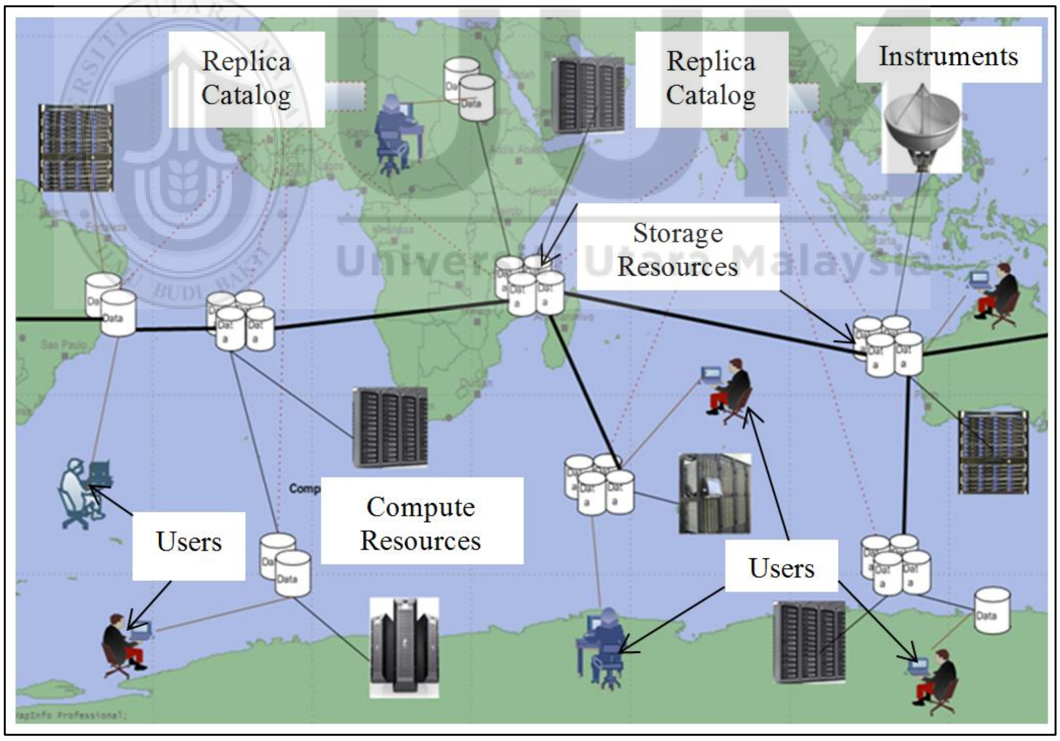


Figure 2.1. An abstract view of data grid system [30]

The DG system depicted in Figure 2.1 spans different countries, the components of which include high-speed networks linking computational and storage resources. Some of the common services offered on DG systems include mechanisms for data resource discovery, management and transfer to requesting clients [1], [9], [26]. Nevertheless, the discovery and transfer services are based on requests of an application, which forms part of a collection of other related services. These related services may include, amongst other things, management of replica consistency, data cleaning and analyses [24]. DG systems are typically equipped with a security layer that mediates all operations, to ascertain that all requests originate from genuine sources, and only approved operations are allowed to accomplish on the platform.

Another crucial service offered on the DG platform is the management of shared collections of data items that originated from diverse sources. These data items are maintained on different types of storage elements (SE). The scalable nature of DG systems allows an authenticated and effortless addition of new nodes to the platform. Due to the dynamic nature of operations platforms, and in case of migrating from one platform to another, DG systems incorporated persistent storage archive as a mandatory service, to preserve the various data collections, related metadata, access permissions, and information regarding versions upgrades [14].

Furthermore, from the abstract view of DG systems in Figure 2.1, major centres communicate with each other via high bandwidth networks represented by thick lines. The thinner lines indicate the low-bandwidth networks that link the subsidiary centers. Each primary data store maintained data items that are produced by either experimentation, scientific instruments, or sensing nodes. These data collections are transmitted to other storage locations on requests, away from where the data originated, by the means of data replication services. Additionally, scheduling mechanisms are deployed to ensure that users' requests are routed to appropriate data sites within the DG environment.

The first point of contact when addressing data requests from users is the local replica catalog. Data from the local storage facilities are obtained and transmit to requesting clients, after authenticating the requester's credentials. However, if the requested data items are not readily obtainable from the local storage facilities, the data need to be obtained from a remote repository. Subsequently, the data may have to be routed to a super computing facility or a cluster facility for processing purposes. In the case of DGF systems, these computing facilities may be part of a DG infrastructure located in another region. The processed data items may then be subjected to series of activities, which include sharing, visualisations, analysis and duplications on local storage facilities [14]. As mentioned earlier in the introductory Chapter One, data replication improves data locality in both DG and DGF systems, but comes with a price, thus need to be done under some guiding principles, which will be discussed later in Section 2.6 of this chapter.

2.3 Data Grid Projects

The organization of data items from originating sources determines to a large extent, the type of DG model design. Thus, numerous DG models exist globally, for various kinds of operations, which depend on some factors such as the data origin, distributed or single source, size of the data, and mode of data sharing. Data Grid is part of the Grid Computing paradigm, which presents an evolving structure for accessing distributed resources, complex computational and data resources, across independent organizations. Their federation is fomred by linking various DG platforms. More discussion on DG federation is given in Section 2.5 of this chapter. Data Grids present a robust data management systems for global sharing, distributing and maintaining data that reside on storage systems belonging to various managerial domains. A DG platform offers consistent namespaces for its users, as well as other entities such as the digital objects, storage space for maintaining persistent identities and access rights, which enables replicating certain data items as well as scheduling of users' jobs, while taking into accounts the different user's time zones. The grid infrastructures are analogous to the electrical power grids that afford users with a persistent access to electrical energy, with little concern over the source of such services. Examples of grid projects, which combined the computational powers of worldwide-distributed computers include BeInGrid, BIRN (2005), GridPP, TeraGrid [65], ChinaGrid, and APACGrid) [59].

Data Grids are developed based on the need for a global-scale data management services including access to data, integration, processing and archiving through distributed data warehouses. Some notable amongst DG projects include EGEE, LHC Grid, and GriPhyN [65]. The Net-Solve and Grid-Solve are two notable Application Service Grids developed to provide access to remote applications, modules, and libraries that are hosted on Data Centers (DC) or Computational Grids (CG) [65]. Access Grid is an example of interaction grid with a focus on interaction as well as collaborative visualization amongst users.

Knowledge Grids aimed towards knowledge acquisition, processing, and management, as well as offers functionalities for business analysis via unified data mining services. Utility Grid emphasized on providing all the grid services to its users, which include computing power, access to data services, and utility services that are part of users' benefits when duely subscribed [7]. Grid systems portrayed a layered architecture with Computational Grid (CG) forming the bottom layer, while the Utility Grid is forming the topmost layer [30]. A grid at a higher-level employs the services of grids that operate at bottom layers in the design. For instance, a DG utilizes the services of the CG for data processing and hence builds on it. Also, the grids at the lower layer focus heavily on infrastructure aspects, whereas those at the higher layer focus on users and quality of service delivery. The next sub-sections explain these models in further details.

2.4 Data Grid Models

As opined in Section 2.3 of this chapter, DG systems are modeled according to the underlying projects that created them. The various DG models depict the mode of data flow in a system. For instance, in a centralised DG platform, all requests for data items are directed to the central storage facilities. A variety of models is in place for the operation of a DG services. Figure 2.2 shows four notable DG models that are used to deploy DG infrastructure around the world today, together with their characteristics elements.

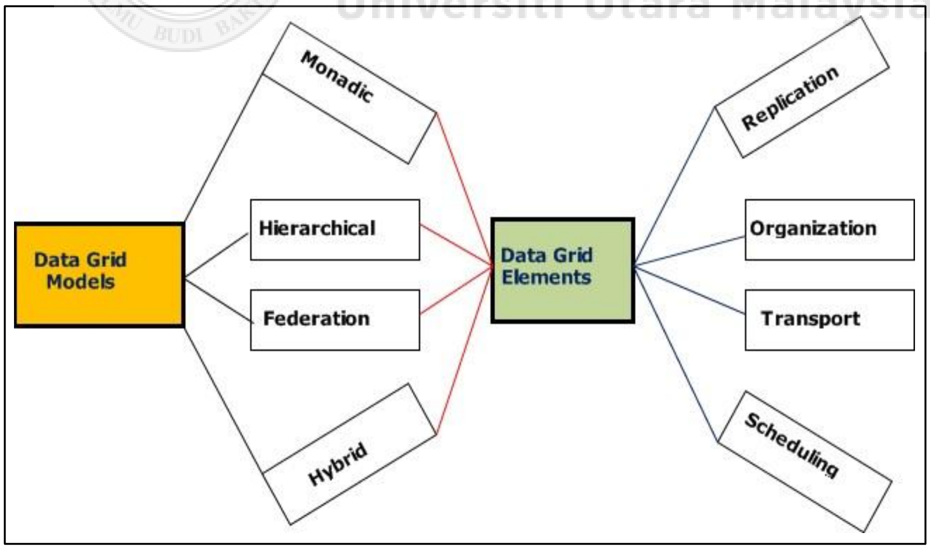


Figure 2.2. DG models and their characteristic elements [30]

The models architectures are dependent upon specific factors namely; data source; whether single or distributed; as well as the size and method of sharing the data items [30]. The model on the Figure 2.2 are explained susequently in the next paragraph.

- *Monadic (Centralized).* This model is regarded as the common method of deploying a DG infrastructure, whereby the entire data resources are stored on centralised storage facilities. The data stores accept users' requests for data, then deliver the requested data items if available. The mornadic model offers a single access point, by which users retrieve data items. Thus, the distinction between monadic and the other DG models is that, in the other DG models, data items could be retrieved fully or partially from a variety of access points within the system.
- *Hierarchical.* This model presents data to collaborators in the form of a hierarchy. Data items flow from originating sources to major centres and to sub-centres.
- The *Federation* model [30] is dominant in DG platforms deployed by organisations or institutions that are willing to collaborate by sharing some of the data on their localised data stores. Examples of DG federation include the Bioinformatics-based Research Networks (BIRN) federation [30], NASA backup federation [15], NPACI driven federation [9], NARA federation [9], BaBar federation [9], PDB federation [9], SIOExplorer federation and Web cache federation system [9]. Section 2.6 (Data Grid Federation Scenarios; page 29) gives more explanations on these models.
- *The hybrid model* combines the features of the other DG models.

This study focuses on designing a data replica creation and eviction mechanism for DG federation systems. The proposed mechanism aims to improve on the performance of DGF system regarding jobs completion times, bandwidth usage storage element usage and computing element usage. The next section gives an overview of DG federation models and scenarios, as well as their characteristic features, some of which were inherited from the traditional DG systems.

2.5 Overview of Data Grid Federation Systems

Data Grid Federation refers to a model for distributed resource management that group a pool of computing sites into clusters or regions [1], [9]. DGF is the focus of this research, and thus, the overall goal is centred on enhancing the performance of the system regarding jobs times, bandwidth, storage and computing element usage, by improving on the data replication mechanisms within the DGF platform.

DGF offers a global platform for collaboration and resources sharing systems that consists of various DG systems, which are linked via WAN connections [1], [9], [30] [66]. The platform enables complete decentralisation of resource control and offers better scalability, as well as a self-organize-able and fault-tolerant computing environment.

DGF provides an alternative to the unbalanced resources sharing constraints imposed by the centralized DG systems [3], [30], [45], [67]. The logical coupling of DGF systems could take any of the following forms namely a Hierarchical Peer-to-Peer [22], Centralized Peer-to-Peer [22], Decentralized Peer-to-Peer [22] or Absolute Peer-to-Peer system [68].

Depending on the type of logical coupling, different DGF scenarios exist, which serve the global community on various works of life. According to researchers in

[1], [9], the feature of grid computing is moving towards federated infrastructure, integrating many global communities and numerous users. The next section explains the point of difference between the conventional DG platforms and the federated DG systems, together with scenarios in which DGF systems are deployed.

2.6 Data Grid Federation Scenarios

This Section highlights some DGF systems that have been in operations over the years. The overview will be used to lay the foundation on the inherent characteristics of the DGF systems for proper justification and motivation for the study. According to research in [68], DG platforms could be federated over dedicated Peer-to-Peer grid systems, with one or more servers, in which case the platforms are referred to as centralized Peer-to-Peer DGF systems. As expected, decentralized Peer-to-Peer DGF systems are not strictly controlled by dedicated servers. The decentralized Peer-to-Peer systems can be a single tier or multi-tier (hierarchical) in their topology. The hierarchical Peer-to-Peer DGF systems combine both centralized and decentralized architectures to take advantage of both topologies [68].

Research conducted by the authors in [1], [30], explained DGF as a model for distributed resource management that group a pool of computing sites into clusters. In addition, the researchers explained that DGF could be seen as a platform for sharing of vast resources globally that incorporates individual DG systems linked via peer level connections. The peer level connection approach enables complete decentralization of resource control and offers better scalability as well as a self-organisable and fault-tolerant computing environment. According to research conducted severally by researchers in [1], [30], [35], [69], they maintained that a DGF system offers solutions to the uncoordinated resources sharing constraints imposed by the centralized DG systems. In [1], it was reported that the Grid Computing in general, emanates from the existing distributed computing paradigm, which encapsulates the ever-increasing internet-based communities, with diverse topology and computing resources that may span over different administrative domains. In addition, according researchers in [1], the distinction between traditional DG system and DG Federation is that: Whereas DG systems offer the capability for management of data resources on distributed storage facilities; the DGF platform provides a means of managing the data on multiple DG systems. The implication of this interpretation regarding DG systems and their federations suggests that DGF is an architectural framework for logical coupling of DG resources that are under different organisational domains. The various organisations are likely to run on distinct administrative and time domains, and may transparently share their resources based on given policy and schedule the grid jobs based on an established Quality of Service (QoS).

The DGF resource types include computing machines, commutating clusters, online instruments for scientific experiments, storage facilities, data resources as well as various types of applications. The consumers of these resources can utilise them to solve data-intensive applications.

Furthermore, the researchers in [1] explained that DGF systems present a loosely coupled architecture, whose logical coupling is usually determined by a pre-determined appropriate federation mechanism [13], [15]. Going further, they reported that, federating multiple DG systems imposes controls on users' registrations, which limits their ability to register across other regions. Consequently, their ability to access some resources, data files, and other utilities is also limited.

In addition, since DG federation brings together sites from geographically distributed resources [1], it imposes a number of issues namely increase in the distance between the regional sites and availability issues resulting from the loosely coupled nature of the regional sites. In addition, some jobs require more than one file to execute, and some files require other files or partial replicas for their proper execution, which brings about the issue of file dependability. Furthermore, since the data required for some jobs may not be available within the region of request, transferring such data to local region will have to consider workload or storage space availability.

Fortunately, however, the issues regarding sites distance and availability as well as files dependability and workloads could be resolved by deploying a robust data replication system, which will incorporate all these factors into one mechanism. These four factors further distinguish DGF systems from the traditional DG systems. Furthermore, the four factors are used by this thesis as the design metrics elements to help evaluate the performance metrics. Section 2.7.5 explains more on site availability, while Section 2.7.6 explains more on file dependability issues.

Also, the limitations due to users' inter-registration could be partly resolved by embarking on identity federations, as explained in research conducted by [70], as well as by integrating storage facilities and computing centers of different sizes, power, and architecture according to [71]. The later type of federation leads to Cloud Data Centers. The revolving point for these constraints could be associated with the users or imposed automatically by the regional DGF managers. The researchers further explained that the constraints might be restricted to either no inter-registration, partial inter-registration or complete inter-registration by the users.

According to research in [1], [9], more than 1,500 methods of federating DG systems may result, if users' registration metrics or constraints are combined in a variety of ways. Finding the best approach for achieving grid federation presents a dynamic research area because of the numerous ways the federation could be implemented. Thus, in recent times, research conducted by the authours on DGF systems in [15], explained a means by which various DG installations are integrated using identity federation. Their research case study was of National Aeronautics and Space Administration (NASA) on how to achieve a secure information sharing amongst the partnering organizations. Similarly, researchers in [72] buttressed the importance of identity federation for NASA's future, to provide viable collaboration competencies between NASA and the various partner organizations.

Each organization taking part in a DG federation maintains ownership and controls of its data resources [1]. The strengths of collaboration depend to some extents on the integration limits imposed by the various stakeholders. These limits may include site autonomy, users' inter-registration, and replication threshold as well as synchronization degree. With proper credentials, researchers from partner organisations may take advantage of the DGF platforms for their various data needs. The researchers delved further by discussing ten scenarios of DGF systems that have been deployed over the years, to tackle real-world issues in sciences [1].

Some notable instances of the real world DGF scenarios are highlighted in Subsections 2.6.1-2.6.7. Furthermore, researchers in [38] reported that each DG platform in a DGF system is referred to as a zone or region. The various regions manage their data stores, metadata indexes, users' credentials, list of resources and utilities within the system. Some of the federation scenarios alongside their areas of applications identified by this research include: Peer-to-peer or Napster federation [1], BIRN federation [12], CMS federation [73], NASA backup federation [15], [72], NPACI driven federation [9], NARA federation [9], BaBar federation [74], PDB federation [75], SIOExplorer federation and Web cache federation system [9]. The next sections discuss the type of Peer-to-Peer Federation models in more details, as well as research efforts in the development of a middleware architecture for federating different DG systems.

2.6.1 The Peer-to-Peer or Napster Federation System

The Peer-to-Peer federation of clusters reported by the researchers in [3], depicted this infrastructure as the type of model implemented by Napster [1]. This model is shown in Figure 2.3.

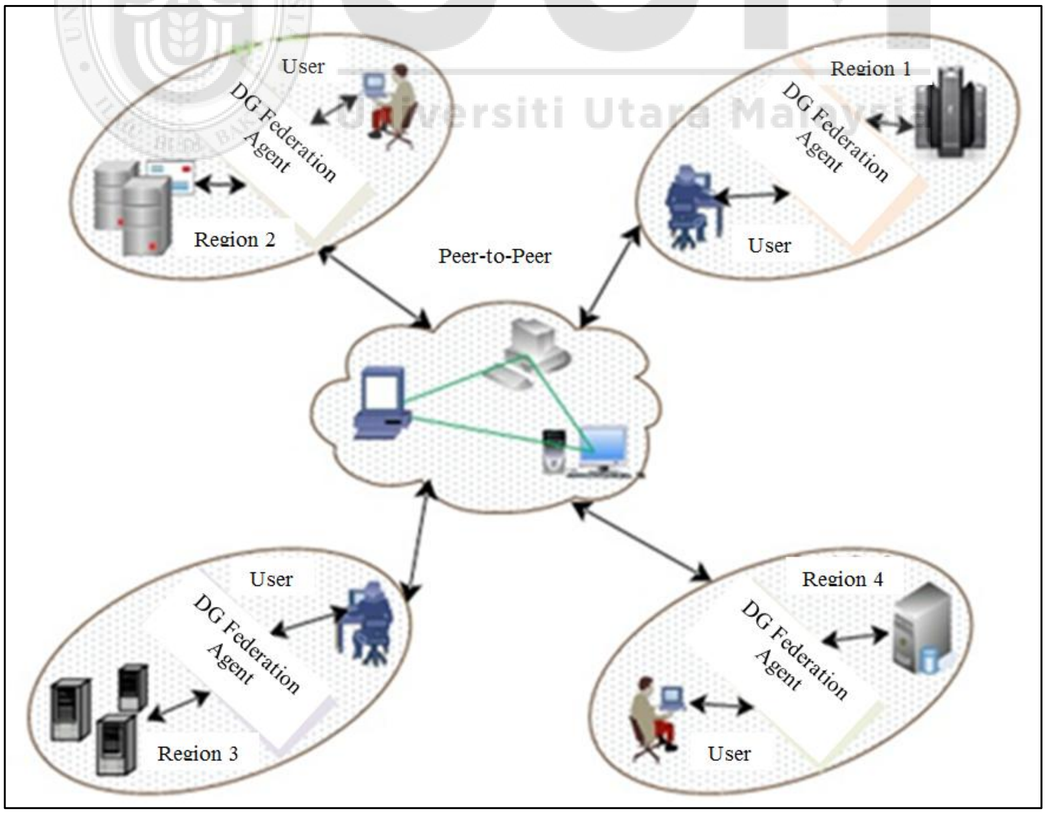


Figure 2.3. The Peer-to-Peer federation model [3]

The Napster federation is also termed as the “Free-floating Zones.” This platform has numerous individual regions without a central region. The individual regions are peers to their sister regions. The system comprises of few resources, which are accessed by limited number of registered users. The Peer-to-Peer federation model is predominantly adopted by organisations that are willing to share their data items from their local stores. The various regions are regarded as individual DG systems that operate on their own (similar to personal computer). The regions are losely coupled, thus the regions connect on occassions for data exchange or collaborative purposes. The occasional exchange is synonymous to data exchane with colleagues via auxiliary storage devices. Also, the regions are autonomous and control the kind of data shared with the collaborators.

2.6.2 The CMS Federation System

The Compact Moun Solenoid (CMS) Data Grids Federation, is explained by researchers in [62], [76] as a hierarchical model. In the CMS federation, data files originate from a major region. The data objects are then replicated to minor regions down the lower level in the hierarchy. The minor regions can manipulate the data along with the associated metadata and may wish to share all or subsets of the data to other clients. The CMS federation is dedicated to providing a structured analysis environment to physicists at the LHC centers, whose activities are focused on analysing the data from Physics experiments.

2.6.3 The BIRN Federation System

The Bio-Informatics Research Network (BIRN) reported by the researchers in [1] and [12], is a DGF platform with a focus on Bio-Informatics research data. This platform is termed as a resource interaction model, in which data objects are shared amongst multiple regions. The sister regions can replicate the data for use by their numerous users. This type of federation proves more effective in situations where the partner regions are placed far from each other, but would like to make access to data objects easier for the numerous users from other regions. The users can duplicate the data for offline usage, and may share the files to other users of the same mutual interests. The same process is carried on the associated metadata, which would then be synchronised across the regions. The BIRN started operations with a handful of sites and presented application-oriented test beds with a central coordinating center. The federation was initiated by the European National Institutes of Health in the year (2001) [12], with an anticipated growth that will span many regions, which will provide an open framework for global sharing of relevant data.

2.6.4 NASA Backup Federation System

The National Aeronautics and Space Administration (NASA) backup federation model, also called the Archival Zone or Back-up Zone, as reported by [1] and buttressed by the researchers in [15]. The recent development in NASA federation system portrays identity federation [15], [72], in which members of one organisation can use their credentials to access information hosted or managed by a partner organisation in a separate security domain.

The NASA federation was achieved via a pre-defined set of authentication information to the hosting organisation. Thus, different organisations can share information beyond the boundaries of their individual DG firewalls, which reduces the cost of credential management, improve security and provide a reduced sign-on experience to users. In this platform, there could be multiple regions, which share data objects with an auxiliary region referred to as the archival zone. The various partner regions donate data objects that are used to populate the archival zone. These archives serve as data backup for the entire regions, which may be stored on disks or other auxiliary data storage syetems.

2.6.5 The BaBar Federation System

The BaBar federation model is popularly known as the Replicated Data Zones [1], [73]. In the BaBar federation, each region works independent of the others, but maintains the same set of data objects as well as the associated metadata across the partner regions. In this type of federation, each region operates autonomously, and users' credentials in the sister regions are useful only within their regional boundaries, but not permitted to cross over to the partner regions. However, individual users may wish to obtain accounts with other regions apart from their domain, which will enable them to access as well as replicate data from those regions, as permitted by their subscriptions status. The advantage of this type of federation is that the regions could save network bandwidth, while sharing data over a WAN connection.

2.6.6 The Earth System Grid Federation

The Earth System Grid Federation (ESGF) was formed using a distributed and federated software platform [77]. The federation composed of numerous sites that are geographically apart. The sites interoperate via common interfaces, services as well as protocols for collaboration purposes. Each site maintains its data objects together with the related metadata independent of the partner sites. In addition, the sites can join or leave the federation at will. The ESGF Peer-to-Peer platform is an instance of globally operated systems; by which scientific data objects are accessed via web-based application interfaces. A few of the most noticeable implementations of ESGF data systems include NASA Global Change Master Directory (GCMD), which is a well established platform for geo-scientific metadata records, and British Atmospheric Data Center (BADC) for data collections relating to global atmospheric conditions.

2.6.7 Data Grid Federation Middleware and Frameworks

Software for joining different DG platforms, otherwise known as FedMi was developed according to research in [78]. Figure 2.4 shows an abstract architecture of FedMi proxy server package.

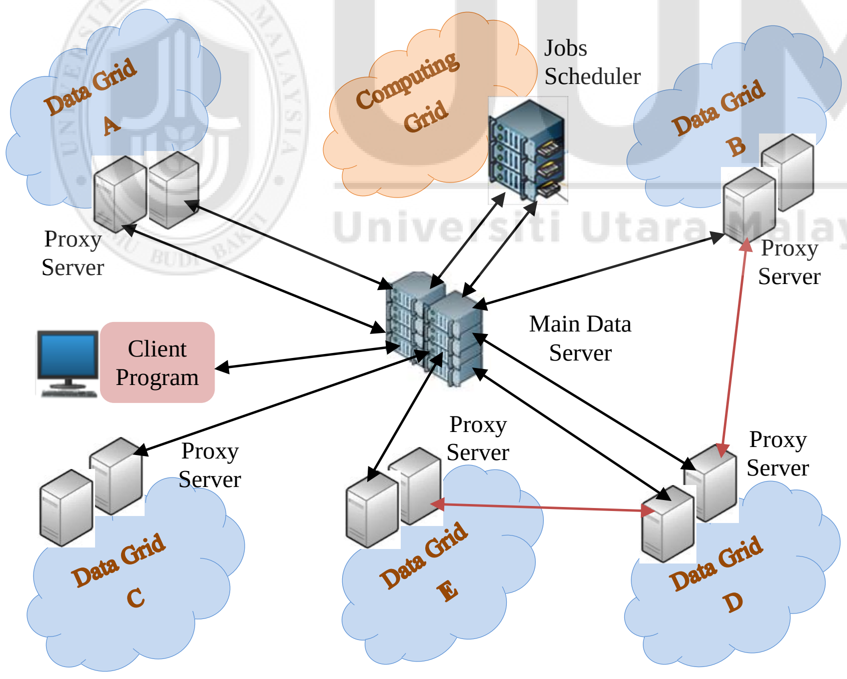


Figure 2.4. A Federation middleware for integrating heterogeneous data grids [78]

FedMi's goal was to achieve reliable collaboration between numerous DG systems from heterogeneous domains. The software comprises a system of proxy server package, which runs on top of the underlying implementations of various DG platforms. A common interface was integrated into the proxy server package, which enables users from various DG platforms to collaborate. For integrating a DG platform into the federation system, the DG administrator needs to implement the interface that will translate the basic instructions supported by FedMi package, into native instructions understandable by the underlying DG installation. By this way, the complexities of their architectures are hidden away from the various partnering organisations, thereby providing a high-level abstraction of their heterogeneity.

Also in [79] the researchers proposed a framework for joining various types of service grids from multiple domains, for efficient management of corporate intelligence data objects. The architectural framework provides support for synchronising various services, such as service-registries, service-composition, access-control, and monitoring, which are under the management of a service grid operator. The authors applied the proposed framework to the language-service domain for the establishment of a language grid. Similarly, in [3] the researchers proposed a framwork that consists of a Peer-to-Peer federation agent, which joins resources together from distributed cluster platforms, to enable a collaborative environment amongst the various partner organisations. The framework provides mechanism for cooperative and coordinated sharing of distributed clusters, aimed at resolving the problems of application-specific non-coordinated resource allocation nature of the tradional DG environment, which makes scheduling processes independent of the others in the system. The non-coordinated nature of the tradional DG systdms can exacerbate the load sharing and utilisation problems of distributed resources due to substandard schedules that my occur in the systems. To overcome these limitations, the framework allows resources to be used transparently from the federation, when local resources are inadequate to satisfy user’s demands.

2.7 Data Replication in Data Grid Federation Systems

The two principal issues that surround DGF systems involved how to integrate various DG platforms to form a federation sysyem over WAN connections, and how to make data more available as well as accessible via replication. Although developing an efficient middleware for federating DG systems presents open research issues [1], this research will not delve deeper into the aspects of middlewire development. The core of this research is for data replication, which has been previously introduced under Research Motivation (Section 1.2, page 6). Data replication generates multiple copies of the existing data objects to provide access opportunities from remote sites [80]. Subsequently, this section will discuss further into the concepts of data replication and its roles in improving the overall performance of DGF systems. Before discussing relevant literature on data replication, the next section discuss replica management architecture for DG systems.

2.7.1 Replica Management System and Replica Eviction

A DGF system joins together a globally distributed data objects, from diverse administrative domains. Every partner in the federation requires persistent access to various data objects of mutual interests. Data replication serves as a pre-requisite technique for preserving network bandwidth cost, and an essential factor for maintaining data reliability. Replication also ensures scalable collaboration amongst the partner regions . Replica eviction removes unwanted replicas from the system. Both data replication and replica eviction are part of the replica management system [81]. Figure 2.5 shows a system of replica management architecture, which consists of storage facilities that are linked via high-performance data trasport protocols.

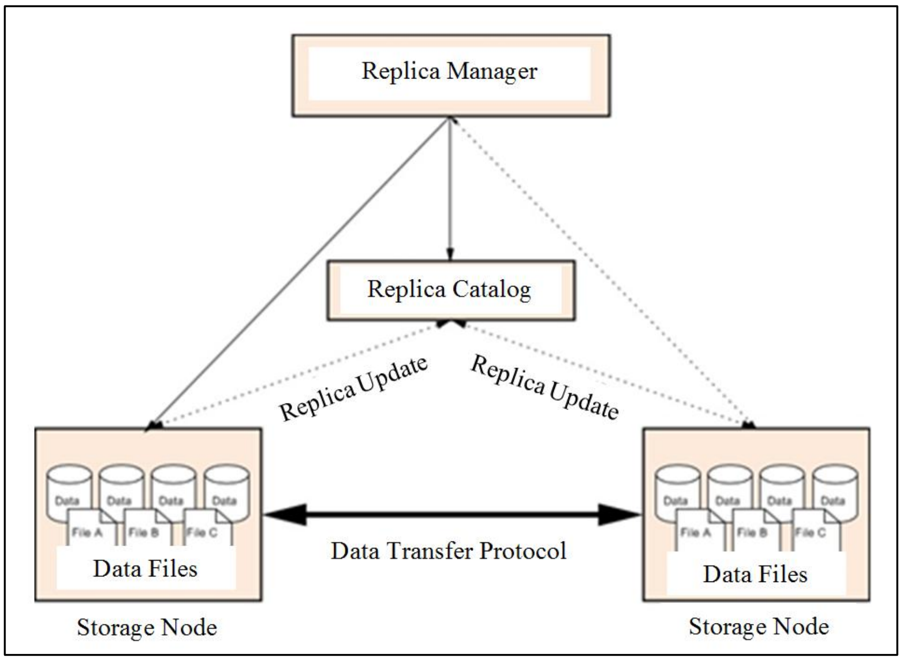


Figure 2.5. Replica management architecture [30]

In addition, the performance of a replication mechanism could be affected by the available storage space as well as the network bandwidth between source and target sites [82]. Thus, there is a need for a replica management system [83], [84], which will guarantee access to data items, as well as the management of the available storage spaces.

In a typical replica management system, replicated data objects are managed by the replica manager, based on storage space availability of the relica sites. In the replica manager, there is a catalog or a directory that keeps track of the locations of files replicas. Various applications may query the replica catalog to find if replica copies exist for particular data objects in the system. The query will return the number as well as location information for the duplicate files. The application software on the client machines will usually incorporate library suites that query the catalog for possible existence of certain data objects within the system. The replication methods as well as the replica management system are amongst the essential components of dynamic replication strategy. An important function of replication mechanism entails the ability to minimise the time required to access data objects in the system, which is part of the core objectives of replica optimisation services. If all requests for data objects are sent to the appropriate replica sites, files' access time could be reduced to minimal [18].

The access history for various data objects is collected into a statistical data form, and the repeatedly used files are replicated in advance. The replication mechanism decides which files to replicate, when to replicate, and where to put new files replicas. The static and dynamic replication systems are the two terms used to classify replica placement mechanisms in a DG platform.

A replication mechanism is said to be static, if the duplicate data objects are placed statically within the system. In other words, if more space is needed, or if such data objects become obsolete, then the files have to be manually evicted from the system. Static replication system is not suitable in a dynamic environment, such as the DGF systems, where users exhibit certain degree of unpredictable behaviour, which will consequently undermine the potentials of data replication systems.

In contrast, a dynamic replication system duplicates data objects as well as evicts them automatically, if the need arises or when the data objects become obsolete in the system. Data objects could be obsolete in the system, if the users stopped patronising the files. However, other files may still need these files, even if the users stopped accessing them directly. Thus, file eviction should consider replica dependencies, prior to evicting files that are regarded as obsolete. The proposed DRCEM mechanism ensures the benefits of replication are not tempered with despite the possible changes in users' behaviours to form the popular data [60], [85]. Assessment of popular data file can be done by considering file access by users or relevance of the data file to other files in the DGF system. Dynamic replication promotes file popularity, as well as load balancing by distributing file replicas to lightly or moderately loaded sites within a given region. The benefits of data replication cannot be over-estimated. When data are replicated on sites closer to clients, bandwidth consumption is drastically reduced. Also, in a distributed environment such as the DGF environment, the number of replica sites could be increased to boost up data availability and improve system performance.

Where relevant literature is not readily available on data replication in DGF environment, the research falls back to data replication in the traditional DG systems, and explore the possibility of tailoring to suit the target environment. Besides, DGF is a collaboration of DG systems. Thus, a mechanism that works in DG environment will certainly work in DGF environment with some modifications.

2.7.2 Stages of Dynamic Replica Creation

Dynamic replica creation (DRC) encapsulates three main stages as follows:

a. The decision for replication: firstly, given the jobs type and data objects to operate on, then there is need to specify the method for replicating the data objects, in other words, whether to actually replicate or evict the files.

b. Determining the number of replica files: after deciding on which files to replicate or evict, the next stage is to find the required number of the affected files that need to be created or evicted from the system.

c. Placement of file replicas into appropriate locations: the mechanism for replication should determine the appropriate locations to put the new file replicas, or the locations from where to evict the old replicas.

2.7.3 What Triggers Data Replication in a DGF Environment

When a request for data objects is sent to a given site, but the data objects are not available on that particular site, the situation could trigger a replication process. This kind of strategy is also called an unconditional strategy, where every data request results to a replication process. Thus, along with replication, there is possible eviction of data objects. Among the popular policies used commonly in operating systems are the Least Recently Used (LRU) and the Least Frequently Used (LFU) mechanisms [9], [51], [86]. These mechanisms are used to evict data objects to claim space for new significant data objects. The above-mentioned strategies are also used in DG systems for evicting obsolete data objects [44], [50], [51], [52]. In the LRU mechanism, the required file replicas are obtained by the queried site. If space is not sufficient in the target location to hold the file replica, the mechanism will evaluate the existing data objects to determine the data objects that are least significant and evict such data objects.

2.7.4 Replica Optimisation Process

Replication helps to improve data locality, which may result to reduction in the time required for a job to execute in DG systems [87]. By replication, duplicate copies of data objects are stored at various locations within the local storage, for easy recovery of such data, in situations where the data objects are lost or unavailable. Additionally, replication tends to save sites bandwidth, which may reduce congestion when demand for data objects increases in the system. Nevertheless, despite its advantages, replication also has some issues. These issues could be triggered by factors such as the insufficiency of storage space at local sites, as well as the bandwidth between various sites [88]. Additionally, the files in a DG system are typically produced in larger sizes [69], [89]; consequently, placing file duplicates to each site and maintaining an indefinite number of duplicates is not a good practice, particularly in DGF platforms [10]. Hence finding the adequate (optimal) number of duplicates as well as finding the ideal location to host them is a vital process [90]. This process is referred to as the replica optimisation problem. Data optimisation and replica optimisation are synonymously used in the literature, to refer to the problems of optimising the number of replicas in DG systems [90].

In DGF systems, finding required number of replicas is equally as crucial as finding the locations for hosting the file replicas [11], where the sites are located far apart with large number of data objects. The way to formulate Replica Optimisation Process (ROP) involves amongst other things; architecture (topology) of the underlying system, users' locations, users' requests for popular data objects, and ideal locations suitable for hosting file replicas. The objective of ROP could be centred on the minimisation of certain costs regarding access to data objects, storage space and file transfer from source to destination sites. The individual cost or a combination of more than one cost is thus minimised or maximised, which depends on the laid down objectives of the replication mechanism. For instance, the cost due to communications, the cost due to extreme storage element usage, the cost due to infinite number of duplicate files, and the cost due to maximum number of locations to host duplicate files. Literarily, these costs are respectively refereed to as *read cost, storage cost, replica placement cost* and *site cost*. Various studies have attempted to address the various costs related to replica optimisation in DG systems [27], [42],

[91], [92]. In the past, some researchers proposed mechanisms that distribute file replicas to locations where the read cost (RC) is minimal [40], [42], [54], [92], which minimised the time required to transfer data objects over the network to requesting clients. Regarding the replica placement cost (RPC), some researchers considered hosting data objects on storage locations that optimise storage usage in the system [27], [93], [94], [95].

The storage cost is synonymously refereed to as the file size, the site throughput, or the fact that a file duplicate resides at an ideal location, which also means cost of placing replica files [90]. Associated to read cost is the access cost, which refers to the duration it takes to access data objects, stored in a replicated site [34]. In this thesis, an enhanced scheme for replica placement cost is integrated in the proposed DRCEM mechanism, which considered, access cost (storage cost) and site distance in finding the best locations where to put file replicas within a federated DG system.

1. Replication Benefits

Some of the crucial benefits of data replication are:

- Improves performance: Replication can enhance the system’s performance, regarding jobs completion times and bandwidth consumption. Data files could be placed at sites locations that are closer to the users’ access points. Therefore, response time, file transfer time and the overhead will reduce, thereby improving on the jobs completion times. Thus, users can access data items within a given region at the same degree of response time, if essential file replicas are duplicated evenly over the network via replication.
- Enables balancing of sites workloads within the DGF regions: Replication can offer load balancing between the regional sites, in such a way that same data

items could be simultaneously served by multiple sites. Therefore, this gesture will minimise the burden on the originating sites, where the main data items are been hosted.

However, despite its numerous advantages, data replication in DGF systems is constrained by issues such as the storage resources usage, sites workloads and bandwidth consumption. Data replication mechanisms should consider some significant factors to control the cost of replication. The factors impact directly on the systems' performance. Thus, excessive use of replication is not recommended, to conserve DGF network and other resources. The next subsection discusses the factors for data replication.

2. Replication Factors

The following essential factors should be considered when developing a replication mechanism in DGF systems [8], [26]:

- Which data file should be reproduced?

  Most of the existing replication mechanisms select the files for replication using popularity of the stored data files as a measure for the selection. A common method for assessing popularity of data files on a given storage element is to calculate the total requests made to the data files over a specified period. Other mechanisms replicate only shared files or files that are very rare in the system.

- Which sites should hold the replica files?

  The decision on where to place a certain file replica is very crucial. Essential copies of files replicas need to be stored on storage facilities closer to sites that may request such files in the future, so that delays due to searching and downloading of the files will be minimised. Moreover, some dynamic characteristics of DGF sites, such as workloads, storage availability and site failures, should be considered while placing files replicas. Thus, mechanism for improving data availability should measure the site availability or failure prior to placing new file replicas in the DGF system: If an essential file replica is stored on a site, which has a low degree of availability (frequent failures), there is need to look for an alternative file replica to preserve the file's desired availability.

- When and how replication is to be carried out?

  It is crucial to decide between static, eventful or periodic replication: The trigger for replication is equally important, while contemplating on when to perform replication. A replica is said to be static if it is unmanaged, which means that the content is not changed through time. A dynamic replication takes place either eventfully or periodically, depending on what triggers it. Eventful replication occurs at the time of data request. Periodic replication occurs at the background, at certain interval, to balance the data availability of the system. Static replication is not an option in DGF systems.

- What file should be evicted and from which sites?

  Since eviction removes files permanently from the system, caution is needed not to evict files with links to other files. In other words, file eviction should not be guided by users' frequent access only, but also how frequent a file is accessed by other files within the system. If a certain data file is not directly accessible by a user, but is always invoked when a user accessed another file, which provides partial service to the file accessed by the user, deleting such partial service provider may not be in the best interest of replica eviction decision.

2.7.5 Availability of Data and Replica Sites

Availability refers to capability to deliver suitable services, despite underlying constraints in the system. This situation is otherwise, tagged as the readiness for correct service delivery [58]. The lifespan of a Peer-to-Peer connection in DG systems is classified into a set of 'up states' and 'down states' [68]. Furthermore, data availability means having the required data accessible at all times, by users and the critical applications [17]. Also, availability is the condition wherein consumers can persistently access a given resource. Therefore, in DGF environment, availability means that if a data item is available, then the users including applications can have constant access to the data items. Any condition that renders the data resource inaccessible causes the opposite of availability, otherwise known as "unavailability". Replica availability is the ability of a file replica to provide proper services to the users, despite certain underlying constraints. The availability of a replica can be drastically affected by the frequent failures of the host site. In other words, it is worthwhile to consider replica availability alongside site availability.

In [19], the researchers opined that a site in a federated DG platform is less consistent in its behaviour, even in a dedicated server-based federation system [19]. That is because the central servers only control specific behaviours, such as access to centralized billing software or a centralised database. However, control over characteristic sites behaviours regarding disconnecting and reconnecting back to the federation is very limited. This is simply due to fact that the user, who controls the site, can shut down the system or application at any moment or uninstall it permanently. Thus, a reasonable solution to deal with sites failures is for replication mechanism to find best sites where to place replicas, that is, sites with less unavailability measures. Although site availability has been addressed previously by research in [19], its application in replica placement was however overlooked by modern dynamic replication mechanisms.

Sites availability can be affected by the period considered as “valid availability”. In addition, repair times, insufficient bandwidth; time differences (due to geographical locations) could lead to sites availability issues, which need to be considered while contemplating on sites availability measurements. In these regards, this research does not explicitly differentiate between these variations regarding availability. Thus, availability based on the Telecommunications industry interpretations reported by authors in [19], is the degree of operability of a system or sub-system, which is determined from the ratio of total time of operation during a given interval, to the length of the measured time interval.

2.7.6 The Concept of Replica Dependency

As explained by research in [61], replica dependencies are considered as the type of implicit relationships typical of interactions between software objects or artifacts that evolved together over a specified period. Direct logical dependencies (DLD) are defined for pairs of files in an association rule of the form $F_1 \Rightarrow F_2$, meaning that when F_1 occurs, F_2 also occurs. In this notation, F_1 and F_2 are two disjoint sets of items. Furthermore, F_1 and F_2 are called the antecedent left-hand-side, (LHS) and the consequent right-hand-side, (RHS) of the rule, respectively. In software development process, the density of dependencies amongst sites increases the likelihood of synchronization failures, as argued by research in [96]. Based on this notion, the work of researchers in

[97] proposed a more comprehensive measure based on the DLD concept, called clustering of logical dependencies (CLD). Unlike the DLD, the CLD measure encapsulates the degree to which the files that have direct logical dependencies to the given file F_i, may have indirect logical dependencies (ILD) among themselves. Thus, in this thesis, the indirect logical dependencies (ILDs) amongst replica files are considered in addition to DLDs for proper file evaluation. Computation of Logical dependencies of file replicas are dealt with in Chapter Four (Section 4.5.6, page 165).

2.7.7 Concept of Grid Jobs and Job Schedules

Although this thesis is concerned about data replication, it is pertinent to discuss the concept of grid jobs, and job scheduling briefly. Grid job scheduling has been described by authors in [39] as the process of scheduling jobs to specific available physical resources, trying to minimize a given cost function specified by the user. In a simple DG installation, users may submit their jobs directly to a machine suitable for running the jobs. However, the larger systems, such as the DGF platforms, would usually include a robust job scheduler for mapping various jobs onto the grid environment. According to the researchers in [98], different resources types are available on the grid for collaborative purposes. These resources are typically accessed from applications that serve as an interface to the users. The term application refers to the highest piece of the task within the DG system. In some situations, however, the term job is used synonymously to refer to as application. An application may be broken down into any quantity of individual jobs. In a typical grid environment, jobs might require the outputs from other jobs to execute. Figure 2.6 , shows a concept of grid's jobs scheduling.

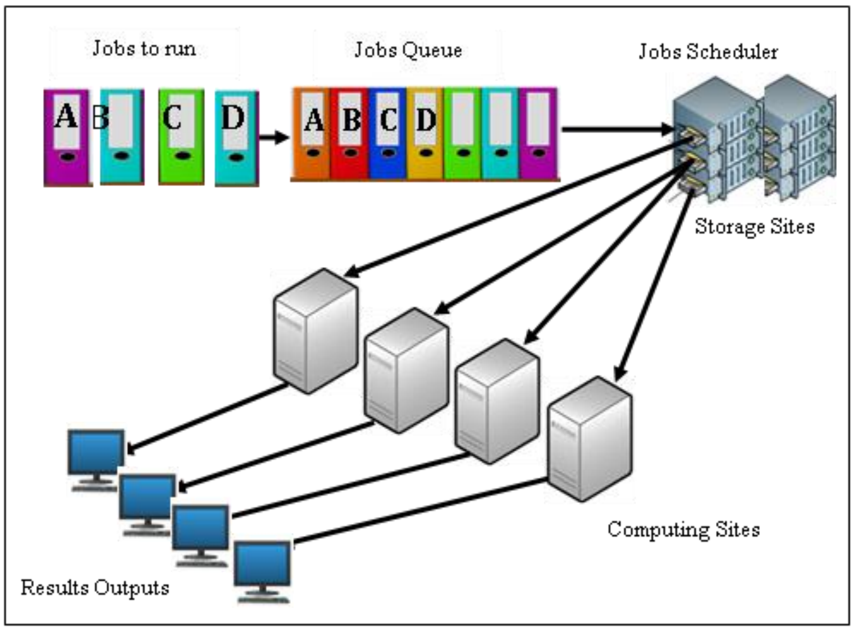


Figure 2.6. A Concept of grid's jobs scheduling [98]

The terms such as transactions, work units or submissions are readily used by the grid industry to depict the concepts of grid jobs. Jobs could also mean the same thing as programs that are run to accomplish some tasks in the grid environment. The jobs may perform computational tasks, run one or multiple system-specific commands, perform data operations, or control machinery.

Some jobs may have high data affinity compared to others. The reason manifests in the requirements for extensive data objects as well as numerous file replicas that are needed by some jobs, which differentiate the jobs with compute intensive jobs. In addition, these replicas are distributed as well as situated over regions that are globally scattered. The schedulers need to consider bandwidth and transfer time amongst computational sites while retrieving data objects from storage facilities [83], [99]. Therefore, mechanism for scheduling grid jobs should be aware of replication mechanism that is closer to the computational sites, for improved system's performance.

Grid jobs may have explicit dependencies that could inhibit them from running in parallel. The situation requires some form of data duplication. For instance, some data items may have to be copied onto the target machine, on which the jobs will run. Jobs may require additional sub-jobs, depending on the range of data they process. These dependency behaviours of data files can influence how data are evicted from the grid environment as well as the popularity of the replica files.

2.8 Related Work on Data Replication

The core of this thesis is to develop an enhanced dynamic data replication and eviction mechanism for improving the performance of DGF systems, in terms of jobs times, network usage, storage and computing element usage. This section discusses the relevant literature, based on which the research goals are formulated. That is, the section looks at critical, relevant studies, and thereby helps in justifying the gaps in the chosen research area. Various researchers made some efforts in trying to improve data locality and availability within both the conventional DG environment and its federations. The replication mechanisms are affected by the underlying grid topology, whether centralized, hierarchical and decentralized or federation systems [69]. Thus, the basic difference between DG and DGF systems regarding data replication mechanisms is the way these mechanisms handle replica placement decision and replica management regarding replica eviction [50]. Some mechanisms consider replica placement and management on the regions of the DGF systems [87], as against the entire federation system [69]. In addition, while some research works concentrate on popularity-based data replication mechanisms, others worked on availability-based replication mechanisms. These are critically examined in Subsection 2.8.1 and 2.8.2, respectively.

Researchers in [87], proposed a data replication mechanism for addressing the problem of data availability within DGF environment based on files' access history within regions with high bandwidth concentration. The mechanism improves access latency and remote site access, by selecting a file for replication using access history. However, restricting data access to sites with high bandwidth may overburden the system, and hence results in reducing the overall system performance.

Data availability within DGF regions was considered by the research in [50] aimed at improving data locality by proposing a mechanism, which was called Least Access Lowest Weight (LALW). The LALW mechanism duplicates data objects based on access frequency with the aim of improving upon access time and transfer time. However, even though the mechanism identified the region within which to place file replicas; it falls short of specifically determining the ideal locations within the regions, which are suitable for hosting the duplicated data objects. Also, the replicating mechanism failed to determine sites distances, which will have a drastic effect on jobs completion time. The other shortcoming is that LALW failed to consider logical dependencies amongst the replica files, which is a crucial parameter in file evaluation for determining file popularity and its worth in relation to users and other files. In addition, the mechanism did not consider response time, which affects the decision for selecting file replicas, as well as the jobs completion time.

In [26], the author proposed a replica creation algorithm for DG systems (DRCM), by capitalizing on the shortfalls of the LALW mechanism. The replica selection decision was based on files with high weight or rate of growth based on the LALW mechanism [50]. The mechanism aimed to minimize network bandwidth consumption and storage cost. However, it failed to find the availability of the site to hold file replica and did not consider response time as well. Another issue is that DRCM did not properly address the issue of file dependability, as it only considers direct logical dependability, ignoring indirect (clustered) logical dependability between replica files.

Also, in [54], the researchers proposed a GA-Based replica placement mechanism for DG systems to address the issue of management of large data files in DG environment that provides massive data resources across geographically distributed systems. Their work aimed at improving data resource sharing capability via replication. The mechanism identifies a suitable location to place file replicas, using five design metrics (read cost, storage cost, sites' workload, and replication site). However, the mechanism failed to consider resource failure, site distance and inter-dependability amongst replica files. In [8], the researchers proposed an enhancement of the LALW mechanism, which was referred to as the "Enhanced LALW" mechanism or ELALW. The mechanism selects a file to replicate by considering how many times the file would be requested in the near future, which was based on the work of researchers in [50]. ELALW is critically analysed in Subsection 2.8.1 of this chapter, since its one of the core reference work in this thesis.

The researchers in [38] proposed an agent-based dynamic algorithm for replication in hierarchical DGF systems. The mechanism was anchored around a central master site, and various regions controlled by static header sites. The mechanism selects best replica site based on bandwidth capacity, load capacity, and computing capacity of the site. One pressing issue in this approach is that replication within the various regions is controlled by static headers. If a header fails, the complete replicas within that regions will not be accessible. This situation requires dynamic headers within the various regions, which will minimise rate of failures in the system. The dynamic headers can be created by storing index information within the different sites of the regions, and a dynamic header selection algorithm, so that once the current header sites stop responding, one of the neighbouring sites could be selected as the header site. This thesis assumes that there is dynamic header selection mechanism within the DGF system. Thus, it is not a concern in this research. Further, data replicas are stored at the header sites. This may not provide the required data availability to serve the numerous users. Also, file dependability was not considered while evaluating popular file for replication. The replica placement decision also failed to address the issue of distance between replica sites as well as site availability, while determining the suitable sites to host new replicas.

The work of researchers in [100] would have resolved the problem of static headers by proposing a mechanism that implements dynamic headers within the regions. However, the mechanism creates another problem similar to [87] in that, replica placement decision is based on sites with high bandwidth concentration. Also, the mechanism did not consider the distance between replica site and the requesting site while placing replicas. Thus, the region with high bandwidth can be put far away from the requesting client, and may not be available all the times. Another shortcoming is that the replica creation decision does not consider periodic replication; it is based on the event of a request for a file coming from clients. Also, replica placement based on high bandwidth region may have the adverse tendency of creating bottlenecks as the number of inter-cluster hits for region increases. Also, the distance of a region from requesting a client with high bandwidth may impede on access time and job completion time. Also, the site availability may be another issue, if not evaluated before placing data replicas. In [101], the writers proposed algorithms for replica cost optimisation in DG systems of tree network architecture, which are constrained by quality of service (QoS) as well as network bandwidth costs. The algorithms' primary objective was to minimise the cost of replication, which includes additional costs due to communications and storage usage, while the anticipated user-oriented QoS regarding users' distance from the server machine is assured. The authours considered bandwidth limitation as a global QoS regarding the network, which can impact on the DG systems' users. Additionally, the authours evaluated the heuristic algorithms regarding replication cost, network bandwidth usage, and data availability. The afore-mentioned variables are considered significant for evaluating DG systems' performances. Although the work in [101] fares well regarding replication cost, bandwidth usage, and data availability, yet there is a need for improvement, regarding site availability, site distance and inter-dependability of replica files. The next subsection discussed relevant works concerning popularity-based data replication mechanisms, to clarify gaps in the existing literature further.

2.8.1 Popularity Based Data Replication Mechanisms

Central to replicating data file is to improve data locality and availability by maintaining numerous duplicates of the data objects at strategic locations within the DG sites [3], [69]. Many mechanisms for replication have been proposed for Peer-to-Peer DGF systems by different researchers, notable amongst which include researches in [102], [103], [104] for improving data availability and reliability within the systems. Replicas can synchronize with other replicas within the regional sites. However, to decide on improving data availability, the importance or relevance of such data objects needs to be established. The idea of file importance brings about the concept of file popularity. Many researchers attempted to address the issue of file popularity in DG systems [8], [38], [40], [44], [51], some of which may require modifications for deployments in DGF platforms. Notable amongst these approaches are discussed in the following subsections.

According to researchers in [40], DG platforms deal with an enormous volume of data on a regular basis. Thus, the authors proposed a popularity-based replication mechanism, which computes an appropriate number of replica copies and determines the DG sites for replica placement. The authours named the mechanism as Latest Access Largest Weight (LALW) mechanism. The mechanism correlates each past access to data objects with a different weight, which helps to determine the significance of each data objects in the system over certain period. A data object with more recent access has a larger weight, which signifies that the data object is more relevant to the system's users.

The authours used OptorSim simulator for evaluating the system'sperformance. The results of simulation show that LALW effectively enhances the ENU. In effect, the LALW approach finds popular files and duplicate them to suitable regions, without adding too much burden on the system's network. Although the LALW mechanism determines the region where the replica needs to be placed as well as the number of file replicas to be created, it failed to determine the appropriate sites within the regions where the file replica should be placed. Thus, this approach may need some modifications for deployment in DGF environment, since detemining the appropriate regional sites suitable for replica placement is a core requirment in DGF systems.

In [8], the researchers proposed an enhancement of the LALW mechanism [40], which was referred to as the “Enhanced LALW” mechanism or ELALW. The mechanism selects a file to replicate by considering how many times the file would be requested in the near future, which was based on the work of researchers in [50]. Also, the mechanism makes replica placement decision by considering factors such as replica size, replica copies, site with least transfer time, awaiting requests for file replicas as well as the storage usage.

Further, the ELALW mechanism aimed to improve jobs time, network bandwidth and storage element usage. Simulation results from OptorSim showed that the mechanism fares well regarding the measureable metrics compared to other mechanisms. A drawback with the ELALW mechanism is that the cost of replication due to the distance of replica site from requesting site was not adequately addressed. As a resolve, there is need to modify the mechanism to determine not only the distance between requesting site, but also the percentage availability of the sites to hold the file replicas. This mechanism could be enhanced by this research to locate and evict the unpopular files for replica placement decision. The enhancements will focus on finding the network distance between replica site and requesting site, file dependability measure as well as site availability.

The study conducted by the work in [26], portrayed a common method used to improve the performance of data access in distributed systems. The study examined some algorithms for data replication that have been proposed by other researchers. Also, the study proposed a dynamic replica creation algorithm using the concepts of exponential decay/growth based on the LALW mechanism [40]. The main contribution of that reserch was to find popular files for replication based on access history, using the concept of exponential decay/growth. The theoretical ascertion of exponential growth/decay entails that the rate of growth is proportional to the current file size, as well as the access history of the file.

The theory is popular considering that each file's popularity increases with an increase in its' access rate and decreases by the decrease in its' access rate. The popular file is determined by taking the average after totalling the various file's access rates at different time intervals. The mechanism was compared with the popular LALW mechanism and yielded similar results as the LALW mechanism. Thus, the exponential decay/growth theoritical framework could be enhanced to evaluate popular files by considering logical dependencies along with the file's access rates, for proper file evaluation.

In [38], an Efficient Dynamic Replication Algorithm (ERDA), which uses agent for DG systems, was proposed. The algorithm resolves the problem of centralized arrangement of sites within clusters, by enhancing the replica placement strategy in [69]. The EDRA mechanism was implemented in a hierarchical DG system and selects best replica sites based on the network bandwidth, sites workloads, and storage capacity. Sites workload helps in balancing the loads on the DG sites, which is done by distributing loads evenly on the sites. EDRA was evaluated using OptorSim simulator, based on the CMS model. The performance was evaluated using access time, network usage and storage usage measureable metrics. The simulation results obtained were compared with BHR, LRU, No Replication, and EDRA strategy, which was tested using different jobs ranging from 100 to 500 jobs. The results showed an improved efficiency of EDRA mechanism regarding jobs time, network, and usage of storage facility. This approach has potentials in load balancing, but replica placement failed to consider other important variables such as the site distance and site availability. This mechanism could also be enhanced to achieve better performance in DGF systems, regarding load balancing.

2.8.2 Availability Based Data Replication Mechanisms

The distance covered by users to search for suitable data objects depends on how the replica files are distributed within the system [36]. Simply stated, increasing the number of duplicate files would lead to a corresponding increase in data availability within the system. However, due to the larger sizes of DGF data resources, it would be very expensive to store an infinite number of file duplicates. Therefore, it is vital to control the number of file duplicates to preserve storage space. In these regards, a replication mechanism, which improves data availability as well as system's performance, without excessive use of storage and bandwidth resources is mostly desirable. For instance, the work of researchers in [27] creates replica copies for maintaining a certain level of data availability within a Peer-to-Peer system, so that each site within the system is at liberty to duplicate data objects of interest. In that research, data availability was based mainly on the failure rate of the host sites. Thus, the authors developed scheme for increasing the number of duplicate file copies based the fact that each peer site could duplicate data objects of interest, based on an established system's threshold for data resource availability. Unfortunately however, the scheme failed to determine the exact number of file replicas needed to balance the storage usage, improve system's performance, as well as satisfies the excessive demands from the users. In addition, a scheme for replica eviction was not an integral part of the mechanism. Consequently, file duplicates will continue to accumulate in the system even though they become obsolate, thereby affects the replica placement cost. The next paragraphs examined critically, some of the notable research works regarding data avaiability in the traditional DG systems, that may be adopted in DGF platforms.

In [49], the researchers proposed a replication scheme, the aim of which was to ensure desired availability of data with minimum replication, without degrading system performances regarding load balancing, response time (Jobs times), and improving data availability. The researchers proposed a replica placement and replacement strategy that maintains desired data availability with minimum possible replication, despite the constraints of sites failures and sites loads within the federation. Their main contribution aims at a replica placement and replacement decision that takes into account the desired data availability, as well as sites stability and failures. The mechanism focuses on a hierarchical topology of federated clusters as shown in Figure 2.7. This topology is in use by several existing distributed systems, such as the Internet [23], [49].

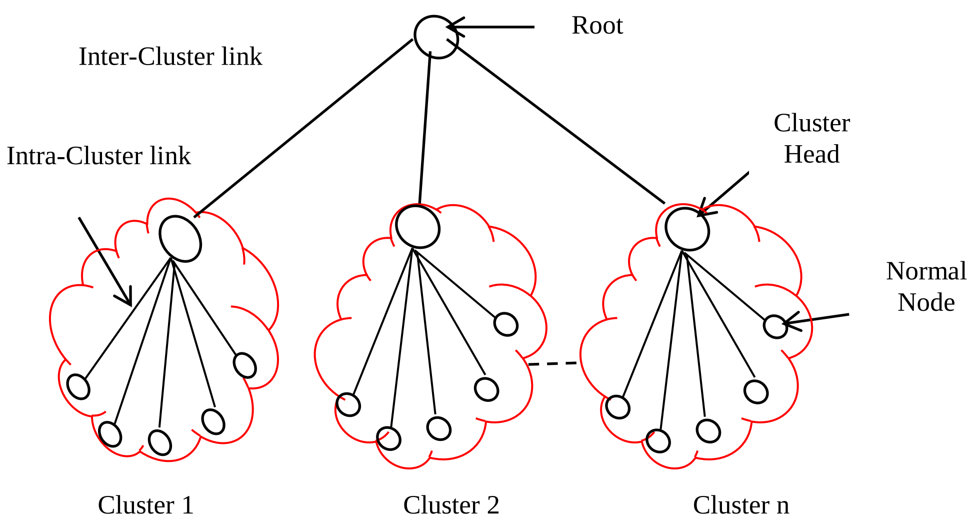


Figure 2.7. A DGF Architecture based on cluster federation topology

Their work covers placement of replicas in distributed systems and data grids, using a cluster federation topology, with a single root to the link between the various clusters. The topology does not have a dedicated server, but a root site (super site) that binds the various clusters together. Each cluster has a header called cluster-head (CH), which liaises with other clusters on behalf of members of the cluster. The header site manages the other sites via a routing table. Also, the header site contains metadata and information on file replicas that are in the cluster. The file replicas are stored on the storage elements within the cluster. In this model, sites have predictive behaviours, and if a failure is detected, the sites will be re-adjusted to keep the topology connected. The strength includes improving data availability with minimal replica placement. Also, sites availability is monitored by header sites to ensure stability. However, the availability relates more to the topology stabilization and ensuring an available number of replica files, considering the current workload of the sites. Thus, their work aims to optimise the number of replicas. However, the scheme fails to keep track of percentage site availability over a specific period, which will help to decide on where to place replica files for the benefit of both users and the federation system as a whole. In addition, workload was not entirely addressed, such that highly loaded site that offers lower replication cost should be made lightly or moderately loaded by replica eviction or placement. Thus, some of the good features could be improved for deployment into a DGF platform.

The work of researchers in [105], proposed a replica management solution to optimise files' replicas by reducing useless or unnecessary replicas. To this end, the researchers defined two mathematical frameworks, which determined the appropriate number of replicas to achieve a given level of performance, without compromising system's performance. The researchers observed that since every replica must perform all updates at some point in time, replica updates may create saturation point beyond which replication process does not increase system throughputs. Furthermore, if the number of replicas exceeds the optimum threshold, the unnecessary replicas would create an overhead as a result of additional message communications. Thus, the idea is to maintain minimum replica that can efficiently serve the needs of the DG users. Furthermore, the authors asserted that replication could tolerate failures to some extent, because if a replica site fails, another replica file can be used to replace it on another site. However, the number of replication must be monitored and adjusted concerning the failure frequency, for reducing the cost of replica consistency management. Then, given a rate of site failures, their goal was to estimate how much replicas are needed to ensure system availability. This differs significantly with determining percentage site availability for replica placement decisions. Instead of waiting for a replica file to fail and replace it with an active one, it is pertinent to take record of previous sites failures, determine sites with high percentage failures, then avoid such sites for replica placement. Such gesture will ensure all replicated files are placed on sites that are active almost all the times.

In [102], the work explained that efficient data sharing in global Peer-to-Peer systems is difficult due to the unpredictable site failure, insufficient bandwidth as well as unreliable network connectivity amongst the peer sites. Placing data replicas on various sites can improve data availability and enhances the response time. Thus, determining when and where to place file replica for satisfying the performance needs of numerous users, in large distributed systems, with unpredictable user-behavior and dynamic network characteristics, is a difficult task. The researchers proposed an approach, in which peer sites create replicas automatically in a decentralized fashion, as required to meeting availability goals. The framework aimed at maintaining a certain level of data availability in the system. The model was evaluated using simulations, and the results showed that the strategy could be used for determining the required number of file replicas in the system. However, if data replicas should be created in every DGF site, the storage cost will escalate beyond desirable limits.

In [106], the work presents a survey of recent dynamic data replication strategies. The authors studied as well as classified the strategies according to the underlying DG platforms, in which the strategies were deployed. Also, the authors discussed the strategies based on some crucial metrics, which include jobs times, access time, storage and bandwidth management. Furthermore, the impact of DG architecture on dynamic replication performance was investigated in a simulation environment. Furthermore, the writers highlighted some critical issues and open research problems on dynamic replication in DG systems.

The authors also studied some factors that influence the performance of DG systems in general. Considering the dynamic nature of the DG systems, sites can join or leave the system at any time. As a result, the number of active sites at any given time may vary. Thus, dynamic replication strategies should consider these dynamic aspects of the DG systems, as well as consider how files are accessed by the users. Also, mechanisms for replication should consider storage usage, and bandwidth consumption. Consequently, the benefits of replication should always outweigh the overheads.

2.8.3 Replica Placement Mechanisms

According to researchers in [89], replica placement is the process of identifying where to place copies of replicated data files within a DG system. In addition, transferring a data file from a site to a client consumes an amount of bandwidth. A significant issue here entails where to put the new file replicas to minimize the amount of bandwidth used [89], which means the storage is of great importance when it comes to replica placement [26]. The cost of transferring a file from the underlying site to the other locations is referred to as the replica placement cost (RPC), which is determined by the file access cost multiplied by the distance between replica sites [26]. The next paragraphs highlight some existing mechanisms that addressed crucial issues regarding replica placement in DG systems.

Regarding the issue of bandwidth, the researchers in [87] proposed a replica placement approach, to minimize access time by going round network congestion. The apparoach was titled: Bandwidth Hierarchy based Replication (BHR). The BHR minimizes the required time for accessing and transferring the file replica. The mechanism places a replica at a location of high bandwidth concentration. But, such method considers only the transfer costs of the underlying file replica, and does not guarantee to minimise the overall replication cost.

The authors in [107] proposed a replica placement approach with load balancing capabilities. The approach places the most frequently accessed file closer to DG users and makes replication decision by considering the access load and the storage load of the candidate’s replica servers as well as their sibling sites. The strategy does not consider site distance as well file request to other files. In other words, the strategy ignored the possibility of logical dependencies amongst replica files.

The authors in [27] proposed an algorithm that can evaluate the replica placement in multi-tier DG systems. The aim was to maximise the gap between the cost of replication and the replication benefits, which are determined by the storage cost and the transfer time cost. The benefit is accorded to the users by the decrease in a transfer time off their jobs. The transfer time is the period for transferring from the current location to the new location. Again, replica eviction mechanism was not incorporated. Thus the “storage cost” may escalate due to files accumulations.

In [59], the researchers proposed a dynamic replica placement scheme that takes into considerations the dynamic nature of replica sites in the DG environment. The dynamic nature of DG sites entails leaving the grid and possibly joining again at a later time. Thus, the work investigated two parameters. The first is the number of request for each file by the neighbouring sites. The second is the effectiveness of each site involving the number of times the site failed to respond to a file request due to its absence from the grid. This approach made efforts regarding replica availability but failed to consider the site availability as a function of replica placement decision making.

The researchers in [8] proposed a replica placement mechanism that places the popular file to a suitable site by considering the access frequencies for each replica file. Access frequency is considered as an essential factor that should be taken into consideration while deciding on replica placement. However, some critical factors such as the overall replication cost (storage cost and read cost), site distance and site availability, should also be considered [19].

2.8.4 Replica Management Stage

At this stage, the replicas have already been distributed to different locations based on the described replica creation mechanisms. If multiple replicas exist, a replica management service is required [84], [101]. Replica Management Service (RMS) discovers the available replicas and selects the best replica that matches the user's quality of service requirements, and then adjusts the location of those replicas. Thus, there are two main phases in the Replica Management Stage (RMS); namely, Replica Selection (RS) and Replica Maintenance (RM), and each of which will be discussed briefly in the next subsections.

2.8.5 Replica Selection

In a typical grid environment, replication systems create multiple copies of the same data file and distribute these copies (replicas) to different site locations. These site locations vary in their capabilities, resources, and network. Thus, there is a significant difference in selecting a replica location, amongst many locations that are widely distributed [42]. The replica selection strategy is the process of choosing a replica from among those spreading across the grid sites based on some characteristics functions [42], [52].

The big challenge of any replica selection strategy is defining the appropriate criteria to determine the best replica location, and the selection algorithm used for replica selection strategy. One common replica selection criteria reported by existing literature is the response time [29], however, replica selection should consider site distance as well [8].

2.8.6 Replica Maintenance

According to research in [42], the candidate site that holds replicas may currently not be the best sites to look for replicas in subsequent periods, due to the dynamic nature of DG systems. Therefore, replica maintenance needs to consider site parameters, such as availability, distance and workload. If a particular site metric degrades in its performances, the maintenance services should update replica location services, or consider moving the file to a preferable site. For instance, changes in site bandwidth, the distance between replica sites, or resources failure. In either case, replica maintenance needs to dynamically move valuable replicas to sites with favorable performances. Although, replica maintenance service is not in the scope of this thesis, its mention is included here to help understand the importance of the performance metrics considered by this thesis.

The authors in [89] proposed a dynamic replica maintenance strategy called Dynamic Maintenance Service (DMS) to improve the performance of the DG environment. DMS decides where to place the replicas based on two main parameters: request frequency and free storage space. However, the replica eviction scheme is not considered; instead, the system does not locate the replica at a site unless there is enough storage space even if it stands to benefit the overall system performance. Concerning changes in the distance between replica sites, the replica maintenance phase will adjust/move important replicas to the appropriate location based on the information collected relating to some effect factors [89] [108]. Replicas should be adjusted to the appropriate locations that are closer to the computing devices to adapt the current network environment to reduce time when the computing device accesses the data, as well as to maintain optimal performance of the network environment [108]. If on the other hand, the network environment is changed, which makes the same replica sites not always being the best choice to download data while reducing transmission time. Therefore, according to research in [89], replicas should be adjusted to the appropriate locations for achieving optimal access times. Based on the literature review in this chapter, it was concluded that there are many issues regarding the current replica placement mechanisms in DGF systems, and there is a need to enhance these mechanisms in various aspects.

Obviously, to get benefits from replication mechanism, the file transfer time, bandwidth and storage costs need to be minimised. From the literature, it was observed that there was deficiency in current replication mechanisms for the efficient management of DGF resource usage, as most of the mechanisms focused on the traditional DG systems. However, in the absence of many replication mechanisms devoted to DGF system, this thesis revamped to related mechanisms in the traditional DG systems. Table 2.1 gives parameter wise summary of some of the related approaches critically reviewed by this study.

Table 2.1

Summary of strengths and weaknesses of related literature in data replication

Approach	**Topology**	**Replication policy**	**Metrics**	**Simulator**	**Strengths**	**Weaknesses**
Dynamic data grid replication strategy based on internet hierarchy [87]	Tree-level hierarchical Data Grid	Bandwidth hierarchy replication	Access time effective network usage	OptorSim	Reduces access time and bandwidth consumption	Search for the required file is limited to sites with high bandwidth
A dynamic data replication strategy using access-weights in data grids [8]	Hierarchical Data Grid	Latest access largest weight	Network usage, storage usage, jobs times	OptorSim	Increases efficient network usage.	Failed to address resources failure, node distance, file dependability

Table 2.1 continued.

Implementing data placement strategies for the cms experiment [109]	CMS Data Grid	File popularity	Access latency, network utilization	OptorSim	Minimizes pre-placement of data and automatic replication of hot datasets	Did not address resource constraints, such storage capacity while automatic replication
A GA-Based Replica Placement Mechanism for Data Grid [54]	Hierarchical Data Grid	Replicates popular files based on file value and weights	Access time, storage network usage, usage	OptorSim	Works on both dependent and independent files	Failed to address resource failures, site distance and file inter-dependability
Managing data replication and placement based on availability [83]	Cluster federation with header sites to link between clusters	Takes into account data availability based on host sites stability	Access time, network distance	FTSim simulator	Improves data availability with minimal replica placement, Ensures site stability.	Availability related only to topology stabilization, not linked to replica placement
Efficient Dynamic Replication algorithm using agent for Data Grids [38]	Hierarchical Data Grid Federation (Central node, Regions Headers)	Agent-based replication	Mean job time, network usage, storage usage	OptorSim	Finds the region where place file replica	Not consider site distance, storage latency & site availability
Dynamic Data Grid Replication Algorithm Based on Weight and Cost of Replica [41]	Hierarchical Data Grid	Lowest Weight and Lowest Cost	Job times	OptorSim	Processes jobs faster than DHRA algorithm by 33%	Failed to address storage usage, network usage, resource failures, node distance
Dynamic data storage & placement system based on category & popularity [51]	Hierarchical Data Grid	Replication based on File category and popularity	Tob time, network usage, and storage usage	OptorSim	Better memory utilization than HDFS	Did not consider site failures, file dependencies, replica consistency

Table 2.1 continued.

Exploiting cms data popularity to model the evolution of data management for run-2 and beyond [62]	Based on EU Data Grid	Future data predictions based on file popularity	Network utilization, storage usage, job throughput	OptorSim	CMS analytics: collecting /transforming data, and predicts future data	Storage for storing the analysis data is constraints , thus stores data externally.
RPLB: A replica placement algorithm in data grid with load balancing [110]	Based on EU Data Grid	Replica placement based on highest degree and the highest frequency	Access time, Storage usage, job throughput	OptorSim	generates less number of replicas, achieves load balancing	Not consider: node distance, storage latency & node availability
Implementation of sub-grid-federation model for performance improvement in federated data grid [100]	DG Federation	Network core area	Network consumption, access time	OptorSim	Locates best replicas at sites with the highest bandwidth, fewer remotes sites access	Did not consider: File dependencies, Node distance, site availability
Towards efficient location and placement of dynamic replicas for geo-distributed data stores [6]	Geo-distributed data stores	Identifies popular data in the cluster, and replicates closer to the clients	Memory usage, read-latency, error rate	Synthetic simulator	Allows users to locate closest data copy with minimal overhead	Automatic placement of popular data, overlooks possibility of resource failures
Data replication approach with consistency guarantee [18]	Based on EU Data Grid	Replica region selection, placement, & update	Network usage, response time, system load & system availability	Gridsim	Performs ok when all the sites have facility to hold data copies	Failed to address storage constraints

The table explicitly indicates the existing approaches and their respective strengths and weaknesses in data replication related tasks. It also indicates the replication policy adopted by the existing mechanisms, the grid topology along with the measurable metrics. Furthermore, the table reveals the type of simulation environment suitable for evaluating the performance of data replication mechanisms. The outcomes of the literature review as summarised in the Table 2.1, are based on the related works in both DG and DGF systems. In addition, it was apparent that the existing mechanisms kept ignoring resources failures, site workloads, as well as inter-dependencies amongst replica files, while deciding on where to place file replicas. Thus, more enhancements needs to be done on the existing mechanisms to address the goal of minimising replica placement cost, without degrading system performance, regarding jobs times and bandwidth usage. Thus, the table serves as input as well as reference data feed in the research design section of Chapter 3 (Section 3.2: Research Design). The thesis selects relevant mechanisms from existing literature in DG system that could serve for DGF system, since the latter is formed by joining various units of the former. In the next section, this thesis discusses the DGF simulations tools for more details.

2.9 Data Grid Federation Simulation Tools

This study has conducted an extensive analysis of some of the widely used grid simulators available in the market. These will be deliberated upon in this section. The authors observed that simulating DGF environment entails looking at the DGF system regarding regions, with each region representing an individual DG system. Thus, this research looked into the simulators used for the individual DG systems, and selects the most popular amongst them. Sequel to the dynamic behaviours of distributed systems, a significant challenge to any simulation process is the validity question: does a simulation shows what would happen in the real testbed environment? Apparently, there seems to be no ideal simulation tool for any broad scientific area [111]. This is one reason there exist many simulators each focused on a specific aspect, which makes validation and verification possible [111]. Indeed, it is not wrong to say that grid installations are becoming the prevalent and most complicated distributed systems worldwide. Thus, it is not surprising there exist numerous different simulation tools [111].

Various grid simulation tools have been developed over the years based on the type of application run on the target grid platform. Prominent among them are: OptorSim [37]; Bricks [112]; ChicagoSim [73]; EDGSim [73]; GangSim [113]; GridNet [73]; GridSim [114]; SimJava [73]; HuskySim [73]; MicroGrid [115]; PlanetLab [73]; Emulab [73]; SimGrid [115]; and SimBOINC [116]. The next sub-section compares the various simulation tools based on design motivations, grid platform and support for replication and scheduling mechanisms, which facilitates the choice of OptorSim simulator for performance evaluation in this thesis.

2.9.1 Comparisons of Various Grid Simulation Tools

In this subsection, the research examines critically the various types of simulation tools based on their design motivations, grid platform usage and support for replication and scheduling optimisations. This gesture helps in choosing the appropriate simulation tool for performance evaluation in this thesis, as will be discussed in Chapter Three (Section 3.6.1 page 106) and Chapter Five (page 196). Table 2.2 gives summary of comparisons between various grids simulators based on the afore-mentioned features.

Table 2.2

Comparisons of various grid simulation tools

Simulator	**Design Motivation**	**Grid Platform**	**Suport for Replication & Scheduling Mechanisms**
OptorSim [37]	A discrete event simulator for testing dynamic scheduling and replication mechanisms developed in the framework of the European Data Grid (EDG) project used for optimizing grid performance.	Data Grids and Computational Grids	Data replication and Resource scheduling
Bricks [112]	Resource Scheduling tool for computational environment.	Computational and Data Grids	Scheduling
ChicagoSim [73]	Models computations, network and application behavior of computational grids.	Computational Grids	
EDGSim [73]	A discrete event simulation for high energy physics data analysis of the European Data Grid project. Support for data replication.	Comoutational and Data Grids	Data replication
GangSim [113]	A scheduling algorithm simulator for large systems comprising of hundreds thousands computers and storage systems.	Computational Grids	Scheduling
GridNet [73]	A simulation tool for dynamic data replication mechanisms developed to evaluate the performance of different scalable distributed systems that uses replica placement to meet the need of a large numbers of users who continuously change their data demands.	Data Grids and Computational grids	Data replication
GridSim [114]	A discrete events simulation of scheduling mechanism for large scale resources distributed systems.	Comoutational and Data Grids	Scheduling
SimJava [73]	A discrete events simulation of scheduling mechanism for large scale resources distributed systems.	Comoutational and Data Grids	Scheduling
HuskySim [73]	A discrete event grid simulator, enables simulation of both static and dynamic resource scheduling mechanisms	Computational Grids	Scheduling
MicroGrid [115]	An emulation tool that provides a vehicle for scientific study of grid topologies and applications. It emulates computations and applications performances.	Computational Grids	Scheduling
PlanetLab [73]	Platform for emulating Network, CPUs and applications services.	Computational Grids	Scheduling

Table 2.2 continued.

EmuLab [73]	Platform for emulating network topologies in distributed systems.	Distributed Systems	Scheduling
SimGrid [115]	Simulates networks and computational resources for large-scale distributed systems such as Grids, Clouds, HPC and Peer-to-Peer systems	Computational and Data Grids	Scheduling
SimBOINC [116]	Simulator based on the SimGrid toolkit, designed for the simulation of scheduling strategies in heterogeneous and volatile desktop grids and volunteer computing systems.	Computational and Data Grids	Scheduling

The choice of OptorSim was based on its frequent uses by existing researchers on data replication related issues as seen in Table 2.1, as well as its design motivation and support for both replication and scheduling related issues (Table 2.2). The next subsection presents an overview of OptorSim simulator, the candidate simulator chosen by this research.

2.9.2 The OptorSim Simulator

Amongst these tools, OptorSim is extensively used in simulating DG systems as explained severally by different researchers, such as [38], [40], [92], [117], [118], [119] as well as in DGF related data optimisation problems [38], [100]. OptorSim is a tool designed to test numerous replication optimisation approaches in a simulated grid environment before they are deployed in the real grid system, especially simulating data access optimisation mechanisms. OptorSim uses discrete event simulation framework. The OptorSim architecture for Peer-to-Peer replication optimisation of data resources, which comprises of the access mediator, storage mediator, and the Peer-to-Peer mediator, is shown in Figure 2.8.

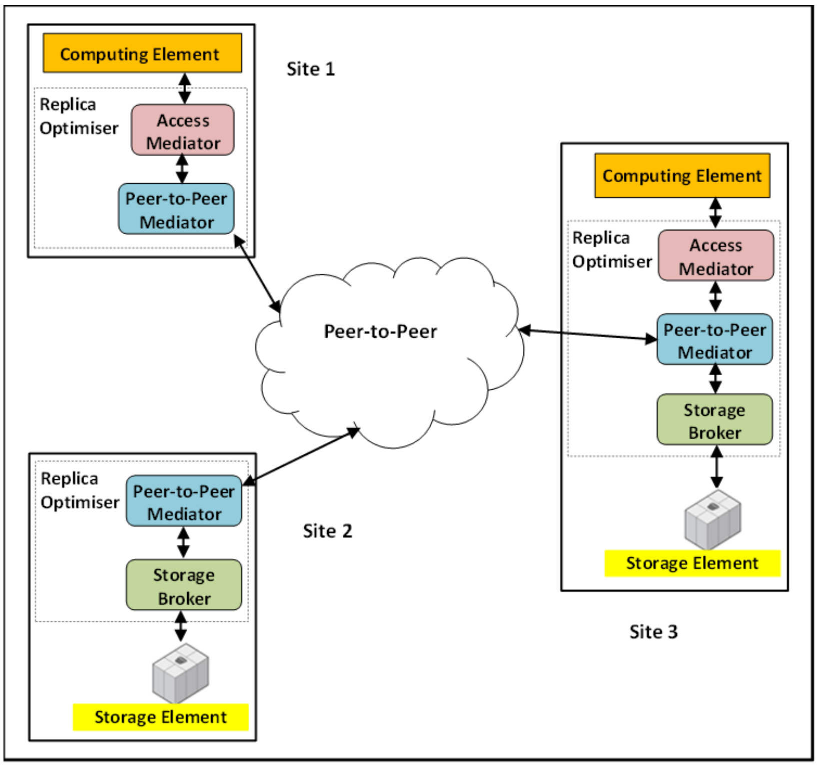


Figure 2.8. Peer-to-Peer architecture of OptorSim replica optimiser

According to research in [117], OptorSim was developed in the framework of the European Data Grid (EDG), as a joint effort of the University of Glasgow and CERN. The OptorSim architecture, shown in Figure 2.8, is based on the EDG model where sites provide computational and data storage resources, both modeled as computing elements (CEs), resource brokers schedule the jobs to CEs and routers without CEs. Each site handles its file content with replica managers; replicas are automatically created and destroyed using replica optimisers with different algorithms. Network topology can be described by enumerating the sites and specifying the bandwidth. Also, there are several file access patterns available for configuration. The OptorSim is equipped with replica optimiser comprising of three building blocks. These are:

- The Access Mediator (AM) - contacts replica optimisers to locate the cheapest copies of files and makes them available locally

- The Storage Broker (SB) - manages files stored in SE, trying to maximise profit for the finite amount of storage space available
- The Peer-to-Peer Mediator (P2PM) - establishes and maintains Peer-to-Peer communication between DG sites

In addition to simulating data replication strategies, OptorSim is designed to also handle job scheduling optimisations application in parallel and distributed computing systems [120]. It is an Open Source Java Application published under General Public License (GPL), thus presents a cost less tool for the simulation activities in this thesis. The OptorSim simulator enables modelling and simulating of entities in parallel and distributed computing (PDC) systems. These entities may include amongst others, users, applications, computational and data handling, as well as resources brokers for design and evaluation of optimisation mechanisms. OptorSim simulator provides a facility for building different classes of various resources that can be combined using resource brokers for solving computational as well as data-intensive applications. A resource can be a computing element (CE) on a single processor or multiprocessor with a shared or distributed memory resource, that is managed by time or space shared scheduling mechanisms.

The processing sites within a CE can be heterogeneous regarding processing capability, configuration, and availability. The resource brokers use scheduling mechanisms or strategies for mapping jobs to CEs to optimise the system or user objectives depending on their goals [38]. Amongst the simulation tools reviewed, OptorSim and GridSim [114], [121] simulators have been widely used to investigate the properties of both computational and DG platforms. However, critical study of these simulators indicated that previous researchers had extensively used OptorSim, to evaluate data replication mechanisms in a DG platforms, as well DGF environment. Also, based on the design motivations, OptorSim has been intended for testing of dynamic data replication mechanisms in DG environment.

2.10 The Existing Data Replication Mechanisms

This section discusses the characteristics features of two existing mechanisms that will be used later in chapter five for performance comparison with the proposed DRCEM mechanism. After critical analysis of the related literature in section 2.8 of this chapter, the two mechanisms DRCM and ELALW were selected for comparison, based on their relevance features as well as significant contributions, which provide the impetus for the various schemes of the proposed DRCEM mechanism. Although the existing mechanisms have made significant contributions in terms of replica evaluation, creation, placement and eviction functions, however, there is need for improvement on these features as earlier opined in the study of related works on data replication (Section 2.8). Table 2.3 outlines some of the significant features of the selected replication mechanisms, based on the triggers for replica evaluation and creation, as well as replica placement and eviction decisions, which form the core components of the replica management stage.

Table 2.3
Charateristics features of DRCM and ELALW on replica management.

Mechanism	**Triggers for Replica Evaluation and Creation Decision**	**Replica Placement Decision**	**Replica Eviction Decision**
DRCM	Triggered by job submission and file importance	Based on number of access (NoA) and the current sites loads	Based on file value and storage cost
ELALW	Triggered by job submission and file importance	Based on number of access (NoA) and site distance	Least frequently used files

In what follows, Table 2.4 outlined some additional characteristics features of DRCM and ELALW, based on the types of files supported, number of files to be replicated, number of created replicas, replica placement function and replica eviction function. The table features help to identify the parameters that are suitable for direct integration into the proposed mechanism and the parameters that require further enhancement prior to integration into the proposed mechanism by this research.

Table 2.4
Additional features of DRCM and ELALW based on files types and required number of replicas.

Mechanism	**DRCM**	**ELALW**
Type of files	Independent and dependent files	Independent files
Number of files to be replicated	Depends on FV and number of existing replicas	Depends on FV and number of existing replicas
Number of created replicas	Depends on FV, and number of existing replicas	Depends on FV and number of existing replicas
Replica placement function	Depends on read cost, workload and places of dependent files	Depends on NoA and node distance
Eviction function	Based on FV	Based on FV

As outlined in Tables 2.3 and 2.4, the existing mechanisms considered both independent and dependent files for replica evaluation and creation, but have failed to consider indirect logical dependencies of files, which is an important feature for efficient replica evaluation and creation decision. In addition, the existing mechanisms place replica files based on read cost, workload and site distance. However, they failed to consider sites availability in the replica placement decision. Furthermore, the mechanisms evict unwanted files based on file value (FV), without regards to inter-dependability amongst file replicas, which may lead to deleting an important file that may be needed later.

The DRCEM seeks to narrow the gap in the existing studies by enhancing both the replica evaluation and creation, replica placement and eviction processes. In addition, the idea of DRCEM was inspired from the real world marketing community, where there is a need for balance between the interests of users and the interests of service providers (resources) [26]. Thus, DRCEM as a part of data management services in DGF systems, provides the grid community with improved essential services compared to the existing DRCM [26] and ELALW [8] mechanisms. DRCEM aims to enhance the existing mechanisms by developing three schemes embodied in one mechanism namely dynamic replica evaluation and creation scheme (DRECS), dynamic replica placement scheme (DRPS), and dynamic replica eviction scheme (DRES). The unique advantages of DRCEM include its considerations for site workload, site distance, direct and indirect logical dependencies of files, as well as availability of replica sites in the course of evaluating, creating, placing and evicting replica files.

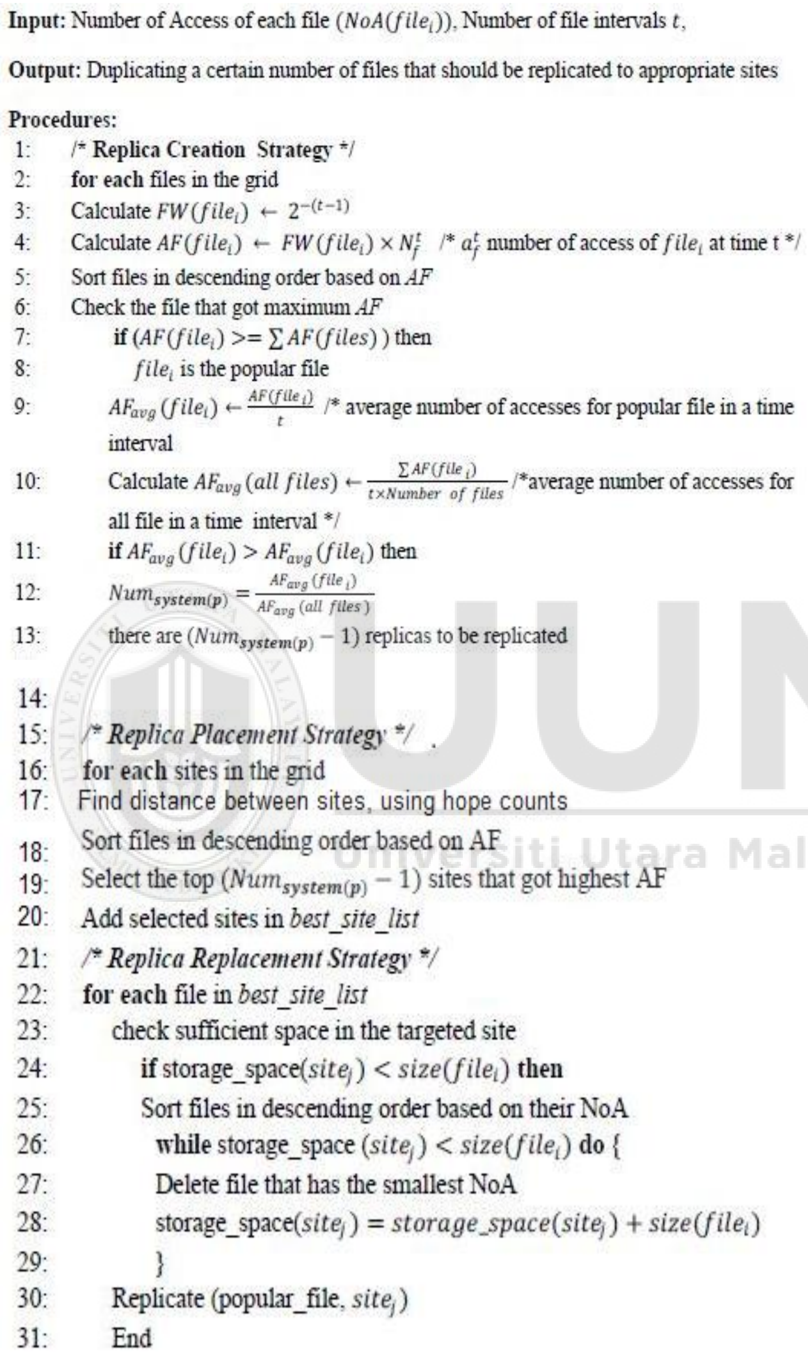
One of the issues addressed by DRCEM over the existing replication mechanisms is that it selects popular files based on file value and indirect logical dependencies, in addition to direct logical dependencies. The existing DRCM mechanism considered only the direct logical dependencies of files, as an enhancement over the LALW mechanism [40]. Thus, there was no particular focus on the dependency measurement, (when a file is dependent on another) for evaluating popular files. In other words, there is need to cconsider both direct logical dependencies (DLD) and indirect logical dependencies (ILD) of data files to determine the significance of the files to both users and the grid system, which also aids in evaluating the data files and determining the popularity of each file. The DLD and ILD compliments file access history in determining the popularity of each data file. Since access history gives records of user-to-files accesses, there is need to determine file-to-file access history, for proper file evaluation.

Regarding sites loads, there is need to consider both lightly and moderately loaded sites, as well as avoid highly loaded sites while placing replicas. In addition, there is need for a replica eviction scheme that will evict unwanted replicas from the system based on file values and dependency factors, thereby improving the storage space availability for placing new replica files within the federation system.

Also, there is need to consider three levels of data files, that is, independent files, directly dependent files and indirectly (clustered) dependent files, whereas the existing works operate only on independent files and directly dependent files. The next two subsections (2.10.1 and 2.10.2) outlined the algorithms for ELALW and DRCM mechanisms, respectively. Figures 2.9 and 2.10 illustrate the algorithms for the ELALW and DRCM mechanisms respectively, which show the differences in the procedures of each mechanism.

2.10.1 ELALW Data Replication Mechanism

The following Figure 2.9 shows an outline of the ELALW algorithm [8].

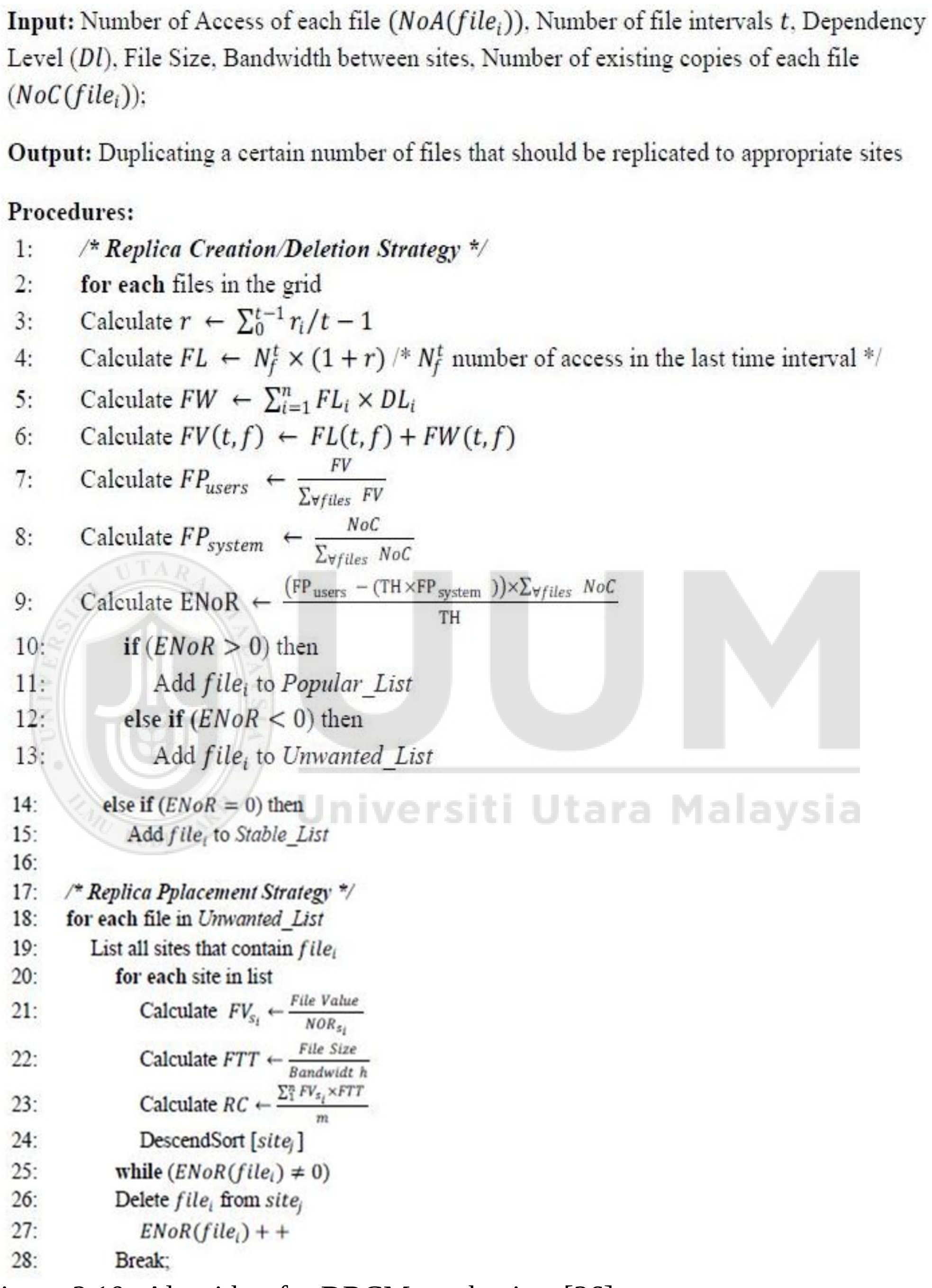


```
Input: Number of Access of each file ($NoA(file_i)$), Number of file intervals $t$,
Output: Duplicating a certain number of files that should be replicated to appropriate sites
Procedures:
1:  /* Replica Creation Strategy */
2:  for each files in the grid
3:  Calculate $FW(file_i) \leftarrow 2^{-(t-1)}$
4:  Calculate $AF(file_i) \leftarrow FW(file_i) \times N_f^t$  /* $a_f^t$ number of access of $file_i$ at time t */
5:  Sort files in descending order based on $AF$
6:  Check the file that got maximum $AF$
7:      if ($AF(file_i) >= \sum AF(files)$) then
8:          $file_i$ is the popular file
9:      $AF_{avg}(file_i) \leftarrow \frac{AF(file_i)}{t}$ /* average number of accesses for popular file in a time
        interval
10:     Calculate $AF_{avg}(all\ files) \leftarrow \frac{\sum AF(file_i)}{t \times Number\ of\ files}$ /*average number of accesses for
        all file in a time interval */
11:     if $AF_{avg}(file_i) > AF_{avg}(file_i)$ then
12:     $Num_{system(p)} = \frac{AF_{avg}(file_i)}{AF_{avg}(all\ files)}$
13:     there are ($Num_{system(p)} - 1$) replicas to be replicated
14:
15: /* Replica Placement Strategy */
16: for each sites in the grid
17: Find distance between sites, using hope counts
18: Sort files in descending order based on AF
19: Select the top ($Num_{system(p)} - 1$) sites that got highest AF
20: Add selected sites in best_site_list
21: /* Replica Replacement Strategy */
22: for each file in best_site_list
23:     check sufficient space in the targeted site
24:         if storage_space($site_j$) < $size(file_i)$ then
25:         Sort files in descending order based on their NoA
26:           while storage_space ($site_j$) < $size(file_i)$ do {
27:           Delete file that has the smallest NoA
28:           storage_space($site_j$) = $storage\_space(site_j) + size(file_i)$
29:           }
30:     Replicate (popular_file, $site_j$)
31:     End
```

Figure 2.9. Algorithm for ELALW mechanism [8]

2.10.2 DRCM Data Replication Mechanism

The following Figure 2.10 shows an outline of the DRCM algorithm [26].



Input: Number of Access of each file ($NoA(file_i)$), Number of file intervals t, Dependency Level (Dl), File Size, Bandwidth between sites, Number of existing copies of each file ($NoC(file_i)$);

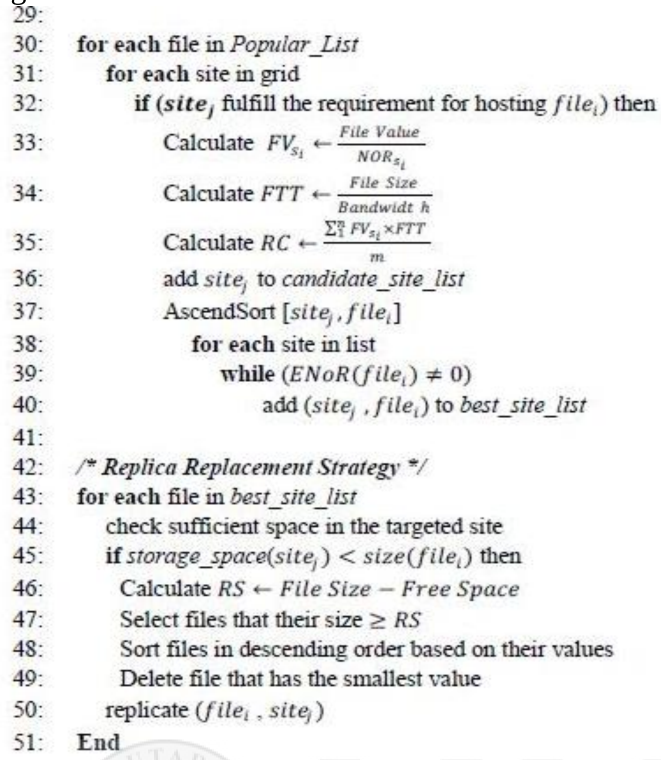
Output: Duplicating a certain number of files that should be replicated to appropriate sites

Procedures:

```
1:     /* Replica Creation/Deletion Strategy */
2:     for each files in the grid
3:     Calculate $r \leftarrow \sum_0^{t-1} r_i / t - 1$
4:     Calculate $FL \leftarrow N_f^t \times (1 + r)$ /* $N_f^t$ number of access in the last time interval */
5:     Calculate $FW \leftarrow \sum_{i=1}^{n} FL_i \times DL_i$
6:     Calculate $FV(t,f) \leftarrow FL(t,f) + FW(t,f)$
7:     Calculate $FP_{users} \leftarrow \frac{FV}{\sum_{\forall files} FV}$
8:     Calculate $FP_{system} \leftarrow \frac{NoC}{\sum_{\forall files} NoC}$
9:     Calculate ENoR $\leftarrow \frac{(FP_{users} - (TH \times FP_{system}))\times\sum_{\forall files} NoC}{TH}$
10:        if ($ENoR > 0$) then
11:            Add $file_i$ to $Popular\_List$
12:        else if ($ENoR < 0$) then
13:            Add $file_i$ to $Unwanted\_List$
14:        else if ($ENoR = 0$) then
15:            Add $file_i$ to $Stable\_List$
16:
17:    /* Replica Pplacement Strategy */
18:    for each file in Unwanted_List
19:        List all sites that contain $file_i$
20:            for each site in list
21:                Calculate $FV_{s_i} \leftarrow \frac{File\ Value}{NOR_{s_i}}$
22:                Calculate $FTT \leftarrow \frac{File\ Size}{Bandwidt\ h}$
23:                Calculate $RC \leftarrow \frac{\sum_1^n FV_{s_i} \times FTT}{m}$
24:                DescendSort [$site_j$]
25:            while ($ENoR(file_i) \neq 0$)
26:            Delete $file_i$ from $site_j$
27:                $ENoR(file_i) + +$
28:            Break;
```

Figure 2.10. Algorithm for DRCM mechanism [26]

Figure 2.10 continued.



```
for each file in Popular_List
    for each site in grid
        if ($site_j$ fulfill the requirement for hosting $file_i$) then
            Calculate $FV_{s_i} \leftarrow \frac{File\ Value}{NOR_{s_i}}$
            Calculate $FTT \leftarrow \frac{File\ Size}{Bandwidth}$
            Calculate $RC \leftarrow \frac{\sum_1^n FV_{s_i} \times FTT}{m}$
            add $site_j$ to candidate_site_list
            AscendSort [$site_j$, $file_i$]
                for each site in list
                    while ($ENoR(file_i) \neq 0$)
                        add ($site_j$, $file_i$) to best_site_list

/* Replica Replacement Strategy */
for each file in best_site_list
    check sufficient space in the targeted site
    if $storage\_space(site_j) < size(file_i)$ then
      Calculate $RS \leftarrow File\ Size - Free\ Space$
      Select files that their size $\geq RS$
      Sort files in descending order based on their values
      Delete file that has the smallest value
    replicate ($file_i$, $site_j$)
End
```

2.11 Chapter Summary

This chapter presents solid background on the issues covered in this thesis. The chapter conducts a critical review of related works in the research domain, and towards the end of the chapter, a comparison was made between proposed mechanism with some selected popular existing mechanisms. The first part of the chapter presented a brief description and characteristics of Data Grids and their federations, as well as some silent issues involved in dealing with Peer-to-Peer connectivities within the federation regions. Further, it was highlighted in this part that DGF systems are unlike other types of distributed systems due to their extensive data requirements, the presence of virtual organizations, broader heterogeneity, autonomous domains and unpredictable behavior of peer sites. The second part presented the need for an enhanced replication mechanism in a DGF environment. Related works in data replication were critically studied and presented. The review highlights related research and recent developments in replication mechanisms in the research domain. The second part indicates that the existing replication mechanisms require further enhancements in some functionalities. The first aspect is concerned with the mechanism as a whole, where more functions need to be included in the replication mechanism, such as: considering files dependencies while deleting replicas, how many replicas should be deleted and finding the places from where to delete the replicas.

To the best of the author's knowledge, no previous work encapsulates all of the core functions in a single mechanism. In this context the functions are: Determining which files to be replicated, places to host the newly created replicas and how many copies required; Determining the percentage site availability, while placing the replicas; Considering workloads, which entails determining the lightly loaded and moderately loaded sites, while placing replicas for load balancing, as well as considering the distance between sites in addition to access time; Determining which files to be evicted, locations from where to evict the files and how many files for replica optimisation, as well as determining the file to be evicted, in the case of insufficient storage space.

The second aspect is concerned with the individual functions of the replication mechanism, where some parameters have been neglected by existing replication mechanisms, which should be considered. For instance, while evaluating the individual files, to determine which file is to be replicated and evicted, the designer needs to consider the indirect logical dependency relationships between files, in addition to direct logical dependencies and the depreciation or appreciation rate of the file replicas. While placing the newly created files, the implementation needs to address the workload, which includes both highly loaded, lightly loaded and moderately loaded site before deciding where to place the replica files. The highly loaded sites should not be considered for replica placement. Going by the relevant literature reviewed in this thesis, it is proposed that there be a need for an enhancement in the replication mechanism that will consider all the core functionalities listed above. Also, the proposed mechanism should include the necessary parameters not considered by previous researchers either due to scope limitations or lack of unawareness, as pointed out in the critical review of the relevant works in this chapter.

Peer-to-Peer communications have been mentioned in many sections of the chapter, to examine the common properties that could affect replicaton in DGF systems. Although this research is not dedicated to replication in Peer-to-Peer computing domain, it was apparent to discuss the concept due to its similarity with the behavior of DGF sites within various regions. As mentioned earlier in the introductory Chapter 1, the sites within the regions of the DGF system communicate with each other, as well as with other sites in the neighboring regions. The nature of the communication is thus categorized into inter-regional and intra-regional communication. This is one of the motivations that informs the decision to include background literature from the dedicated Peer-to-Peer computing paradigm, so as to understand the dynamic nature of peer sites within the DGF system, which may interfere with the goal of achieving efficient replica placement, and consequently impedes in realization of the overall research goal, if not adequately addressed.

CHAPTER THREE

RESEARCH METHODOLOGY

3.1 Introduction

Research on grid systems and related discipline could take any of the forms, i.e., theoretical, experimental, design methodology, or a combination of these methods [122]. The interaction of users and sites are highly sophisticated in DGF environment, resulting in difficulty to predicting their behaviours from the beginning. In effect, studying the properties of DGF components comprising of multiple users and heterogeneous sites by implementing and deploying a prototype solution on a test bed will require many resources, tremendous efforts, and time. Indeed, the immediately available, feasible option will be to design the prototype using a suitable design methodology and later simulate the proposed solution. Thus, this research adopts the design research methodology (DRM) for its exceptional ability to generate a prototype solution close to reality. Also, its ability to provide a mechanism for evaluating some measurable metrics that might be used to elevate the grid performance makes design research method a suitable choice for this research.

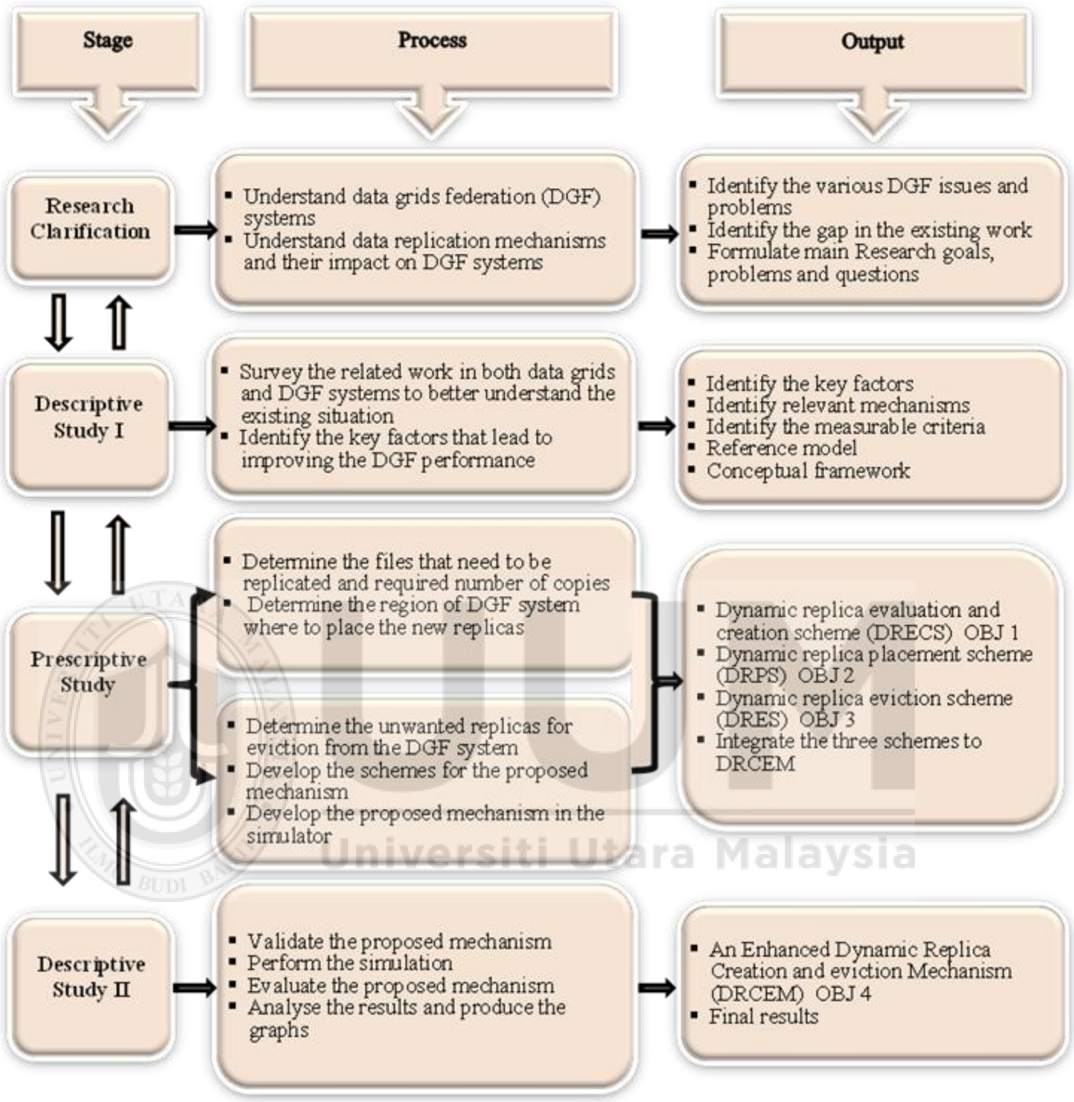
In this research, the DRM is used to draw the action plan for integrating the various components of the study in a comprehensible and logical manner and proceeds by introducing its primary stages. Section 3.2 discusses the overall research design. Section 3.3 discusses the first stage of DRM known as Research Clarification (RC). The section discusses the goal of RC stage, methods to support this stage, and overall deliverables from this phase. Section 3.4 describes the second stage called Descriptive Study-I (DS-I). It discusses steps to obtain sufficient understanding of the current situation, designs a reference model, and proposes a conceptual model.

Section 3.5 explained the method adopted in designing the proposed dynamic replica creation and eviction mechanism in line with the third stage of DRM, namely Prescriptive Study (PS). Section 3.6 outlined the various techniques for performance evaluation, under the Descriptive Study–II, as well as explains the evaluation techniques used by this thesis. Toward the end of the section, the performance evaluation metrics used in this thesis are discussed (Subsection 3.6.3). The chapter concludes with a summary in Section 3.7.

3.2 Research Design

The central aim of this research is to design a dynamic replica creation and eviction mechanism, with the intent to enhance the performance of the existing situations into better performance. The preferred solutions entail improving data availability, minimizing jobs completion times, minimizing bandwidth usage, as well as optimising storage and computing element usage, which will help minimise access latency and improve job throughput of the DGF system. This thesis utilises the DRM approach to guide in achieving the stated research goals.

The DRM approach requires a careful mapping of understanding of the traditional mechanisms and the development of a new one leading to an effective and efficient solution [88], [123]. These requirements are fitted with the design research definition as proposed by research in [122], and buttressed in [124], where design research integrates the development of understanding and the development of mechanisms. The DRM is an approach and a guideline, coined with a set of supporting methods that represent a framework for performing design research. The approach helps to make design research more rigorous, effective and efficient. Its outcomes are both academically and practically more worthwhile [122]. Due to these attractive features, DRM has been adopted for conducting this research. The DRM framework for this thesis is shown in Figure 3.1.



Stage
Process
Output
Research Clarification
Understand data grids federation (DGF) systems
Understand data replication mechanisms and their impact on DGF systems
Identify the various DGF issues and problems
Identify the gap in the existing work
Formulate main Research goals, problems and questions
Descriptive Study I
Survey the related work in both data grids and DGF systems to better understand the existing situation
Identify the key factors that lead to improving the DGF performance
Identify the key factors
Identify relevant mechanisms
Identify the measurable criteria
Reference model
Conceptual framework
Prescriptive Study
Determine the files that need to be replicated and required number of copies
Determine the region of DGF system where to place the new replicas
Determine the unwanted replicas for eviction from the DGF system
Develop the schemes for the proposed mechanism
Develop the proposed mechanism in the simulator
Dynamic replica evaluation and creation scheme (DRECS) OBJ 1
Dynamic replica placement scheme (DRPS) OBJ 2
Dynamic replica eviction scheme (DRES) OBJ 3
Integrate the three schemes to DRCEM
Descriptive Study II
Validate the proposed mechanism
Perform the simulation
Evaluate the proposed mechanism
Analyse the results and produce the graphs
An Enhanced Dynamic Replica Creation and eviction Mechanism (DRCEM) OBJ 4
Final results

Figure 3.1. Research Design

The various aspects of the DRM framework complement each other to produce an efficient and effective solution, for a better/higher performance. Blessing opined that design research must be scientific in acquiring valid results both in the theoretical and practical sense and due to its unique features; it requires a distinct methodology, such as the DRM. The framework illustrates the links between the various stages of the DRM, the methods used in each stage, and the primary deliverables. DRM comes in four different stages, namely Research Clarification (RC), Descriptive Study-I (DS-I), Prescriptive Study (PS), and Descriptive Study-II (DS-II) stages. In the sections that follow, a brief explanation of DRM stages from the perspective of this research area is presented.

3.3 Research Clarification (RC)

This research started with the first stage of DRM called RC, which was used to obtain a precise understanding, as well as a challenging but realistic overall research plan. In general, the deliverables of the RC stage is Chapter One. The RC stage consists of six iterative steps, as shown in Figure 3.2.

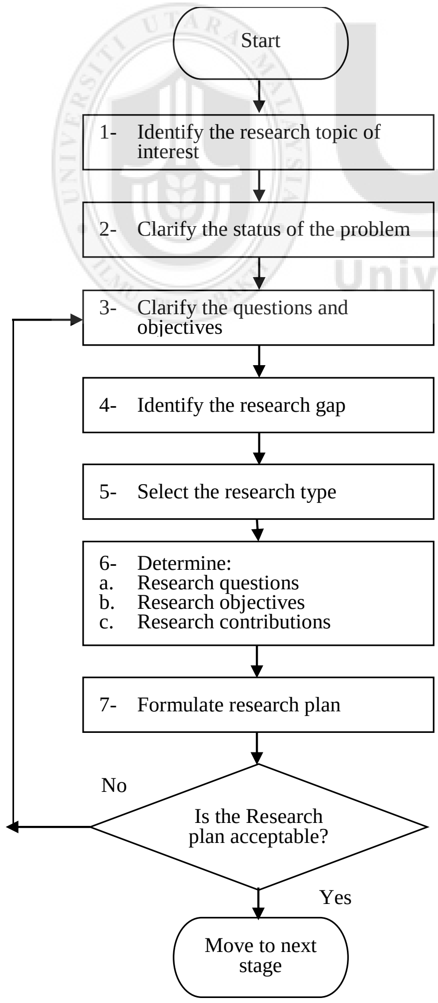

Start
1- Identify the research topic of interest
2- Clarify the status of the problem
3- Clarify the questions and objectives
4- Identify the research gap
5- Select the research type
6- Determine:
a. Research questions
b. Research objectives
c. Research contributions
7- Formulate research plan
No
Is the Research plan acceptable?
Yes
Move to next stage

Figure 3.2. Main steps in the research clarification stage

More specifically, the deliverable is the overall research plans that include the following elements: research focus and motivation, research problem and research questions, related literature, research approach (research type, scope, main stages, and methods), and the area of research contributions and deliverables.

3.4 Descriptive Study-I (DS-I)

The DS-I stage proceeds the RC stage. Whereas the RC stage is concerned with the realistic research plan, the DS-I stage is mostly associated with assessment and clear understanding of the current situations. The DS-I consists of five steps [123], typically made up of several iterations, as follows:

I. Review Literature
II. Determine research focus
III. Develop research plan for DS-1
IV. Undertake Empirical study:
    a. Collect, process and analyse data
    b. Update the conceptual Model
V. Draw overall conclusions:
    c. Combine the results
    d. Complete the conceptual model

The DS-I stage usually involves a critical analysis of related literature as well as empirical studies of the research area. In the course of this study, a detailed analysis of the current mechanisms was discussed as shown in Chapter two under literature review, and many empirical studies were critically evaluated to gain more perceptions into the existing solutions and onward look. Every step of the DS-I stage is geared towards increasing the understanding of the subject matter and may result in further empirical studies or literature reviews, which may lead to further refining and updating of the performance and the conceptual model.

The outcomes of the DS-I stage are a critical examination of related works as well as strengths and weaknesses of data replication mechanisms in DGF systems as presented in chapter two, and conceptual model for the proposed DRCEM mechanism.

3.4.1 An Overview of the Proposed DRCEM Mechanism

In this section, the core components of the proposed DRCEM mechanism are outlined. The development of these schemes entails the design of the core mathematical models for each scheme in the proposed mechanism. The components of the new mechanism, otherwise referred to as Dynamic Replica Creation and Eviction Mechanism (DRCEM) are enhancements on the existing DRCM [26] and ELALW Mechanisms [8]. Hence, the proposed DRCEM is a coin of three schemes, thus:

1. Dynamic Replica Evaluation and Creation Scheme (DRECS), which decides on replica evaluation and creation, determines how many replicas needed to be created; the DRECS evaluates data files based on access history and files logical dependencies, then optimises the number of replicas by creating the required number of replicas to meet users' data needs.
2. Dynamic Replica Placement Scheme (DRPS), finds the best location sites to host the newly created replicas; the best location sites are determined by the site distance, site availability, site workload and replica placement cost.

3. Dynamic Replica Eviction Scheme (DRES), which decides on replicas to evict in case there is need to create more space for the newly created replicas.

3.4.1.1 Conceptual Model of the DRCEM Mechanism

Figure 3.3 shows the conceptual model for the proposed DRCEM mechanism.

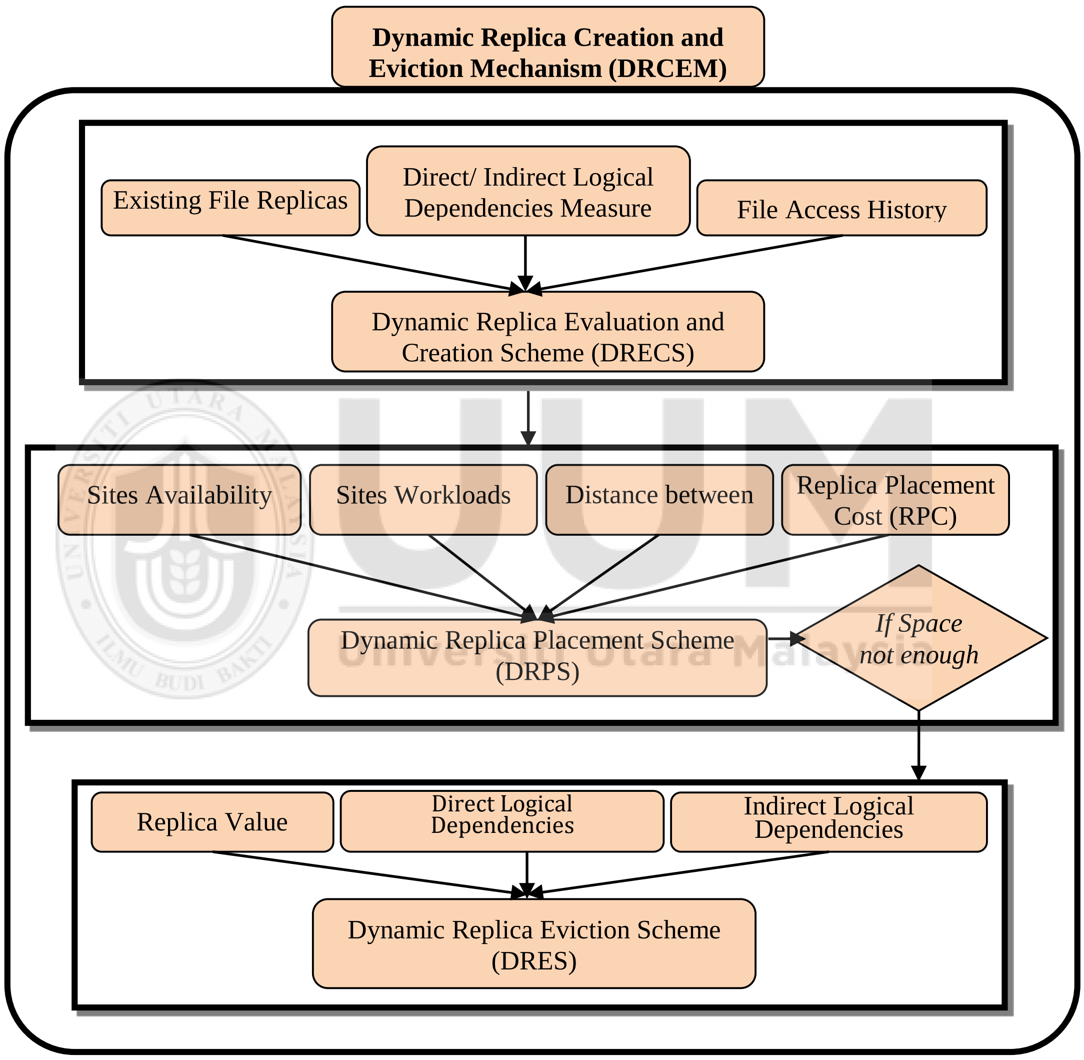

Dynamic Replica Creation and Eviction Mechanism (DRCEM)
Existing File Replicas
Direct/ Indirect Logical Dependencies Measure
File Access History
Dynamic Replica Evaluation and Creation Scheme (DRECS)
Sites Availability
Sites Workloads
Distance between
Replica Placement Cost (RPC)
Dynamic Replica Placement Scheme (DRPS)
If Space not enough
Replica Value
Direct Logical Dependencies
Indirect Logical Dependencies
Dynamic Replica Eviction Scheme (DRES)

Figure 3.3. Conceptual model of the proposed DRCEM mechanism

To achieve the desired goal of improving the performance of the DGF system regarding jobs completion times, network consumption, storage and computing element usage, this thesis develops a conceptual model for the proposed DRCEM mechanism. Some design metrics are carefully selected and used by this thesis to help in realising the effectiveness of the performance metrics. These design metrics include number of jobs, file size, site distance, site availability, site workload and replica placement cost. The conceptual model for the proposed DRCEM mechanism describes the desired and improved solution, following a critical review of related works. The figure consists of three schemes namely DRECS, DRPS, and DRES.

From Figure 3.3, the proposed DRCEM conceptual model suggests that, the mechanism achieves a set of *functional* and *non-functional requirements*. The *functional* requirements of DRCEM include replica evaluation and creation, replica placement, and replica eviction decision.

In the replica evaluation and creation functionality, DRCEM performs replication evaluation and creation processes and determines which files will be replicated as well as how many copies to be created. To make the decision, DRCEM considers popularity of the files combined with their logical dependencies, which is termed as File value (FV), and the existing number of replicas of each file.

The DRCEM considers file appreciation or depreciation, in addition, to file logical dependency to evaluate the files, aim at determining the files' popularity as regards to users' access. File appreciation or depreciation is rooted by the theory of exponential growth/decay [8], [26]. In addition, the DRCEM evaluates data files based on file value and existing number of replicas with the aim of determining the files' popularity relating to other files in the DGF system. Then, based on the two factors, the DRCEM decides on the overall file popularity.

Relating to the replica placement functionality for hosting the newly created replicas, the location sites that provide the least amount of Replica Placement Cost (RPC) for data replicas are chosen. Choosing the best location depends on the parameters namely; site distance, site availability, site workload, inter-dependent files location, and RPC. Eventually, in replica eviction functionality, the eviction function is invoked to evict a victim replica from the target storage element to provide space for the newly created replica. Choosing the victim replica depends on the parameters, namely the value of the files, inter-dependability and site workloads (highly loaded/lightly loaded sites).

The *non-functional* requirements that have been achieved include:

- *Scalability:* The DRCEM can scale well, in the case of increasing the depending parameters, such as file sizes and a number of submitted jobs
- *Interoperability*: The DRCEM should have the ability to work with other existing systems without special efforts, such as replica location services (RLS)
- *Performance:* The DRCEM should show improved performance compared to other similar mechanisms. Therefore, the evaluation metrics are used to measure the performance of the system
- *Feasibility and Simplicity*: The DRCEM should be implementable. Thus the mechanism has been void of ambiguities, to ease the implementation tasks

DRCEM relies on existing data grid core services, such as Replica Location Services (RLS) [52], [63] that provide information related to the physical file locations, and Information Service Provider (ISP) [125] , to provide the network availability and status. Figure 3.4 shows a relationship between DRCEM, Replica Location Services (RLS), Information Service Provider (ISP) and other related entities of the Replica Management Services.

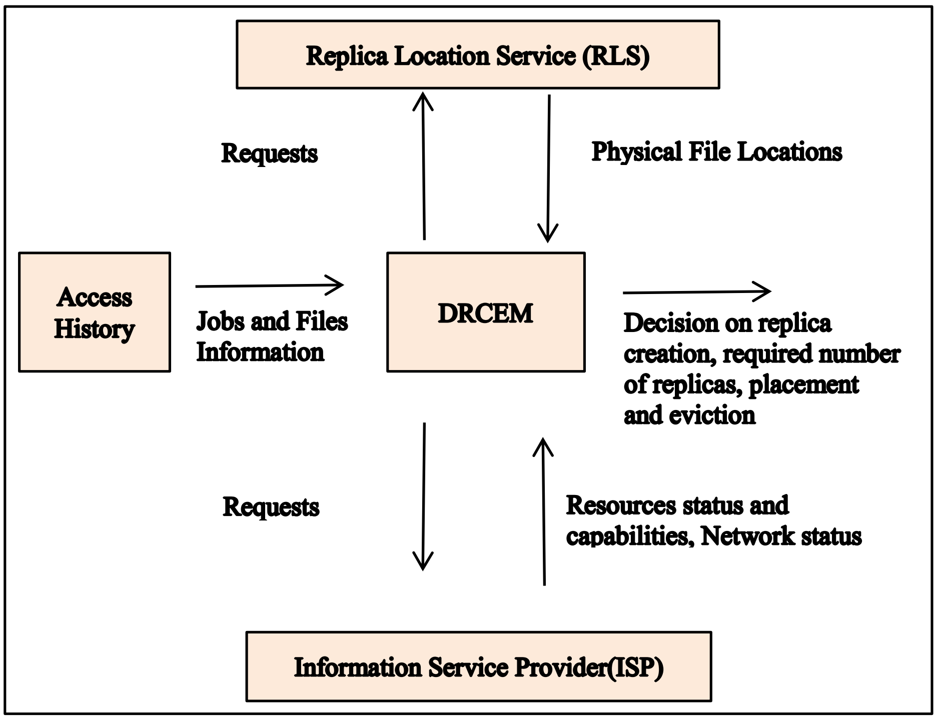

Replica Location Service (RLS)
Requests
Physical File Locations
Access History
Jobs and Files Information
DRCEM
Decision on replica creation, required number of replicas, placement and eviction
Requests
Resources status and capabilities, Network status
Information Service Provider(ISP)

Figure 3.4. Relationship between DRCEM, RLS, ISP and other entities

The DRCEM works in the background of the system in such a way that there is no direct connection with users. Therefore, as shown in Figure 3.4, DRCEM offers the following functionalities:

i- Gathers replica locations information from RLS;

ii- Gathers network bandwidth information from the Network Status;

iii- Gathers jobs information from the history file; and

iv- Makes central decisions on replica evaluation and creation, replica placement, and replica eviction.

3.4.1.2 Framework for the DRCEM Mechanism

The framework of the proposed DRCEM mechanism is shown in Figure 3.5. The framework comprises of three schemes namely, dynamic replica evaluation and creation, dynamic replica placement and dynamic replica eviction scheme.

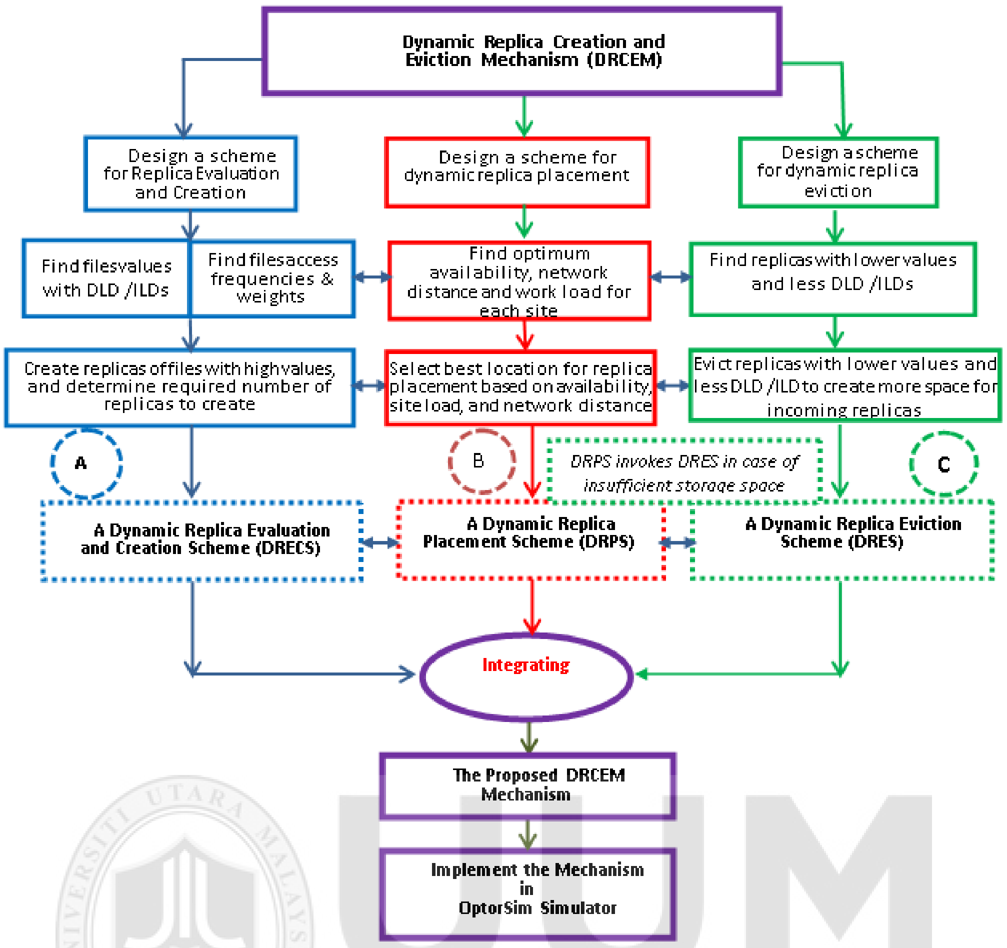

Dynamic Replica Creation and Eviction Mechanism (DRCEM)
Design a scheme for Replica Evaluation and Creation
Design a scheme for dynamic replica placement
Design a scheme for dynamic replica eviction
Find files values with DLD /ILDs
Find files access frequencies & weights
Find optimum availability, network distance and work load for each site
Find replicas with lower values and less DLD /ILDs
Create replicas of files with high values, and determine required number of replicas to create
Select best location for replica placement based on availability, site load, and network distance
Evict replicas with lower values and less DLD /ILD to create more space for incoming replicas
A
B
C
DRPS invokes DRES in case of insufficient storage space
A Dynamic Replica Evaluation and Creation Scheme (DRECS)
A Dynamic Replica Placement Scheme (DRPS)
A Dynamic Replica Eviction Scheme (DRES)
Integrating
The Proposed DRCEM Mechanism
Implement the Mechanism in OptorSim Simulator

Figure 3.5. Framework for the proposed DRCEM mechanism

The framework outlines how to achieve the various stages of replica evaluation and creation scheme, replica placement scheme and replica eviction scheme. Although the schemes appeared to be in parallel, they are actually logically arranged according to their execution as A, B and C on the figure. Specifically, the following methodologies insured to achieve the three components of the framework. Thus;

A. The dynamic replica evaluation and creation scheme (DRECS) aimed at creating and optimising number of replicas within a DGF system. In order to optimise number of replicas, insignificant file replicas will have to be deleted along with the creation process. This process differs with the dynamic replica eviction scheme (DRES) process in (C), whose aim is to create more space for incoming replicas. The replica evaluation and creation scheme encapsulates the following set of activities: Identify previous access frequencies for the files in the federation, then;

i. Compute average access frequencies for all the files in the federation
ii. Compute logical dependencies for all the file
iii. Compute indirect logical dependencies for all the files
iv. Compute file weights
v. Compute files values

B. The dynamic replica placement scheme (DRPS) is achieved via the following set of activities: Identify the availability status for all the sites

i. Determine percentage availability for all the sites
ii. Compute average and optimum availability
iii. Compute current workloads for all the sites; determine lightly and moderately loaded sites
iv. Compute distance between replica sites, using bandwidth information

C. For dynamic replica eviction scheme (DRES), files are evicted to create more space for incoming replica files, in case there is shortage of space to accommodate the newly created replica files. Thus, the dynamic replica eviction scheme is invoked only on the condition of insufficient storage space by the DRPS stage to evict file replicas that have low value and less dependability factor.

The proposed DRCEM mechanism performs these activities seamlessly, without constraining the users and the grid resources. Balance is maintained between replica placement and eviction, by deleting unwanted replicas and creating replicas of popular files. In addition, replica placement aims at making sure all-important replicas are hosted on sites with higher availability in the DGF, regarding online and offline status, and at distances that offer least replica placement cost.

3.5 Prescriptive Study (PS)

The PS is the main stage in DRM, as it includes the design of the proposed mechanism. Figure 3.6 shows a flowchart describing the main steps of the prescriptive stage according to this research phenomenon.

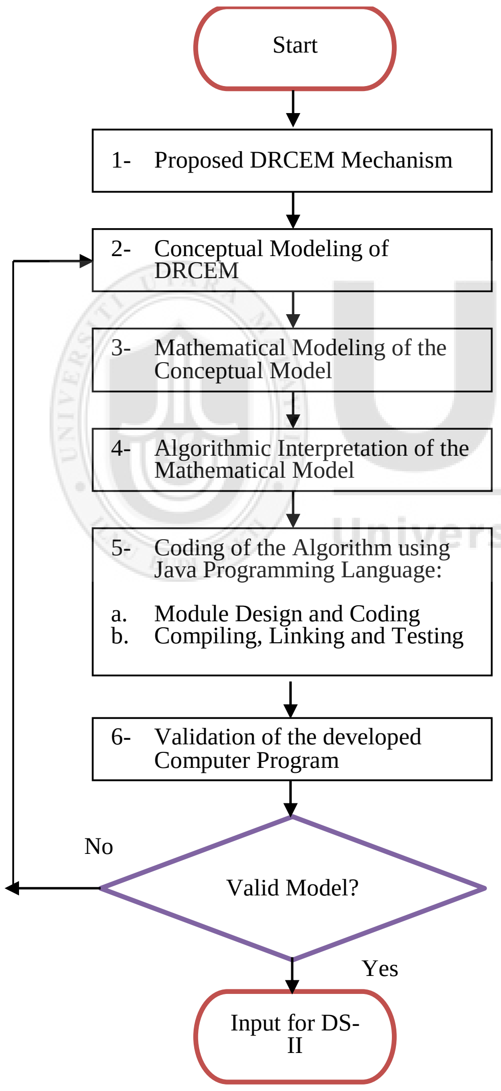

Start
1- Proposed DRCEM Mechanism
2- Conceptual Modeling of DRCEM
3- Mathematical Modeling of the Conceptual Model
4- Algorithmic Interpretation of the Mathematical Model
5- Coding of the Algorithm using Java Programming Language:
a. Module Design and Coding
b. Compiling, Linking and Testing
6- Validation of the developed Computer Program
No
Valid Model?
Yes
Input for DS-II

Figure 3.6. Main steps in the prescriptive study stage [126]

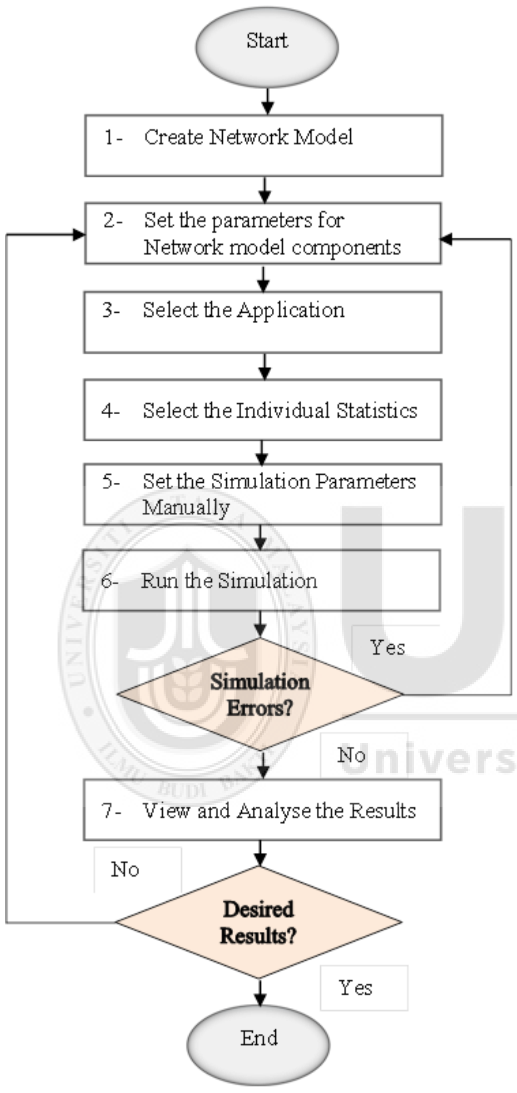
For this research, network modelling and simulation process proposed by research in [126] is adopted. The first block of the PS stage represents specifications of the proposed mechanism. The second and third blocks constitute model development that includes problem analysis, goals determination, and study of related theory. Furthermore, it frequently involves making assumptions and introducing a simplification to reduce the model's complexity.

Blocks 4 and 5 in Figure 3.6 illustrate mechanism implementation, and it depends very much on the choice of the simulation environment. Finally, validation will be covered in detail in the following subsection. The deliverables of the PS stage are Chapter Four and Five (Objectives 2 and 3), which covers the development of the proposed Dynamic Replica Creation and Eviction Mechanism (DRCEM). The design and implementation of the proposed mechanism as well as the performance validation of the proposed mechanism are carried out under the PS stage.

3.5.1 The Procedure for Mechanism Validation

Mechanism validation often connotes authenticating that the mechanism, within its domain of applicability, behaves satisfactorily and consistent with the study objectives [127]. Mechanism validation is the capacity to demonstrate that a computerised mechanism within its domain of applicability possesses a satisfactory range of accuracy [127]. Validation needs to be performed to ensure that the mechanism meets its intended requirements regarding the working methodology and the obtained results, which is all part of the process to building the correct mechanism.

Thus validation ensures that the proposed mechanism is transformed from one form to another while maintaining its sufficient accuracy [127]. In other words, validation assesses the accuracy of transforming a mechanism representation from a flowchart or pseudocode form into an executable computer program. The following Figure 3.7 shows the flowchart of the mechanism validation process.



Start
1- Create Network Model
2- Set the parameters for Network model components
3- Select the Application
4- Select the Individual Statistics
5- Set the Simulation Parameters Manually
6- Run the Simulation
Simulation Errors?
Yes
No
7- View and Analyse the Results
Desired Results?
No
Yes
End

Figure 3.7. Validation process flowchart [127]

The NetBeans IDE for Java Developers provides integrated functionality for Java programs, such as Source Editor, GUI Builder, compiler, debugger, launcher, parser, and makefile generator, which enables the validation process. The NetBeans IDE for Java Developers provides integrated functionality for developing and analyzing Java source codes. In this research, the various schemes for the proposed mechanism are transformed into Java code since OptorSim requires Java as the base programming language. The Code Analysis, which is an integral part of NetBeans, is capable of identifying and exposing syntax errors to the advantage of developers. Code analyser works by scanning Java source codes and checks for potential programming problems as well as syntax and semantic errors. Code Analysis is a crucial feature and was used to ensure that: the mechanism is programmed and implemented correctly, and the mechanism does not contain any errors or bugs. After writing the program codes, validation ensures that the code is correct both syntactically and logically. The validation process was done according to the laid down techniques as outlined by the research in [127], such as fixed value and internal validity tests, which are considered as standard reference for validation of simulation mechanisms. This thesis adopts the relevant techniques concerning validation of DRCEM mechanism, which was done against two existing DRCM and ELALW mechanisms. Two methods are used to validate the DRCEM mechanism, as explained subsequently:

Fixed Value Test: Fixed values of the simulation parameters are chosen (number of file access, file dependency level set to 0), and elementary topology such that expected results can be calculated manually, and then compared with actual results. Number of file access indicates how many times a given file is accessed at the completion of the experiment. By setting the dependency level to zero, it means that file inter-dependencies is set to minimum (zero) value, aimed at keeping the validation experiment as simple as possible.

A simple topology is used for the validation process that consists of twenty (20) sites and five (5) jobs were run, while keeping the dependency level set to zero and maintaining the same number of time intervals between jobs submissions. Then, the DRCEM was invoked to execute the jobs, after the five jobs have been submitted. Then, the number of jobs processed and the jobs completion time for each site were observed and recorded from the simulation, followed by manual calculations of these parameters to ascertain the mechanism's logic was correctly coded. Afterwards, the existing function that is already implemented in Optorsim <*listReplicas (String)*>, was used to verify the manual calculations, and the results yielded the same replica files that were expected to be replicated.

Internal Validity: The simulation was run several times (up to 20 times) to make sure that amount of variability are small. According to research in [123], the acceptable number of times for simulation validation is ten (10) times.

3.5.2 Comparison with Manual Computations of the Validation Data

The previous section explained the procedure for mechanism validation. This section will explain the process by running and comparing the simulation results from validation data with manual computations of the expected outcomes. Table 3.1 shows the data used for the fixed value and internal validity tests in this thesis.

Table 3.1
Fixed value and internal validity tests data

Parameter	**Value**
Number of Jobs	5
File Size	100MB
Number of sites	20
File dependencies	0
Job Delay	2500ms
Scheduler	QAC

Table 3.2

Statistic data for validation result

Site ID	**Number of Jobs processed**	**Job Completion Time (Mill sec)**
Site1	1	191.85
Site3	1	104.731
Site16	2	372.34
Site17	1	40.471
Total	5	709.392
Mean Job Time of all Jobs (Mill sec)		141.000

From Table 3.2, it shows that sites 1, 3, 16 and 17 were selected out of the 20 sites specified in the simulation to run the jobs. The mechanism selects these sites for executing the five jobs based on the selection procedure outlined in the QAC scheduling mechanism [33]. Each of *Site1, Site3* and *Site17* processed one job out of the five jobs, with jobs completion time of 191.85, 104.731 and 40.471 mill sec, respectively. In addition, *Site 16* alone processed 2 jobs, with job completion time of 372.34 mill sec.

Furthermore, from the *total* column in Table 3.2, there are five numbers of jobs, which indicated there was no job loss. Therefore, the issue of job loss will not be considered, as the simulation accounts for all submitted jobs. Also, manual addition of the individual jobs completion time reveals 191.85+104.731+40.471+372.34 = 709.392 mill sec, and dividing this value by the number of jobs (5) reveals 141.787 mill sec. This result is similar to the output from the validation simulation result in Table 3.2. The manual computation of job completion time is performed using Equation 3.3, which is explained further in section 3.6.3 under the performance evaluation metrics. Table 3.2 presents the result from the 20 runs of the validation data, Figure 3.8 plots the mean job completion time for 20 simulation runs.

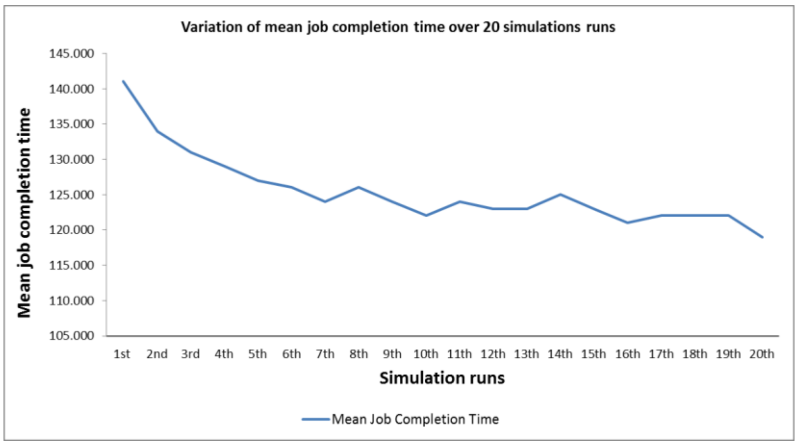


Figure 3.8. Variation of mean job completion time over 20 simulations runs

From the figure, it shows an insignificant variance with a root mean square value (RMS) of 12.99, when compared to the absolute values (which are of order 10^2), it is clear that the variation is in fact negligible. From the start, the values seem to be higher because the storage elements are empty. After the first run, the mean job completion time started dropping due to access history. Although the graph fluctuates at some points, it is however to be expected due to occasional interrupts within the system, which may result to minor delays while executing some jobs. Both DRCM and ELALW exhibits similar pattern of graph as would be explained under performance evaluation in Chapter Five, which further confirms the validation process. What follows is the DS-II stage, which explains the technique used to evaluate the performance of the proposed mechanism (see Chapter Five, Section 5.2).

3.6 Descriptive Study-II (DS-II)

The DS-II focused on the performance evaluation of the designed mechanism. Performance evaluation is a crucial step in evaluating any research. Since there is need to study the behaviours of the system before building such systems, thus the system's performance could be explored using measurement, simulation modelling or analytical modelling [88], [123]. The evaluation of DRCEM performance could be conducted using any of the above three possible traditional methods. However, measurement is most feasible when the actual system or its prototype exists. Thus, the available options are narrowed to two, namely analytical and simulation modelling. The subsequent subsections explain briefly the available techniques typically used for performance evaluation and a choice is made amongst the existing techniques for evaluating DRCEM mechanism.

3.6.1 Procedure for Performance Evaluation

Selecting the evaluation technique is a very crucial step in all performance evaluation projects [123]. The following subsections explain briefly the different types of techniques that are typically employed to evaluate the performance of a mechanism [88], [128].

3.6.1.1 Analytical Modeling

Analytical (also called mathematical) modelling is a set of equations formulated using mathematical symbolism to describing the performance of an actual system [88], [123]. A mathematical model can be investigated using computer programming, which translates the operations by using functional relationships within the system. The results of the mathematical model can be represented using a graphical representation drawn from the output of the running program. Users can adjust the conditions of the system by varying the input or parameters of the program. The technique is best suited for studying the behaviours of systems whose prototypes or actual implementation is yet to be carried out. Modeling helps to understand better the initial view of a system before moving to the implementation process. This technique is often used to study simple systems, where an analytical model is built and validated to explore and solve a particular problem in a system. Once the system complexity increases, this technique would require simplification and assumptions to focus on certain aspects of the system and fix the rest. According to [19], mathematical modelling has several benefits and advantages such as low cost and requires less time. However, it has low accuracy as compared to other techniques in performance evaluation.

3.6.1.2 Evaluation Using Testbed

Performance evaluation using measurement technique proved to be reliable and is preferable for validating a simulation model [122], [123]. However, in practice, this method is often impracticable either because the actual systems may not exist or due to the high cost of carrying out the measurements. Performance evaluation via measurement could be conducted using testbeds or direct implementation of the real systems. Although this technique delivers reliable results, the need for specialised equipment makes it unbearably costly [122], [123].

3.6.1.3 Evaluation Using Simulation

Simulation is a valuable and flexible tool for analysing the performances of computer systems [122]. In most systems designs, the real system or its prototype may not be available. Thus, a simulation model helps in studying the dynamic behaviours and responses of real systems. Also, even if a system lends itself to measurement, it may be preferable to use simulation. Because it accords the developers with a variety of scalable and controllable alternative workloads, as well as environments for comparison [128].

The simulation approach is extensively applied for evaluating and validating the performances of grid systems [123], and this research adopts this method of validation and evaluation. Some of the motivating characteristics features of simulation tools include: (i)- Simulation eliminates the need to build a real system from the onset. (ii)- Simulation provides enabling ground for conducting experiments in a controllable, and repeatable successions. (iii)- No limit to the number of scenarios for experimentation. (iv)- Simulation allows other researchers to reproduce the results of previously conducted experiments [123], [129].

The use of simulation environment to study the performance of the proposed DRCEM system attracts the following: (i)- Provide enabling ground for creating an accurate representation of the DGF model under review. (ii)- Simulators provide integrated development and simulation running on a single machine, which makes it easy for researchers to run their simulations and data analysis within the same environment. (iii)- It becomes possible to create complex topologies via simulators, which would have been otherwise impossible to replicate in a test bed environment. (iv)- Simulation enables researchers to reconstruct a representation of the application model easily. (v)- Simulation Provides vivid access to all the data about the metrics for performance evaluation in a graphical representation. (vi)- Some Simulators are available as freeware. Thus, little cost is needed to simulate systems under different scenarios [128].

3.6.2 The Simulation Environment

In this research, the performance analysis of jobs times, storage, bandwidth and computing element usage is conducted using the OptorSim simulation environment. As explained in Chapter Two, other grid simulators exist, such as

GridSim [114], [121], SimGrid [115], and GangSim [113], with varying characteristics features that differentiate them from one another regarding merits and demerits. This research has conducted a critical review of the most commonly used simulators in grid industry regarding their various functionalities. The initial justification for choosing OptorSim as the main simulator in this research follows straight from Chapter Two (Section 2.8). It was evident that majority of research conducted on DG systems used OptorSim to investigate the perofrmance of the proposed mechanisms. In addition, from Table 2.2, the OptorSim simulator has been designed to test dynamic scheduling and replication mechanisms. In these regards, this research has adopted OptorSim as the main simulation environment for evaluating the proposed mechanism regarding jobs completion time, storage element usage, bandwidth usage and computing element usage. The following subsection explains some variable features of the OptorSim simulator along with the simulation setup used in this thesis.

3.6.2.1 The OptorSim Simulator and Simulation Parameters

The OptorSim provides enabling ground for simulating any grid topology and a list of jobs to process using the integrated and extensible configuration files. It could also simulate background traffics and network usage. The simulator has provision for adding new mechanisms. During simulation, data statistics can be collected according to the measurable metrics specified [130].

A total of 5000 jobs was simulated in this research, with varying number of file sizes ranging from 2.5 GB, 5.0 GB and 10 GB. Table 3.3 presents the simulations parameters used in this thesis that are mostly static throughout the simulation processes.

Table 3.3

Configuration parameters used in the simulations

Parameter	**Value**
Number of Jobs	50, 500, 1000, 2000, 3000, 4000, 5000
Scheduler	QAC Scheduling Mechanism
Site Policy	All Job Types
File Size	10GB, 5GB, 2.5GB
Sites Bandwidth	The same EDG test bed bandwidth configuration file [33]
Storage capacity	One site has the most capacity of 100 GB to hold all the master files at the beginning of the simulation. The others have a uniform size of 80 GB.
Access history length	1000000 Ms
Storage metric (D)	0.67
Max. Queue Size	200 jobs
Job Delay	2500 Ms

Some of the design parameters such as the *file size* and *bandwidth* are variable parameters, which are described in the *jobs.config*, and *bandwidth.config* files of the OptorSim simulator, respectively, as shown on the table. In addition, *number of jobs* is one of the variable parameters that keep changing as the simulation proceeds. A job is submitted to Resource Broker every 25 seconds. Resource Broker then submits to CE according to a QAC scheduling mechanism. There are six job types, and each job type requires specific files for execution. The order of files accessed in a job is sequential and is set in the job configuration file as an input to the simulation. OptorSim is a command-driven software, but a friendly GUI is incorporated into the simulation environment, which may be used by researchers for comfort. However, GUI needs to be enabled from within the parameters config file that came with the OptorSim installation package. Further, the simulator can be run using Java integrated development interface (IDE) such as the NetBeans and Eclipse [33]. In this thesis, the NetBeans IDE is used for the simulation processes.

The simulation process could output relevant statistics depending on the measurable metrics specified within the simulation setup. These include total and individual job times, computing element (CE) usage, number of replications, local and remote files accessed, as well as the percentage of storage element (SE) usage.

The suitable statistical elements are displayed at the grid level, the individual sites and the components of the sites. The statistics can be viewed in real-time using the Graphical User Interface (GUI). The next subsection explains some of the important parameters used for data collection and analysis.

3.6.2.2 The Simulation Parameters used for Data Collection and Analysis

After running the simulation, there is the need to collect desired data that will aid in measuring the performance of the proposed mechanism. Thus, OptorSim simulator outputs relevant statistics depending on the measurable metrics specified within the simulation setup. In this research, the relevant data collected from the simulation output include; individual job times, total job times, number of replications, number of files evicted, local and remote files accessed, storage element sage (SEU), and computing element usage (CEU). These statistical elements are displayed at the grid level and the individual sites, as well as the components of the sites. The graphical user interface helps in displaying the statistics that can be read off in real-time. The collected data statistics are used in the measurement of the data availability as well as evaluation of the performance metrics.

DRCEM is compared with other existing mechanisms as explained in Chapter Two (Section 2.10) and will be discussed in Chapter Five under performance evaluation.

In order to evaluate the system's performance, the simulation was run using different number of jobs and file sizes, then the results were compared with the other existing mechanisms. The parameters that influence replication mechanisms within the simulator includes number of submitted jobs, site policy, job scheduler, access history length, storage metric (D), maximum queue size, and job delay [8]. These parameters are monitored throughout the simulation processes. They complement in the performance evaluation processes. For instance, the numbers of submitted jobs, as well as files sizes are varied, to measure the jobs completion times, storage element usage, bandwidth consumption and computing element usage.

i) Number of Submitted Jobs (Job Workload)

System scalability can be tested by the number of jobs running during the simulation. In the real DG system, according to research in [109], the CMS experiment makes a considerable usage of DG resources for the data storage and online analysis. For instance, on daily basis, an average of 250 users submits 200,000 jobs per day, reaching peaks of 500,000 jobs that access distributed data resources.

Thus, a user submits an average of 800 jobs on daily basis. So following the footprints of the existing works [8], [26], for simulating a different number of submitted jobs, the number of jobs considered for evaluation in this thesis is varied between 50 to 5000, at the intervals of 50, 500, 1000, 2000, 3000, 4000 and 5000 jobs. At the various intervals, values for the measurable metrics are output into tables and graphs for analysis.

ii) Site Policy

Different grid regions are likely to prioritise different kinds of jobs; each job has its requirements, which means that there are sites that may not be able to execute specific jobs. Site policy refers to a type of jobs, which would not be accepted by sites in the system. The effect of site policies on the overall running of the grid is investigated. This was done by defining two extremes of policy namely All Job Types and One Job Type [131]. Therefore, if a site accepts all job types, then the site has All Job Types policy on the underlying grid. Likewise, if a site would accept only one job type, then the site has One Job Type policy. In this thesis, it is assumed that sites can accept all jobs types. Thus, there no need for analysis regarding jobs types, but job numbers and file sizes.

iii) Scheduling Mechanism (Job Scheduler)

Typically, a scheduler submits jobs to the grid sites according to some algorithms that may affect the performance of the replication mechanism [34], [35]. The following are the job scheduler mechanisms implemented in OptorSim. Random: jobs are randomly scheduled to any computing element that can execute the job. Queue Length (QL): jobs are scheduled to the computing element that has the shortest queue of waiting jobs. Access Cost (AC): jobs are scheduled to the Computing Element with the lowest access cost (time taken to access the files required by the job). Queue Access Cost (QAC): jobs are scheduled to the computing element with the lowest queue access cost (sum of access cost for all jobs in the queue at the given computing element). For uniformity in the obtained results, and since the fourth scheduling mechanism combines the second and the third mechanism, this research used the fourth scheduler in all the simulation scenarios.

iv) Access History Length

The access history length is defined as the period from which to maintain file access history. The history of file accesses is used by replication mechanisms to identify the most popular file in the next time window. Therefore, the length of history used in the computations must be carefully chosen to produce an accurate prediction. If the history does not go back in time far enough, the statistics of file access may not be exact, but if the history goes back too far, it may affect simulator performance. Moreover, the information could be overdue and obsolete. Thus, the default length of access history (1000000 Ms) that was configured with the simulator is considered for evaluation throughout the simulation scenarios.

v) Storage Metric (D)

Storage metric is defined as the ratio of the storage element size to the total dataset size [132], which is computed in the following Equation 3.1:

$$D = \frac{Storage\ Element\ Size}{Total\ \ Dataset\ Size} \tag{3.1}$$

If the value of $D>1$, then there is enough space in the storage element to hold all files that a job would require. Hence, there is no need for any eviction and the replication mechanism will have little effect on the performance of the grid. If $D<1$, than the storage element is not capable of holding all of the required files so eviction must take place and choices have to be made on which replicas to keep. In this case, the replication mechanism will be useful. In order to study the effect of storage metric, different file sizes that vary between 2.5 GB to 10 GB were considered and used in the simulation processes.

vi) Maximum Queue Size (MQS)

The MQS is defined as the highest number of jobs a site can keep in its queue. The queue size was fixed at 200 jobs per queue in all of the experiments, to accommodate for jobs, which is the default setting that came with the simulator [131]. This parameter does not affect the results in any way since the *job delay factor* (item vii) takes care of the delay between job submission and retreiving of a job from the queue.

vii) Job Delay

Job delay is defined as the rate at which jobs are submitted to the data grid environment. In this thesis, job delay was fixed at 25 seconds in all of the simulations, as this is the default setting; there is no need to adjust the value.

3.6.2.3 The Simulation Topology

In this subsection, the simulation topology with its associated network connectivity is explained. The sites are connected via varying bandwidth capacities. Some sites are connected with a bandwidth capacity of 45 million bits per second (45Mbps), while others have as high as 2500 Mbps. The variations in bandwidth capacities make data replication decision very crucial.

The study of DRCEM was carried out using two existing studies namely the DRCM [26] and ELALW [8], both of which are based on the EU Data Grid model [16]. In EU Data Grid, a set of high-energy physics analysis jobs was generated from the Compact Muon Solenoid (CMS) [76], [81] experiments in the European Organization for Nuclear Research (CERN) [133], [134] project. Jobs were based on the job types described in [26]. The simulation topology is shown in Figure 3.9, indicating the various sites and bandwidth connectivity amongst the sites.

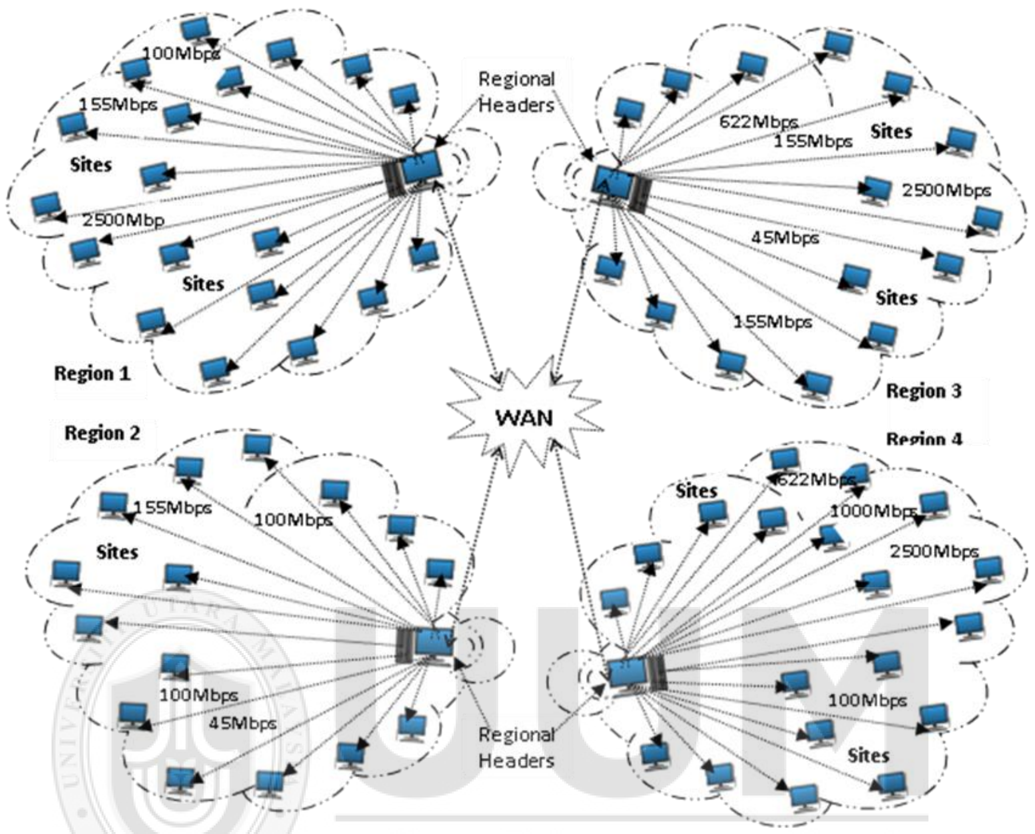


Figure 3.9. The DRCEM test bed showing sites connectivity

Thus, the topology used in this thesis has four clusters, and each cluster has 4000 sites. One site has the most capacity of 100 GB to hold all the master files at the beginning of the simulation. The others have a uniform size of 80 GB. As highlighted earlier in Chapter Two (Section 2.5) the future of Grid Computing is dynamically shifting from standalone grid to a federation grid environment [65] [1]. True to this assertion, the European Data Grid (EDG) has migrated to European Grid Infrastructure (EGI) [135], supported by EGI-InSPIRE [130]. EGI has featured a federation of shared computing, storage and data resources from national and inter-governmental resource providers that deliver sustainable, integrated and secure distributed computing services, to European researchers and their global partners [130], [135]. Thus, the composition of the EU Test bed sites has grown to more than what appeared on the standard CMS Test bed. Also, the topology has advanced to a federation of Data Grid connecting more countries, with over 200 user communities up to 20,000 site users, much more than envisaged by the original EU Data Grid Project [136].

A mechanism for Data Replication will have to consider the fact that EU topology, which evolved from EDG, Enabling Grids for E-science (EGEE) to EGI [130], [135] has now much number of sites than the early stages of grid technology. Therefore, this thesis considers simulating 10,000 sites, to study the scalability of DGF systems as the number of jobs increases, which also serves as a motivation for choosing the topic of this research. Implementing the DRCEM topology in OptorSim does not present much difficulty, as the simulator is designed with a Peer-to-Peer mediator, which establishes and maintains connections between sites.

3.6.2.4 Replication and Scheduling Mechanisms in OptorSim Simulator

The OptorSim simulator is bundled with both replication and scheduling mechanisms, which were deployed based on the EDG test bed [40]. However, the modular architecture of the OptorSim simulator allows researchers to integrate and test new mechanisms for performance evaluation. The scheduling mechanisms are used by the Resource Brooker to allocate jobs; and the replication mechansisms are used by the Replica Manager at each site to make copies of popular data items. The replication mechanisms help to decide when to replicate a file, which file to replicate and which file to delete according to research in [40]. The overall aim is to reduce the time it takes jobs to run and to make the best use of grid resources [37]. In the short term, an individual user wants their job to finish as quickly as possible, however, in the long run, the goal is to have the data distributed in such a way as to improve job times for all users, thus giving the highest throughput of jobs [38]. So far, the following scheduling mechanisms have been implemented in OptorSim:

a. The Scheduling Mechanisms

The scheduling mechanisms incorporated into the OptorSim simulator are Random Scheduling that schedules jobs to random sites [33]. The access cost scheduling (ACS) that schedules jobs to the site where time to access all files required by the job is shortest [33]. The queue size scheduling (QSS) [33], which schedules jobs to sites with the shortest job queue. The queue access cost (QAC) [33] scheduling that schedules jobs to sites where access cost for all jobs in the queue is shortest. In this thesis, the scheduling mechanism used along with the DRCEM mechanism for simulating system's performance is the QAC scheduler. The QAC scheduler has been used by existing works of the researchers in [8], [26] to evaluate their mechanisms.

b. The Replication Mechanisms

The replication mechanisms that are bundled with the OptorSim simulator are No replication, which reads files remotely [33]. The least recently used (LRU) [33], which always replicate, and deletes least recently used file. The least frequently used (LFU) [33], which always replicate, and delete least frequently used file. The economic model (Binomial) [33], which replicates files if economically advantageous, uses binomial prediction function for file values. The economic model (Zipf) [33] that replicates if economically advantageous, using Zipf-based prediction function. In addition, the modular design of the

OptorSim simulator enables new replication/scheduling policies to be developed and integrated the toolkit.

3.6.3 Performance Evaluation Metrics

Some performance metrics are used to evaluate the system performance. Four performance metrics namely jobs completion times; network usage, storage and computing element usage are used to evaluate the system performance. In addition, these metrics are evaluated using design metrics such number of jobs, file size, site distance, site workload and file logical dependencies. More details on the design metrics is given in Chapter 5. What follows is a comprehensive explanation on the performance metrics used in this thesis.

3.6.3.1 Job Completion Time

Job completion time, otherwise known as mean job execution time (MJET) is defined as the average time a job takes to execute, from the moment it is scheduled to Computing Element, to the moment when it has finished processing all of the required files. It is calculated by summing the total time taken by each job and divided by the total number of jobs [49], as shown in the following formula:

$$Jobs\,Completion\,Times = \frac{\sum TotalJobsTime}{n} \tag{3.2}$$

Where,

TotalJobsTime; is the total time taken by each job.

n; is the total number of jobs processed.

3.6.3.2 Effective Network Usage

Effective Network Usage (ENU) is defined as the ratio of files transferred to files requested, so a low value indicates that the optimisation mechanism used is better at placing files in the right places. Thus, ENU is a measure of how well the replication mechanism uses the network [92] is computed as:

$$ENU = \frac{N_{remote\ file\ access} + N_{replications}}{N_{remote\ file\ access} + N_{local\ file\ access}} \tag{3.3}$$

Where $N_{\text{remote file access}}$ is the number of times that Computing Element reads a file from a remote location, $N_{\text{replications}}$ is the total number of file replication that occurs, $N_{\text{local file access}}$ is the number of reads performed locally.

A lower value of ENU would indicate that the utilisation of network bandwidth is more efficient. Thus, to get low ENU value, the value of numerator should be small, which means that the mechanism should not do more replication unless it is beneficial to the entire system [92].

3.6.3.3 Storage Element Usage

Storage element usage of a site is the percentage of capacity reserved by files according to the total capacity for the underlying storage. The average of all storage reserve capabilities in the DGF can reflect the total system storage cost [84]. The Average Storage Usage (ASU) metric is computed by the following equation [140]:

$$ASU = \frac{\sum_{i=1}^{n} \frac{U}{C}(site)}{N} \times 100\% \tag{3.4}$$

Where,

U; is the storage space reserved for data files, *N*; is the number of sites in the data grid and *C*; is the total capacity of the storage facility or medium

3.6.3.4 Computing Element Usage (CE Usage)

CE usage is defined as the percentage of time that a CE is active (transferring or processing data). The CE usage of the whole DGF system is computed by aggregating the CE usage of each CE from the individual regions. The CE usage is a metric that could be of interest to resource owners, as high CE usage would mean that the workload is balanced across the individual DG platforms [49]. Low CE usage, on the other hand, would mean that some CEs have long queues while others are underused.

- **Response time**

Response time can be interpreted as the waiting time at the end of a request submission, and the commencement of the corresponding response from the system. It could also mean the time interval between request submission and the corresponding response from the system [4]. The latter definition is more appropriate if the time between request and getting response takes too long. Response time is mathematically represented as follows [37]:

$$\text{Response time } (T_r) = \text{Waiting time } (W_t) + \text{Service time } (T_s) \qquad (3.5)$$

- **Distance Between Replica Sites**

The distance *D(x,y),* denotes network distance between site *x* and site *y*. Network distance is computed by considering the number of physical devices between sites, using the number of hops with a traceroute command. Also, distance information is captured when a replica is checked for the first time, to reduce the cost. Network distance between sites influences replica selection decision, which is one of the key components of data management in data-intensive applications. Because of the possibility of the existence of several replicas for a given file, network distance is often used as the deciding factor for determining which replica location is the best for the grid users [92].

- **Job Throughput**

Throughput is the rate (requests per unit of time) at which the requests can be serviced by the system [59]. For batch streams, the throughput is measured in jobs per second. For interactive systems, the throughput is measured in requests per second. For CPUs, the throughput is measured in Millions of Instructions per Second (MIPS), or Millions of Floating-Point Operations per Second (MFLOPS). For networks, the throughput is measured in packets per second (PPS) or bits per second (bps). For transactions processing systems, the throughput is measured in Transactions per Second (TPS). The throughput of a system steps up as the load on the system increases from the start. After a certain load, the throughput ceases to rise; in most scenarios, the throughput may start reducing. The nominal capacity of a system is the maximum achievable throughput under perfect workload conditions. Bandwidth is the nominal capacity of networks.

3.7 Chapter Summary

This chapter has described in details the research design in readiness for achieving the research objectives. This research's prime objective is geared towards developing a dynamic replica creation and eviction mechanism (DRCEM) for improving data availability, which in turns improve the performance of DGF systems in terms of jobs completion times, storage element usage, network bandwidth usage and computing element usage. Four principal activities of the research are outlined in this chapter in line with the design research principle. The first activity is the Research Clarification (RC) stage, which discusses methods to support the initial stage of this research. RC was used to identify issues in the DGF systems, identify gap in the literature, and identify the problem, formulate objectives, and research questions that are both academically and practically worthwhile and realistic.

The second activity is called Descriptive Study-I (DS-I), which discusses steps to obtain sufficient understanding of the current situations relating to the DGF systems by using the reference table in Chapter two, which aids in proposing the DRCEM conceptual model. The third activity highlights the methods adopted in designing the proposed DRCEM mechanism for improving DGF performance, otherwise known as Prescriptive Study (PS). It was shown that the proposed DRCEM mechanism encapsulates three schemes namely DRECS, DRPS and DRES responsible for replica evaluation and creation, replica placement and replica eviction, respectively. The PS stage aids in coming up with the methodologies for realising the three schemes of the proposed mechanism. It shows vividly the various activities contained in each of the schemes.

The last activity named DS-II focuses on the evaluation of the proposed DRCEM mechanisms for DGF systems. The chapter also explained the validation and evaluation environment, along with the measurable metrics. The proposed DRCEM mechanism was validated, and parameters used in Chapter Five for performance evaluation are outlined as well explained in this chapter.

CHAPTER FOUR

THE SCHEMES IN DRCEM MECHANISM

4.1 Introduction

The chapter commences by giving brief highlights on the design objectives of the proposed mechanism, as well as explanations on the rationales and goals of the proposed mechanism. The chapter also gives the detailed design of the DRCEM along with design specifications, schemes, flowcharts and algorithms. The implementation of Dynamic Replica Creation and Eviction Mechanism (DRCEM) for improving the performance of DGF system is explained in the chapter's subsequent sections. The central idea of data replication is to preserve some replicas or duplicates of the same data at various locations within the DGF system. Thus, the three DRCEM schemes (see Figure 3.3) are fully developed and explained in this chapter. In addition, numerical examples are given at appropriate subsections of the chapter to illustrate how the various schemes operate to realise the intended functions of the DRCEM mechanism.

4.2 Design Objectives for the Proposed (DRCEM) Mechanism

The core goal of this research is to develop a dynamic replica creation and eviction mechanism (DRCEM), with the aim of minimising jobs completion times, network, storage and computing element usage, within a DGF environment. The term *dynamic* means that the mechanism creates files replicas that are subject to timely maintenance. Unlike *static* replica creation, which does not support timely maintenances in terms of replica updates, the replicas created by DRCEM could accept any forms of updates, either synchronous or asynchronous updates. One of the core benefits of the dynamic replica creation is that the replica files need not to entirely be replaced, in case of changes to the original or source file. In addition, as explained in Chapter 3, the proposed DRCEM mechanism consists of three schemes namely, Dynamic Replica Evaluation and Creation Scheme (DRECS), which determines the file replicas that need to be created, the Dynamic Replica Placement (DRPS) scheme that determines best locations on which newly created replicas should be placed and the Dynamic Replica Eviction Scheme (DRES), which determines the file replicas to be deleted in order to create more space within the federation regions.

These schemes are coined to form the DRCEM mechanism, which leads to realising the design objectives as well as the overall system objectives. The next subsections explain briefly, the performance metrics that formed the design objectives in this chapter, starting with the jobs times.

4.2.1 Access Latency (Tobs Times)

The term access latency (jobs times) refers to access time, considering delays due to the distance between replica site and the site requesting for such replicas. In addition, network bandwidth affects access time for data files, which may drastically affects the time required to execute scheduled jobs. The distance between replica sites plays a vital role in minimising network bandwidth usage [119].

4.2.2 Optimising Storage and Computing Element Usage

The storage cost is the storage space used to store data, and computing element indicates the resources used for computational tasks. Therein, the proposed mechanism utilises storage and computing element as optimally as possible by:

a. Determining the number of replicas that need to be created or evicted such that there is a balance between users' data requirement and system workload
b. Locates the important files replicas closer to the jobs that need them
c. Determining the victim replica that needs to be replaced by the newly created replica, in case there is not enough space at the chosen site to host the new replicas

4.2.3 Minimising Bandwidth Consumption

The term bandwidth relates to the propagation characteristics of communication systems between two sites within the network, and bandwidth quantifies the data rate that a network path can transfer [34], [35].

When the network bandwidth consumption is reduced, the network traffic will be decreased. Network traffic can be expressed as transmission time in the network, which is the period when a user requests a replica until the replica download is completed. Therefore, the proposed mechanism utilizes bandwidth as optimally as possible by:

a. Placing the replicas very close to the sites that request the replicas
b. Distributing the replicas among sites such that the workload is balanced, and thus avoid the network congestion
c. Ensuring that a site only stores a copy of a replica with the aim to distribute the workload among the sites; and
d. The number of times the system performs the replication process is minimised as much as possible

4.3 Detailed Schemes Design for the Proposed (DRCEM) Mechanism

In this subsection, the schemes for the proposed DRCEM mechanism are fully developed. The required inputs to the mechanism include file access history, site availability and site connectivity workload data, as outlined in the appendices section. The design of the proposed DRCEM mechanism commences with mapping out how the mechanism decides which file is important and worth keeping in the system, as well as which file is insignificant. This is achieved via the dynamic replica evaluation and creation scheme (DRECS).

The second stage of the design involves a dynamic replica placement scheme (DRPS), which determines the appropriate locations to put the newly created replica files. The third crucial stage involves a dynamic replica eviction scheme (DRES), which determines the unwanted replicas to be deleted from the DGF system in order to create more space for the newly created file replicas.

Thus, the mechanism takes decision whereby only the most popular files are replicated, and the least popular (insignificant) files are evicted. The evaluation process requires a scheme to calculate the file lifetime (FLT), file weight (FW), and file strengths (FS) and projected number of replicas (PNoR), based on files access histories. The proposed DRCEM mechanism takes account of the relationship between both the grid users and grid system, while deciding which file to replicate and which file to evict. Also, site distance, site availability and workloads are taken into considerations by the mechanism prior to placing file replicas. Thus, the design of the DRCEM mechanism commences with the development of a dynamic replica evaluation and creation scheme (DRECS) of the DRCEM mechanism, as explained in the following subsection.

4.3.1 The Dynamic Replica Evaluation and Creation Scheme

The dynamic replica evaluation and creation scheme (DRECS) decides which file should be created and how many copies to be created. The scheme performs a file-by-file evaluation of all replica files to determine whether a file is worth having compared to all other replica files that are in the system. In a DGF system, when a file is required by a job and is not available in a local storage, it may either be replicated or read remotely. If a file is replicated, the next time it is requested, the job can read it quickly and the time to complete the job could be reduced. However, if replicating a file requires the eviction of other files, future jobs running those files, which were evicted will take longer period.

The DRECS scheme was briefly outlined in Chapter Three (Subsection 3.4.1.2). Further to that, the scheme uses the *file access history* workload data (Appendix A), then performs the following set of activities:

i. Evaluates individual files to determine access frequency for each file, within a specified past time interval (determine popular and unpopular files)
ii. Compute average increase/decrease rate for the access frequencies within the past time intervals, for all the files in the federation
iii. Determines file life time (FLT), which is the file access frequency for the upcoming time intervals
iv. Determines files that have direct links, as well as indirect links to each individual file (computes logical dependencies)
v. Compute file weights (FW), files values (FV), file strengths (FS) and projected number of file replicas (PNoR) required for each popular data file
vi. Performs logical comparison of the *PNoR* for each file with zero (0), based

on which the scheme creates three files (at the first run) namely *Popularity_List*, *Unpopularity_List* and *Stability_List,* to contain the list of popular files, unpopular files and stable files records, respectively.

vii. Create the required file replicas of the popular files based on the *PNoR* value, using the *Popularity_List* file records

viii. Input for dynamic replica placement scheme (DRPS)

The following Figure 4.1 shows the logical flowchart of the dynamic replica evaluation and creation scheme (DRECS).

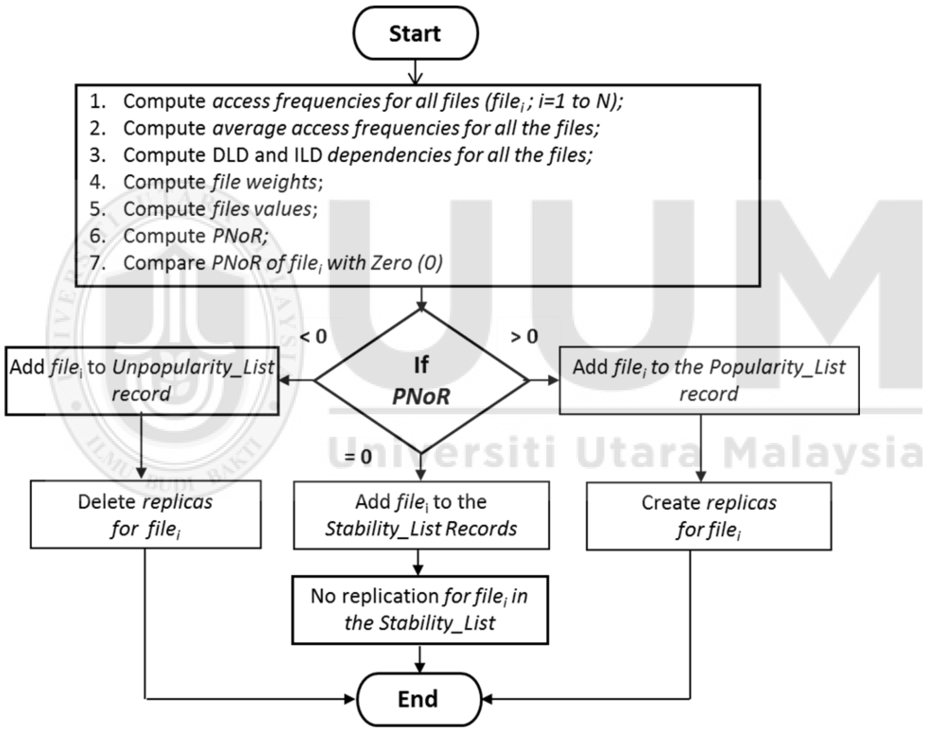

Start
1. Compute access frequencies for all files (file$_i$; i=1 to N);
2. Compute average access frequencies for all the files;
3. Compute DLD and ILD dependencies for all the files;
4. Compute file weights;
5. Compute files values;
6. Compute PNoR;
7. Compare PNoR of file$_i$ with Zero (0)
< 0
> 0
If PNoR
= 0
Add file$_i$ to Unpopularity_List record
Add file$_i$ to the Popularity_List record
Delete replicas for file$_i$
Add file$_i$ to the Stability_List Records
Create replicas for file$_i$
No replication for file$_i$ in the Stability_List
End

Figure 4.1. Flowchart for Dynamic Replica Evaluation and Creation Scheme (DRECS)

The Dynamic Replica Evaluation and Creation Scheme (DRECS) is the first of the three schemes that form the DRCEM mechanism. The scheme commences with the replica evaluation, as the first stage in replica creation. Also, this thesis assumes periodic replica creation, that is, the mechanism conducts background checks on the access frequencies of the existing files, and automatically decides to replicate the most popular files. Before creating a replica of a file, there is need to determine which file(s) need to be replicated, and how many copies.

The replica creation evaluates the individual files and selects the popular file for replication, and unpopular file for eviction based on associated file values. File value is a function of *file lifetime, file weight* and *file age*. Also, file weight signifies the importance of a file to the entire files, which is a function of *file lifetime* and *file dependencies.* Furthermore, *file lifetime* is a function of *file access frequency.* The uniqueness of the DRCEM mechanism is that, it considers all these parameters while evaluating popular files, a functionality which is lacking in the existing mechanisms. The next subsection explains how the DRECS determines popular files.

4.3.1.1 Determining the Popularity of Data Using Access Frequencies

Due to the limited storage capacity, replication decision should be made to conform to users' needs so that high demand files (replicas) should be kept and files (replicas) with less demand are evicted. The high and fewer demand replicas are determined by the frequency at which these replicas are requested by the users. In this thesis, a modified scheme based on the research in [8], is adopted with modifications, for finding file replicas with high demands as well as fewer demands within the DGF environment.

The modified scheme is at this moment referred to as a dynamic scheme for evaluating popular file based on appreciation (increase) rate and depreciation (decrease) rate, which is also based on access frequencies of the file replicas. Replica creation mechanisms intend to identify and select potential popular files because it is believed that popular files in the past time window are likely to remain popular in the future time frame. The proposed DRCEM mechanism is designed by considering the importance of data files to both users and other files within the DGF system.

Regarding file's importance to users, the mechanism takes account of how frequently the user requests a file and determine the change to this request whether the files exhibit a progressive or dwindling change. In addition, the mechanism keeps track of the relationships between data files, and observing the level of both direct logical dependencies (DLD) and indirect logical dependencies (ILD) on a particular file. As explained in Chapter (Section 2.6.7: Concept of replica dependency), DLD refers to relationship between data files, which are directly linked together. Indirect logical dependency (ILD) refers to link to a file via another data file, which is also called transitive or clustered logical dependency.

File access weight indicates the frequency at which a given data file is accessed over a time interval. It also forms the basis for determining the file that needs to be replicated, based on the file's popularity or its importance relating to other files. This research adopts the concept of access history of the stored files to find the most popular file and least popular file within the DGF system.

The statistic required for access frequencies is obtained from the header sites within each region. The header sites manage file information in each of the clusters of the DGF system. A given file record in a cluster header is stored according to the following format: <*File_ID, Cluster_ID, Number*> [8], [26].

The *Number* indicates how many times a given file (File_ID) was accessed by the cluster (Cluster_ID). Increase/ decrease factor determines how popular or unpopular an entity could be after a certain period. By analogy, data file could increase (appreciate) or decrease (depreciate) after a given period. The factor that determines file increase or decrease is the access frequency, which is obtained from the access history. The principle of file increase/decrease applies to file access history based on the enhanced largest access largest weight algorithm (ELALW) [8]. That is, file popularity increases or decreases by the frequency at which the file is accessed over a given period.

Each file popularity increases by the increase in access frequency and decreases by the decrease in access frequency. Thus, this research applies the principle of file increase/decrease on file access history to determine popular files, as well as files with less popularity. The popularity/unpopularity of a data file, determines how relevent/irrelevent such data file is to both users and the DGf system at large. File access history is mathematically modeled by the following formulae [8]: Assuming a_f^t is used to represent the number of accesses for file f at time t, and a_f^{t+1} is used to represent the number of accesses at time $t+1$, and then access history would be given by [8]:

$$a(t) \leftarrow a_0 * e^{-kt} \tag{4.1}$$

a_0 = number of access for file f at time t,

$a(t)$ = number of access for the same file f at time $t+1$, after the first access. By evaluating the trend in Equation 4.1, a sequence is obtained after a time T. Thus, the sequence of access numbers after a time T is given by:

$$a_f^0\ a_f^1\ a_f^2\ a_f^3\ \ldots\ldots a_f^{T-1}\ \ a_f^T \tag{4.1.1}$$

Average access rates for all intervals is calculated as follows:

$$\alpha \leftarrow \frac{\sum_{i=0}^{T-1} \alpha_i}{T} \tag{4.1.2}$$

Where

$$\alpha_{T-1} = lin \frac{a_f^T}{a_f^{T-1}} \tag{4.1.3}$$

Also from equation 4.1.3, the *lin* factor is a natural logrithm, which is used to multify the quantity $\frac{a_f^T}{a_f^{T-1}}$, which is a division of the number of access for file *f* at the current time interval with the number of access for file *f* in the immediate previous time interval.

$$a_f^T = a_f^{T-1} * e^{\alpha_{T-1}} \tag{4.1.4}$$

T = number of past intervals,

F = set of files accessed,

a_f^t = number of access for file *f* at the current time interval (*t*),

a_f^{T-1} = number of access for file *f* in the immediate previous time interval (T^{-1})

e = the increase /decrease factor for the replica files after a given period, which is computed using natural logrithm function.

Note that the natural logarithm of a number is the power to which "*e*" would have to be raised to equal that number. The function of e is similar to its application in determining the rate of population increase or decrease for a settlement, in a typical census aaplication. The number of access for the next time interval is calculated as follows:

$$a_f^{T+1} = a_f^T * e^{\alpha} \tag{4.1.5}$$

For instance, assuming the following set of data files, A, B, and C, and suppose that each of the file record is accessed at least once. According to research in [8] and [26], an increase in file access indicates file popularity, while a decrease in file access indicates file unpopularity. Based on these phenomena, this research finds the most popular file based on its increase rate, while the unpopular file is determined based on its decrease rate. Furthermre, this thesis considers additional factors while evaluating popular files namely DLD and ILD of file replicas.

Thus, using the sample access history workload data file (Appendix A), the most popular and least popular files are computed in stages, which starts by computing the average access frequncies for the upcoming future interval, using the previous history of data access to these files. The average access frequency for the upcoming interval, also known as the lifetime of the data file (FLT), is computed by Equation 4.1.6.

File lifetime (FLT) = average access frequency for the upcoming interval:

$$FLT \leftarrow a_f^{T+1} \leftarrow a_f^{T} * \mathrm{e}^{\alpha} \qquad (4.1.6)$$

Where α has been expressed under Equation 4.1.2 as the average access rates for time intervals between T1 to Tn, where T1 and Tn represents the time intervals under review for the first and the last interval, respectively.

4.3.1.2 Illustrations on How to Determine Access Frequencies

For illustrations on how the proposed DRCEM mechanism computes the average access frequencies for the stored replica files, for five consecutive time intervals, sixteen case files are used in this scenario. Within five consecutive time intervals between T1 to Tn (n=5), access frequencies for the sixteen data files A, B, C to P are pre-determined between T1 to T4.

The next fifth access frequency is computed to determine how frequent the files would be accessed after time T5, by considering the previous access frequencies T1 to T4 and the increase/ decrease factor, e. To illustrate, assuming from Table 4.1, file A was accessed in the passt time intervals 10 times after time T1, 15 times after time T2, 12 times after time T3 and 10 times after time T4. Then the following compuations ensured to find the value of alpha (α) as well as the next number of access to file A, after time T5. Thus, from Equation 4.1.2, the factor alpha (α) is used to determine the average access rates for all intervals as follows:

$$\alpha = \frac{lin\left(10/12\right) + lin\left(12/15\right) + lin\left(15/20\right)}{3} \cong -0.231 \qquad (4.1.7)$$

Also, from Equation 4.1.5, the number of access for file A in the next time interval after T4 is T5, which is computed as follows in Equation 4.1.8:

$$a_f^{T+1} = \; a_f^T \; * e^{\;\alpha} \;\leftarrow\; a_A^{4+1} * e^{\;-0.231} \qquad (4.1.8)$$

But according to the data on Table 4.1, number of access for file A at time T4 = 10, and T5 is unknown. Thus Equation 4.1.8 becomes:

$$a_f^5 \;=\; a_A^4 * e^{\;-0.231} \qquad (4.1.9)$$

$$a_f^5 = \; 10 * e^{\;-0.231} \;\approx\; 07.937 \qquad (4.1.10)$$

Therefore, number of access for file A after time T5 ≈ 08; from Equation 4.1.10. The same process is followed by the mechanism to compute access rates and the number of access for all other files in the system. Thus, given the sequence of acces numbers in Equation 4.1.1, the mechanism uses Equation 4.1.2 and Equation 4.1.5 to compute the files’ life time, which is also denoted as the average number of access frequncies in Equations 4.1.6. The Table 4.1 shows set of values for average access rates and acces frequencies for the past 4 consecutive intervals as well as for the upcoming fifth interval (T5), for the example files A-P, which were computed using Equations 4.1.1 - 4.1.6.

Table 4.1

Computation of popular file using access frequencies

File ID	**No of Access for time intervals T1 to T4**				**Find T5**	**Find alpha (α)=average increase/decrease rate for all previous intervals**	**Find number of access for the next time interval after the four**th **interval (file lifetime)**
	T1	T2	T3	T4	T5	$\alpha \leftarrow \frac{\sum_{i=0}^{T-1}(\alpha_i)}{T}$	$a_f^{4+1} \leftarrow a_f^4 * e^{\alpha}$
A	20	15	12	10	08	-0.231	07.937 ≅ 08
B	17	20	24	15	14	-0.042	14.383 ≅ 14
C	15	13	20	30	38	0.231	37.796 ≅ 38
D	14	18	21	16	17	0.045	16.728 ≅ 17
E	15	19	17	14	14	-0.023	13.682 ≅ 14
F	20	16	14	11	09	-0.1993	09.0125 ≅ 09
G	20	17	15	24	14	-0.042	14.383 ≅ 14
H	15	13	20	30	38	0.231	37.796 ≅ 38
I	14	18	21	16	17	0.045	16.728 ≅ 17
J	20	16	14	11	09	-0.1993	09.0125 ≅ 09
K	14	18	21	16	17	0.045	16.728 ≅ 17
L	20	12	15	10	08	-0.231	07.937 ≅ 08
M	13	15	30	20	38	0.231	37.796 ≅ 38
N	20	17	15	24	14	-0.042	14.383 ≅ 14
O	20	15	12	10	08	-0.231	07.937 ≅ 08
P	14	18	21	16	17	0.045	16.728 ≅ 17

From Table 4.1, the least popular file is the file that gets the least number of access in the next coming interval. In this case, files A, L and O have the least amount of access. Thus, A, L and O are the unpopular candidates files for eviction to create more space for upcoming replicas, while file C, H, and M are the candidate's files for replication with the highest number of access (38) in the upcoming (fifth) time interval (T5). The unpopular files are the files that have least number of requests from the users over the specified time interval measured, thus are considered less signifant in the system.

4.3.1.3 Framework for Determining the Required Number of Files to Replicate

Having known the popular file regarding the past time window and future access frequencies, the proposed mechanism needs to determine how many copies of the popular file should be created. The projected number of replicas to be created depends on some factors such as the file value (FV), file strengths (FS) in relation to both users ($FS_{<users>}$) and other files ($FS_{<system>:}$) as well as the threshold value (TH) and existing number of copies of the file replicas (ENoC) [40].

The term file strength signifies how freqently a file is accessed by both users and other files in the system, which indicates how important a file is to the users and the system. The DRCEM computes the projected number of replicas (PNoR) by considering the importance or strengths of the popular file regarding users and the system as follows [40]:

$$PNoR_{si} \leftarrow \frac{(FS_{<users>} - (TH * FS_{<system>})) * \sum_{\forall < files>} ENoC}{TH} \qquad (4.2)$$

Where;

PNoR is the projected number of required replica copies.

TH: is the threshold value, usually assigned by the systems administrator; set to 50% in this thesis to control the number of replications.

$FS_{<users>:}$ is the file strength regarding users, which indicates the importance of file to users and is computed as follows [40]:

$$FS_{<users>} \leftarrow \frac{FV}{\sum_{\forall < files>} FV} \qquad (4.2.1)$$

Where

FS: file strength regarding users, and

FV: is the value in respect to the DGF system (Section 4.3.2.10)

$FS_{<system>}$: is the file strength regarding the DGF system as a whole and is computed as follows [40]:

$$FS_{<system>} \leftarrow \frac{ENoC}{\sum_{\forall < files>} ENoC} \qquad (4.2.2)$$

where,

FS: is the file strength regarding the DGF system as a whole, and

ENoC: existing number of replica copies of the underlying file.

The objective here is to strike a balance between the users and the DGF system either by increasing or decreasing the number of existing copies of replica files ($FS_{<system>}$) to meet the volume of demand on files within the DGF ($FS_{<users>}$). The balance occurs when users and systems strengths are identical as expressed by the following Equation 4.2.3 [40]:

$$FS_{<users>} = FS_{<system>} \qquad (4.2.3)$$

However, a common scenario is when there are many requests, but few replicas exist [11], [40]. This is because storage capacities and other DGF resources are limited. Therefore, in this thesis, the focus is on the most occurring scenario, so Equation 4.3.3 will become Equation 4.2.4 as follows [40].

$$FS_{<users>} = TH*FS_{<system>} \qquad (4.2.4)$$

where TH is the threshold value that determines the required percentage number of copies that are supposed to exist to meet the user's request of the popular file.

4.3.1.4 Scenarios Used to Determine the Required Number of Files to Replicate

In this thesis, number of popular files for replication is controlled by both the *TH* value and *PNoR* value. As explained under Equation 4.2, the system administrator specifies the threshold value as a percentage value, which is usually set to 50% of the storage resources. However, the *TH* value varies according to the DGF situation, such as the current bandwidth, the type of the running applications and jobs, as well as the workload of the system (number of jobs and number of files).

In this thesis, the *TH* value is set to 50%, so that number of replication should not occupy more than 50% of the storage resources. This is to make sure storage resources are not drastically constrained by unnescessary number of files replicas. Regarding the use of *PNoR* value to control the number of popular files to replicate, this thesis considers three different scenarious.

a. *Scenario 1*: if the $PNoR > 0$, then the system will replicate copies of the file, based on the computed *PNoR* value of the affected file
b. *Scenario2*: if the $PNoR < 0$, then the system will evict copies of the existing file replicas, based on the computed *PNoR* value of the file
c. *Scenario3*: if the $PNoR = 0$, then neither replication nor eviction is required fof the underlying file replicas

4.3.1.5 Illustrations on How the Required Numbers of Files are Computed

In this subsection, the thesis illustrates how the required number of popular files are computed using *TH* value and *PNoR* valu*e.* At the start of the computation, *FV* and the *ENoC* are used as inputs.

The *FV* (refer to Section 4.3.2.10 Mathematical Framework for Determining File Value) and *ENoC* represents file values and existing number of replica copies, respectively. Assume a region within DGF system has 16 files, and their corresponding *FV* and *ENoC* exist as shown in Table 4.2.

Table 4.2

Example of 16 files together with their values and existing number of copies

File ID	File value in respect to DGF (FV)	Existing number of replica copies (ENoC)
File1	25.58	1
File2	32.20	2
File3	58.90	1
File4	36.33	3
File5	27.95	4
File6	47.04	1
File7	14.00	3
File8	68.74	1
File9	26.51	1
File10	31.08	6
File11	42.56	1
File12	38.34	2
File13	51.73	4
File14	53.94	1
File15	08.00	3
File16	33.07	2
Total	595.97	36

Thus, the relationship between *ENoC* and *PNoR* is that the former is used as input for determining the later. For implementation, the *ENoC* are obtained from the access history workload data file. However, for illustrative purposes, the existing number of copies in Table 4.2 are assigned arbitrarily for the sixteen case data files. From Table 4.2, the assumption made was that the threshold value *(TH)* set by system administrator is 50%, that means from Equation 4.2.4, the $FS_{<users>}$ should be twice $FS_{<system>}$ value. The primary concern here is to determine which files need to be replicated and which files need to be evicted. The first step is to calculate the strengths of each file regarding both users and system according to Equations (4.2), (4.2.1), and (4.2.2). For example, the strengths of *File1* regarding both the users and system as well as the projected number of the required replicas (*PNoR*) are computed as follows.

$$FS_{(File1)<users>} \leftarrow \frac{25.58}{595.97} \leftarrow 0.043$$

$$FS_{(File1)<system>} \leftarrow \frac{1}{36} \leftarrow 0.028$$

$$PNoR_{(File1)} \leftarrow \frac{(0.043 - 2 * 0.028) * 36}{2} \leftarrow \text{-0.234} \cong 0$$

The computed *PNoR* value is rounded to the nearest whole number since it represents projected number of replica copies, which must be an integer value. In this situation the projected number of replica for *File1* is null. Thus, no replication will occur for *File1* in this scenario. Thus, the mechanism computes *FS* values and *PNoR* values for all files, as shown in Table 4.3.

Table 4.3

Example of how DRCEM computes PNoR values of data files

File ID	Users Strengths	System Strengths	PNoR
File1	0.042922	0.027778	-0.227
File2	0.05403	0.055556	-1.027
File3	0.09883	0.027778	0.779
File4	0.060959	0.083333	-1.903
File5	0.046898	0.111111	-3.156
File6	0.07893	0.027778	0.421
File7	0.023491	0.083333	-2.577
File8	0.115341	0.027778	1.076
File9	0.044482	0.027778	-0.199
File10	0.05215	0.166667	-5.061
File11	0.071413	0.027778	0.285
File12	0.064332	0.055556	-0.842
File13	0.0868	0.111111	-2.438
File14	0.090508	0.027778	0.629
File15	0.013423	0.083333	-2.758
File16	0.055489	0.055556	-1.001

From Table 4.3, the results show that *File3*, *File8*, and *File14* each has *PNoR* value approximately equal to 1, thus needs to be replicated by one copy each, while 1 copy of *File2* and *file16*, 2 copies of *File4*, 5 copies of *File10*, 3 copies of

*File*5 and *File7* need to be evicted from system to create more space. The negative value indicates that these files contain excess number of file replicas, therefore the excess need to be evicted.

Also, the *PNoR* with positive values indicates that their present qunatity is not sufficient to meet the required data availability within the system. Therefore, more copies of these files need to created. On the other hand, the *PNoR* for *File1* = -0.227 and that of *file9 =-0.199,* which approximately equal to 0, thus no action will befall thes files, as they are considered to be in a stable state. In effect, there will be three kinds of files, the first kind contains files that need to be replicated, the second contains files that need to be evicted, and the third kind contains files that will neither be replicated nor evicted from the system, in others words, these kind of files are considred stable as the case with *file1* and *file9.*

Note that file weight (FW) and file value (FV) are used by both the DRCES, DRPS and DRES schemes, thus the design of these parameters are reserved for the next subsection. This does not effect the logical flow of the DRCEM program in the Java Programming Language environment, which allows modular programming design.

4.3.2 The Dynamic Replica Placement Scheme

The Dynamic Replica Placement Scheme (DRPS) finds the best site to place the newly created file replicas. Placing replicas closer to sites that will likely request the file benefits the users regarding data locality [141]. Thus, while selecting best locations for placing file replicas, in addition to considering the dynamic behavior of the peer sites, five parameters need to be determined. These are replica placement cost (RPC), file transfer time (FTT), site workload, the distance between sites, availability of replication site (SA), and DLD/ ILD of replica files. Thus, DRCEM considers the aforementioned parameters to obtain a more efficient location for placing file replicas in the DGF environment. The scheme uses the *Popularity_List* file record, and then performs the following set of activities:

i. Computes current workload for all sites

ii. Compute current disk space (CDS) and optimum disk space (optimum CDS)

iii. Get storage requirements from files sizes

iv. Compute distance between replica sites (D) using bandwidth information

v. Compute replica placement cost from FV, FTT & D and compute RPC optimum

vi. Determines percentage availability for all sites

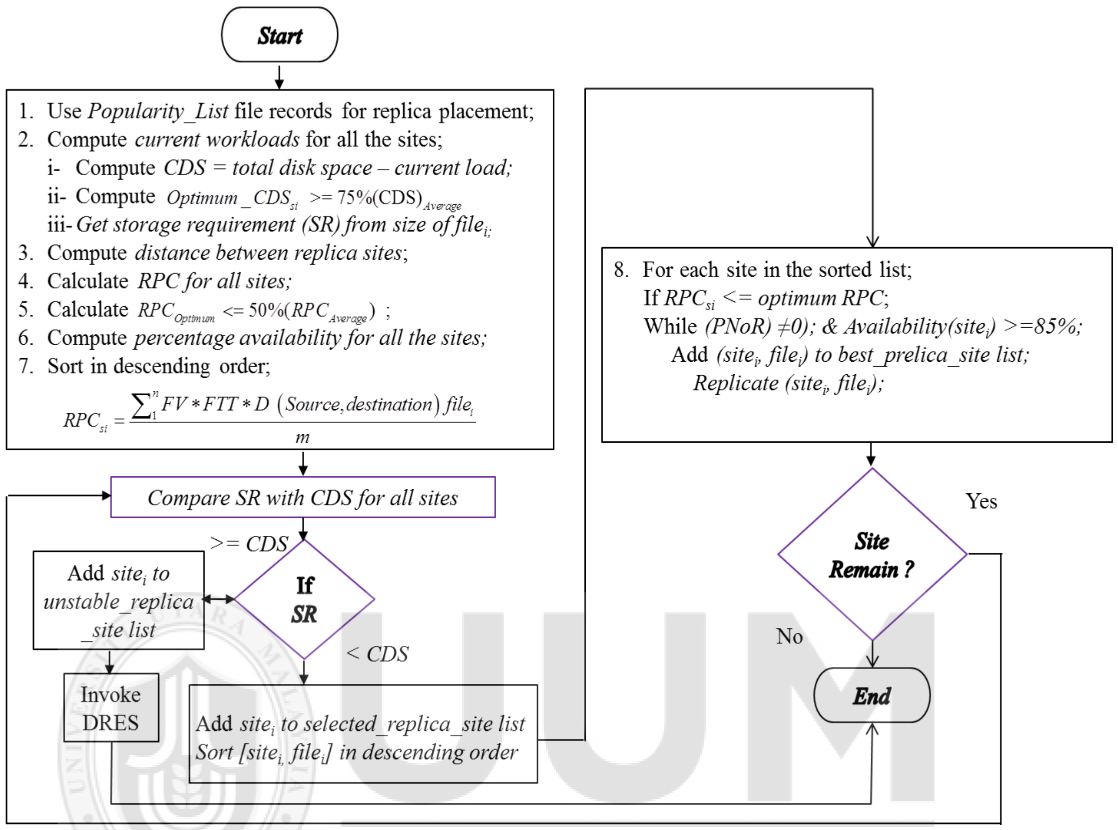
vii. Add sites whose CDS > SR to *selected_relica_site* list

viii. Add sites whose CDS <= SR to *Unpopularity_List (unstable_relica_site)*, and then invoke the DRES scheme

ix. Add sites whose RPC <= optimum RPC; PNoR ≠ 0 and Availability >= 85% to the *best_replica_site* list

x. Distribute the computed popular files from DRECS scheme to the various sites in the *best_replica_site* list

The following Figure 4.2 shows the logical flowchart of the dynamic replica placement scheme (DRPS).



Start
1. Use Popularity_List file records for replica placement;
2. Compute current workloads for all the sites;
i- Compute CDS = total disk space – current load;
ii- Compute $Optimum\_CDS_{si} >= 75\%(CDS)_{Average}$
iii- Get storage requirement (SR) from size of $file_{i;}$
3. Compute distance between replica sites;
4. Calculate RPC for all sites;
5. Calculate $RPC_{Optimum} <= 50\%(RPC_{Average})$;
6. Compute percentage availability for all the sites;
7. Sort in descending order;
$RPC_{si} = \frac{\sum_1^n FV * FTT * D\ (Source, destination)\ file_i}{m}$
Compare SR with CDS for all sites
>= CDS
If SR
< CDS
Add $site_i$ to unstable_replica _site list
Invoke DRES
Add $site_i$ to selected_replica_site list
Sort [$site_i$, $file_i$] in descending order
8. For each site in the sorted list;
If RPC_{si} <= optimum RPC;
While (PNoR) ≠0); & Availability($site_i$) >=85%;
Add ($site_i$, $file_i$) to best_prelica_site list;
Replicate ($site_i$, $file_i$);
Site Remain ?
Yes
No
End

Figure 4.2. Flowchart for Dynamic Replica Placement Scheme (DRPS)

The next sub-section address how to determine the RPC for placing newly created replicas.

4.3.2.1 Determining the Replica Placement Cost (RPC)

For data-intensive jobs, the replication scheme should make sure the file replica is placed closer to the jobs schedules, where the data required for running the jobs can easily be accessed [101]. The replicating mechanism proposed in this thesis for these types of data-intensive jobs is based on the “replication cost” of placing the data for the jobs, where the data could be obtained easily. Replication cost here does not refer to monetary cost, but the estimated time to copy all the files required for the job into a local store or storage facility so that the relevant jobs can access the files directly. From a set of sites that are suitable for hosting the replica files (according to site availability and workload constraints), the replica file is placed to site that offers minimum replication cost. The RPC is the total cost of transferring a file from the underlying site to the remote locations, which is determined by the file access cost and the distance between replica sites, which is computed according to Equation 4.3, based on the work in [54].

$$RPC_{si} = \frac{\sum_0^n FV_{si} * FTT * D\left(Source, destination\right) file_i}{m} \tag{4.3}$$

Where,

n = total number of sites within the grid,

m = number of sites that request the replica from the given location,

FV_{si} = File value in respect to a particular site, which is determined according to the following formula outlined by the research in [26].

$$FV_{si} = \frac{FileValue}{NoR_{si}} \tag{4.3.1}$$

Where,

FileValue is the file value in respect to the DGF system as a whole, which is determined from Equation 4.7

NoR_{si} = number of requests for the popular file from a specific site, *s*

$D\left(x, y\right) file_i$ = the distance between replica site and the site requesting $file_i$ measured in hops; number of physical elements along the path between source and destination site. The next subsection illustrates how to compute file transfer time (FTT).

4.3.2.2 Calculating the File Transfer Time (FTT)

FTT is defined as the data transmission via a wide area network, which depends on the network bandwidth and the size of the file. It is computed by the following equation [40].

$$FTT = \frac{FileSize}{Bandwidth} \tag{4.3.2}$$

Where,

FileSize is the size of $file_i$ whose *FTT* is to be determined, and *Bandwidth* is the network Bandwidth between the host site of $file_i$ and the site from where the data is retrieved.

4.3.2.3 Illustrations on RPC computations for Replica Placement.

The *RPC* is computed from access cost (*FV & FTT*) and site distance (*D*), as expressed by Equation 4.3. This subsection illustrates how the proposed mechanism computes *RPC* for determining appropriate sites to place newly created replicas. From equation 4.3, *RPC* is computed using FV_{si}, *FTT* and distance between sites. For illustrations, the parameters used for computing RPC values for the 16 example files are collected in the following Table 4.4.

Table 4.4

Example of how DRCEM computes RPC values of data files

Site ID	**File ID**	**File Size (MB)**	**Bandwidth MBPS**	**FV in respect to DGF system**	NoR_{si}	**FV**$_{si}$	**FTT (Milsec)**	**Site Distance (hops)**	**RPC (Milsec)**
0	File1	100	100	38.94	26	1.498	1000.00	2	115.231
1	File2	100	150	32.20	18	1.789	666.67	1	66.260
2	File3	100	622	58.90	21	2.805	160.77	5	107.371
3	File4	100	100	36.33	19	1.912	1000.00	2	201.263
4	File5	100	45	27.95	14	1.996	2222.22	1	316.825

Table 4.4 continued.

5	File6	100	250	47.04	38	1.238	400.00	1	13.032
6	File7	100	100	14.00	0	0	1000.00	5	0.000
7	File8	100	622	68.74	31	2.217	160.77	4	45.991
8	File9	100	100	26.51	10	2.651	1000.00	1	265.100
9	File10	100	150	31.08	23	1.351	666.67	1	39.160
10	File11	100	150	42.56	26	1.637	666.67	2	83.949
11	File12	100	45	38.34	30	1.278	2222.22	3	283.800
12	File13	100	622	51.73	14	3.695	160.77	4	169.727
13	File14	100	100	53.94	40	1.349	1000.00	2	67.450
14	File15	100	100	08.00	0	0	1000.00	2	0.000
15	File16	100	250	33.07	16	2.067	400.00	3	155.025

From Table 4.4, some assumptions are made regarding the *NoR* and site distance values. Regarding the *NoR* values for the individual files, its assumed that total number of requests made to a file includes both direct requests and indirect requests, which are determined by file weight (FW). A single or several other files may have made these requests. However, since there is no break down on file-to-file data requests from a real life scenario, this thesis assumed the calculated FW values for the *NoR* values. In other words, the calculated FW in Section 4.3.2.8 rounded to the nearest whole number, is used to represent *NoR* for a file for computation of RPC.

The second assumption on the site distance is that, distance is zero, if request for a popular file emanates from the same site. That is, if the source and destination sites are the same. Furthermore, due to the lack of actual data for sites distances from real life scenario, arbitrary distance values are assigned to the various sites for illustrative purposes. In other words, arbitrary distances are assigned between the sites in the absence of actual site-to-site distance amongst the various sites. However, for the implementation in this thesis, EDG test bed bandwidths values have been used as the distances between the sites.

Now, from Equation 4.3.1 and from the Table 4.4, file value for *file1* in respect to another file is computed as follows:

$$FV_{file1} = \frac{38.94}{26} = 1.497 \cong 1.50$$

$$FV_{file2} = \frac{32.20}{18} = 1.789 \cong 1.79$$

Similarly, all other file values in respect to other files are computed in the same pattern. Also, from Equation 4.3.2, FTT for the individual file is computed as follows:

$$FTT_{file1} = \frac{100}{100} = 1.00\,\text{secs}$$

$$FTT_{file2} = \frac{100}{150} = 0.67\,\text{secs}$$

From the FTT computation, *File1* has same value for size and bandwidth, thus FTT is 1.00 second. Similarly, *File2* has size of 100 MB and bandwidth value of 150 Mbps, resulting to 0.67 second. Thus, higher bandwidth results to lower FTT and lower bandwidth results to higher FTT. In addition, file size ranges from 2.5 GB to 10 GB in all the experiments conducted in this thesis. However, for convenience, the illustration uses 10 GB file size scaled down by 100%, which gives 100 MB. This has been the practice for the EDG simulations due to large files sizes [33]. The EDG simulation in OptorSim uses scaled down representation of the actual file sizes for convenience. Furthermore, since the FTT is measured in millisecond, the above value needs to be converted to millisecond as well. Thus, 1.00 Second = 1000.00 Millisecond. Similarly, FTT for all the other files is computed in the same pattern.

The number of physical elements (hops) between source and destination is used as the distance between the sites, and this is obtained from the sample site connectivity workload data in Section 4.3.2.7, Table 4.5 on page 162. The bandwidth between sites varies, which imitates the EDG test bed bandwidth connectivity. For the actual implementation, the EDG bandwidth configuration file is used as the bandwidth between the various sites. Furthermore, the bandwidths values are used as the distances between the various sites. Now, to find RPC from Equation 4.3, for number of sites ranging from 1 to *n*, the computations proceed as follows:

$$RPC_{site0} = \frac{1.498 * 1000 * 2}{26} = 115.231 \; milli\,\mathrm{secs}$$

$$RPC_{site1} = \frac{1.789 * 666.67 * 1}{18} = 66.260 \; milli\,\mathrm{secs}$$

Thus, from the table, total RPC is the sum of all the computed RPC for sites from 0, to *n*. average RPC is total RPC/number of sites.

The average RPC is used to determine the optimum RPC, as outlined by Equation 4.3.3 as follows:

$$RPC_{Average} = \frac{TotalRPC_{\forall allsites}}{NumberofSites} \qquad (4.3.3)$$

Where

ToTalRPC is the total cost of replica placement for all sites

NumberOfSites represents the total number of sites in the DGF system

Also, the optimum RPC is determined from average RPC as follows:

$$RPC_{Optimum} <= 50\%(RPC_{Average}) \qquad (4.3.4)$$

Where

$RPC_{Optimum}$ is the replica placement cost that provides the ideal cost of replication, which is determined from 50% of the $RPC_{Average}$ (Equation 4.3.3).

4.3.2.4 Determining Workloads for all Regional Sites

The workload of a site is defined as the number of requests that can be fulfilled by the underlying regional site [142]. The candidate site should not exceed a specific amount of workload that was assigned to it. For load balancing, this thesis considers lightly and moderately loaded sites for replica placement, in the context of storage space. The proposed DRCEM mechanism computes the current disk space (CDS) for all sites (Equation 4.4) and divides by the number of sites to obtain the $CDS_{Average}$ (Equation 4.4.1). Then computes the $Optimum\_CDS_{si}$ >= 75% *(Average CDS)* (Equation 4.4.2) as a proportion of CDS on the Average CDS. The Equations 4.4 to 4.4.4 are outlined as follows.

$$CDS = TotalDiskSpace - CurrentLoad \tag{4.4}$$

$$CDS_{Average} = \frac{TotalDiskSpace_{\forall allsites}}{NumberofSites} \tag{4.4.1}$$

$$Optimum\_CDS_{si} >= 75\%(\mathrm{CDS})_{Average} \tag{4.4.2}$$

$$Lightly\ Loaded\ Site > Optimum\,CDS \tag{4.4.3}$$

$$Moderately\ Loaded\ Site = Optimum\,CDS; \tag{4.4.4}$$

Lightly loaded site are those sites whose CDS is less than (>) the optimum CDS (Equation 4.4.3). Moderately loaded sites are those sites whose CDS equals (=) the value of the optimum CDS (Equation 4.4.4). Highly loaded sites contain CDS greater (>) than the optimum CDS. In this thesis, replica placement avoids sites that are highly loaded. The rationale is to control the amount of files duplicated and to balance the load amongst the regions sites.

Thus, DRCEM compares storage requirement (SR) of the file replica with current disk space for all the SEs. If the *SR* is less than or equal to (<=) *Optimum_CDS*, then places the replica file. Otherwise, replica eviction will be invoked, to create more space, to accommodate the new replica file. The next subsection explains how the proposed mechanism computes the distance between sites to compliment replica placement.

4.3.2.5 Computing Site Distance Using Bandwidth Information

D(x, y) represents network distance between sites *x* and *y*, computed using the network bandwidth information. Also, distance information is captured when a replica is checked for the first time, to reduce the cost of checking. To show the effectiveness of any dynamic replication mechanism, a site needs to be able to identify the nearest replica. This problem could be solved by using the least number of hops probes with a traceroute command. The nearest replica is one, which is the least number of steps away from the site. In case there is a tie between two or more replicas, one of them is selected randomly [142].

The distance between sites has been considered by [8] in relation to replica placement, using hop counts. For small grid installations, finding the distance may not present many difficulties. However, when dealing with larger network as the case with this work, there is a need for a robust mechanism that not just finds the distance, but finds it fast enough to narrow access time for data replicas. In this thesis, finding the shortest distance is by using a modified version of the Dijkstra's algorithm. Dijkstra's algorithm (Figure 4.3, Page 152) is used to find the shortest paths between sites in a graph, which may represent, for example, road networks for "Intelligent Map" path planning as reported by the work in [143]. In similar vein, the algorithm is used to find the shortest distance in computer networks. Dijkstra's algorithm uses data structures such as Array, Set, Stack, Heap, List, and Queue implementations to store the distance information [146], which in this case the bandwidths connecting the various sites are used as the distance information. This thesis adopts Dijkstra's algorithm based on its popularity in the literature reviewed by this research and less time complexity of *O(|E|+|V|Log |V|)*. The modified version of the Dijkstra's algorithm is shown in Figure 4.3.

```
1.  function Dijkstra (Graph, source);  //the function takes inputs graph and source
    site, then compute shortest distances between the sites
2.  use site connectivity sample workload data file;
3.  compute TLC = DLD+ILD;  //compute total logical connections
4.  compute TLC_Average; for all sites
5.       $TLC\_Average=\sum_{i}^{n} TLC \div \sum_{i}^{n} Sites$;
6.  set dist[source] ← 0  // Initialization
7.  create vertex set Q
8.  group vertices based on TLC value
9.  for each group,
10.     set source ←ILD (site_i) => TLC_Average
11.     for each vertex v in Graph:
12.         if v ≠ source
13.             set dist[v] ← INFINITY  //Unknown distance from source to v
14.             set prev[v] ← UNDEFINED  // Predecessor of v
15.                 do Q.add_with_priority(v, dist[v])
16.     While Q is not empty:  // Main loop
17.         u ← Q.extract_min()  // Removes and returns best vertex
18.             for each neighbour v of u:  // Only v that is still in Q
19.             alt ← dist[u] + length(u, v)
20.                 if alt < dist[v]
21.                     dist[v] ← alt
22.                     prev[v] ← u
23.                     do Q.decrease_priority(v, alt)
24.             End if
25.         End for
26.     End while
27.   End if
28.  End for
29.  End for
30.  Return dist[ ], prev[ ]
31. //Iterative deepening depth-first path-finding (IDDF)
32. S ← empty sequence   // list of vertices
33. u ← target
34. while prev[u] is defined:  // Construct the shortest path with a stack S
35.         insert u at the beginning of S  // Push the vertex onto the stack
36.         u ← prev[u]  // Traverse from target to source
37.   insert u at the beginning of S  // Push the source onto the stack
```

Figure 4.3. Dijkstra's algorithm for finding distances between replica sites

Figure 4.3 continued.

```
38.
39.    function IDDFS(root);  //Iterative deepening depth-first search
40.    calls DLS if depth is known before hand
41.      for depth from 0 to ∞
42.      found ← DLS(root, depth);
43.         if found ≠ null
44.    returns found
45.
46.    function DLS(node, depth)  //Depth-limited search function
47.      if depth = 0 and node is a goal;
48.      return node
49.        if depth > 0
50.            for each child of node
51.             found ← DLS(child, depth−1)
52.                      if found ≠ null
53.           return found
54.    returns null
55.    END
```

From the pseudo-code in Figure 4.3, *Graph* is the set of vertices of the input graph and *source* is the starting site or vertex. If only the shortest path between *source* site and *target* site *needed to be found*, the algorithm can be terminated to stop the search after line 15 if *u* = *target* (line 20), and the rest of the algorithm is ignored. The algorithm will perform a number of iterations to find shortest paths between given vertices. Other algorithms exist such as Johnson algorithm and Floyd Warshall algorithm, with time complexities of $O(|V|^2 log|V+|V||E|)$ and $O(n^3)$, respectively, which are higher than Dijkstra's [143]. The Dijkstra algorithm exists as original and the common variant. The original variant found the shortest path between two sites, while the common variant sets a single site as the "source" site and finds shortest paths from the source to all other sites in the graph, producing a shortest paths collection, using the specified data structure [143], [146]. This thesis impalements the common variant of Dijkstra's algorithm using min-priority queue, with some modifications. The min-priority queue is implemented using heap data structures due to its low time complexities. The advantage of Dijkstra's algorithm is that it finds shortest path in $O(|E|+|V|Log(|V|))$ if a min-priority queue is used [143]. However, the algorithm fails if there is negative edge in the graph; but none exists in the case of this thesis. Thus, $F(E, V) = O(|E| + |V| \log |V|)$, which means that $f(V, E)$ is “big oh” of $(|E| + |V| \log |V|)$. This means that $f(E, V)$ is asymptotically smaller than or equal to $(|E| + |V| \log |V|)$. Therefore, in an asymptotic sense $(|E| + |V| \log |V|)$ is an upper bound for $f(E, V)$ [143]. The meaning of asymptotically smaller implies that 1 less than *logn*, less than *n*, less than *nlogn*, less than *n2* as follows: $1 < logn < n < nlogn < n2 < n3 < 2n < n!$, in that order [143].

Furthermore, “*amortized time*” is the way to express the time complexity, when an entity exhibits bad time complexity only occasionally, besides the time complexity that happens most of the time**.** Thus, *amortized time* is the average time taken per operation, if many operations are performed at an instance. Considering that priority queue permits decrease-key operation in just *O(1)* amortized time, justified its implementation along with the Dijkstra's algorithm.

The implementation of Dijkstra’s algorithm into the DRCEM mechanism, along with a min-priority queue abstract data type, gives faster computing time than using a basic queue. The min-priority queue (line 16-26) offers an abstract data type that provides three basic operations thus; *add_with_priority()*, *decrease_priority()* and *extract_min()*. Such a data structure can offer low run times due to lower time complexities, leading to faster computing time than using a basic queue [53], [55]. One of the major modifications done on the Dijkstra's algorithm by this thesis to solve the issue of finding shortest distance between replica sites, entails how to select the source site from a group of sites, to construct site-distance graph. The first modification starts from line 2 through to line 10. The second modification (line 35-58) is the integration of an Iterative deepening depth-first path-finding (IDDFS) algorithm that returns the shortest path trees after the first part of the algorithm finished execution at line 34.

As regards to the first modification, the algorithm starts by computing the total logical connections *(TLC) ← (DLD+ILD* using site connectivity sample workload data file in line 2 and 3. Line 4 computes the *TLC* average. In line 6, *source* distance value is initialized to zero, that is, distance from source site to itself is set to zero. Line 7 creates a set of vertices from connectivity workload data file, and group the vertices according to *TLC* value in line 8. Line 9 begins the first *For Loop*, which selects the site with *TLC* value greater than or equal (=>) to the *TLC_average* and assigns the site as *source* site in that group. If more than one site satisfies this condition, one site among them is selected randomly.

The second *For Loop* began with an *if* condition to compare each vertex *v* with *source* site within the group. Then, it proceeds to extract distance information for each vertex and insert the distance value of each vertex within the group into the array set *Q*, with a priority *D(v),* using a min-priority queue data structure (line 16). By initializing the source to zero, the rest of the array elements are set to infinity (∞) for the remaining sites as expressed by line 13 and 14. The main loop commences with a while *Q is not empty* construct at line 19. The code *u←vertex* in *Q* with min *dist[u]* in line 20, searches for the vertex *u* in the vertex set *Q* that has the least *dist[u]* value from the sample site connectivity workload data file (see Appendix B on page 249 for the connectivity workload data), and returns the best vertex with min value. The *length(u, v)* in line 22 returns the length of the edge joining (i.e., the distance between) the two neighbouring sites *u* and *v*. The variable *alt* on line 22 is the length of the path from the root site to the neighbouring site *v*, if it were to go through *u*. If this path is shorter than the current shortest path recorded for *v*, that current path is replaced with this *alt* path value. The *prev* array is populated with a pointer to the *next-hop* site on the *source* graph to get the shortest route to the source. Figure 4.4 shiows graph abstraction for computing distance between sites.

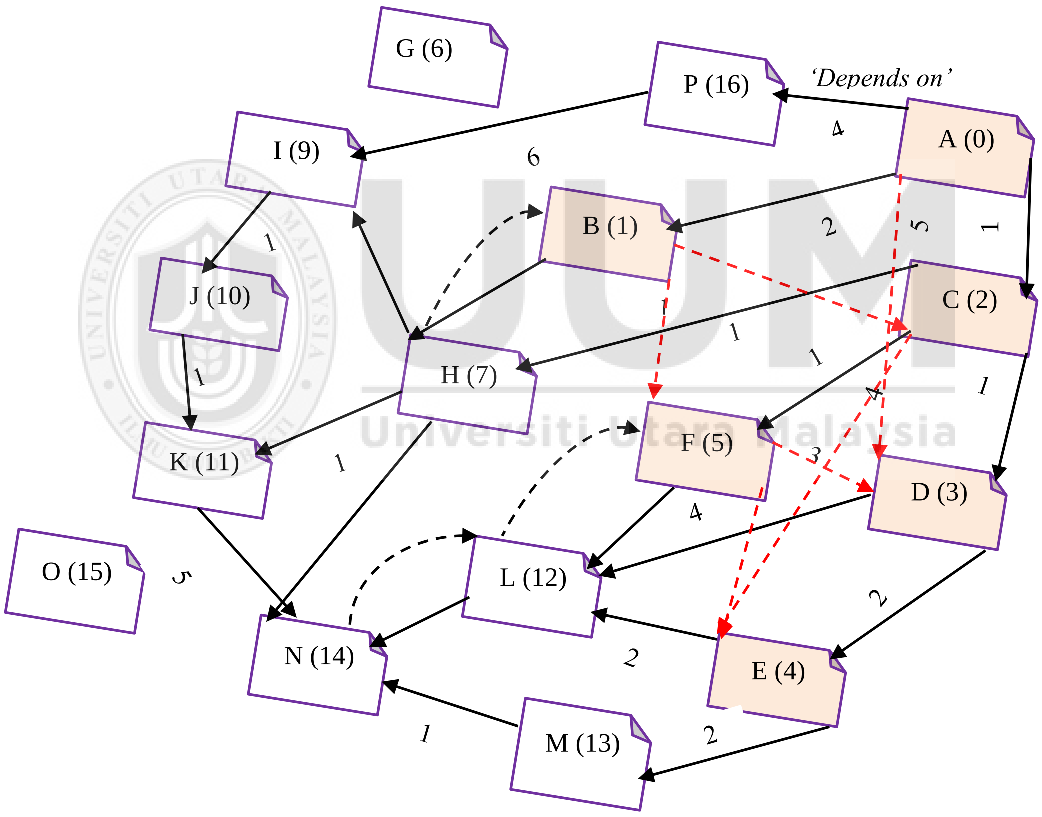


Figure 4.4. Graph abstraction for computing distance between sites

From Figure 4.4, the coloued boxes and coloured lines for sites A, B, C, D, E and F indicate an instance of a shortest-path tree from the given starting vertex A to the other vertices B ... E in the graph. Also, the numbers in brackets represent the site identifications (IDs), the numbers along the edges represent distance between respective sites. For instance, the distance between site A and B is 2, distance between A and C is 1, distance between A and D is 5. Similarly, the distance between site B and D is 2 and distance between B and E is 1. Furtermore, the distance between C and D is 5 and the distance between C and E is 4. These distances are arbitrarily set for illustration purposes. However, in the actual implementation, the bandwidths values between the various sites are used as the distances between the sites. The distance information along the paths connecting the various sites are used to find the shortest distance between the sites suitable for replica placement. As mentioned earlier, for the actual implementation, the distance values are replaced with bandwidth data connecting these sites. The bandwidth data is obtained from the EDG testbed bandwidth configuration file that came with OptorSim simulation tool v2.1 [33]. Since the various EDG sites are not connected via same bandwidth, that is, some sites have higher bandwidth than others, thus became suitable for use in the experiments as distances connecting various sites.

Site A is marked as the source site. The process is synonymous to assigning the site with highest number of logical connections, or the site with highest request for data as the source site, then finding the shortest paths from this source to all other sites in the region. It works by building a shortest-path tree from a given starting vertex to every other vertex in a graph, using *weights* allocated to each edge; in this case, the *vertices* are sites, and the *weights* are the bandwidth available between each site. In this scenario (Figure 4.4), the graph shows a construct with a vertex for each site carrying data file. Thus, from Figure 4.4, it was assumed that site *A* is the source site, and all other sites are destination sites. The numbers between the vertices indicate the bandwidth between the individual sites. Making A to be source site is arbitrary. Other sites could be made as sources sites, as well. The distance information is stored using a min-priority queue. Thus, the modifications on the Dijkstra's algorithm include the computation of TLC values, the use of priority queue *Q* for holding the set of sites or vertices *v* and the integration of the Iterative deepening depth-first path-finding (IDDFS) scheme for returning the shortest paths trees. For illustrations, this research demonstrates how distances between replica sites tare represented using graph abstraction method. The next subsection describes the second modification on the algorithm; the iterative deepening depth-first path-finding algorithm.

4.3.2.6 Finding Shortest Paths Using an Iterative Deepening Depth-First Path-Finding Algorithm.

After computing the various distances, the shortest path can be read from *source* to *target* by reverse iteration after line 34 of the algorithm. The *S* in (line 35) is the sequence list of vertices constituting the shortest paths from *source* to *target*, or the empty sequence if no path exists. The function *DLS(node, depth)* is called the depth-limited search function (line 50-59), which is used to impose depth limit on the search locations. The DLS is used in conjunction with IDDFS, to save time and conserve memory. This is achieve by deciding to only search up to the specified depth *L*, that is, the algorithm does not expand beyond depth *L* based on the DLS function. However, if solution is deeper than depth *L*, then the algorithm increases *L* iteratively to cover the depth. A more general formulation would be to find all the shortest paths between *source target* (there might be several alternative paths of the same length). Then instead of storing only a single site in each entry of *prev*, the algorithm would store all sites satisfying the relaxation condition. For example, if both *r* and *source* connect to the *target* and both of them lie on different shortest paths through the *target* (because the edge cost is the same in both cases), and then the algorithm would add both *r* and *source* to *prev [target]*. When the algorithm completes, *prev[]* data structure will describe a graph that is a subset of the original graph with some edges removed. Its key property will be that if the algorithm runs with some starting site, then every path from that site to any other site in the new graph will be the shortest path between those sites in the original graph, and all paths of that length from the original graph will be presented in the new graph. Now the iterative deepening depth-first path-finding algorithm [53] was used on the new graph, to find and return all the shortest paths between two given sites, which commences after line 35 of the algorithm.

4.3.2.7 Mathematical Framework for Replica Site Availability

Sites availability has previously been discussed in Chapter 2 (Section 2.7.5 Availability of Replica Site, page 48). Site availability is expressed as a percentage, based on a number of hours put up by sites online. A replication site is a location that is hosting the replica of the file being replicated. In this thesis, replication sites should have percentage availability of greater than or equal to (>=) 85%, prior to selecting it amongst the best locations for replica placement. While measuring availability for the various sites within the federation system, there is need to consider all of the following questions: what are the units or elements of measurement; what parameters are included in the measurement; what tools are used to collect the parameters; what algorithm is used to calculate the availability, and over what period is the availability presented and considered valid. To measure site availability, there exist two methods namely; by measuring the *response time* of a site and by measuring *percentage packet loss* of a site [19], [21]. The two methods involved using probe techniques, such as the Packet Inter-Network Groper (PING) command and observe the response time, or sending data package to individual sites and observe the precentage of packet loss at the receiving site. Due to the ambiguities of measurement involving packet loss, this thesis considers sites availability based on response times.

Site availability involves how to calculate the availability of a system that has been in operation for some times. In so doing, a mathematical framework is typically employed by network administrators to measure site availability. Such framework uses site availability record, which is based on both *uptime* and *downtim*e of the site under review. The availability of replication sites is of utmost importance in this thesis, which is computed according to Equation 4.5 [19].

$$SA = \frac{MeasuredTime - UAS}{MeasuredTime} * 100\% \qquad (4.5)$$

Where:

- *SA* = the time frame within which a site is operational and is expressed as a percentage value
- *MeasuredTime* (Uptime), is the amount of time during the period in question that the system was up

- *Unavailable Seconds* (UAS) or Downtime, is the amount of time during that same period that the system was down

Also, from the SA value, the percentage value is computed according the following Equation 4.5.1.

$$SA_{si} = SA_{si} * 100\% \qquad (4.5.1)$$

The method employed for calculating site availability is based on historical *up* or *downtime* status of the grid sites. In the real world scenario, dedicated software for failure reporting, analysis and corrective actions system, such as the Orion software deployed on BOINC platform [67] are used to capture such historical data for managerial decision making. For instance, the Orion could be set to poll for the current status of the grid sites using dedicated polling feature, and log files are read using log reader that is bundled with the Orion software [146].

Thus, to determine site availability, a background PING command is sent to sites by the software, and if the site responds to a PING within the default time interval, the site is considered up, and a value of 1 is recorded in the response time view. Otherwise, a value of 0 is recorded in the response time view, if the site failed to respond to a PING command within the default interval; thus, the site is considered down [146].

The assumption in this thesis is that Data Grid installations are equipped with such monitoring software and that sites availability records are readily available. What needs to be done is to formulate a mathematical framework that extracts the availability records for the individual sites and determine their percentage availability over a certain period. Equipped with this percentages, the proposed mechanism decides on where to place incoming data replica, in addition to considering network distance and replication cost.

Availability is also defined as the probability of the system being found in the operating state at some time *t* in the future given that the system started in the operating state at time *t = 0*. Failures and down states occur, but maintenance or repair actions always return the system to an operating state [146]. This thesis does not consider maintenance period as a separate entity. Thus, the mathematical framework defined in Equation 4.5 is maintained.

Sites availability in this thesis is determined by using existing data from the public failure time trace archive (FTA) records. FTA is an online public repository of availability time traces taken from diverse parallel and distributed system [47], [48]. Table 4.5 shows site status records used by this research to determine availability measure of individual sites for data replication considerations.

Table 4.5

Sample site status record for availability workload from TeraGrid

#event id	Component id	Site id	Platform id	Site name	event type	start time	stop time	Event end reason
0	0	927	10	"tg-login1"	1	1.15E+09	1.17E+09	NULL
0	0	928	10	"tg-login2"	1	1.15E+09	1.17E+09	NULL
0	0	929	10	"tg-login3"	1	1.15E+09	1.17E+09	NULL
0	0	930	10	"tg-login4"	1	1.15E+09	1.17E+09	NULL
0	0	739	10	"tg-c740"	1	1.15E+09	1.17E+09	NULL
0	0	748	10	"tg-c749"	1	1.15E+09	1.17E+09	NULL
0	0	962	10	"tg-s148"	1	1.15E+09	1.17E+09	NULL
0	0	234	10	"tg-c235"	1	1.15E+09	1.17E+09	NULL
1	0	234	10	"tg-c235"	0	1.17E+09	1.17E+09	NULL
2	0	234	10	"tg-c235"	1	1.17E+09	1.17E+09	NULL
0	0	233	10	"tg-c234"	1	1.15E+09	1.17E+09	NULL
1	0	233	10	"tg-c234"	0	1.17E+09	1.17E+09	NULL
2	0	233	10	"tg-c234"	1	1.17E+09	1.17E+09	NULL
0	0	236	10	"tg-c237"	1	1.15E+09	1.17E+09	NULL
1	0	236	10	"tg-c237"	0	1.17E+09	1.17E+09	NULL
2	0	236	10	"tg-c237"	1	1.17E+09	1.17E+09	NULL
0	0	235	10	"tg-c236"	1	1.15E+09	1.17E+09	NULL
1	0	235	10	"tg-c236"	0	1.17E+09	1.17E+09	NULL
2	0	235	10	"tg-c236"	1	1.17E+09	1.17E+09	NULL
0	0	230	10	"tg-c231"	1	1.15E+09	1.17E+09	NULL

From Table 4.5, the data of interest include *Site_id, Site_name, event_type, start_time* and *stop_time.* The record contains up to 260,000 Sites. However, this research is interested in 10,000 sites only. The reason for limitting to 10,000 sites is to tally with the capacity of the topology been simulated. Also, the data is raw, thus need to be analyzed for further usage. From Table 4.5, it could be seen that the data gives start time and stop time probes for each site at various time intervals. These probes are summed up for the individual site to get total availability for the site. For instance, the *event_type* contains availability (uptime) and unavailability (downtime) records for the sites, with one (1) and zero (0) indicating availability and unavailability, respectively. The *start_time* and *stop_time* indicates the start and stop of the probes, respectively. From Equation 4.5, assuming the following sites (927=A, 928=B, 929=C, 930=D, 739=E, 748=F, giving the *measuredTime* and *UAS*, the following Table 4.6 gives percentage availability for the six case sites A to F.

Table 4.6
Sample availability computation for six sites

Sites	**MeasuredTime (Secs)**	**Unavailable Seconds UAS**	**Availability %**
A	60	5	85%
B	60	7	79%
C	60	3	90%
D	60	6	82%
E	60	6	82%
F	60	5	85%

From Table 4.6, the eligible sites for replica placement include site A, site C and site F with percentage availability of 85%, 90% and 85%, respectively. These sites have attained the minimum availability specified for replica placement by this thesis. Note that, 97% availability translates to a total d*owntime* of nearly 11 days a year, while 99.91% availability is a little less than eight hours over a year. In practice, a replica file may seem to have high access frequency even though it is hosted on site with frequent failures. The chances are that the access may have been erratic most of the times due to site failures. Therefore, there is the likelihood of transferring whole or a fraction of the data file to the requesting client. In such cases, the access counter will increment, when in actual sense the file has not been fully transmitted, or data loss may have occurred along the process [146]. This situation needs to be checked and curtailed by making sure that important replica are not placed on site with frequent failure or unavailability status. Also, an attempted but failed data transfer may have, for instance, two status as shown in Table 4.7.

Table 4.7

Example of file transfer status

File	Date accessed	Status	Comment
A	12/11/2016	Complete	-
B	13/11/2016	Failed	Network failure
A	13/11/2016	Failed	Corrupt data
C	13/11/2016	Complete	-
C	13/11/2016	Failed	Unplanned maintenance
A	14/11/2016	Failed	Site stops responding

The course of failure could be due to either of the following factors: data corruption, network failure, site failure (shuts down or become unavailable), and maintenance activities at the site location. All the above factors could contribute to failure, which may cause data not to be readily available to the requesting clients. However, whatever may have been the course of failure, this thesis considers it as unavailability of the replica site. Thus, failure is modeled based on the available record obtained from the TeraGrid FTA record [47], [48].

4.3.2.8 Framework for Determining Files Weights

The data files that are used in this research are in the form of source code modality. Thus, there is a possibility to have some files that need other files to be executed or compiled. In other words, there might be a dependency relationship between files [26]. The dependency level differs from one file to another, i.e., the importance of a file to the environment is not the same. The concern is to determine the importance of a file to the whole files system, which is termed as file weight (FW). The following equation computes the file weight:

$$FW \leftarrow \sum_{i=1}^{n} FLT_i * ILD_i \qquad (4.6)$$

where,

FW = *file weight*

n: total number of files in the grid federation system,

FLT: file lifetime (computed previously)

ILD: file dependency level of other files on the given file; *ILD* is zero, if dependency does not exist.

4.3.2.9 Framework for Determining Files Logical Dependencies

Before determining file weight (FW), file lifetime and the dependency level between the various files need to be established. File lifetime has been computed in Subsection 4.3.1.1. of this chapter. In addition, this thesis concerns about the *indirect logical dependencies* of data files, in addition to the *direct logical dependencies*. Logical dependencies are considered as the type of implicit relationships typical of interactions between software objects or artifacts that evolved together over a given period [61].

Direct Logical Dependencies (DLD) are defined for pairs of files in an association rule of the form $F_1 \Rightarrow F_2$, meaning that when F_1 occurs, F_2 also occurs. In this notation, F_1 and F_2 are two disjoint sets of items. Furthermore, F_1 and F_2 are called the antecedent (left-hand-side, LHS) and the consequent (right-hand-side, RHS) of the rule, respectively. In software development process, the density of dependencies amongst sites increases the likelihood of synchronisation failures, as argued by researchers in [96]. Based on this notion, researchers in [97] proposed a more comprehensive measure based on the DLD measure, called clustering of logical dependencies (CLD), wich is also another name for indirect logical depedencies (ILD), or transitive dependences. Unlike the DLD, the CLD measure encapsulates the degree to which the files that have direct logical dependencies to the given file f_i, have indirect logical dependencies (ILD) among themselves. In this thesis, ILD is computed from the CLD measure for a proper file value evaluation. In the graph-theoretic relations, the ILD measure for a given file f_i is computed as the density of connections among the direct neighbours of file f_i [97]. Thus, the indirect logical dependency measure is mathematically expressed by Equation 4.7 [97].

$$\mathrm{ILD}(f_i) \leftarrow \frac{2|\{e_{jk}\}|}{k_i(k_i-1)} \tag{4.7}$$

where

k_i is the number of files or neighbours that a particular file f_i is connected to, through logical dependencies

e_{jk} is a link between files j and k which are neighbours of file f_i.

The work of researchers in [26], considered only the direct logical dependencies between files, in a bid to find file value (FV). Thus, the dependency measure did not consider the links between neighbouring sites, meaning that it did not capture the *ILD* amongst the files. Thus, ignoring the *ILD* may not serve the level of dependency measure needed to provide the actual file value (FV). To go round the problem, in addition to capturing the direct links between file f_i and neighbouring sites (J, k), this thesis consisders the links between the neighbours (j, k) themselves into considerations, while determining the associated file dependencies.

In the context of this study, a logical dependency from a file f_2 to another file f_1 is denoted by $F_1 \Rightarrow F_2$, that is, an association rule in which the antecedent and consequent are both singleton sets containing f_1 and f_2, respectively. It is also assumed that both f_1 and f_2 may have their neighbours, which have to be taken into considerations.

Thus, from Equation 4.7, the ILD values for the example files A to P are computed for illustrations here. The values of this measure range from 0 to 1. Also, the total logical conections (TLC) are computed using data on Figure 4.3. The *TLC* values are used by the Dijkstra's algorithm to speed up the process of finding shortest distance. Table 4.8 illustrates the omputations of *DLD, ILD* and TLC for the 16 data files in our example.

Table 4.8

Indirect logical dependencies computations for 16 sites

File_ID	**Number of direct neighbors that file f_i is connected to (k_i)**	**Number of indirect neighbors connected to file f_i via j & k (e_{jk})**	**Total logical connections (TLC) = DLD+ILD**	**ILD measurement** $ILD(f_i) \leftarrow \frac{2\lvert\{e_{jk}\}\rvert}{k_i(k_i-1)}$
A	3	5	8	0.536
B	3	6	9	0.597
C	4	6	10	0.299
D	3	5	8	0.536
E	3	5	8	0.536
F	3	6	9	0.597
G	0	0	0	0.000
H	3	3	6	0.366
I	2	2	4	0.693
	3	3	6	0.366
K	3	7	10	0.648
L	4	7	11	0.324
M	2	4	6	0.693
N	3	6	9	0.597
O	0	0	0	0
P	3	3	6	0.366

As mentioned earlier the total logical conections (TLC) are computed using data on Figure 4.3. The *TLC* values are used by the Dijkstra's algorithm to speed up the process of finding shortest distance. To simplify the process of finding the shortest distance, the set of vertices in Figure 4.4 are grouped based on their respective *TLC* values and a subset is created for each group. Equations 4.7.1 and 4.7.2 are used to compute *TLC* and *TLC_Average*, respectively.

$$TLC \ = \ DLD + ILD \tag{4.7.1}$$

$$TLC\_Average \ = \sum_{i}^{n} TLC \div \sum_{i}^{n} Sites \tag{4.7.2}$$

Where *DLD* refers to number of direct neighbours of $site_i$,
ILD refers to number of indirect neighbours of $site_i$.

Note that *ILD* differs from *ILD* measure, in that the later is a fractional value from 0 to 1, whereas the former indicates number of indirect links to $site_i$. For each group, the site with *ILD* value => *TLC_average* is set as the source site. The value for *TLC_average* is obtained by adding all the *TLC* values and divide by the number of sites. In this thesis, graph abstraction [147] is used to indicate components dependencies amongst the data files within the DGF system using the ILD values computed from Equation 4.7. Graph abstraction method exemplifies a simple logic for articulating file dependencies on graph sites. One of the most widely used applications of the graph abstraction in computing discipline is to track components dependencies. For instance, dependency tracking on the compilation process for all files in application programs that are developed on a daily basis. These dependencies manifest inside programs used for developing tailored packages, such as Netbeans Integrated Development Environment (IDE). The tracking helps in minimising the number of files that must be recompiled or adjusted, after effecting some changes to the initial package [147]. Similarly, dependency tracking for data replicas in DGF system helps to identify the relevance or importance of a file in relation to other files within the system. To illustrate how file weight/importance is calculated, suppose that files A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, and P exist, and exhibits some dependencies on one another, their dependencies at time *t-1* is demonstrated on the graph abstraction shown in Figure 4.5.

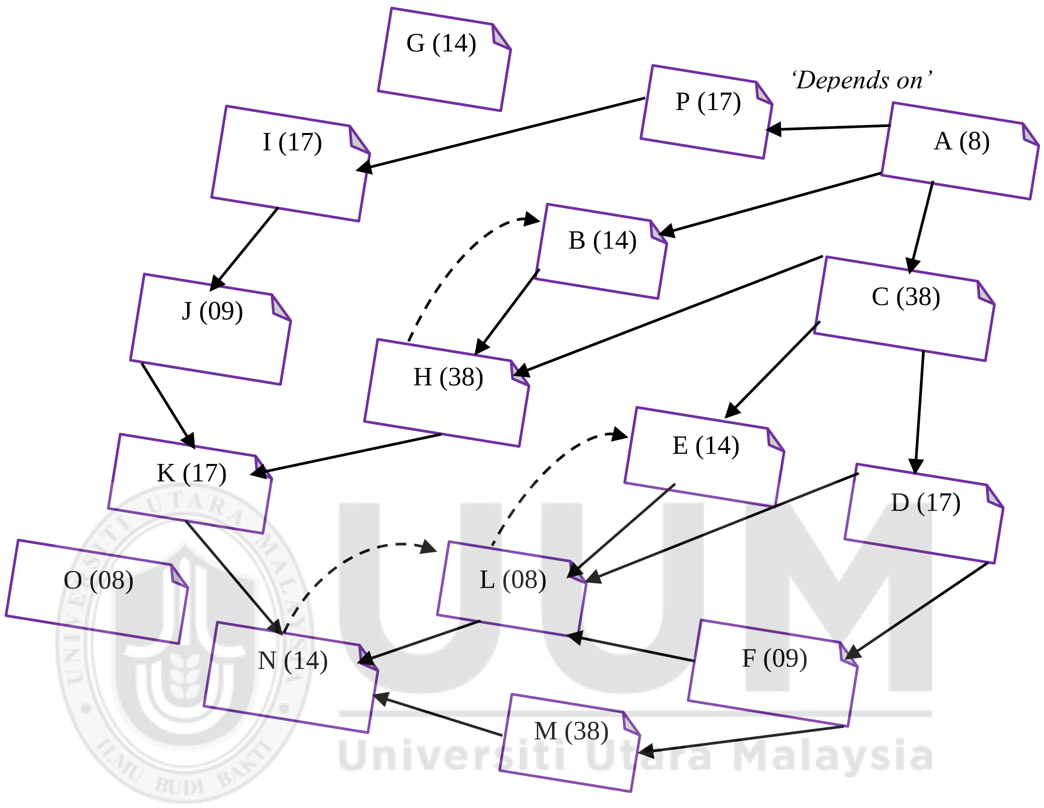


Figure 4.5. Graph abstraction showing dependencies amongst 16 data files

From Figure 4.5, the graph abstraction shows a construct with a vertex for each data file. The arrow lines in the graph indicate which files are dependent on other files. For instance, the forward arrow means file A is used by file B, C and P, while the opposite direction of the arrow means that file B, C and P depend on file A. The choice of which direction to point the arrows is somewhat arbitrary. However, it is important to note that the arrows mean *used by*, while the opposite directions mean *depends on*. The computed ILD relations in Table 4.8 (Page 167) means that file L is more important than other files as there are four files

(E, F, D, N) that directly depend on file L, while the rest have less than four. Regarding FLT, file C, H, and M are more important than the rest, with FLT of 38 each. However, the next level of decision is made after computing file weight (FW), which is determined by file lifetime and ILD dependents amongst sites. Hence, the FW for files A, B, C, D, to P are obtained according to Equation 4.6, and the computed values for FW are shown in Table 4.9.

$$FW \leftarrow \sum_{i=1}^{n} FLT_i \times ILD_i$$

$FW(FLT_A, ILD) = (0.366*17) + (0.299*38) + 0.597*14 = 25.94$

$FW(FLT_B, ILD) = 0.366*38) + (0.536*08) = 18.20$

Table 4.9

Computed file weight values for 16 data files

File ID	FLT	ILD measure	Neighbouring sites	ILD_i due to neighbouring sites	$FW \leftarrow \sum_{i=1}^{n} FLT_i * ILD_i$
A	08	0.536	P, C, B	0.366*17, 0.299*38, 0.597*14	25.94
B	14	0.597	H, A	0.366*38, 0.536*08	18.20
C	38	0.299	A, E, D	0.536*08, 0.536*14, 0.536*17	20.90
D	17	0.536	C, L, F	0.299*38, 0.324*08, 0.597*09	19.33
E	14	0.536	C, L	0.299*38, 0.324*08	13.95
F	09	0.597	D, L, M	0.536*17, 0.324*08, 0.693*38	38.04
G	14	0.000	0	0	0.00
H	38	0.366	B, C, K	0.597*14, 0.299*38, 0.648*17	30.74
I	17	0.693	J, P	0.366*09, 0.366*17	9.51
J	09	0.366	I, K	0.693*17, 0.648*17	22.80
K	17	0.648	J, H, N	0.366*09, 0.366*38, 0.597*14	25.56
L	08	0.324	E, F, D, N	0.536*14, 0.597*09, 0.536*17, 0.597*14	30.34
M	38	0.693	F, N	0.597*09, 0.597*14	13.73
N	14	0.597	K, L, M	0.648*17, 0.324*08, 0.693*38	39.94
O	08	0	0	0	0.00
P	17	0.366	A, I	0.536*08, 0.693*17	16.07

From the computed FW values, file F and N proved to be more important than others, due to the indirect logical dependencies around file F and N, with 38.04

and 39.94 file weights, respectively. The next stage determines the file value, which is the peak of economic importance associated with data files, whose magnitude is determined from the file lifetime (FLT) and the file weight (FW).

4.3.2.10 Mathematical Framework for Determining File Value

The file value for each data file within the federation is computed by taking into account both users' behaviour in accessing the files and files' behaviour in relation to other files. Thus, *file lifetime* (FLT) and *file weight* (FW) are used to compute the file value (FV). As mentioned in Chapter Three, the age parameter proposed by the work of researchers in [139] is not incorporated in our framework due to time constraints and lack of enough viable data on file age (FA) from the site status record at our disposal. File value is used to indicate the capacity of demands on a file in the DGF system, based on which, certain file(s) will be replicated or evicted by the proposed mechanism. The higher the file value, the more relevant the file is. The three parameters (FLT, FW and FV) described file's behaviour in relation to both users, other files, and the grid system as explained by researchers in [139] reporting from [26]. These parameters are further broken down into five parameters as follows:

a. File-to-User relationship - the behaviour of a file being requested by users, and notes the change to this request; whether is an increase or decrease change. The File-to-User relationship provides the lifetime (FLT) of a file
b. File-to-File relationship - behaviour of a file directly or indirectly requesting other files and is represented by file weight (FW)
c. File-directly-to-File relationship - behaviour of a file directly requesting other files and is represented by direct logical dependency (DLD)

d. File-indirectly-to-File relationship – denotes a file indirectly requesting other files and is represented by indirect logical dependency (ILD)

e. File-to-Grid relationship – this denotes the age (FA) of a file, in the entire grid system.

Thus, in this thesis, file value (FV) is computed from the following parameters; file lifetime and file weight (FLT*ILD) [26], which is mathematically expressed by the following Equation 4.8:

$$Filevalue\ (t, f) = FLT\ (t, f) + FW\ (t, f) \qquad (4.8)$$

Where

FileWeight = *FLT* * *ILD*, but *FLT* and *FW* have been computed previously.

The computed file value (FV) from Equation 4.8 is presented in Table 4.10.

Table 4.10
Calculating file values for 16 data files

File ID	File weight (FW)	File lifetime (FLT)	File value (FV)
A	25.94	08	38.94
B	18.20	14	32.20
C	20.90	38	58.90
D	19.33	17	36.33
E	13.95	14	27.95
F	38.04	09	47.04
G	0.00	14	14.00
H	30.74	38	68.74
I	9.51	17	26.51
J	22.80	09	31.08
K	25.56	17	42.56
L	30.34	08	38.34
M	13.73	38	51.73
N	39.94	14	53.94
O	0.00	08	08.00
P	16.07	17	33.07

From Table 4.10, computed file values indicated that file H is more important than the rest of the files with FV of 68.74, followed by file C with FV of 58.90. It is clear that even though file F has high FW of 38.04, the small value of FLT brings down the corresponding computed FV to 47.04, thereby making it the 3rd most important file in this scenario.

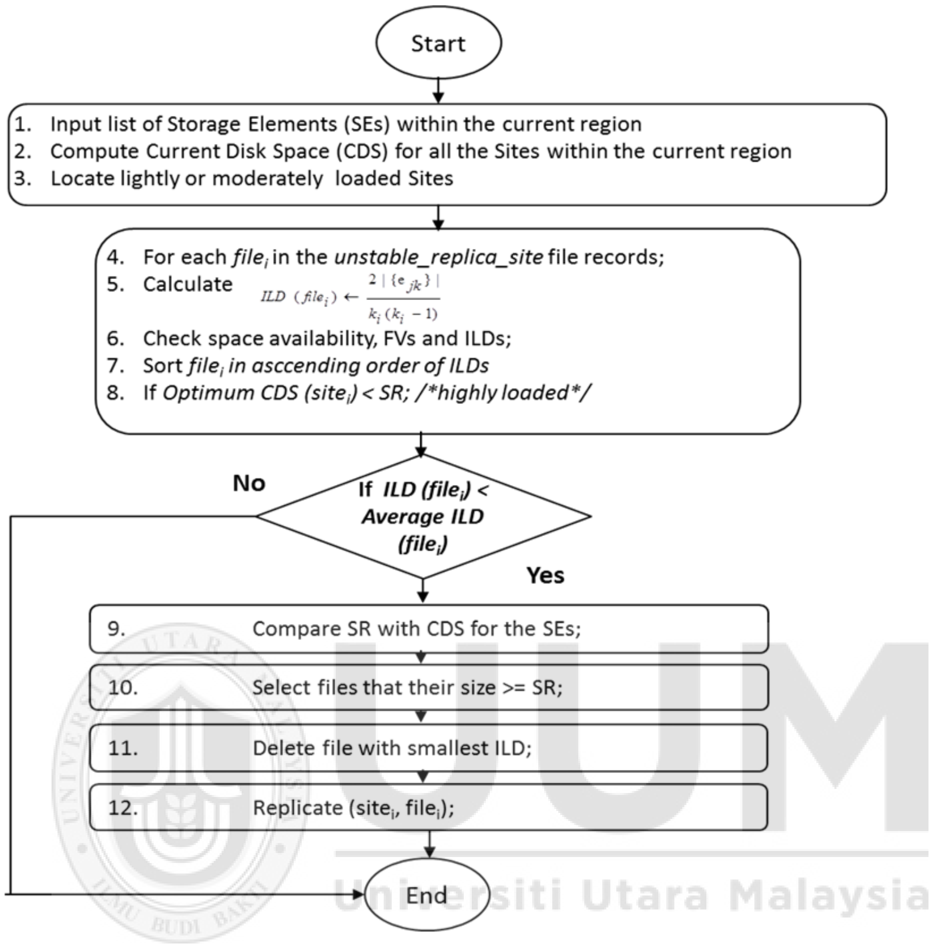
4.3.3 The Dynamic Replica Eviction Scheme

The dynamic replica eviction scheme (DRES) evicts insignificant (unwanted) replicas from the *Unpopularity_List (Unstable_replica_site)* file records to obtain more space for the newly created files replicas. In addition, the DRES scheme part of the DRCEM mechanism considers file dependencies while deciding which file to evict, so that files with high dependency level are not carelessly evicted from the system.

The unwanted replicas are the replicas of the file that get negative entry for the projected number of replica (*PNoR*)values, and the system decides to delete them to reduce storage cost. Therefore, the location sites from which the replica to be deleted are required. The calculations used here are the same as the ones used in the dynamic replica placement scheme (DRPS), but with some minor modifications. The scheme uses the *Unstable_replica_site* file record, and then performs the following set of activities:

i. Get list of storage elements from the current region, and their CDS
ii. Locate all the lightly and moderately loaded sites
iii. Compute the ILD for all the files in the Unstable_replica_site file record
iv. Isolate sites with minimum CDS (highly loaded sites)
v. Isolate files with minimum ILD (less dependability) factor to other files, whose sizes >= SR
vi. Delete files with minimum ILD from the sites with minimum CDS (highly loaded sites)
vii. Place popular file replicas on the newly created spaces within the SEs

The following Figure 4.6 shows the logical flowchart of the dynamic replica eviction scheme (RDES).



Start
1. Input list of Storage Elements (SEs) within the current region
2. Compute Current Disk Space (CDS) for all the Sites within the current region
3. Locate lightly or moderately loaded Sites
4. For each $file_i$ in the *unstable_replica_site* file records;
5. Calculate $ILD\ (file_i) \leftarrow \frac{2\,|\{e_{jk}\}|}{k_i\,(k_i - 1)}$
6. Check space availability, FVs and ILDs;
7. Sort $file_i$ in asccending order of ILDs
8. If Optimum CDS ($site_i$) < SR; /*highly loaded*/
No
If ILD ($file_i$) < Average ILD ($file_i$)
Yes
9. Compare SR with CDS for the SEs;
10. Select files that their size >= SR;
11. Delete file with smallest ILD;
12. Replicate ($site_i$, $file_i$);
End

Figure 4.6. Flowchart for Dynamic Replica Eviction Scheme (DRES)

For removing a file from the site, the ILD value of all the files stored on the site is compared with one another. The files with less ILD values, which consequently have less weights are deleted and replaced with the newly created files replicas as seen on the flowchart for the DRES scheme. Having discussed all the schemes that collectively interact to form the DRCEM mechanism, the next subsection outlines the overal algorithm for the mechanism.

4.3.4 The Complete Algorithm for the Proposed DRCEM Mechanism

The previous Section 4.3 (Subsections 4.3.1-4.3.3) explained the detailed design of the DRCEM schemes. The following Figure 4.7 gives the outline of the complete algorithm for the proposed DRCEM mechanism, which encapsulates all the schemes described above.

```
Input: Number of access for each file ($NoA(file_i)$), number of access intervals $t$,direct logical dependencies (DLD) indirect logical dependencies (IDL), file size, bandwidth between sites, number of copies of existing replicas of each file ($ENoC(file_i)$);
Output: Creating copies of popular files replicas for placement to suitable locations,
Procedure:
1.  /* Dynamic Replica Evaluation and Creation Scheme (DRECS)*/
2.  Use file access history workload data file;
3.  for each file in the grid regions, given the number of access within the past time intervals (T1, T2…Tn);
4.  /* evaluate individual files  to determine popular and unpopular files */
5.  Find alpha  $\alpha \leftarrow \frac{\sum_{i=0}^{T-1} \alpha_i}{T}$  ; /*average increase/decrease rate for all intervals*/
6.  Calculate  $FLT \leftarrow a_f^{T+1} \leftarrow a_f^{T} * \mathrm{e}^{\alpha}$ ; /file life time, which is also file access frequency for the upcoming time intervals*/
7.  Calculate $ILD(f_i) \leftarrow \frac{2\,|\{e_{jk}\}|}{k_i(k_i-1)}$ ; /Indirect logical dependencl for individual files*/
8.  Calculate $FW \leftarrow \sum_{i=1}^{n} FLT_i \times ILD_i$ ; /* file weight */
9.  Calculate $FV\ (t, f) \leftarrow FLT\ (t, f) + FW\ (t, f)$ ; /*file value*/
10. Calculate $FS_{<users>} \leftarrow \frac{FV}{\sum \forall_{<files>} FV}$ ; /*file strengths (FS) regarding users*/
11. Calculate $FS_{<system>} \leftarrow \frac{ENoC}{\sum_{\forall <files>} ENoC}$; /*file strengths (FS) regarding grid system*/
```

Figure 4.7. Algorithm for DRCEM Data Replication mechanism

Figure 4.7 continued.

```
12: Calculate   $PNoR \leftarrow \frac{(FS_{<users>} - (TH * FS_{<system>})) * \sum\forall_{<files>} ENoC}{TH}$;   /*
    Projected no of replicas*/
13: /* Compare file PNoR (which is a measure of file strengths) with zero */
14:     if ( PNoR > 0) then
15:         Add $file_i$ to the Popularity_List File Records
16:     else if ( PNoR < 0) then
17:         Add $file_i$ to Unpopularity_List File Records  //for eviction at later time
18:     else if ( PNoR = 0) then
19:         Add $file_i$ to the Stability_List.
20:
21:     /* Replica Placement Scheme begins here*/
22:     /* Use the Unpopularity list File Records*/
23:     for each data file in the Unpopularity_List
24:         get all the sites containing $file_i$
25:         for each site in the list
26:             Calculate $FV_{si} = \frac{FileValue}{NoR_{si}}$ ;
27:             Calculate $FTT = \frac{FileSize}{Bandwidth}$;
28:             Calculate Distance between Sites D (Source, destination) $file_i$ ;
29:             /* Use Site Availability Workload Records File (Appendix C) */
30:             Calculate $SA = \frac{MeasuredTime - UAS}{MeasuredTime}$; /* Site Availability */
31: ....../* Use Site Connectivity Workload Records File (Appendix B) */
32:     Calculate $RPC_{si} = \frac{\sum_1^n FV_{si} * FTT * D\ (Source, destination)\ file_i}{m}$;
33:     Compute: $RPC_{Average} = \frac{TotalRPC_{\forall allsites}}{NumberofSites}$ ;
34:     $RPC_{Optimum} <= 50\%(RPC_{Average})$ ;
35:         Arrange $site_i$ in descending order of $RPC_{Optimum}$;
36:         while ( PNoR ($file_i$) < 0); // delete files with negative PNoR
37:         delete $file_i$ from $site_i$ ;
38:         PNoR ($file_i$) + +;
39:     Repeat for all regions in the DGF system
40:     Break //DRECS ends here after determining the required repicas to create

41: /* Dynamic Replica Placement Scheme (DRPS) */
42: /* Use Popularity_List File Record */
43: for each file in the Popularity_List File Records
44:     for each site in the DGF regions
45:     check $site_i$ against the requirements for hosting $file_i$
46:     /* the requirements s are RPC, Current Load and Site Availability */
```

Figure 4.7 continued.

```
47:        Calculate $FV_{si} = \frac{FileValue}{NoR_{ni}}$ ;
48:        Calculate $FTT = \frac{FileSize}{Bandwidth}$;
49:        Calculate Distance between Sites D (Source, destination) $file_i$ ;
50:        /* Use Site_Availability work load record file */
51:        Calculate $SA_{si} = \frac{MeasuredTime - UAS}{MeasuredTime}$ ; /* Site Availability */
52:        Percentage $SA_{si} = SA_{si} * 100\%$
53:        Calculate $RPC_{si} = \frac{\sum_1^n FV * FTT * D\ (Source, destination)\ file_i}{m}$ ;
54:        Calculate $RPC_{Average} = \frac{TotalRPC_{\forall allsites}}{NumberofSites}$ ;
55:        Compute $RPC_{Optimum} <= 50\%(RPC_{Average})$ ;
56:        Calculate $CDS = TotalDiskSpace - CurrentLoad$ ;
57:        Calculate; $CDS_{Average} = \frac{TotalDiskSpace_{\forall allsites}}{NumberofSites}$ ;
58:        $Optimum\_CDS_{si} >= 75\%(CDS)_{Average}$ ;
59:        Arrange the sites according to increasing order of optimum CDS values;
60:            $Lightly\ Loaded\ Site > Optimum\ CDS$ ;
61:            $Moderately\ Loaded\ Site = Optimum\ CDS$ ;
62:        Arrange the sites according to decreasing order of optimum RPC values;
63:        Arrange the sites according to decreasing order of percentage %SA value;
64:        get storage requirement (SR) from the size of $file_i$ ;
65:        /*Initialize replica placement with load balancing*/
66:  /*Compare SR of each $file_i$ replica with optimum CDS for all the SEs */
67:  if the SR < optimum CDS then
68:      if RPC for $site_i$ <= optimum RPC
69:      add $site_i$ to the selected_replica_site list;
70:      Sort [$site_i$, $file_i$] in descending order;
71:          for each site in the sorted list;
72:          while (PRNoR ($file_i$) $\neq 0$); AND Availability($site_i$) >= 85%
73:              add ($site_i$, $file_i$) to the best_replica_site list;
74:              replicate ($site_i$, $file_i$);
75:              End
76:          End
77:      End
78:  End
79:  Repeat for all regions in the DGF system;
80:  End
81:  Break
```

Figure 4.7 continued.

```
82:  /* Dynamic Replica Eviction System (DRES) */
83:   /*Invoked when there is not enough storage space for replica placement*/
84:   for each $file_i$ in the best_replica_site or unstable_replica site records;
85:    Calculate $ILD\ (file_i) \leftarrow \frac{2|\{e_{jk}\}|}{k_i(k_i-1)}$;
86:      Check space availability, FVs and ILDs for $site_i$, $file_i$;
87:      Sort $file_i$ in asccending order of ILDs
88:        if optimum $CDS_{si}$ < SR ($file_i$); /*highly loaded*/
89:          if ILD ($file_i$) < Average ILD ($file_i$); /*less file dependability*/
90:            Select files that their size >= SR;
91:            Delete files with smallest ILDs;
92:        Replicate ($site_i$, $file_i$);
93:   Repeat for all regions in the DGF system
94:   End
```

4.3.4.1 Explanations on the Unique Features of for Proposed DRCEM Mechanism

The DRCEM mechanism encapsulates some unique features, which makes it performs differently from the existing mechanisms. These features include an enhanced method of file evaluation, which considers logical dependencies as well as the importance of a file to both users and the DGF system as a whole. Also, another unique feature is the site availability parameter, which determines the percentage availability of replica sites, prior to placement of newly created replicas. Furthermore, DRCEM finds replica placement cost for all sites, such that sites with minimum replica placement costs are considered for replica placement within the DGF system. Thus, from Figure 4.7, DRCEM commences operation between line 2 and 9, with evaluation of popular files based on FLT, ILD, FV, and FW, which is a strong improvement over DRCM that evaluates popular files based on FLT, FV, and FW only, without considering inter-dependencies amongst the replica files. Between line 10 and 11, DRCEM identifies the file strength for users, which is denoted by FS_{user} and file strength for system that is denoted by FS_{system}. Based on FS_{user}, FS_{system}, and threshold value (TH), the projected number of replicas (PNoR) is calculated as shown in line 12. File strength indicates how important a file is to the users and the system in general.

Regarding the users, file strength indicates how frequently a file is accessed by users of the system. For the system, file strength indicates how frequently a file is accessed by other files in the system. Then, based on *PNoR* value, the files are categorised into three groups as shown between lines 14 to 20, for determining the required number of replicas to either create or evict. The computation for projected number of replicas is similar to DRCM and ELALW mechanisms, with a strong additional parameter for indirect logical dependencies parameter for evaluating file value. In this thesis, the TH value is set to 50%, so that number of replication should not occupy more than 50% of the storage.

Comparing with ELALW mechanism shown in Figure 2.10, page 82 (between lines 2 and 10), and file evaluation depends on a number of accesses, which reflects users' perspective only. It is worth mentioning that computation time of DRCEM in this phase is more than computation time of ELALW, as it determines the group of files to be replicated or evicted based on their indirect logical dependencies. This additional activity tends to draw heavily on the computing element usage compared to the existing mechanisms. Nevertheless, DRCEM accelerates the process of placing replicas, by controlling unwanted replication, which improves the system's performance significantly. Also, DRCEM invokes the replica eviction scheme (DRES), to evict the unwanted replicas in replica placement decision phase, before deciding where to place the newly created replicas. Contrary to ELALW that has no separate replica eviction function, the DRES scheme stands a unique feature of the DRCEM mechanism. The commands between line 21 and 27 in Figure 2.10, shows the steps for deleting replicas in the case of insufficient storage space in ELALW mechanism.

In line 26 there is *While Loop* means that deleting replicas will be continuing until free space is available. The DRCEM's DRECS scheme sets stage for the dynamic replica placement scheme (DRPS) by computing the essential parameters required for placing new replica as seen between line 22 and 36. These include computation of sites distances, sites availabilities and replica placement cost. In addition, unnecessary replicas are evicted from the regions to create more space for incoming replica files as expressed by lines 37-40.

Also, the dynamic replica eviction scheme (DRES) is helpful in the replica placement process in case of insufficient storage, so that less significant replicas (less value, less weight, and low dependency level) are evicted, so that high valued files are not carelessly evicted from the sites as outlined between line 84 and 96. The Current Disk Space (CDS) in line 57 is a measure of the available storage capacity of a site as well as load balancing factor, and is calculated as follows [137]:

$$CDS = \mathcal{S}_{reg} - \mathcal{S}_{usage} \tag{4.9}$$

Where; $\mathcal{S}_{reg}$ is maximum storage capacity of a site and $\mathcal{S}_{usage}$ is storage space occupied by resources on the site.

The optimum CDS is computed as 75% of the average CDS, to ensure that only lightly and moderately loaded sites are considered for replica placement. The 75% is considered by this research as optimum considering that no SE will be 100% free. If the CDS is higher than the size of the file to be replicated, then the replica is placed on the site. If however, the site has insufficient space, other sites are contacted within the region. If all the sites within the region returned a CDS less than the file size, then the mechanism will delete old files or replicas from the site. For computing the required number of replicas, the DRCEM mechanism uses threshold value (TH), file strengths in respect to both the DGF system and the users of the system, to compute the desired number of replicas to create or to evict from the system. The TH value is set to 50% of the system resources in order to control the number of files replicated. For computing the required number of replicas using PNoR values , there may exist three possible scenarios; these are:

Scenario 1: if the $PNoR > 0$, then the system will replicate *PNoR* replicas of the underlying file.

Scenario 2: if the $PNoR < 0$, then the system will delete *PNoR* of existing replicas.

Scenario 3: if the $PNoR = 0$, then neither replication nor deletion is required.

4.4 DRCEM Data Replication Mechanism Implementation

In this section, the implementation of the new schemes for the proposed DRCEM mechanism along with the existing ELALW and DRCM mechanisms are explained. The code implementation and the network structure are outlined, followed by a detailed explanation on the connectivity amongst sites within same and different regions. The schemes for DRCEM, ELALW and DRCM mechanisms are programmed in Java high level programming language, and were implemented in the relevant sections of the OptorSim simulator. The next subsection explains how DRCEM was implemented into the OptorSim simulator, along with the existing mechanisms used for comparison.

4.4.1 Diagramatic Representation of DRCEM Integration with OptorSim Simulator

OptorSim is capable of simulating many aspects of the grid system, and these aspects are divided into packages, where each of which contains a collection of related classes. The diagram shown in Figure 4.8 describes the six packages within OptorSim and relationships among them.

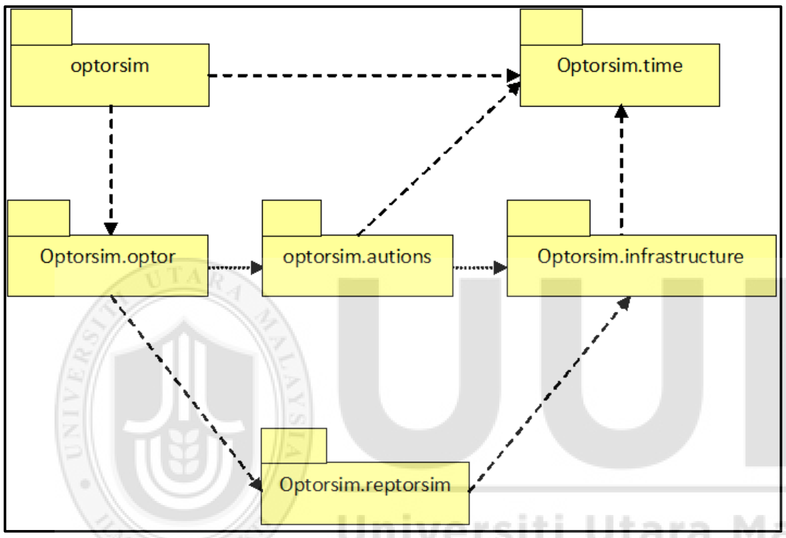


Figure 4.8. OptorSim UML showing relationships amongst the six packages

Each of the packages in Figure 4.8 could evolve to accept new classes for solving problems related to the grid systems. Starting at the lowest level, the *optorsim.time* package deals with how time is measured during the simulation. The *optorsim.Infrastructure* stimulates the underlying grid infrastructure including the network, grid sites, and the basic components of the sites: Computing Elements (CEs) and Storage Elements (SEs). The Peer-to-Peer connectivity and messaging system along with the auctioning process are contained in the *optorsim.Auctions* package. The functionality of the replica management component including Replica Location Service (PLS) is performed by the *optorsim.reptorsim* package, while the replica optimisation strategies are simulated in the *optorsim.optor* package. *Optorsim* is the highest-level package, which simulates the grid resources and users, as well as controls the Graphical User Interfaces (GUIs).

Three replication mechanisms have already been implemented in OptorSim simulator, namely LFU, LRU, and Economic models. In this research work, three additional mechanisms are implemented namely ELALW and DRCM mechanisms that are used for performance comparison, and the DRCEM, which is the proposed Mechanism.

The DRCM, ELALW and DRCEM along with the other replication mechanisms that have already been implemented in OptorSim, are both written in Java high-level programming language. The mechanisms are implemented into OptorSim simulator via the *optorsim.optor* package. The resulting modules are named as *DRCMOptimiser, ELALWOptimiser, and DRCEMOptimiser*.

The library classes for these mechanisms in effect extend the functionalities of the existing *skelOptor* class, which is also a functional part of the *optorsim.optor* package. The *DRCEMOptimiser* contains the class objects used by the DRCEM mechanism for achieving the research objectives. These classes include *+getFileValue*, *+getReplicasToCreate(), +getSiteDistance(), +getRPC(), +getSiteAvailability()* and *+getBestLocation(),* among others. The functionalities of these classes include computation of file value, obtaining file replica to create, obtaining site distance, computing the RPC, in that order. Figure 4.9 presents a UML class diagram showing the existing mechanisms (DRCM, ELALW, FLU, RLU Economic, Ecobin and Ecozip) and the proposed DRCEM mechanism implementation into OptorSim simulator.

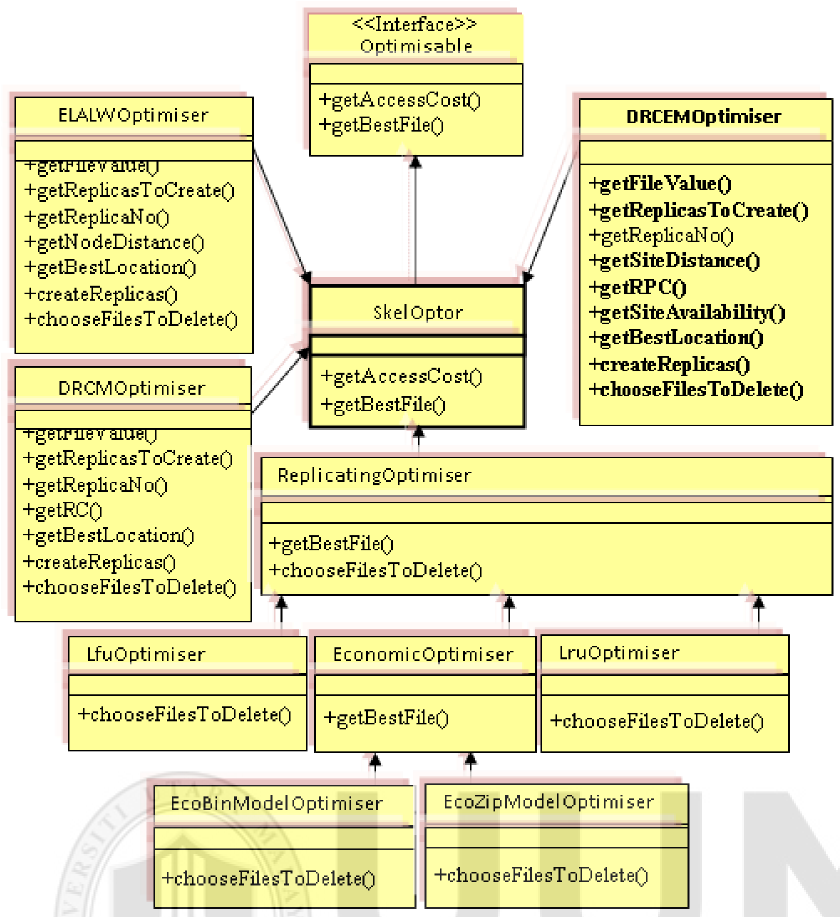


Figure 4.9. OptorSim UML class diagram showing DRCEM implementation

From Figure 4.9, the module by the extreme right side shows the objects contained in the proposed DRCEM mechanism, which also shows the main contribution of this research in the OptorSim Package. The two modules at the extreme left side show the other optimisers for DRCM and ELALW mechanism added into OptorSim package, which are used to compare the performance of the proposed DRCEM mechanism. The remaining *LfuOptimiser*, *EconomicOptimiser*, *LruOptimiser*, *EcoBinModelOptimiser* and *EcoZipModelOptimiser* are part of OptorSim package that was used for simulating job scheduling and data replication strategies on EDG test bed [119] [132]. The *optimisable* is the main interface, which relates to both the existing and newly added optimiser via the *skelOptor* class. In other words, all the optimisers extend the functionality of the *skelOptor* class.

4.4.2 Diagramatic Representation of DRCEM Simulation Processes in OptorSim

The simulation process commences when a user submits a job to the grid via the Resource Broker (RB), which in turn looks for appropriate CE to execute the job. Depending on the scheduling mechanism specified by the user, the RB then schedules the job to the CE, by following the relevant commands in the scheduling mechanism. The user usually specifies the desired scheduling mechanism in the OptorSim parameters' file before the simulation starts. The CE starts executing the submitted job by processing all the files needed to execute that job. If there are dependent files or partial replicas required to accomplishing the job, these will be fetched by the RB.

Thus, it is important that files required to accomplish jobs are located closer to the jobs that use them to save bandwidth usage and hence saves jobs completion time. This is the reason why the study considers distance between replica sites as one of the design variables for performance metrics evaluation. In the OptorSim parameters file, the access pattern defines the order by which jobs should be processed, following which the CE processes the files.

At this stage, local optimiser specified in the parameters file is invoked to find the best replica for the file. The CE then reads the file and processes it, before calling for the next file, until all the files for the job have been processed, in that order. Based on the OptorSim architecture, each site has its replica optimiser termed as a local optimiser, and its primary role is to find the best replica and replicate it in the local SE according to the chosen mechanism. The simple optimiser is used as a local optimiser that finds the best file replica and read off the required files remotely. Thus no replication occurs. In this thesis, the replication decision is made by the proposed DRCEM mechanism. At regular time intervals, DRCEM gets information of the files from Replica Catalogue (RC), which will inform the decision to replicate or evict certain files based on the outcome of the evaluation processes (see Section 4.3.1.1: Determining the popularity of data files based on access frequencies, page 130). The following Figure 4.10 shows implementation of DRCEM optimiser in OptorSim and how it interacts with relevant components to accomplish simulation processes.

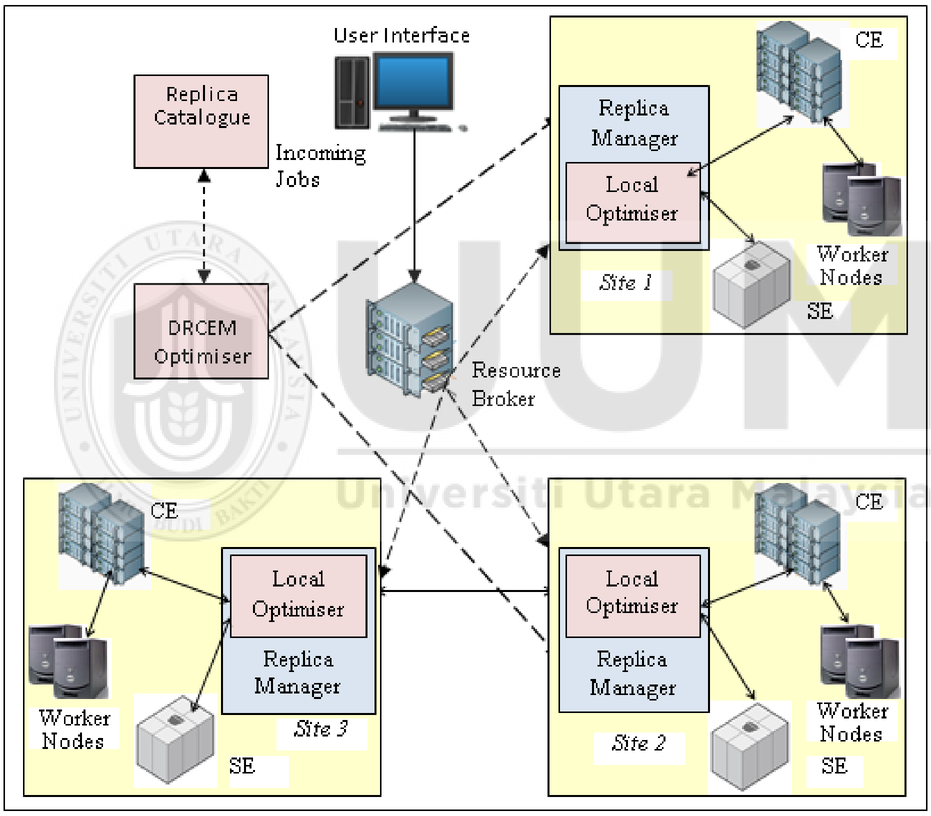


Figure 4.10. DRCEM integration into OptorSim package

As explained previously in Section 4.4.1, the integration of DRCEM optimiser into OptorSim is done via *skelOptor* class through to the *optimisable* main interface, where users can access and use the optimiser. Then a pointer is added from the replica manager to the newly added optimiser, so that it could be selected like the other existing mechanisms to execute submitted jobs. The replica catalogue (RC) of the replica manager (RM) holds mappings of logical file names to physical file names [83], stores information regarding files evaluation in the system, and then accordingly makes required replication or eviction decision, if it is necessary to do so. After new file replicas are created or evicted, the RC is updated to reflect the affected files.

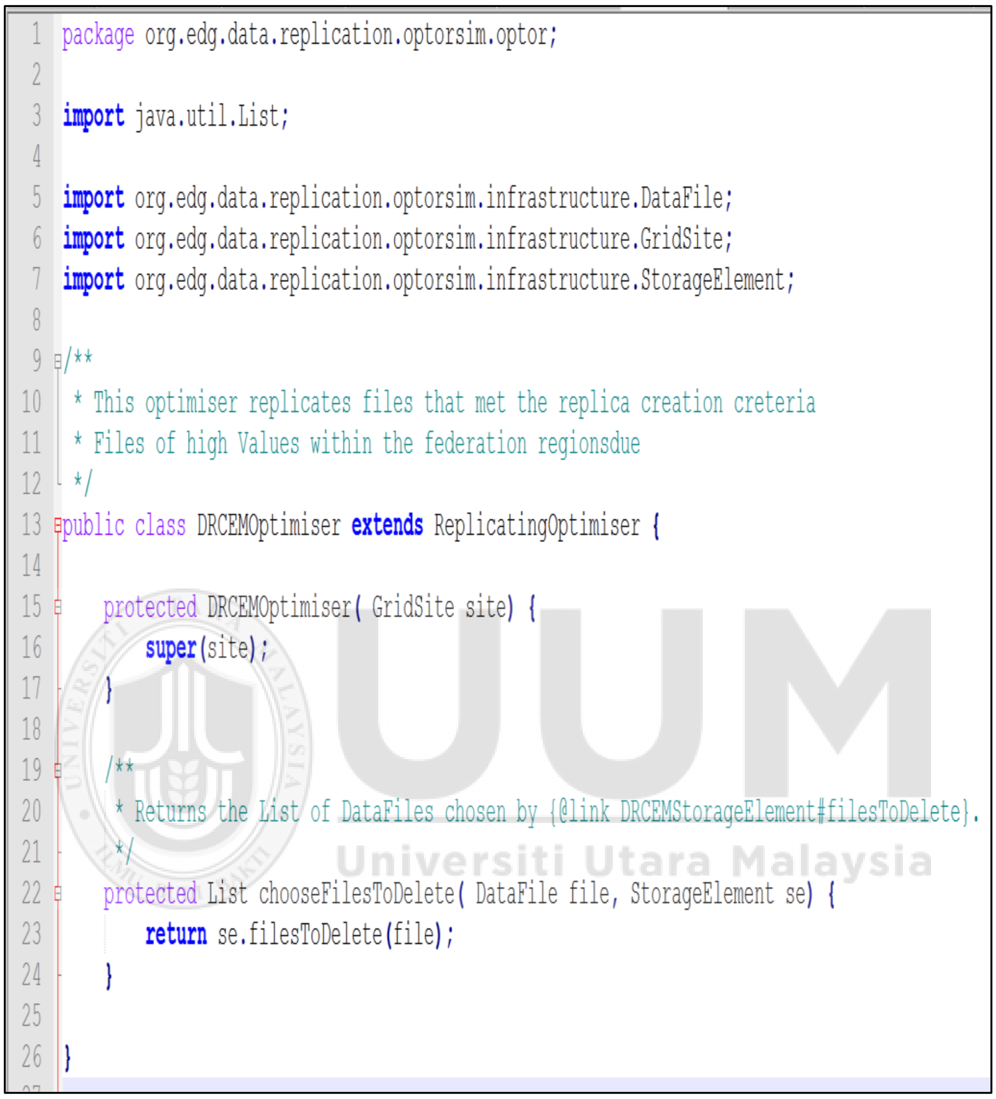
4.4.3 DRCEM Programming and Codes Integration in OptorSim Simulator

The previous subsections (4.4.1-4.4.2) explained diagrammatically, the process of integrating the DRCEM optimiser into the OptorSim simulator and the simulation processes, respectively. In this subsection, the principal implementation tasks made by this research in the process of integrating the DRCEM mechanism in OptorSim simulator are explained, which involved programming and integrating the programm codes into the appropriate sections of the OptorSim packages as indicated on Figure 4.10. The program codes for DRCEM mechanism have been validated earlier using the methods outlined in Chapter Three (Section 3.5.1, Page 100).

As stated previously in Chapter 3, the DRCEM schemes were programmed using Java high level programming and implemented in OptorSim simulator, which is managed using NetBeans Integrated Development Environment. In addition, discussion on the OptorSim simulator was given Chapter Three (Subsection 3.6.2.1, Page 109) of this thesis.

The implementation commences by developing a *DRCEMOptimiser* class for the proposed DRCEM mechanism, and integrates the optimiser into *optorsim.optor* package, via the *ReplicatingOptimiser* class. The *DRCEMOptimiser* extends the functionalities of the *ReplicatingOptimiser* class.

Figure 4.11 shows the integration of *DRCEMOptimiser* into *optorsim.optor* package, via the *ReplicatingOptimiser* class.

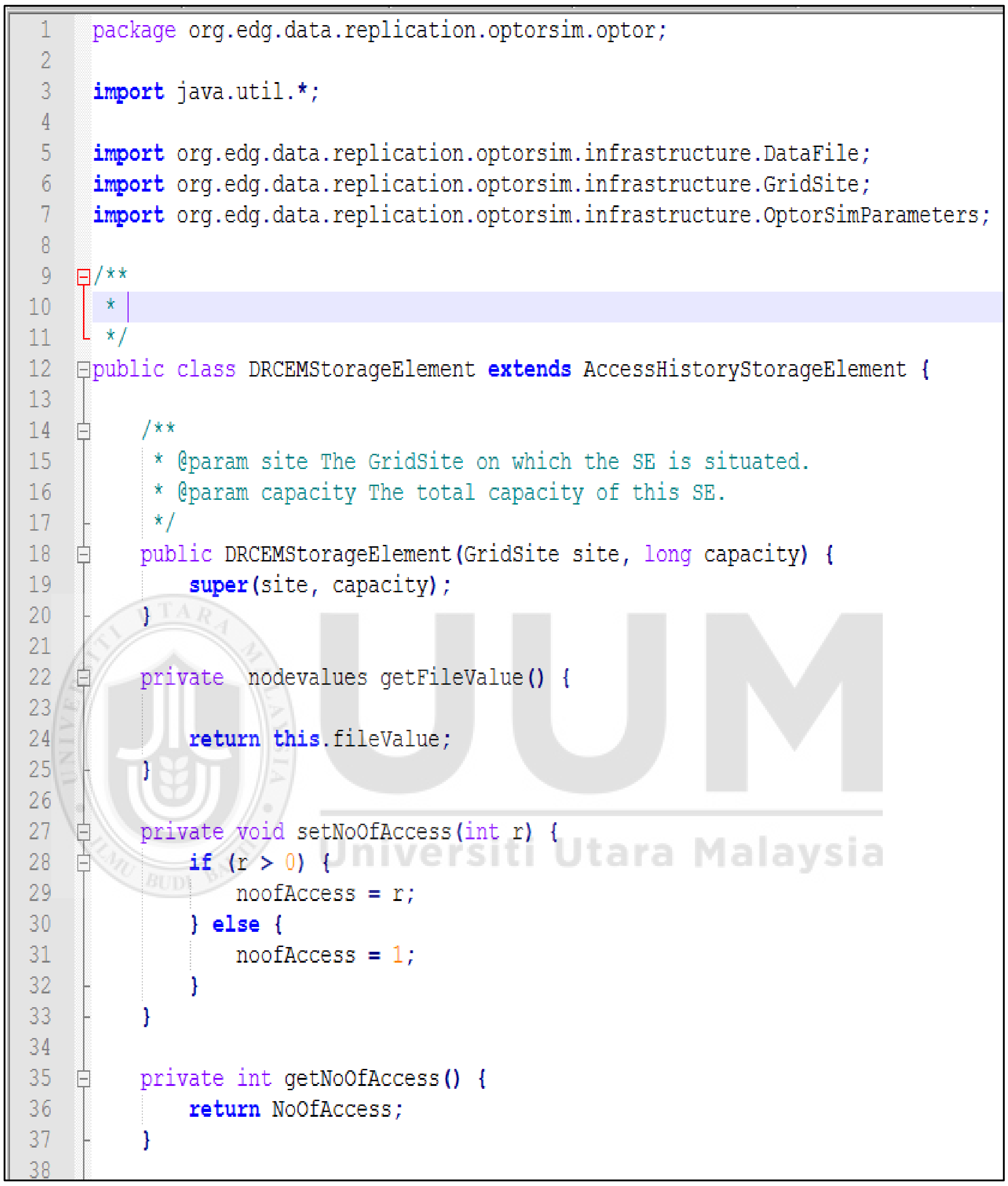


```java
package org.edg.data.replication.optorsim.optor;

import java.util.List;

import org.edg.data.replication.optorsim.infrastructure.DataFile;
import org.edg.data.replication.optorsim.infrastructure.GridSite;
import org.edg.data.replication.optorsim.infrastructure.StorageElement;

/**
 * This optimiser replicates files that met the replica creation creteria
 * Files of high Values within the federation regionsdue
 */
public class DRCEMOptimiser extends ReplicatingOptimiser {

    protected DRCEMOptimiser( GridSite site) {
        super(site);
    }

    /**
     * Returns the List of DataFiles chosen by {@link DRCEMStorageElement#filesToDelete}.
     */
    protected List chooseFilesToDelete( DataFile file, StorageElement se) {
        return se.filesToDelete(file);
    }

}
```

Figure 4.11. DRCEMOptimiser implementation into ReplicatingOptimser class

Similar to other integrated mechanisms, DRCEM has its storage element function that is used when the optimiser decides to make any file replication or eviction. The *DRCEMStorageElement* class performs this functionality.

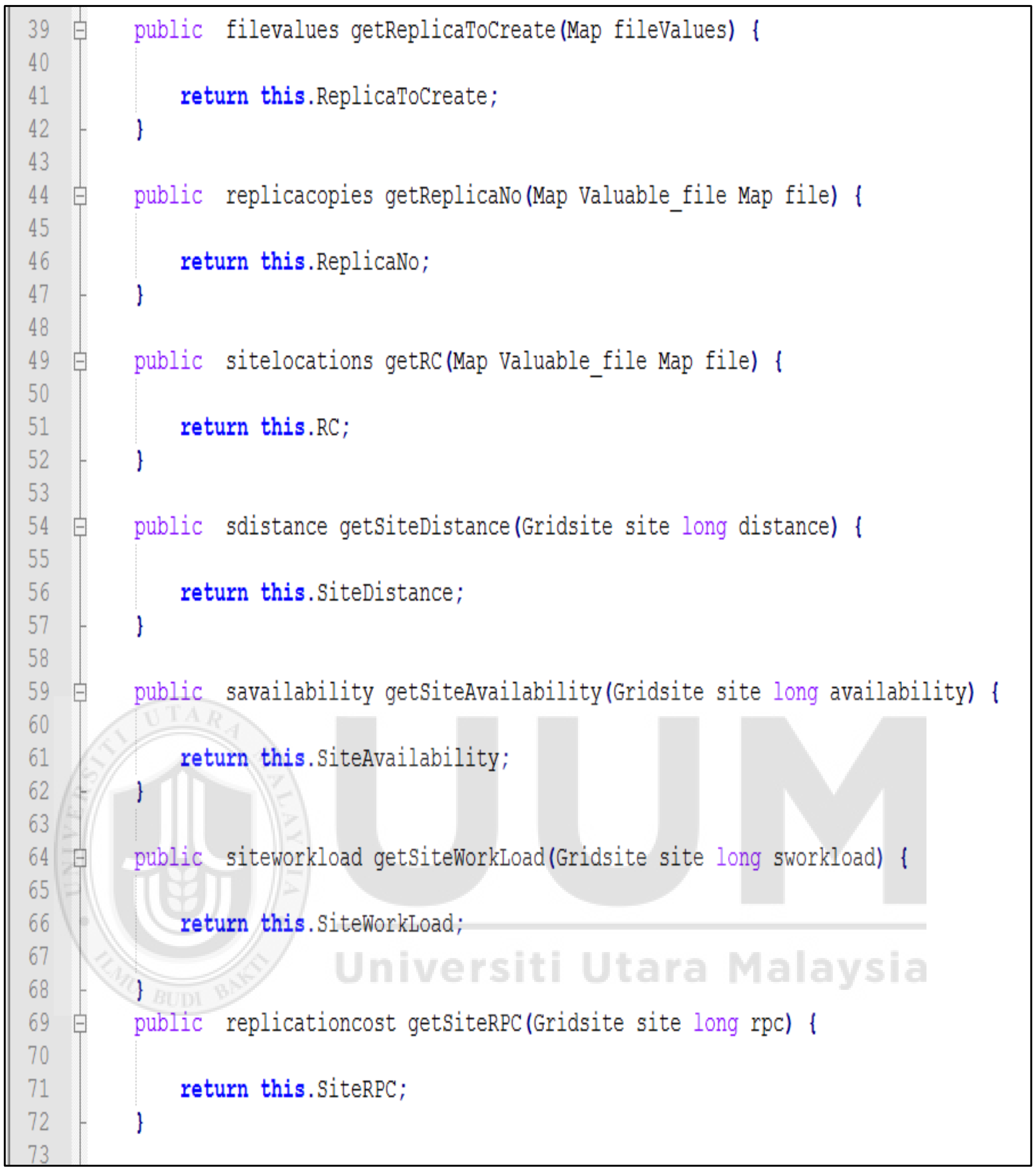
The following Figure 4.12 shows the integration of *DRCEMStorageElement* into *AccessHistoryStorageElement.*



```
package org.edg.data.replication.optorsim.optor;

import java.util.*;

import org.edg.data.replication.optorsim.infrastructure.DataFile;
import org.edg.data.replication.optorsim.infrastructure.GridSite;
import org.edg.data.replication.optorsim.infrastructure.OptorSimParameters;

/**
 *
 */
public class DRCEMStorageElement extends AccessHistoryStorageElement {

    /**
     * @param site The GridSite on which the SE is situated.
     * @param capacity The total capacity of this SE.
     */
    public DRCEMStorageElement(GridSite site, long capacity) {
        super(site, capacity);
    }

    private  nodevalues getFileValue() {

        return this.fileValue;
    }

    private void setNoOfAccess(int r) {
        if (r > 0) {
            noofAccess = r;
        } else {
            noofAccess = 1;
        }
    }

    private int getNoOfAccess() {
        return NoOfAccess;
    }

```

Figure 4.12. DRCEMStorageElement implementation into AccessHistoryStorageElement

Figure 4.12 continued.


```
public  filevalues getReplicaToCreate(Map fileValues) {

    return this.ReplicaToCreate;
}

public  replicacopies getReplicaNo(Map Valuable_file Map file) {

    return this.ReplicaNo;
}

public  sitelocations getRC(Map Valuable_file Map file) {

    return this.RC;
}

public  sdistance getSiteDistance(Gridsite site long distance) {

    return this.SiteDistance;
}

public  savailability getSiteAvailability(Gridsite site long availability) {

    return this.SiteAvailability;
}

public  siteworkload getSiteWorkLoad(Gridsite site long sworkload) {

    return this.SiteWorkLoad;

}
public  replicationcost getSiteRPC(Gridsite site long rpc) {

    return this.SiteRPC;
}
```

Thus, there are two main classes that need to be created, namely *DRCEMOptimiser* and *DRCEMStorageElement* class. The *DRCEMOptimiser* class directs the *DRCEMStorageElement* class to store replicas that are created by the optimiser as well as to remove files. The *DRCEMStorageElement* will then execute the commands and thus stores or removes the particular files. Therefore, the *DRCEMStorageElement* extends the functionalities of the *AccessHistoryStorageElement* class in the *optorsim.optor* package.

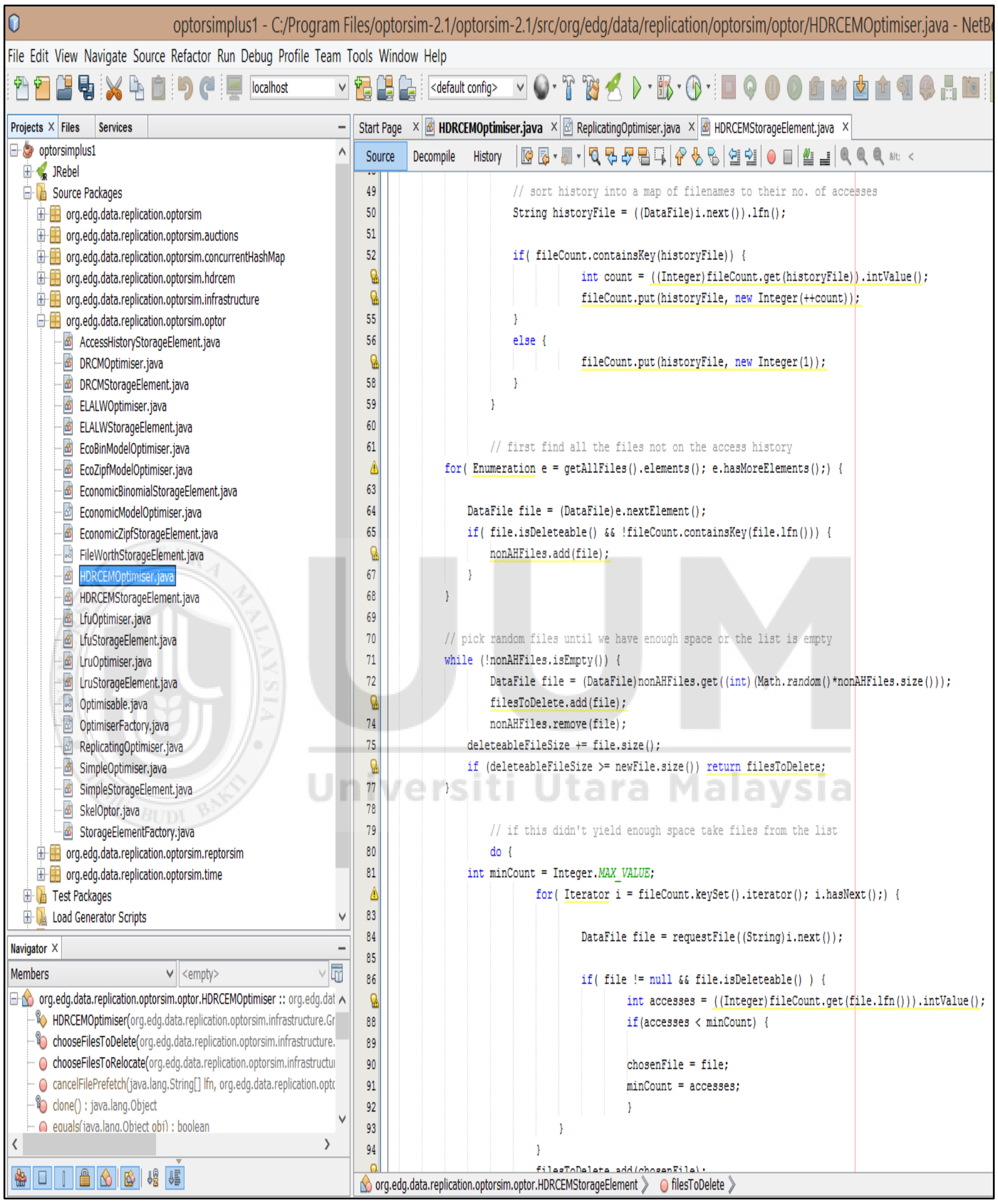
The implementation of *DRCEMStorageElement* class into *optorsim.optor* package via the *AccessHistoryStorageElement* class was a success, and free from programming errors. The *DRCEMStorageElement* class includes the main methods that manipulate the stored data, including, <*getSiteDistance*>, <*getSiteAvailability*>, <*getWorkLoad*> *and* <*getRPC*>.

The DRCEM mechanism computes sites distances using the <*getSiteDistance*> method. Site failure is determined by using the <*getSiteAvailability*> method. The site workloads and replica placement costs are computed using the <*getWorkLoad*> and <*getRPC*> methods, respectively.

In the same vein, similar classes were developed for both ELALW and DRCM mechanisms namely *ELALWOptimiser* and *DRCMOptimiser*, respectively, which were implemented in OptorSim via same interface. While interacting with DRCEM, the various methods contained in the mechanism perform to achieve the desired objectives of this thesis. These methods include program logics to calculate site distance, site availability site workload and replica placement cost, as specified by the DRCEM schemes.

Furthermore, the overall program code for the DRCEM mechanism has been implemented in the OptorSim package using the NetBeans environment, under the *OptorSimPlus1* project, which is free from programming errors. The *OptorSimplus1* project is part of this thesis' efforts in the process of programming and codding the DRCEM mechanism, based on the original OptorSim-2.1 package [33].

The following Figure 4.13 presents a part of DRCEM’s implemented code after completing the integration processes.



optorsimplus1 - C:/Program Files/optorsim-2.1/optorsim-2.1/src/org/edg/data/replication/optorsim/optor/HDRCEMOptimiser.java - NetB

```
optorsimplus1
  JRebel
  Source Packages
    org.edg.data.replication.optorsim
    org.edg.data.replication.optorsim.auctions
    org.edg.data.replication.optorsim.concurrentHashMap
    org.edg.data.replication.optorsim.hdrcem
    org.edg.data.replication.optorsim.infrastructure
    org.edg.data.replication.optorsim.optor
      AccessHistoryStorageElement.java
      DRCMOptimiser.java
      DRCMStorageElement.java
      ELALWOptimiser.java
      ELALWStorageElement.java
      EcoBinModelOptimiser.java
      EcoZipfModelOptimiser.java
      EconomicBinomialStorageElement.java
      EconomicModelOptimiser.java
      EconomicZipfStorageElement.java
      FileWorthStorageElement.java
      HDRCEMOptimiser.java
      HDRCEMStorageElement.java
      LfuOptimiser.java
      LfuStorageElement.java
      LruOptimiser.java
      LruStorageElement.java
      Optimisable.java
      OptimiserFactory.java
      ReplicatingOptimiser.java
      SimpleOptimiser.java
      SimpleStorageElement.java
      SkelOptor.java
      StorageElementFactory.java
    org.edg.data.replication.optorsim.reptorsim
    org.edg.data.replication.optorsim.time
  Test Packages
  Load Generator Scripts
```

```
org.edg.data.replication.optorsim.optor.HDRCEMOptimiser :: org.edg.dat
  HDRCEMOptimiser(org.edg.data.replication.optorsim.infrastructure.Gr
  chooseFilesToDelete(org.edg.data.replication.optorsim.infrastructure.
  chooseFilesToRelocate(org.edg.data.replication.optorsim.infrastructu
  cancelFilePrefetch(java.lang.String[] lfn, org.edg.data.replication.optc
  clone() : java.lang.Object
  equals(java.lang.Object obj) : boolean
```

```java
                // sort history into a map of filenames to their no. of accesses
                String historyFile = ((DataFile)i.next()).lfn();

                if( fileCount.containsKey(historyFile)) {
                        int count = ((Integer)fileCount.get(historyFile)).intValue();
                        fileCount.put(historyFile, new Integer(++count));
                }
                else {
                        fileCount.put(historyFile, new Integer(1));
                }
            }

            // first find all the files not on the access history
        for( Enumeration e = getAllFiles().elements(); e.hasMoreElements();) {

            DataFile file = (DataFile)e.nextElement();
            if( file.isDeleteable() && !fileCount.containsKey(file.lfn())) {
                nonAHFiles.add(file);
            }
        }

        // pick random files until we have enough space or the list is empty
        while (!nonAHFiles.isEmpty()) {
                DataFile file = (DataFile)nonAHFiles.get((int)(Math.random()*nonAHFiles.size()));
                filesToDelete.add(file);
                nonAHFiles.remove(file);
            deleteableFileSize += file.size();
            if (deleteableFileSize >= newFile.size()) return filesToDelete;
        }

                // if this didn't yield enough space take files from the list
                do {
            int minCount = Integer.MAX_VALUE;
                        for( Iterator i = fileCount.keySet().iterator(); i.hasNext();) {

                                DataFile file = requestFile((String)i.next());

                                if( file != null && file.isDeleteable() ) {
                                        int accesses = ((Integer)fileCount.get(file.lfn())).intValue();
                                        if(accesses < minCount) {

                                        chosenFile = file;
                                        minCount = accesses;
                                        }
                                }
                        }
                        filesToDelete.add(chosenFile);
```

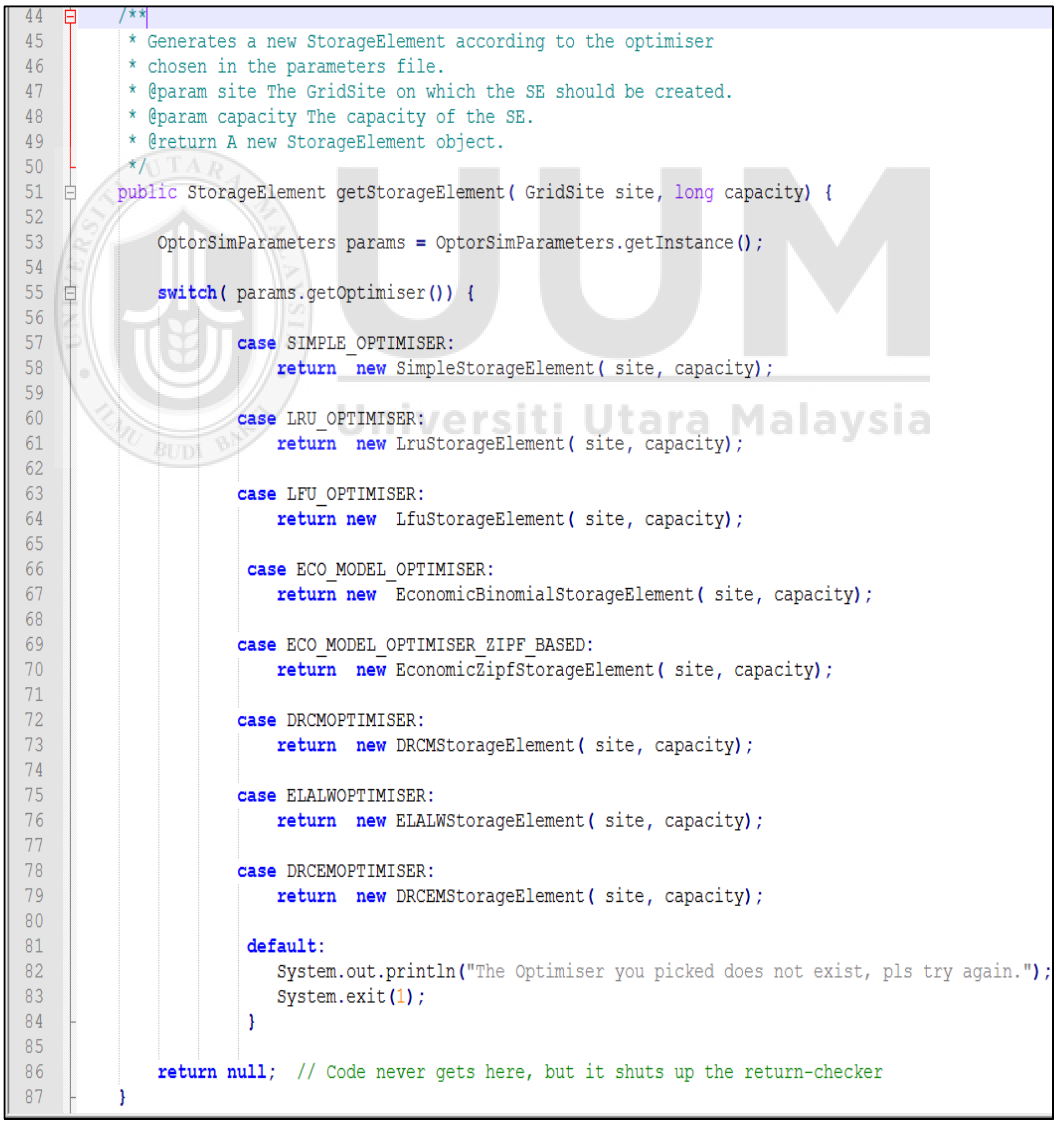
org.edg.data.replication.optorsim.optor.HDRCEMStorageElement 〉 filesToDelete 〉

Figure 4.13. DRCEM entire code in the OptorSim simulator

From Figure 4.13, the code integration confirms the following expectations:

- DRCEM has been correctly coded into the NetBeans environment
- The DRCEM code implementation is free from program bugs and errors

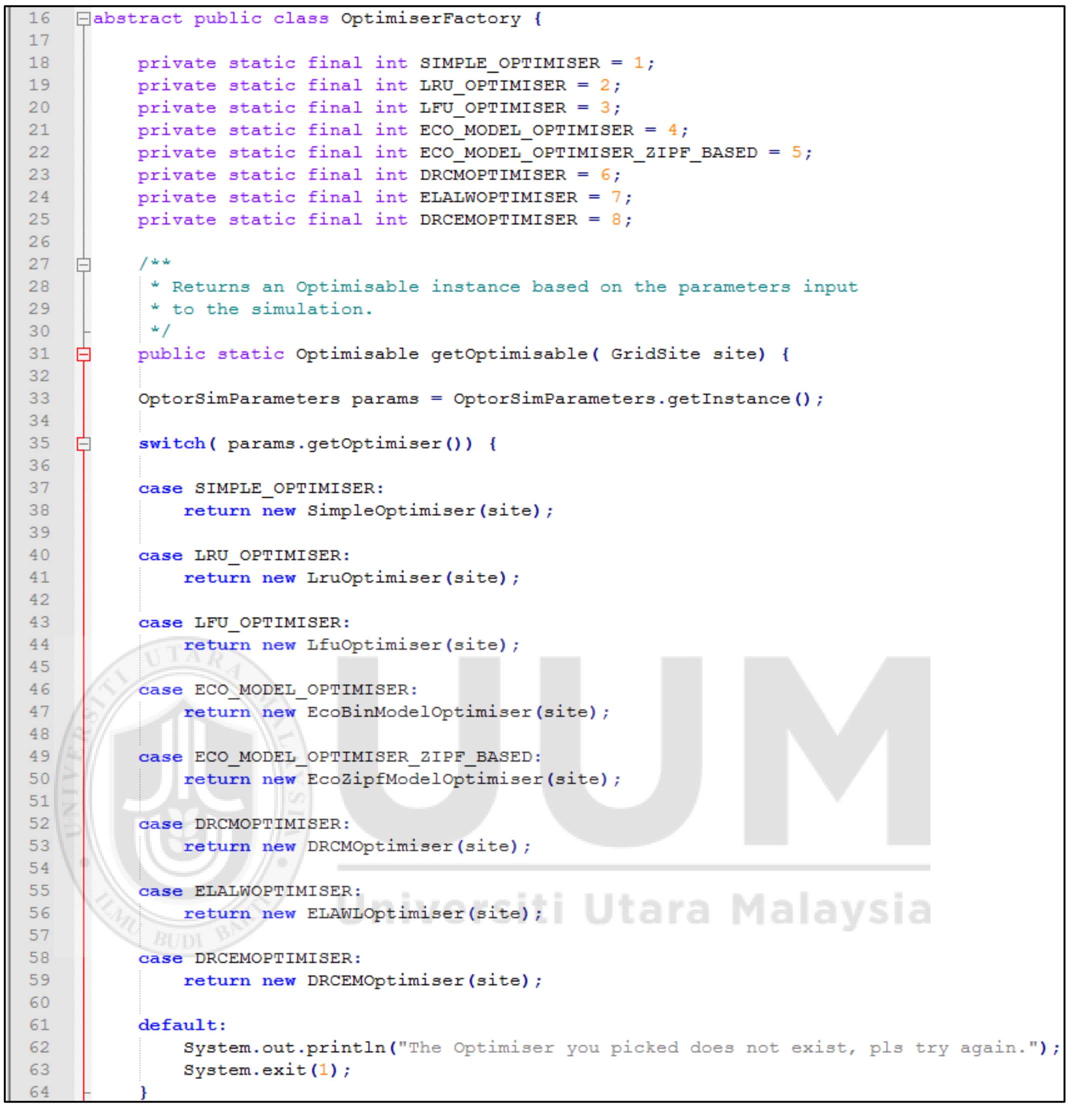
Part of the implementation process involves including the DRCEM mechanism into the parameters file, so that it could be selected for the simulation purpose. To that effect, DRCEM is implemented into the *OptimiserFactory* class and *StorageElementFactory* classes respectively. OptorSim is already bundled with five optimisers. Thus, this thesis implements *ELALWOptimiser* and *DRCMOptimiser* into the simulator, in addition to the proposed *DRCEMoptimiser*. Figure 4.14 and Figure 4.15; show the implementations of *DRCEMOptimiser*, *ELALWOptimiser* and *DRCMOptimiser* into *StorageElementFactory* and *OptimiserFactory*, respectively.



```java
/**
 * Generates a new StorageElement according to the optimiser
 * chosen in the parameters file.
 * @param site The GridSite on which the SE should be created.
 * @param capacity The capacity of the SE.
 * @return A new StorageElement object.
 */
public StorageElement getStorageElement( GridSite site, long capacity) {

    OptorSimParameters params = OptorSimParameters.getInstance();

    switch( params.getOptimiser()) {

        case SIMPLE_OPTIMISER:
            return  new SimpleStorageElement( site, capacity);

        case LRU_OPTIMISER:
            return  new LruStorageElement( site, capacity);

        case LFU_OPTIMISER:
            return new  LfuStorageElement( site, capacity);

         case ECO_MODEL_OPTIMISER:
            return new  EconomicBinomialStorageElement( site, capacity);

        case ECO_MODEL_OPTIMISER_ZIPF_BASED:
            return  new EconomicZipfStorageElement( site, capacity);

        case DRCMOPTIMISER:
            return  new DRCMStorageElement( site, capacity);

        case ELALWOPTIMISER:
            return  new ELALWStorageElement( site, capacity);

        case DRCEMOPTIMISER:
            return  new DRCEMStorageElement( site, capacity);

         default:
            System.out.println("The Optimiser you picked does not exist, pls try again.");
            System.exit(1);
         }

    return null;  // Code never gets here, but it shuts up the return-checker
}
```

Figure 4.14. Implementation of DRCEMOptimiser into the StorageElementFactory class

The following Figure 4.15, show the implementations of *DRCEMOptimiser*, *ELALWOptimiser* and *DRCMOptimiser* into *OptimiserFactory* class.



```
abstract public class OptimiserFactory {

    private static final int SIMPLE_OPTIMISER = 1;
    private static final int LRU_OPTIMISER = 2;
    private static final int LFU_OPTIMISER = 3;
    private static final int ECO_MODEL_OPTIMISER = 4;
    private static final int ECO_MODEL_OPTIMISER_ZIPF_BASED = 5;
    private static final int DRCMOPTIMISER = 6;
    private static final int ELALWOPTIMISER = 7;
    private static final int DRCEMOPTIMISER = 8;

    /**
     * Returns an Optimisable instance based on the parameters input
     * to the simulation.
     */
    public static Optimisable getOptimisable( GridSite site) {

    OptorSimParameters params = OptorSimParameters.getInstance();

    switch( params.getOptimiser()) {

    case SIMPLE_OPTIMISER:
        return new SimpleOptimiser(site);

    case LRU_OPTIMISER:
        return new LruOptimiser(site);

    case LFU_OPTIMISER:
        return new LfuOptimiser(site);

    case ECO_MODEL_OPTIMISER:
        return new EcoBinModelOptimiser(site);

    case ECO_MODEL_OPTIMISER_ZIPF_BASED:
        return new EcoZipfModelOptimiser(site);

    case DRCMOPTIMISER:
        return new DRCMOptimiser(site);

    case ELALWOPTIMISER:
        return new ELAWLOptimiser(site);

    case DRCEMOPTIMISER:
        return new DRCEMOptimiser(site);

    default:
        System.out.println("The Optimiser you picked does not exist, pls try again.");
        System.exit(1);
    }
```

Figure 4.15. Implementation of DRCEMOptimiser into OptimiserFactory class

From Figure 4.14 and Figure 4.15, the *DCREMOptimiser* code implementation into *StorageElementFactory* and *OptimiserFactory* classes indicates that the code is free from both syntax and semantic errors, which further confirmed the validation process of DRCEM mechanism explained previously in Chapter Three. The next section summarises the chapter.

4.5 Chapter Summary

This chapter presented the design of replication mechanism called an Enhanced Dynamic Replica Creation and Eviction Mechanism (DRCEM). The Chapter outlined the primary objectives of the design, which include minimising the jobs completion time, minimising the network bandwidth consumption, minimiszing the storage element usage as well as the computing element usage. For achieving the above objectives, the chapter explained the design of DRCEM mechanism, which incorporates three main schemes, thus: (i) - *Dynamic Replica Evaluation and Creation Scheme (DRECS).* In this part file replicas are evaluated to determine their popularity as well as compute the required number of replicas to be created. (ii) -; *Dynamic Replica Placement Scheme (DRPS):* This part determines the sites from or to which the replicas are to be placed. (iii) - *Dynamic Replica Eviction Scheme (DRES)*: This part is invoked when there is not enough space in the selected site to host the newly created replicas, and then there is a need to create more space to accommodate the newly created replicas.

These schemes have been successfully designed in this chapter, and numerical examples have been provided in its various sections to illustrate how each scheme in the proposed mechanism performs. The step-by-step implementation of the entire DRCEM mechanism was carried in this chapter along with brief on how the mechanism interacts with the various components of the OptorSim simulator for executing user’s jobs. The next chapter discusses the overall performance of the DRCEM along with comparison on existing mechanisms evaluated through simulation experiments, based on the selected performance metrics (Jobs Completion Times, ENU, SEU, and CEU).

CHAPTER FIVE

THE DRCEM PERFORMANCE EVALUATION ALONG WITH COMPARISON ON EXISTING MECHANISMS

5.1 Introduction

In a bid to evaluate the new DRCEM mechanism, this thesis conducted a comparative evaluation experiment between ELALW mechanism and the DRCM mechanism as explained earlier in the introductory chapter. This chapter presents a series of experiments with the aim of examining the efficiency of DRCEM in different situations. The presented experiments with their results aimed at evaluating the system performance are based on the selected performance metrics namely jobs completion times, network bandwidth usage, storage element usage and computing element usage. The DRCEM performance evaluation along with comparison on existing mechanisms is presented in this chapter. As mentioned ealier in Chapter 3 (Section 3.6.3: Performance Evaluation Metrics, Page 119), the performance metrics are evaluated using design metrics, which include number of jobs, file size, site distance, site workload, file logical dependencies and type of job scheduling mechanism. The next section explains the steps taken by this thesis and compares the simulation results with the existing mechanisms.

5.2 Comparison of DRCEM with ELALW and DRCM Mechanisms

In this section, DRCEM is compared against ELALW and DRCM mechanisms using the configuration and parameters files outlined in Chapter Three, Section 3.6.2.1 (Table 3.3).The next sub-section is the analysis of number of jobs effects on the amount of replications performed by each of the mechanisms under review.

5.2.1 Analysis on Number of Jobs and Effects on Replications

In this subsection, the comparison is based on the behaviours of DRCEM for submitted number of jobs in a certain workloads for a particular performance metric. The effects of number of jobs submitted to the DGF environment on the amount of replication are analysed. The performance of DRCEM and other replication mechanisms are measured, using the Queue Access Cost scheduling mechanism with the other parameters and varying the number of jobs submitted from 50 to 5000. Summary of results from the simulation is given in Table 5.1.

Table 5.1
Summary of results from the simulations of 50 to 5000 jobs

Number of Jobs	**DRCEM Replications**	**ELALW Replications**	**DRCM Replications**
50	189	215	197
500	375	348	334
1000	395	428	413
2000	452	536	596
3000	478	652	624
4000	526	678	666
5000	568	765	705

In the following Figures (5.1-5.3), the effects of *number of jobs* on *number of replications* are analyzed based on 10 GB, 5 GB, and 2.5 GB file sizes, for DRCEM, ELALW and DRCM mechanisms, respectively.

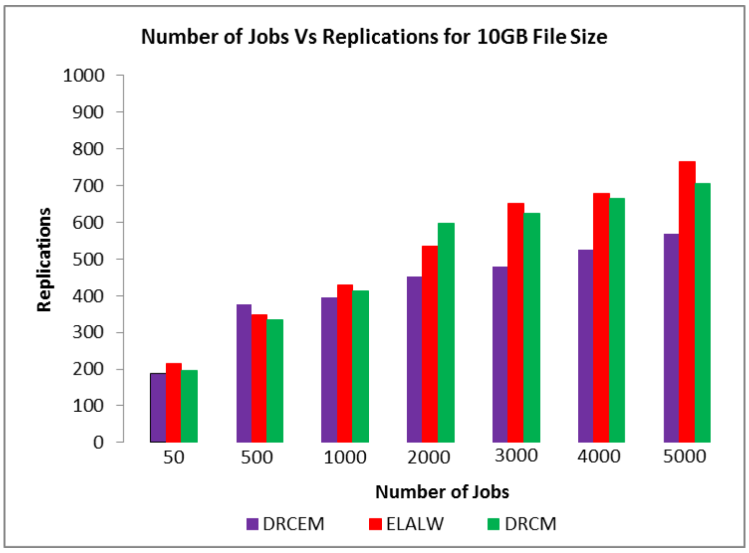


Figure 5.1. Effect of jobs numbers on replications for 10 GB file size

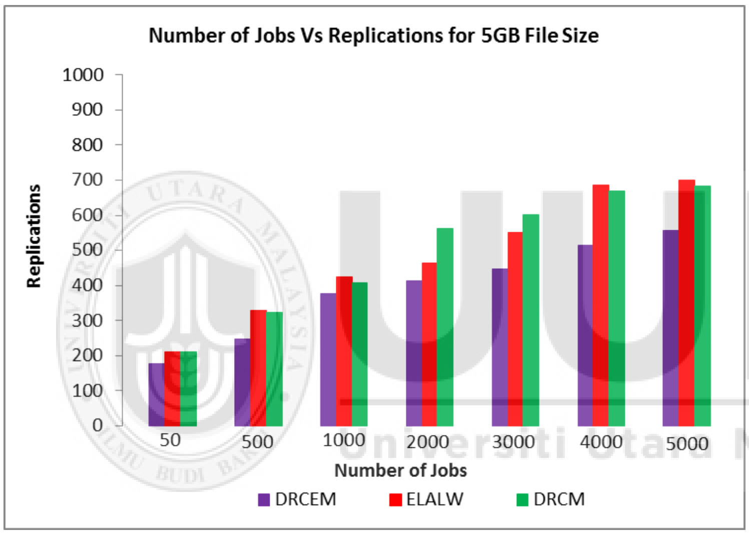


Figure 5.2. Effect of jobs numbers on replications for 5 GB file sizes

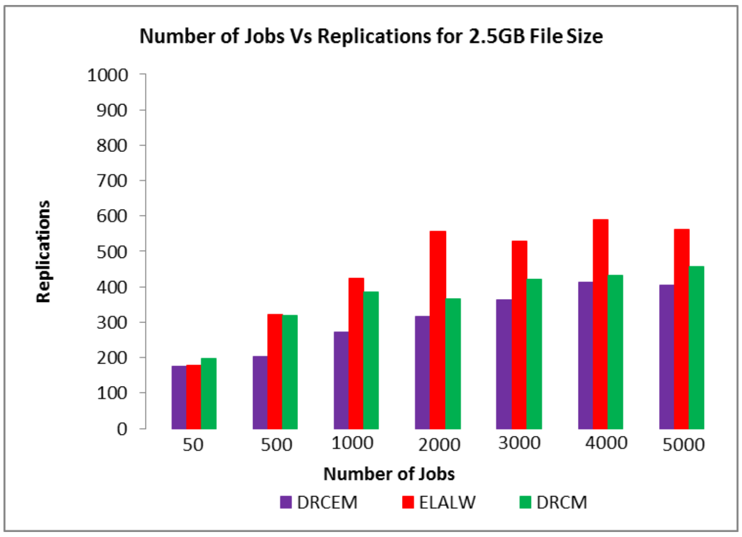


Figure 5.3. Effect of jobs numbers on replications for 2.5 GB file sizes

From Figures 5.1-5.3, both mechanisms showed similar pattern of increasing number of replications as the number of jobs rises from 50 to 5000. In addition, it is evident that file size does not have much effect on the number of replications. However, jobs numbers do affect the numer of replications in all the mechanisms. As the job numbers increase from 50 to 5000, the number of replications also increases. However, at higher number of jobs, there is decline in the number of replications. This is because the mechanisms keep a history of all access to data files. Thus, the locations of various replication sites are taken into considerations from the previous access history. This tends to narrow down the number of replications subsequently.

Although from the beginning, the replications seem to be raising steadily with jobs numbers. As the number of jobs reaches 2000 (Figure 5.3), the replication rises at much lower rate, indicating an arithmetic pattern. From the simulation results in Figures 5.1-5.3, DRCEM shows least amount of replications as the number of jobs rises from 50 to 5000, compared to DRCM and ELALW.

Although the number of replications done by DRCEM is lower than that of the existing mechanisms (DRCM and ELALW), this does not slow down the jobs completion times, as would be seen in subsection 5.2.3, page 200 under the analysis of the number of jobs on jobs times. The reason has been that DRCEM replicates under strict conditions of site availability, site workloads and relative distances between replica sites.

This condition made sure that although replication is minimised to conserve storage resources, it also strategically places replicas at locations that offer best faster access times to the advantage of the users. In addition, along with storage conservation, DRCEM does not consume much bandwidth, due to the reduced number of replications, as would be seen in (subsection 5.2.4, page 204) under the analysis on the effect of a number of jobs on the effective network usage. The next subsection, explains the analysis on the effect of number of jobs on jobs completion times.

5.2.3 Analysis on the Effect of Number of Jobs on Jobs Times

The simulation results of DRCEM and existing mechanisms for 5000 jobs of 10 GB, 5 GB and 2.5 GB file sizes are presented in Table 5.2, Table 5.3 and Table 5.4, respectively.

Table 5.2
DRCEM, DRCM and ELALW on 5000 jobs with 10.0 GB files size

Number of Jobs	**Metrics**	**DRCEM**	**ELALW**	**DRCM**
5000	Jobs Completion Times	4217.91	6066.16	5703.63
	Effective Network Usage (ENU)	78.13	86.04	83.59
	Storage Element Usage (SEU)	18.35	48.96	47.26
	Computing Element Usage (CEU)	55.89	50.24	51.56

Table 5.3
DRCEM, DRCM and ELALW on 5000 jobs with 5.0 GB files size

Number of Jobs	**Metrics**	**DRCEM**	**ELALW**	**DRCM**
5000	Jobs Completion Times	1207.50	2386.30	2233.56
	Effective Network Usage (ENU)	0.7336	0.7908	0.7781
	Storage Element Usage (SEU)	8.64	23.54	23.78
	Computing Element Usage (CEU)	35.33	28.33	33.50

Table 5.4

DRCEM, DRCM and ELALW on 5000 jobs with 2.5 GB files size

Number of Jobs	**Metrics**	**DRCEM**	**ELALW**	**DRCM**
5000	Jobs Completion Times	198.05	440.76	366.57
	Effective Network Usage (ENU)	0.6058	0.6148	0.6123
	Storage Element Usage (SEU)	2.73	10.06	3.22
	Computing Element Usage (CEU)	25.33	18.33	23.50

The efficiency of the proposed DRCEM mechanism is computed using the standard Equation 5.1 and compared with DRCM and ELALW mechanisms.

$$Efficiency = \frac{metric\ value_{existing\ mechanism} - metricvalue_{HDRCEM}}{metric\ value_{existing\ mechanism}} \times 100 \qquad (5.1)$$

From Table 5.2, and using Equation 5.1, for instance, DRCEM outperforms ELALW by 30.47% and DRCM by 26.05% in Jobs Completion Times metric, regarding efficiency. Also, the efficiency of DRCEM over ELALW and DRCM regarding Storage Element Usage is 42.10% and 40.01%, respectively.

Regarding the Effective Network Usage, the performances of both mechanisms under study are at close range. However, due to the less number of replications by DRCEM, its performance over ELALW shows the efficiency of 4.55% and 2.28% over DRCM. Regarding the Computing Element Usage metric, the efficiency of DRCEM over ELALW is 23.65%, while it is 19.12% against DRCM mechanism.

In the same vein, from Table 5.3 and Table 5.4, the efficiencies of DRCEM against ELALW and DRCM, are computed and presented in the following paragraph.

Table 5.5 shows the efficiency of DRCEM with percentage values, against existing mechanisms.

Table 5.5

Efficiency of DRCEM against ELALW and DRCM mechanisms

	Percentage Efficiency (%)	
Metrics	**DRCEM vs. ELALW**	**DRCEM vs. DRCM**
Jobs Completion Times	30.47%	26.05%
Storage Element Usage (SEU)	42.10%	40.01%
Effective Network Usage (ENU)	4.55%	2.28%
Computing Element Usage (CEU)	23.65%	19.12%

In what follows, the result presented in Table 5.5 is discussed and analysed in further details. Figures 5.4-5.6, present the analysis on effects of a number of jobs on jobs completion time based on 10 GB, 5 GB, and 2.5 GB file sizes, respectively.

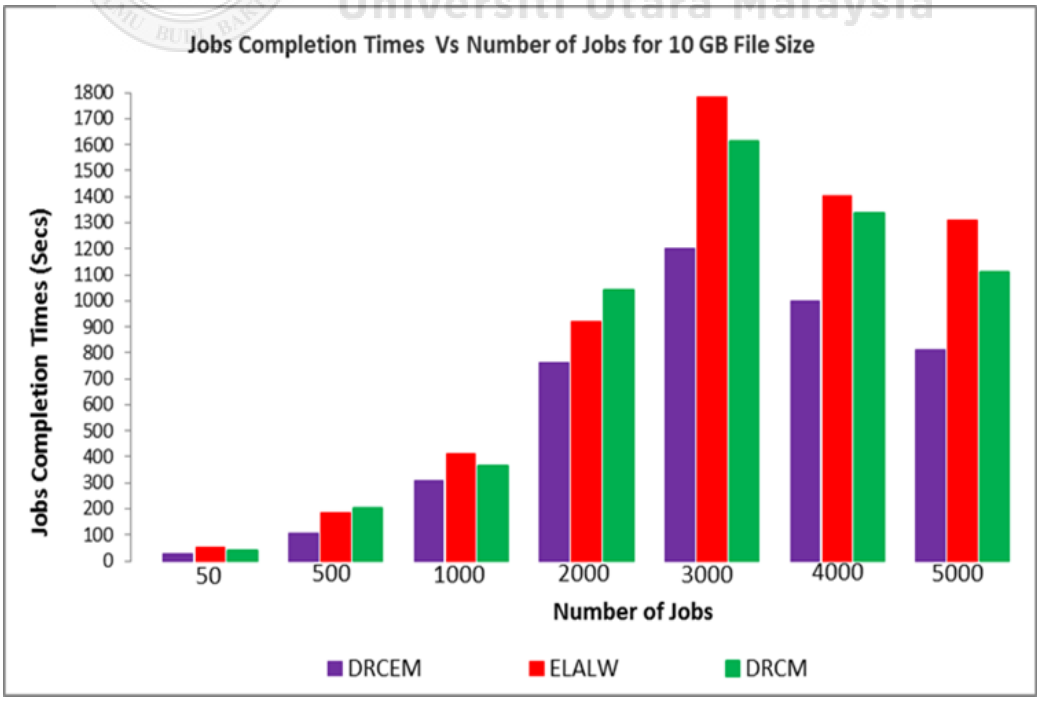


Figure 5.4. Jobs times for different number of submitted jobs of 10 GB files size

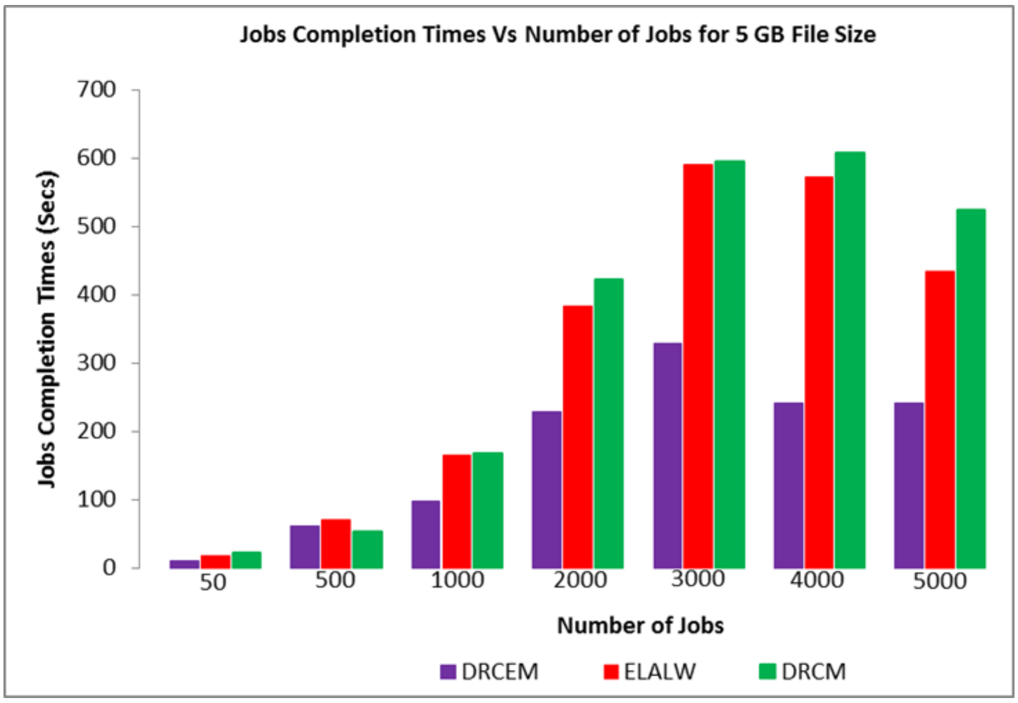


Figure 5.5. Jobs times for different number of submitted jobs of 5 GB files size

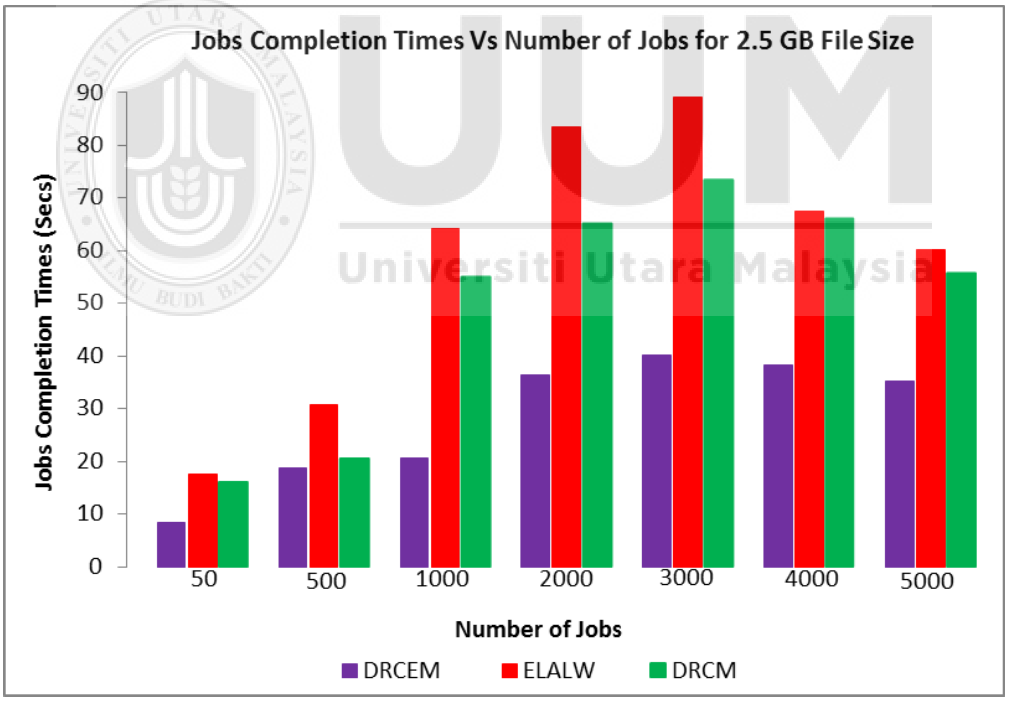


Figure 5.6. Jobs times for different number of submitted jobs of 2.5 GB files size

From Figures 5.4-5.6, it could be seen that DRCEM achieves less jobs completion times compared to the existing mechanisms. A better mechanism is the one with fewer amounts of jobs completion times. Therefore, it suffices to say that DRCEM performs the best among the compared mechanisms. DRCEM consumes 26.05% less

Jobs Completion Times compared to DRCM, and 30.47% over ELALW. The lower jobs completion times is due to the replication decision that has been made by DRCEM, in which decides to replicate a group of valuable files at the same time, as well as considers sites with fewer workloads and high availability. As a result, replicas of popular files are spread in the DGF environment and increase data availability. On the other hand, ELALW replicates only one popular file at one decision, while DRCM replicates group of files, without due regards to the site availability and distance between replica sites.

Previously in Section 5.2.1, it was established that DRCEM performs less number of replications, compared to ELALW and DRCM. The reason has been that DRCEM replication is guided by site distance, site availability and workload (only lightly/moderately loaded sites are considered). In this case, performing fewer replications has two advantages, thus; saves storage by not performing unnecessary replications and boast jobs time; whereas in the other mechanisms, valuable time is used to perform unnecessary replications. Although replication is done to improving data availability, this needs to be with caution in order: not to constraints the storage; not constraints the users by slowing down their jobs times and not constraints the grid federation by creating bottlenecks.

5.2.4 Analysis on the Effect of Number of Jobs on the Effective Network Usage

Performing replication process affects the ENU metric, the ENU is calculated by the Equation 3.3, and the large value of $N_{\text{replications}}$ will increase the ENU value. Figure 5.7-5.9 shows the results of ENU metric; there was little difference between the three mechanisms under review regarding ENU metric.

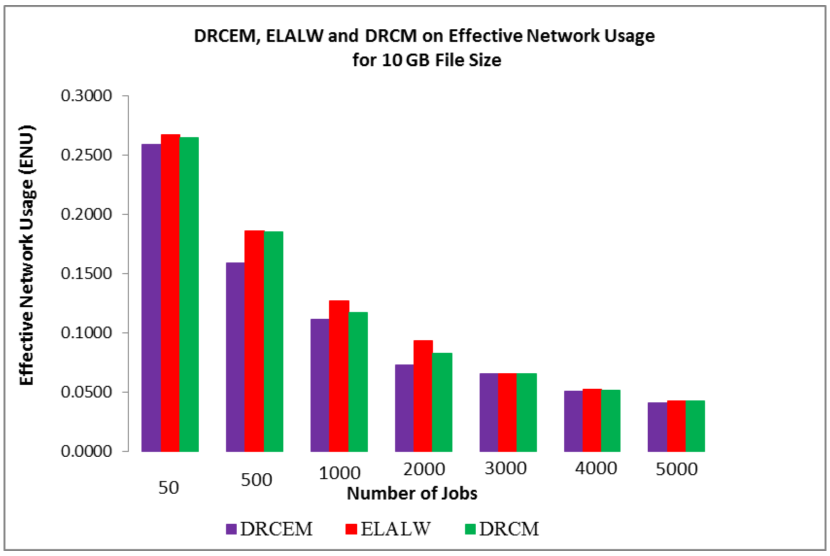


Figure 5.7. ENU for different number of submitted jobs of 10 GB files size

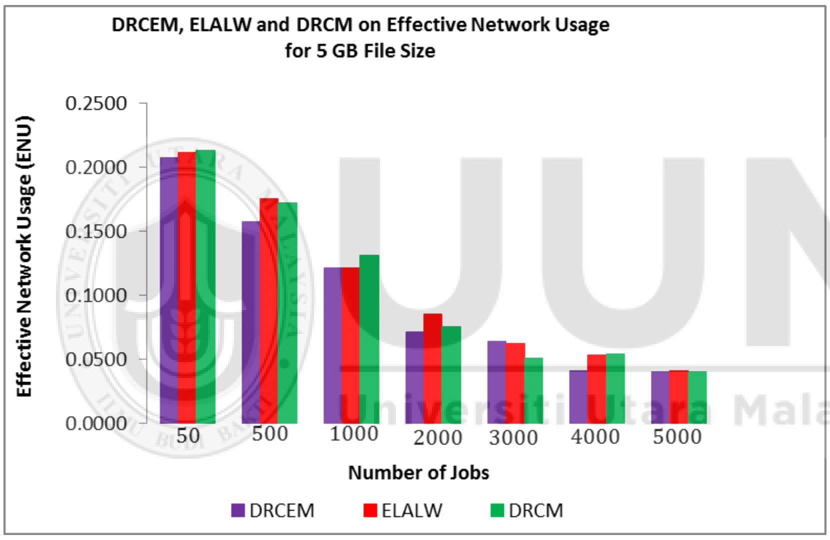


Figure 5.8. ENU for different number of submitted Jobs of 5 GB files size

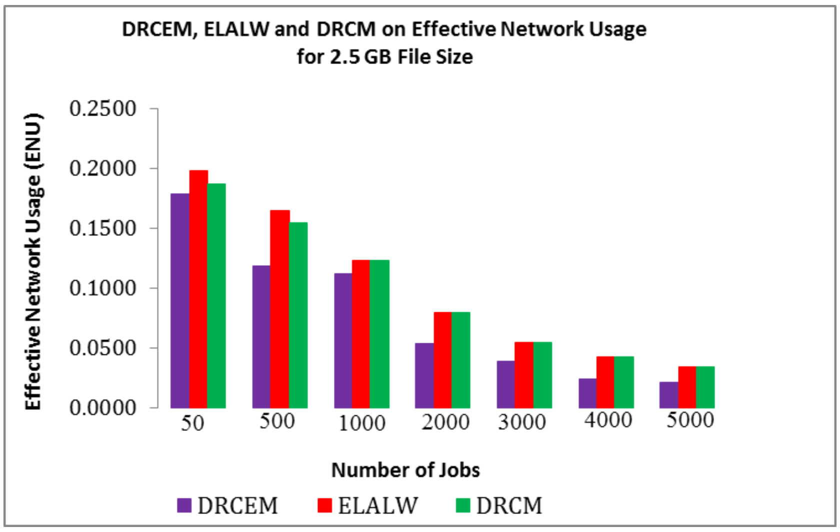


Figure 5.9. ENU for different number of submitted jobs of 2.5 GB files size

The DRCEM makes the decision of replication or deletion every constant time interval and start to replicate or delete the replicas. Thus, at the beginning of the simulation, the number of replication seems to be high, as a result of which ENU becomes higher. However, as the simulation goes with time, the number of replication decreases, with a resultant decrease in ENU value. Nevertheless, DRCEM shows slightly better network usage of about 4.55% efficient over ELALW, which is almost twice that of DRCM, which has percentage ENU of 2.28%. Thus, both ELALW and DRCM indicate poor usage of the network bandwidth compared to the DRCEM mechanism.

The effective network usage for 50, 500, 1000, up to 5000 jobs, shown in Figures 5.7-5.9, decreases with the number of jobs submitted. This is as might be expected, since the access histories used by the replication mechanism to make replication decisions take time to build up and stabilises. The existing mechanisms, though, show much lower usage with an increased number of jobs, with a factor of 2 differences between the ELALW and the DRCM mechanism.

The main advantage of the DRCEM mechanism is that it uses up considerably less network bandwidth than the DRCM and ELALW mechanisms. Thus, the results in Figures 5.7-5.9 indicate that file size affects network consumption. Thus, similar to storage consumption, network usage could be conserved, when processing a large file. This is done by splitting the file into smaller chunks and made to run on different machines. As could be seen in the figure, network usage is higher at the beginning of the simulation. However, as the job processing builds up with time, there is a decline in the network usage.

5.2.5 Analysis on the Effect of Number of Jobs on Storage Element Usage

Looking at the data in Figures 5.10-5.12 illustrate that DRCEM uses the least amount of storage (SEU) by outperforming ELALW by 35%, and DRCM by 27%. The reason for DRCEM outperformance has been that, the replication process is strictly guided by the percentage availability of the replica sites, site distance and workloads, which makes replication to be lower than both DRCM and ELALW, without constraining the jobs completion times, as evident in the previous analysis on number of jobs versus jobs times (page 202). The following Figures 5.10-5.12 compare DRCEM with existing mechanisms (DRCM & ELALW) on the storage element usage for 10 GB, 5 GB, and 2.5 GB file sizes, respectively.

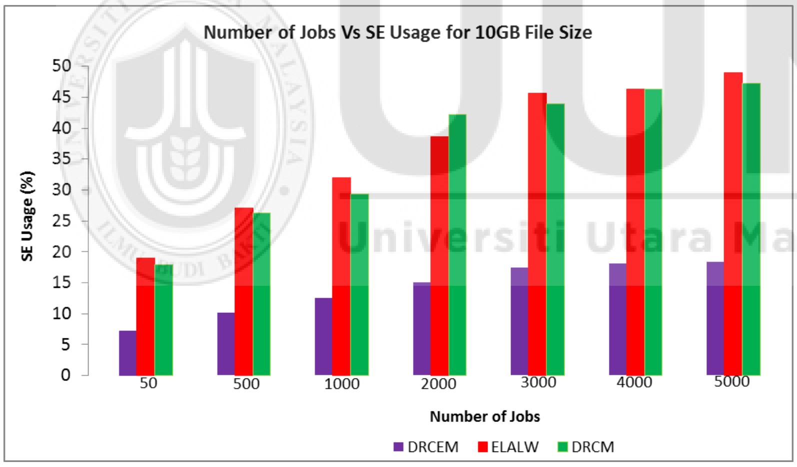


Figure 5.10. SE Usage of DRCM and existing mechanisms for 10 GB files size

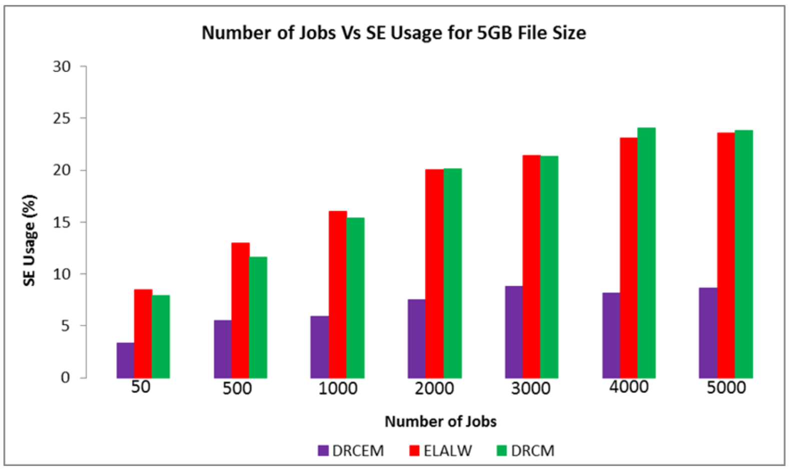


Figure 5.11. SE Usage of DRCM and existing mechanisms for 5 GB files size

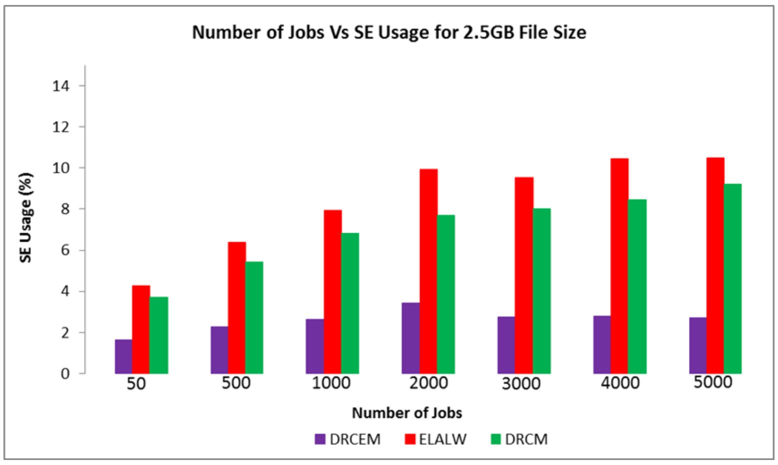


Figure 5.12. SE Usage of DRCM and existing mechanisms for 2.5 GB files size

Furthermore, the reduction in the use of storage is more pronounced with the decrease in file sizes, as seen in Figures 5.10-5.12. This indicates that file size can have a drastic effect on storage consumption. Thus, to minimise storage usage, large file for a job could be split into smaller chunks and distribute to different sites, for faster execution and conservation of storage usage.

5.2.6 Analysis on the Effect of Number of Jobs on Computing Element Usage

The following Figures 5.13-5.15 show DRCEM results and existing mechanisms on CE usage for different files sizes.

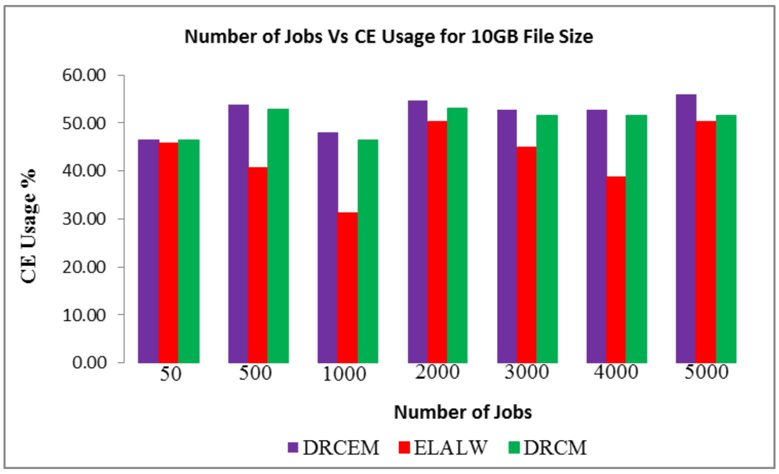


Figure 5.13. DRCEM results and existing mechanisms on CE usage for 10 GB files size

The Computing element usage is less influenced by the replication process, compared to the scheduling process. Each job in the federation needs to be scheduled on an appropriate computing element, which makes a number of jobs run on every computing element a concern. The following Figure 5.14 shows DRCEM results and existing mechanisms on CE usage for 5 GB files sizes.

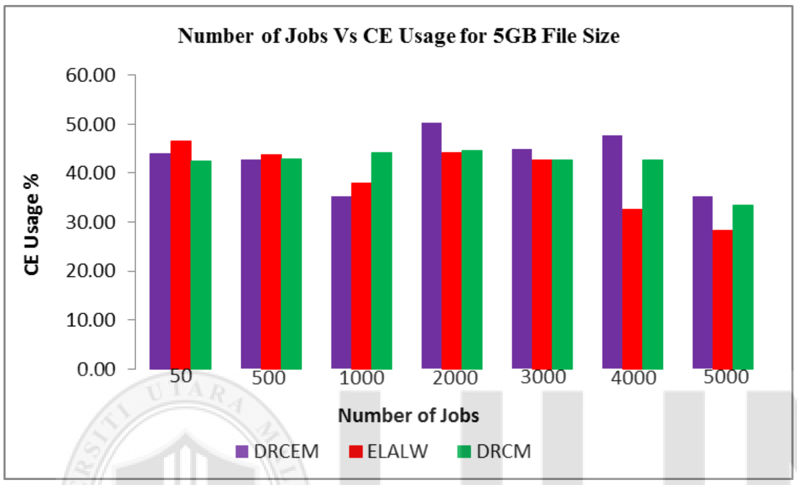


Figure 5.14. DRCEM results and existing mechanisms on CE usage for 5 GB files size

The following Figure 5.15 shows DRCEM results and existing mechanisms on CE usage for 2.5 GB files sizes.

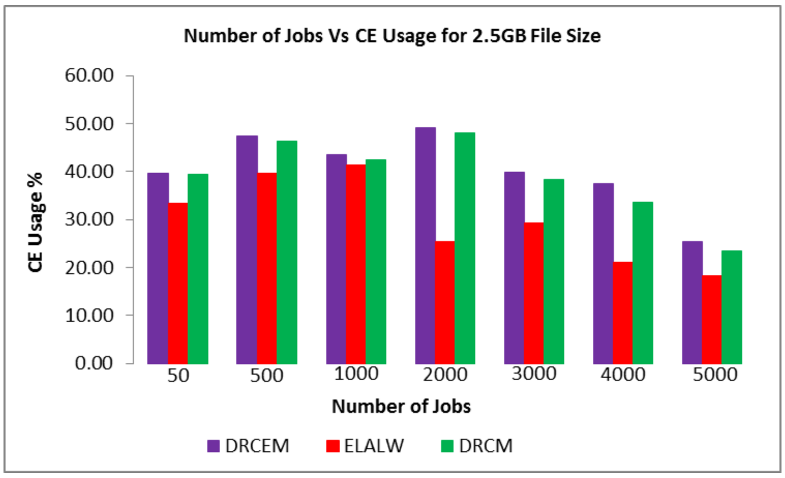


Figure 5.15. DRCEM results and existing mechanisms on CE usage for 2.5 GB files size

The number of jobs scheduled on a computing element by a job scheduler determines the impact of CEU metric. In this thesis, Queue Access Cost (QAC) has been used as a job scheduler, which sends the job to appropriate computing elements that are closer to the data, with less workload and higher percentage availability.

The DRCEM mechanism records the highest value regarding CEU metric, as it has the highest CEU value against ELALW by 23.65% and DRCM by 19.12%. This is because DRCEM allocates the file replicas of the sites taking into account the workload of the sites in the federation and locations of existing file replicas with direct logical dependence on the parent files. The workload information influences the scheduling algorithm to strike a balance while distributing the jobs, as the jobs are sent to the computing elements that are closer to the data and have high percentage availability. In addition, it is observed that the results patterns in Figures 5.13-5.15 showed some oscillations. This was because of some sites CEs may encounter occasional interrupts during the simulations, which may tie down the CE momentarily, or boast the CE's performance in case no interruption occurred.

5.3 Analysis on the Effects of File Dependencies on the Performance Metrics

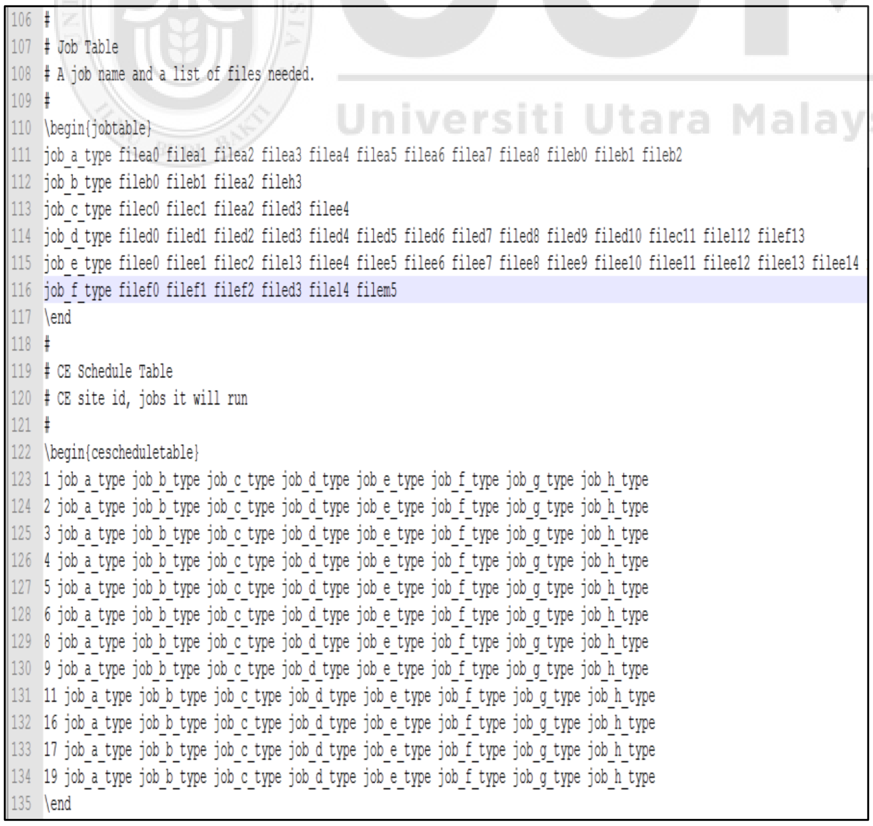
In this section, the same configuration and parameter files as in configuration parameters used in running 5000 Jobs (Table 3.3 Configuration parameters used in the simulations, Page 110) are used, with additional parameters for file dependability. As stated earlier in Chapter Three, this thesis considers both direct and indirect logical dependability based on the assumption that some of the files in the grid depend on one another. Also, other files may logically depend on these dependent files. In this scenario, the thesis used the same types of jobs and files as in the simulator to illustrate the dependability measure. Table 5.6 illustrates types of jobs and the number of files they depended on for running the jobs.

Table 5.6

Sample types of jobs and number of files they depended on for running the jobs

Job_ID	**Number of files or neighbors that file *f_i* is connected to (*k_i*)**	**Link between files *j* and *k* which are neighbors of file *f_i* (e_{jk})**	**Number of dependent files**
A	3	5	8
B	3	6	9
C	4	6	10
D	3	5	8
E	3	5	8
F	3	6	9

From Table 5.6, the number of dependent files for each job is calculated by summing up the direct neighbours and the indirect links. Figure 5.16 is an instance of job configuration file showing logical dependencies amongst the data file, which illustrates the implementation of these jobs in the OptorSim configuration file.



```
#
# Job Table
# A job name and a list of files needed.
#
\begin{jobtable}
job_a_type filea0 filea1 filea2 filea3 filea4 filea5 filea6 filea7 filea8 fileb0 fileb1 fileb2
job_b_type fileb0 fileb1 filea2 fileh3
job_c_type filec0 filec1 filea2 filed3 filee4
job_d_type filed0 filed1 filed2 filed3 filed4 filed5 filed6 filed7 filed8 filed9 filed10 filec11 filel12 filef13
job_e_type filee0 filee1 filec2 filel3 filee4 filee5 filee6 filee7 filee8 filee9 filee10 filee11 filee12 filee13 filee14
job_f_type filef0 filef1 filef2 filed3 filel4 filem5
\end
#
# CE Schedule Table
# CE site id, jobs it will run
#
\begin{cescheduletable}
1 job_a_type job_b_type job_c_type job_d_type job_e_type job_f_type job_g_type job_h_type
2 job_a_type job_b_type job_c_type job_d_type job_e_type job_f_type job_g_type job_h_type
3 job_a_type job_b_type job_c_type job_d_type job_e_type job_f_type job_g_type job_h_type
4 job_a_type job_b_type job_c_type job_d_type job_e_type job_f_type job_g_type job_h_type
5 job_a_type job_b_type job_c_type job_d_type job_e_type job_f_type job_g_type job_h_type
6 job_a_type job_b_type job_c_type job_d_type job_e_type job_f_type job_g_type job_h_type
8 job_a_type job_b_type job_c_type job_d_type job_e_type job_f_type job_g_type job_h_type
9 job_a_type job_b_type job_c_type job_d_type job_e_type job_f_type job_g_type job_h_type
11 job_a_type job_b_type job_c_type job_d_type job_e_type job_f_type job_g_type job_h_type
16 job_a_type job_b_type job_c_type job_d_type job_e_type job_f_type job_g_type job_h_type
17 job_a_type job_b_type job_c_type job_d_type job_e_type job_f_type job_g_type job_h_type
19 job_a_type job_b_type job_c_type job_d_type job_e_type job_f_type job_g_type job_h_type
\end
```

Figure 5.16. An instance of job configuration file showing logical dependencies

From Figure 5.16 for instance, *job_a_type*, denoted by A contains (3+5 = 8) dependent files. Similarly, both *job_b_type* denoted by B, and job_f_type denoted by F contains (3+6 = 9) dependent files. In addition, *job_c_type* denoted by C, contains (4+6 =10) dependent files. Both job D and E contains eight dependent files each.

From Figure 5.16, for example, *job_a_type* depends on eight files namely, *filea0, filea1, filea2, filea3, filea4, filea5, filea6, filea7, filea8, fileb0, fileb1* and *fileb2*. Similarly, *job_b_type* depends on four files namely; *fileb0, fileb1, filea2* and *fileh3*. The *Job_a_type* indicates name of a job of type *a* submitted to the DGF environment for execution.

Other jobs types include *job_b_type*, *job_c_type*, *job_d_type*, *job_e_type*, up to *job_h_type,* in that order. In the simulator, each file has a name, and each job is associated to a particular file name. The simulation results of DRCEM and the existing mechanisms under review are presented in Table 5.7, based on the specified performance metrics.

Table 5.7
Performance of DRCEM and the existing mechanisms with file dependencies

Number of Jobs	**Metrics**	**DRCEM**	**ELALW**	**DRCM**
5000	Jobs Completion Times	5683	8245	7845
	ENU	82.25	94.26	88.37
	SEU	20.53	50.65	49.61
	CEU	56.78	52.35	53.67

The assumption made in this thesis is that the total percentage logical dependency of a data file varies from 0% and 75%. The maximum logical dependency of a data file should not be above 75%. This was in accordance with the existing DRCM mechanism, which reported that dependency label beyond 75%, may result to instability on the dependent files performances. When a file does not have any dependents, the minimum logical dependency of zero is attained. For instances, the logical dependency value of *job_b_type* equals the sum of percentage logical dependencies of all the *b* dependent files thus; 8% of *fileb0*, 7% of *fileb1*, 8% of *filea2*, and 8% of *fileh3* resulting to total percentage logical dependency value of 31%, which is less than 75%. To compare the performance of the DRCEM mechanism against the existing mechanisms, the efficiency values are computed using Equation 5.1 and the results are presented in Table 5.8.

Table 5.8
Efficiency of DRCEM against ELALW and DRCM on file dependencies

Performance Metrics	**DRCEM vs. ELALW**	**DRCEM vs. DRCM**
Jobs Completion Times	31.07%	27.56%
ENU	12.74%	6.93%
SEU	59.47%	58.62%
CEU	-8.46%	-5.79%

5.3.1 Effects of File Logical Dependencies on Jobs Completion Times

From Table 5.7 & 5.8, the results of experimenting with logical dependencies amongst replica files are outlined. It is observed that the jobs completion time increases with the increase in the dependencies amongst the files. The increase in *jobs completion time* is because of the additional burden of both direct and indirect relations with other files, whose execution times are added to the execution time of the parent file. The following Figure 5.17 shows that DRCEM achieves faster jobs completion times, with percentage efficiency of 31.07% and 27.56% over ELALW and DRCM mechanism, respectively.

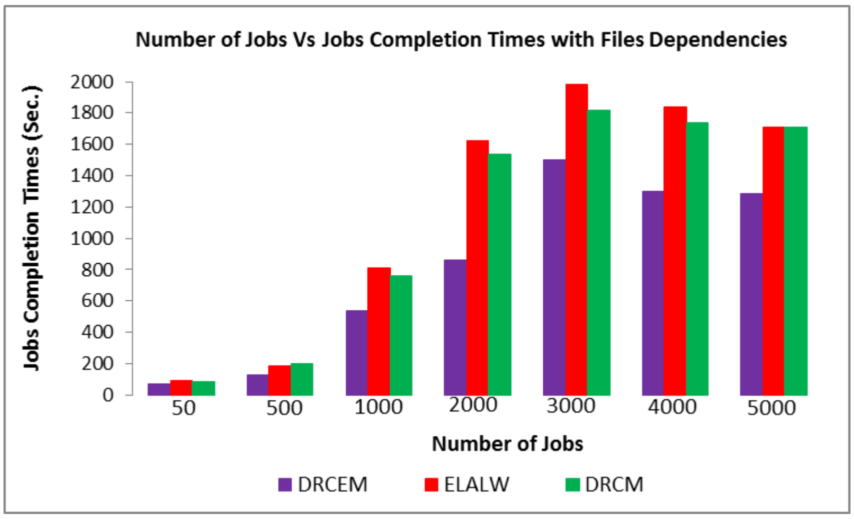


Figure 5.17. DRCEM jobs completion times with files dependencies

In effect, the time required to execute a job is increased with the corresponding increase in the dependency values. The result shows that DRCEM performs better than the existing mechanisms regarding jobs completion time, ENU and SEU, by completing 5000 jobs at a lesser time, consuming fewer networks and less storage facility. However, on the part of CEU, a negative value is obtained for the efficiency of DRCEM against both ELALW and DRCM mechanisms.

The interpretation of the negative value is that DRCEM recorded higher CEU usage than both ELALW and DRCM, which resulted in efficiency value of -8.46% and -5.79%, respectively. Although DRCEM recorded higher CEU, this does not make it worst mechanism going by its better performance in jobs completion time, SEU, and ENU. Usually, a trade-off has to be made amongst some of the performance metrics. It is unusual for a mechanism to obtain lower values for all the performance metrics. Whereas lower values indicate better performance in some metrics such as jobs completion times and ENU, other metrics perform better at higher values, as the case with CE usage.

5.3.2 Effects of File Logical Dependencies on Effective Network Usage

The effect of files logical dependencies on ENU is shown in Figure 5.18. The result indicates that DRCEM makes better use of the network bandwidth. This is due to the DRCEM's replication behaviours of placing file replicas for a job closer to the files logical dependents. Because the CE requires not only the original file that has been requested by the users but also all of its dependent files, to execute the job. Thus, file worth is determined by taking into considerations the number of logical connections it has with other files.

Data transfer time would have been a serious issue if the replica placement failed to consider logical file dependencies while deciding on where to place replica files. In other words, placing dependent files far apart will result to longer data transfer time for all the replicas required to execute a specific job.

DRCEM outperforms both ELALW and DRCM by 12.74% and 6.93% regarding ENU, respectively. Placing replica files closer to their logical dependents reduces file transfer time, thereby narrowing bandwidth consumption.

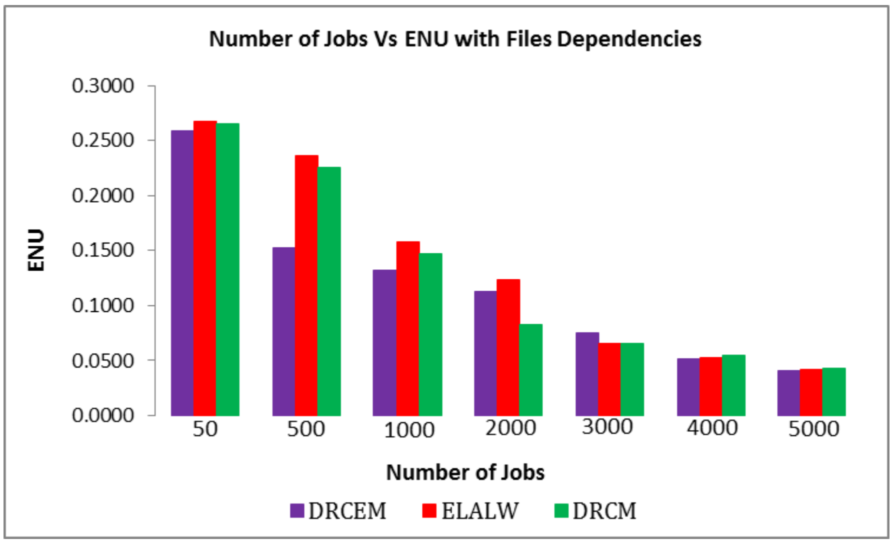


Figure 5.18. DRCEM ENU and existing mechanisms with files dependencies

5.3.3 Effects of File Logical Dependencies on Storage Element Usage

The following Figure 5.19 shows the result of using logical file dependencies on storage element usage for simulating 5000 jobs, at the interval of 50, 500, 1000, up to 5000.

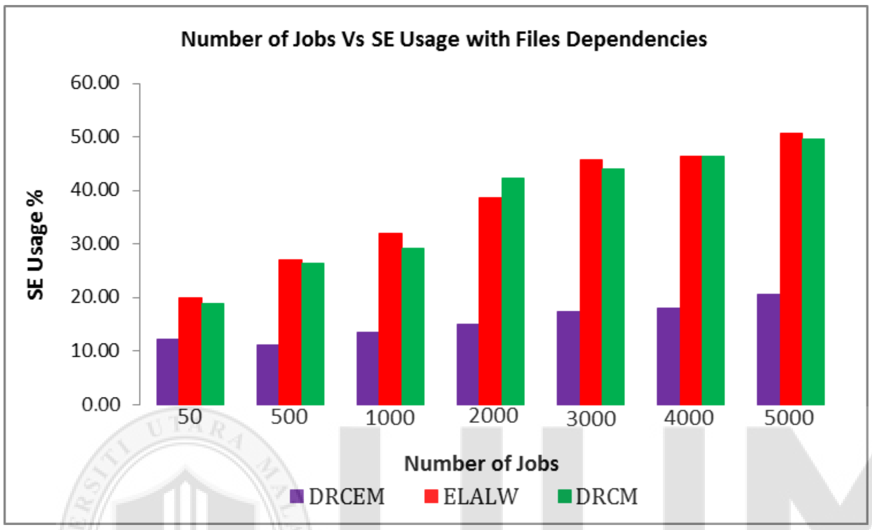


Figure 5.19. DRCEM SEU and existing mechanisms with files dependencies

The storage management of DRCEM shows better performance compared to the existing mechanisms with 59.47% percent efficiency over ELALW and 58.62% percent efficiency over DRCEM. This is one of the advantages of controlling the number of replications; saves storage space, which is a significant issue in the federation environment.

5.3.4 Effects of File Dependencies on Computing Element Usage

The performance of DRCEM regarding the use of computing element is shown in Figure 5.20. DRCEM recorded higher CEU usage than both ELALW and DRCM mechanism, which resulted in negative efficiency value of -8.46% and -5.79%, respectively.

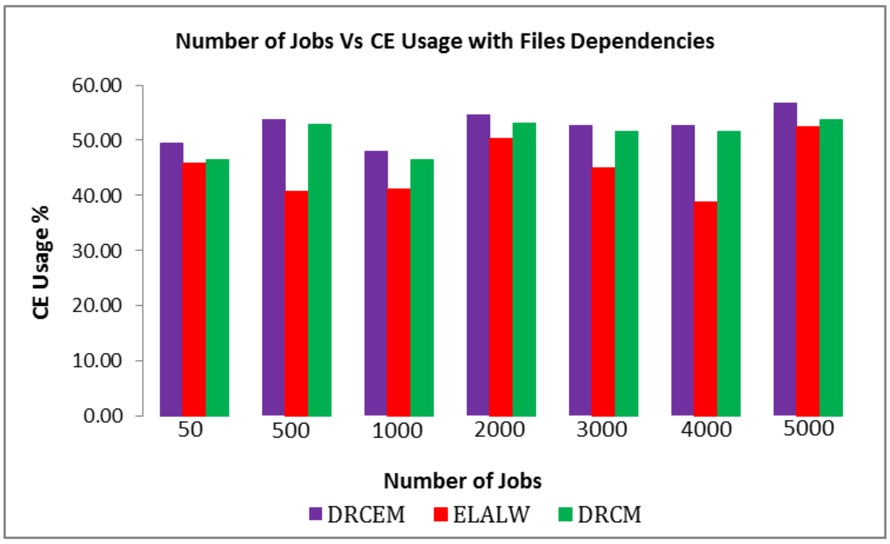


Figure 5.20. DRCEM CEU and existing mechanisms with files dependencies

The DRCEM higher CEU usage is to be expected due to the number of computations needed to take care of the logically dependent files that are required to execute the jobs successfully. Although DRCEM recorded higher CEU, this does not make it the worst mechanism considering its better performance in MJET, SEU, and ENU.

5.4 Effect of Site Availability on Replications

Figure 5.21 shows the performance of DRCEM against existing mechanisms on replications.

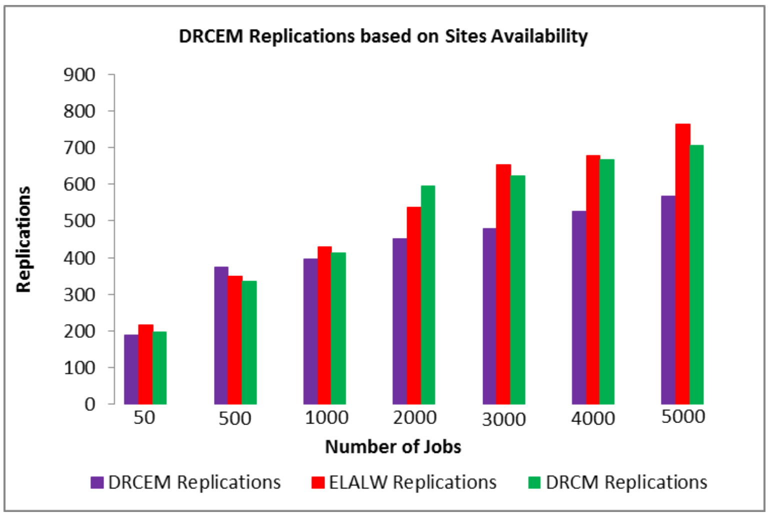


Figure 5.21. DRCEM and existing mechanisms on number of replications

Site availability is one of the major deciding factors for replica placement decision. That is to say before the mechanism decides where to place new file replica, the percentage availability of the site locations is compared with the percentage availability of all the sites within the regions. If the availability is greater than or equal to 85%, then replica placement takes place. Otherwise, loops until this condition is satisfied. The result indicates that for 5000 jobs, DRCEM replicates 568 data files to sites locations with higher availability. On the other hand, ELALW and DRCM replicate 765 and 705 files respectively.

In addition to site availability, DRCEM replication considers site workload and distance, in order not to constrain the storage resources; a reason why the number of replication is lower than the existing mechanisms. Thus, the total replication is done under 50% of the system resources. Thus, replication should not exceed the threshold value set by the system admin, which in this thesis; the assumption was that the threshold value set by system admin is 50%.

5.5 Analysis on Indirect Logical Dependability

As explained in Chapter Three, DRCEM evaluates popular files based on access frequencies and indirect logical dependencies, against direct logical dependencies as the case with the DRCM mechanism. The result indicates that ILD values increase sharply with corresponding increase in the indirect dependents links between replica files. ILD values have effect on the file value evaluation, as would be seen in Figure 5.26, the higher the ILD values, the higher the file values. The following Figure 5.22 shows DRCEM ILD values computations for 16 case data files. The figure indicated that dependability increases with an increase in the number of both direct and indirect dependent files.

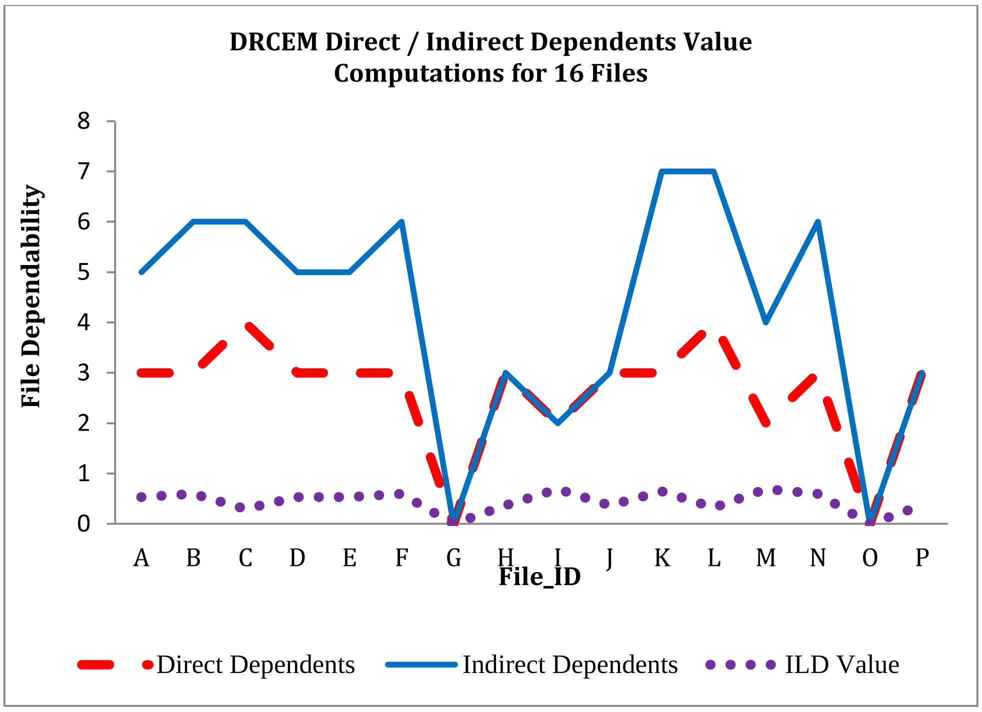


Figure 5.22. DRCEM ILD values computations for 16 case data files.

This shows that DRCEM has potentials over existing mechanisms by considering both direct and indirect relations while evaluating files for replica creation, as seen in the following Figure 5.23.

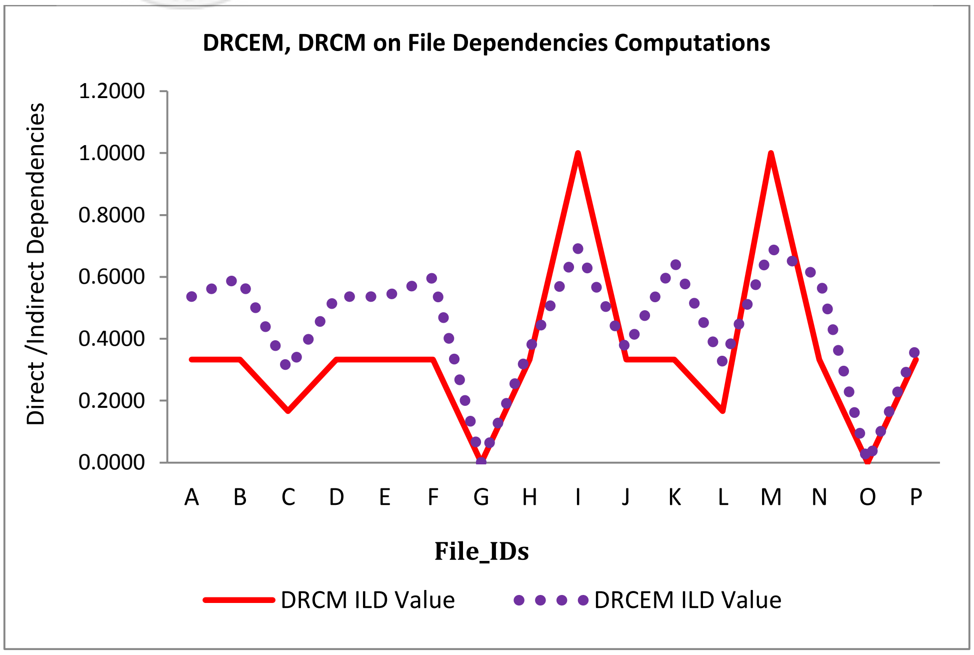


Figure 5.23. DRCEM, DRCM file dependencies computations for 16 case data files

5.6 Analysis on Access Frequencies and File Weights

The results of DRCEM FLT and FW are shown in the following Figures 5.24 and 5.25.

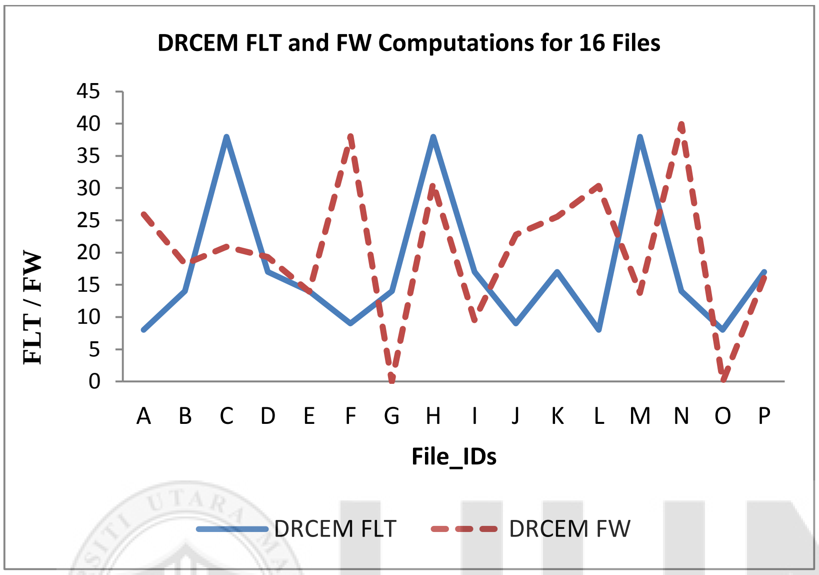


Figure 5.24. DRCEM FW values computations for 16 case data files.

Figure 5.25 compares DRCEM and existing mechanisms on FLT and FW computations.

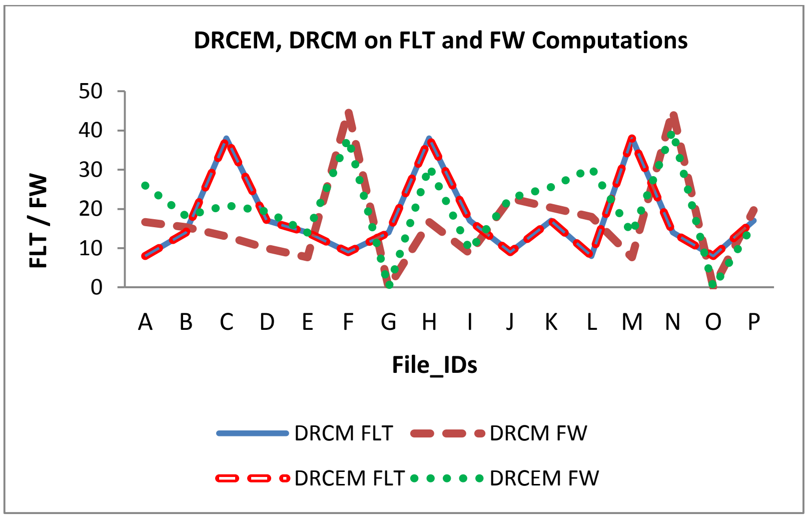


Figure 5.25. DRCEM, DRCM FW values computations for 16 case data files.

From the results of Figure 5.24, it is clear that FLT, which is determined from access frequencies, has a positive effect on FW. It is evident that FW rises with a corresponding increase in FLT. Thus, finding popular file by DRCEM mechanism makes an effective decision by considering indirect logical dependencies, in addition to the direct logical dependencies. Because FW is determined from FLT and ILD, and FV is determined from FLT and FW, which means ILD assumes a significant part in determining the importance of data files in relation to other files within the federation sites.

5.7 Analysis on Access Frequencies and File Values

ELALW mechanism evaluates data files based on half-life algorithm, which assumes the value of data files depreciates by half of its original value. To assess the files in the present time window, the file values within the fast time intervals are computed, and this is referred to as the file access frequency. As a result, the values of the file will be quite large even though the files are not being accessed anymore. On the other hand, DRCEM, similar to the DRCM mechanism, evaluates data files based on file appreciation/depreciation principle. Figure 5.26 shows DRCEM result for evaluating 16 data files, compared with the existing DRCM and ELALW mechanisms on FLT, ILD, FW and FV.

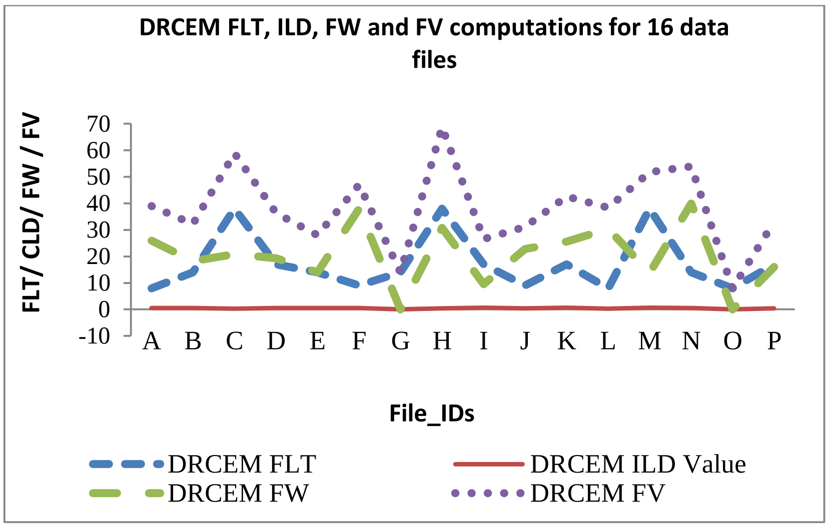


Figure 5.26. DRCEM popular file evaluation computations for 16 case data files.

The average appreciation/depreciation rate is calculated and substituted in the file evaluation formula, to evaluate the files in a current time interval; see how DRCEM evaluates data files in Table 4.1 (on page 136). Difference between DRCEM and the existing mechanisms is that, compared with the values of a recent number of access for the files, DRCM generates file values without taking into considerations, the indirect (transitive) logical dependencies amongst these files. On the other hand, DRCEM generates file values by considering both direct and indirect logical dependencies between replica files.

Thus, the file values generated by DRCEM are more realistic than that of DRCM and ELALW mechanisms, which operates similar mode of generating file values, using direct logical relationships only. Figure 5.27 shows comparative results for finding FV, between DRCM and DRCEM in evaluating 16 data files.

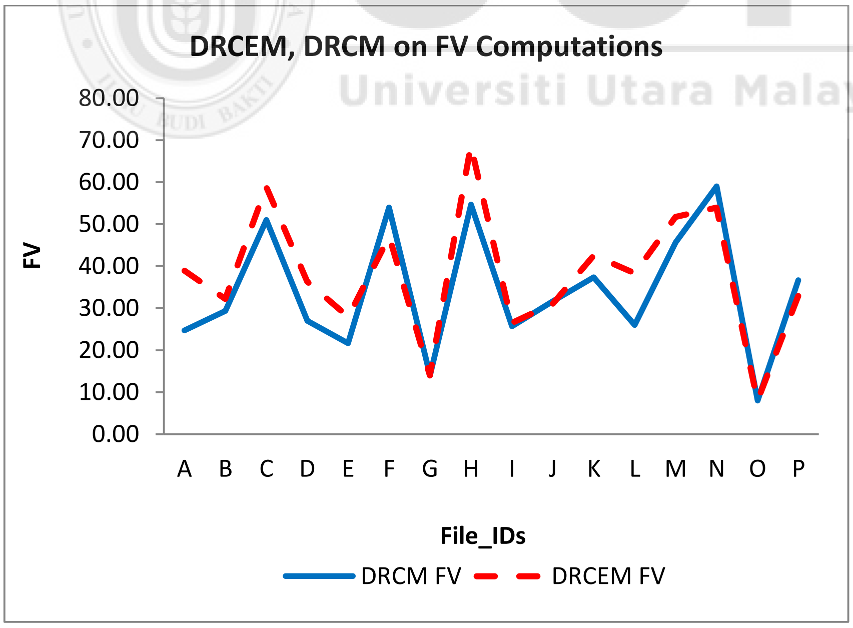


Figure 5.27. DRCEM and DRCM on FV computations for 16 case data files.

5.8 Chapter Summary

This chapter evaluates the performance of the proposed mechanism for dynamic replica creation and eviction, otherwise known as DRCEM mechanism. The performance evaluation commenced with Section 5.2, which discussed the DRCEM performance and the existing using different design metrics. The performance evaluation considers the four primary performance objectives aimed to minimise the jobs completion times, the network bandwidth consumption, the storage element and computing element usage. To evaluate the above objectives, this thesis uses different measurement metrics namely number of submitted jobs, different file sizes, and distance between replica sites, site availability, site workloads and replica placement cost.

Reducing jobs completion times and bandwidth consumption could be achieved by reducing the distance between source site for the job and all other sites hosting the dependent files required by the job. Furthermore, jobs completion times and bandwidth consumption could be minimised by reducing the site workload, reducing replication cost as well maintaining high level of site availability. In addition, minimising storage usage could be attained by controlling the number of files replications as well ensuring load balancing in the system, which is done at the expense of computing element usage. In other words, although the mechanism saves storage by avoiding unnecessary replications, however, there is excessive usage of computing element due to additional task of locating files closer to the jobs files that needed them, as well as computations for site availability and workloads.

Furthermore, load balancing is achieved by avoiding highly loaded sites and considering only lightly or moderately loaded sites for replica placement. In this research, the DRCEM covers both aspects, which provides improved performance over the existing mechanisms. The extensive experiments in this chapter have shown that the performance of DRCEM is better than ELALW and DRCM in all the tested performance metrics.

CHAPTER SIX

CONCLUSION AND FUTURE WORK

6.1 Introduction

This chapter presents an overview and conclusion on the research work carried out in the thesis. The research contributions are supported by the simulation results, which are highlighted. The feasibility of applying the proposed DRCEM mechanism in the real works of life is presented. Recommendations on several possible future research directions to realise and extend the work in this thesis are also identified and recommended at the end. The chapter begins by reviewing the process involved in the implementations of the DRCEM mechanism and the mechanisms used for comparisons. The implementation began with the outline of the schemes for the proposed DRCEM mechanisms in Chapter 4. The schemes were fully developed, programmed and deployed to OptorSim simulator for performance evaluation.

In addition, numerical examples were given at relevant sections of Chapter 4 to illustrate how the three schemes of the DRCEM mechanism interact to achieve the stated objectives. After the implementation processes, simulations were run and results were collected for performance evaluation. It was seen that both DRCEM and existing mechanisms produced similar results patters for different number of submitted jobs. In addition, it was shown that DRCEM performs better than the existing mechanisms in terms of number of replications performed with increasing number of submitted jobs.

The performance evaluation was carried out in Chapter 5, which revealed that DRCEM performs well compared to ELALW and DRCM in terms of the measured metrics (jobs completion times, network bandwidth usage, storage element usage and computing element usage). It was however shown that DRCEM performs fair compared to existing mechanisms regarding computing element usage, due to excessive computations of dependent files and related computational tasks. Also, it was established that DRCEM achieves faster jobs with minimal number of replications compared to the existing mechanisms. In other words, DRCEM runs jobs faster by not wasting precious time making unnecessary replications, considering that files replicas are placed closer to jobs that need them.

In practice, data replication presents a dynamic means of making data more available and placing the data closer to the numerous users of the DGF system. The need for data replication is to spread duplicates of the identical data items to diverse sites, to facilitate disaster recovery operations, in the case of data unavailability due to site failure, site maintenance, unexpected downtimes or demands from the users.

Therefore, the proposed DRCEM mechanism has been designed and implemented to annul some of the scoring effects of limited resources that impede on data exploration in the established domain, which calls for the need of an enhanced data replication mechanism. The main problem addressed by this thesis regarding replica creation, placement and eviction decision include file sites communications, file dependencies, sites failures, sites distances and sites workloads, aim at satisfying both the DGF and its users constraints. Sites communications have been minimised by replicating important data items to appropriate sites with the regions of the federation, closer to users.

Satisfying the DGF system and users constraints is a measure of reducing the stringent cost functions involving the storage usage, bandwidth usage, and file transfer time costs. As regards to users satisfaction, reduction in file transfer time has significantly impacted in reducing the cost for placing file replicas, which aids in minimising the overall jobs times of the users. This problem has been addressed by existing DRCM and ELALW mechanisms, but their works still needed enhancements, as has been discussed under the review of related literature in Chapter 2 of this thesis.

In addition, deciding on replica placement by considering site availability along with some critical parameters has a significant effect on the overall system performance compared to the existing mechanisms. These parameters include the transfer time of data file among sites, workload of each site, and the distance between replica sites.

On the part of DGF storage resources, the constraints due to unbalanced workloads are addressed by considering only lightly and moderately loaded sites for replica placement. The load balancing issue even though has been addressed by previous researchers, however, requires further enhancements to include other relevant factors. Regarding bandwidth usage, the replica placement cost considers site distance by placing file copies closer to sites that frequently need them, which minimises bandwidth usage in the system.

Thus, DRCEM mechanism takes seriously the issues surrounding user's satisfaction and balancing grid storage usage, by looking at both users' requirements and DGF storage resource limitations. Creating and evicting file replicas present an interesting research problem in DGF systems, due to inter-dependability of the replica files. In this thesis, it was observed that replica creation decision by evaluating files based on direct logical dependencies and indirect logical dependencies, in addition to access frequencies, gives added advantage over the existing works.

Replica eviction from a storage element needs to be done with caution in order not to delete a relevant file, which may be needed later. In this thesis, it was observed that considering the file dependability combined with file value in replica eviction presents better enhancements in the performance of the system than considering only file value and size, as in the existing DRCM and ELALW mechanisms.

Also, load balancing is an important issue addressed by this research, which considered lightly and moderately loaded sites, while deciding where to place newly created replica. The load balancing is another vital function similar to the one performed by replica replacement proposed by DRCM, but with the added functionality of taking notes of files with high dependability measure, while evicting replica files.

6.2 Revisiting the Research Objectives

This section discusses the research contributions, by giving comprehensive explanations on the contributions in its various subsections. The significant contribution of this research work is related to the proposing an enhanced replica creation and eviction mechanism that enhances the performance of the DGF system, by reducing jobs completion times, minimising bandwidth usage and optimising the storage and computing element usage within the DGF regions. Therefore, the contributions accorded by this thesis are of utmost significance, which are highlighted above and are explained further, in the following subsections.

6.2.1 An Enhanced Replica Evaluation and Creation Scheme

In this research, an enhanced dynamic replica evaluation and creation scheme (DRECS) is developed, which evaluates replica files based on their file values and logical dependencies. The scheme determines how many copies of the replicas files is required to meet with data demands within the DGF environment. The scheme incorporates a mathematical framework for file logical dependency into the existing popularity-based file evaluation framework proposed by the existing ELALW and DRCM mechanisms, for finding file values within the regions of the established research domain. The framework is based on the exponential growth/decay of file replicas, which is referred to as the file appreciation and depreciation with logical dependency framework.

The first part of the framework establishes files importance by observing the users behaviours of accessing the files over time. The second part of the framework establishes files importance by observing the files behaviours in connection to other files within the regions, otherwise known as file dependability measure. In effect, these two frameworks are jointly employed to determine the value of the popular file, which establishes file’s importance regarding the users as well as the DGF system in general. This is a significant enhancement over the existing works ELALW and DRCM mechanisms, which considered only the users’ behaviours of accessing files replicas over time, to determine the significance of a file to the grid system.

Unlike the existing works, the proposed DRCEM mechanism takes decision on creating more replicas of the important files to meet the demands of the users, based on the outcomes of the two frameworks. Also, the replica eviction provides additional function of creating more space for new incoming files replicas, incase there is not enough space to accommodate the newly created file replicas. However, this is done with due considerations to files with logical connectivities to other files.

6.2.2 An Enhanced Dynamic Replica Placement Scheme

The principle of replication suggests an economical way of hosting the newly created replicas, so as not to constraints the storage resources, as well as not to draw heavily on the bandwidth resources, while accessing such files. Furthermore, creating many replicas is not the best option here, but determining the number of replica copies that satisfies the desired data availability in the DGF system. Therefore, a mechanism that generates a minimum number of replicas, but achieves minimum jobs completion times is more significant than a mechanism that generates more replicas, but ties down the system performance. Thus, the new dynamic replica placement scheme (DRPS) proposed by this work selects the suitable location sites that provide the least transfer time, by placing file replicas on sites that offer minimum replica placement cost. Minimum replica placement cost is achieved by considering file transfer time and distance between replica sites.

The replica placement cost encapsulates file access cost, which is the time required to convey the replicas from source to destination sites. In other words, this denotes the elapsed time from the moment a site sends a request for a data file till the time the entire data file is transferred to its location. Thus, the best location sites is achieved by considering file access cost, file transfer time, site workload, sites distance, site availability, and logical inter-dependencies of replica files.

DRCEM performs less number of replications, compared to ELALW and DRCM, because replication is guided by the aforementioned factors, which help to perform adequate number of replications that improves the jobs execution times. Controlling the number of replications has two advantages. In the first place, it saves storage by not performing unnecessary replications, and secondly, it boasts jobs times. In contrast, the existing works waste precious time by performing unnecessary replications. Furthermore, although replication is known for improving data availability, this needs to be with cautions; not to constraints the storage and the users by slowing down their jobs times, and not to constraints the DGF system by creating bottlenecks.

6.2.3 An Enhanced Dynamic Replica Eviction Scheme

The dynamic replica eviction scheme (DRES) is invoked by the DRPS scheme, if the target storage at the site that is selected for placing newly created replica is full. The DRPS first tries to locate sites that are lightly or moderately loaded for replica placement. If none exists, then DRES is invoked to evict insignificant file replicas from the highly loaded sites, to create more space for the newly created file replicas. The DRES scheme determines the victim file to be evicted based on three design parameters, namely file value, direct logical dependability, and indirect logical dependability of the file replicas.

6.2.4 The DRCEM Mechanism

The core advantages of the developed DRCEM mechanism are: firstly, the new mechanism embodies the above three schemes in one mechanism, which are considered as essential functions in replication mechanism for DGF systems. Secondly, the new mechanism performs replication of multi-files in one decision-making, i.e., when the DRCEM decides to carry out the replication or eviction process, the decision will include several files in a one decision.

6.2.5 Implementation of DRCEM in OptorSim Simulator

The implementation could serve as a starting point for other researchers in this domain for further research work, benchmarking or modification purposes. It presents a noble mechanism that encapsulates some essential schemes involved in the replica evaluation, replica creation, replica placement and replica eviction decision. The replica placement achieves load balancing, by avoiding sites that are highly loaded and considering only sites that are lightly or moderately loaded for replica placement. The implemntation of a load balancing scheme strengthens the replica placement, by making sure both the DGF system and the users are not constrained, while accessing data files. Considering the dynamic behaviour of the DGF sites, candidate site on which replicas are currently hosted may not be the suitable site from where to retrieve the replica due to factors such unavailability and site distance, which may interfere with the jobs times. Therefore, the mechanism ensures that replicas are placed on sites that accord better responses considering the prevailing network conditions and peer sites behaviors. Another significant contribution of this thesis, which makes it unique from existing works, could be seen in the implementation of replica maintenance, which encapsulates replica optimisation that maintains required number of data availability with a minimum number of replication as possible.

6.3 Revisiting the Research Contributions

Further to the Research Contributions itemised in the introductory Chapter One (Section 1.7, Page 16), this section is aimed at buttressing more on how the research has contributed to the knowledge domain, which is in more ways than one. It suffices to say that the following contributions have been accorded by the research.

i. The evaluation of popular files by computing logical dependencies amongst the data files, stands as a major contribution to the research domain, which prviously has been based on files access frequencies. File access frequency has been predominantly used by previous researches in determining the popularity of a data file. Finding popular files using access frequencies without logical dependencies may not produce a realistic conclusion on which file is popular or not. This is because, access frequency indicates how frequently a user accesses a particular file, but does not include how frequently such file accesses a another file. Thus, to establish the realistic popularity of a file, its logical connections to other files in the system need to be included in the computation.

ii. Another major contribution by this research is its ability to resolve the problem of sites failures by computing sites availability, which before now, has been a soring issue in determining the appropriate locations sites for replica placement decision. Previous researches have made tremendous efforts in finding appropriate sites locations for replica placements. However, their efforts concentrated on sites workloads and distances. Thus, this research contributed significantly, by computing sites failures along with the workloads and distances prior to making replica placement.

iii. Computation of distance between replica sites has been enhanced via the use of a modified Dijkstra's algorithm, which previously has been based on hops counts. Although using hops counts may suffice in small grid installations, large grid systems may require a robust mechanism such as the case with this research. Thus, this research computes sites distances using bandwidth information with the modified Dijkstra's algorithm, which adds robustness and more efficiency compared to the existing practice.

iv. The issue of replica eviction has been enhanced in such a way that important replicas that may be needed later, are not caressly evicted from the system. Previously, replica eviction has been based on LRU or LFU, which evicts data files based on how frequently they have been accessed by the users within a specified time frame. Considering that some files may connect to other files via partial replica or threading issue, this research considered modelling the possible logical dependencies amongst replica files, which produced positive effects in the various computations of files values, files life times and files wieghts within the DGF system, as seen in the results analysis of Sections 5.5 -5.7.

v. Also, the research has made significant cotribution in making space more available within DGF systems, through the provision of a dedicated dynamic replica eviction scheme, which is invoked when there is need for more storage space by the replica placement stage. The method used by existing works to create space usually evicts least frequently or least recently accessed files by the users, which is an integral part of their mechanism, may result to concurrent operations issues, by attempting to evict a file that is currently being accessed by another file. As resolve to that, a dedicated mechanism for eviction, which checks file dependencies prior to eviction, will surely be more efficient in creating the desired space within the system without creating bottlenecks.

vi. The issue of what file to replicate has been significantly addressed by making sure that only important files are replicated; thus avoiding redundant replications, which may consume the much needed storage resources. This could be seen in the fewer number of replications done by the mechanism compared to the existing mechanisms.

6.4 Future Research Works

The coverage of this research work has unlocked a number of avenues for further study. Despite the rigorous experiments, comprehensive discussions and thorough analysis of the obtained results, yet, this research is subject to further enhancement in many aspects. This section highlights on the possible areas of improvement in this thesis. As the future of Grid Computing is shifting to federation-based computing infrastructure, which evolves to provide a platform, which will support a variety of distributed computing infrastructure, including High Powered Computing (HPC), High Throughput Computing (HTC) and the Cloud Computing, this research could be expanded to cover the full pledge European Grid Infrastructure (EGI) platform.

This thesis addressed resources failure regarding site unavailability. In the future, there are plans to consider modelling of the various failure causes into a single mechanism for a more efficient replica placement decision making. Also, as part of future work, the DRCEM mechanism will be implemented in Cloud Data Centre environment, based on the current EGI platform.

There are plans to extend this replica creation mechanism (DRCEM) to cover and include the replica management strategies, namely replica selection strategy and replica maintenance strategy. Replica selection strategy is responsible for finding the best replica location out of many replicas, which are spread across the DGF to provide the users with the essential replicas in the minimal response time possible, while running their jobs.

Also, the mechanism could be enhanced to include job scheduling scheme. In this research, the mechanism used for job scheduling is the existing Queue Access Cost heuristic implemented in the OptorSim simulator. A scheduling scheme that considers resources failures, site distance as well as site workloads will perform well if joined with the replica placement decision proposed by this work.

The second scheme that could be included in this mechanism is replica maintenance scheme. Due to the dynamic nature of the DGF system, regional sites that host the file replicas may not be the best locations to fetch the replica in the future due to resources failures. Therefore, there is a need to include a replica relocation policy to consider relocating important file replicas that reside on site locations that have high failure rates.

The replica placement avoids sites with high failures; however, if after placing the file replicas, the site changes its availability status, such files should be relocated to other sites with high availability. In other words, important files could be relocated to location sites that could provide better replication cost, taking into considerations the dynamic system behaviours regarding site distance, site availability and site workload.

REFERENCES

[1] R. W. Moore, A. Jagatheesan, A. Rajasekar, M. Wan and W. Schroeder, "DATA GRID MANAGEMENT SYSTEMS", In *NASA/IEEE MSST 2004 Twelfth NASA Goddard Conference on Mass Storage Systems and Technologies*, pp. 1-15, April 2004.

[2] I. Foster and C. Kesselman, *The grid: blueprint for a new computing infrastructure*. Amsterdam: Kaufmann, 2007.

[3] R. Ranjan, A. Harwood and R. Buyya, "A case for cooperative and incentive-based federation of distributed clusters", *Future Generation Computer Systems*, vol. 24, no. 4, pp. 280-295, 2008.

[4] M. Rahmani and M. Benchaiba, "A comparative study of replication schemes for structured P2P networks", In *Proceedings of the 9th International Conference on Internet and Web Applications and Services*, pp. 147-158, 2014.

[5] W. He, H. Li, L. Cui and S. Lu, "Maximizing the Availability of Process Services in Mobile Computing Environments", *2016 IEEE International Conference on Services Computing (SCC), San Francisco, CA*, pp. 483-490, 2016.

[6] P Matri, A Costan, G Antoniu, J Montes and M. S. Pérez, Towards Efficient Location and Placement of Dynamic Replicas for Geo-Distributed Data Stores. In *Proceedings of the ACM 7th Workshop on Scientific Cloud Computing*, pp. 3-9, ACM, June 2016.

[7] M. Qureshi, M. Dehnavi, N. Min-Allah, M. Qureshi, H. Hussain, I. Rentifis, N. Tziritas, T. Loukopoulos, S. Khan, C. Xu and A. Zomaya, "Survey on Grid Resource Allocation Mechanisms", *Journal of Grid Computing, Vol. 12, No. 2*, pp. 399-441, 2014.

[8] N. Mansouri and A. Asadi, "Weighted data replication strategy for data grid considering economic approach", *International Journal of Computer, Control, Quantum and Information Engineering Vol. 8, No. 8*, pp. 1254-1263, 2014.

[9] A. Jagatheesan and R. Moore, "Data grid and grid flow management systems", *Proceedings, IEEE International Conference on Web Services, 2004*, San Diego, CA, USA, 2004, pp. xxix-xxix, doi: 10.1109/ICWS.2004.1314713

[10] V. Khurana, M. Berger and M. Sobolewski, "A federated grid environment with replication services", In *Next Generation Concurrent Engineering, Omnipress, .(ibid*.), 2005.

[11] V. Khurana, "A FEDERATED GRID ENVIRONMENT WITH REPLICATION SERVICES", Doctoral Dissertation, Texas Tech University, 2005.

[12] W. Jiang, Q. Dai and Y. Zhou, "HUST-BioGrid: The deployment and evaluation of a bioinformatics grid platform", *2010 3rd International Conference on Biomedical Engineering and Informatics, Yantai*, pp. 2785-2789, 2010.

[13] B. Bihani, B. K. Oliver and C. Liu, "Oracle International Corporation", *SYSTEM AND METHOD FOR SUPPORTING DATA GRID SNAPSHOT AND FEDERATION*, U.S. Patent 20,160,092,540, 2016.

[14] R. W. Moore, A. Rajasekar and M. Wan, "*Data Grids, Digital libraries, and persistent archives: An integrated approach to publishing, sharing and archiving data"*. Proceedings of the IEEE (Special Issue on Grid Computing), Vol. 93, No. 3, 2005.

[15] A. Bass, and D. Kay, "*Applying Identity Federation to Enable Secure Information Sharing*", A Case Study on Identity Federation between NASA and Lockheed Martin, Lockheed Martin in collaboration with NASA ICAM, TSCP, USA, November 2013.

[16] D. G. Cameron, R. Carvajal-Schiaffino, A. P. Millar, C. Nicholson, K. Stockinger, and F. Zini, "Evaluating scheduling and replica optimisation strategies in OptorSim," *Journal of Grid Computing*, pp. 57-69, March 2004.

[17] M. Lei and S. V. Vrbsky, "A Data Replication Strategy to Increase Data Availability in Data Grids", In *GCA*, pp. 221-227, June 2006.

[18] J. H. Abawajy and M. M. Deris, "Data Replication Approach with Consistency Guarantee for Data Grid", *Computers, IEEE Transactions on, Vol. 63, No. 12*, 2975–2987, 2014.

[19] A. H. Monshi, "*Calculating the Availability of Nodes in a Peer-to-Peer Backup System*", Master Thesis 2011, Uppsala University, Department of Information Technology, URN: urn:nbn:se:uu:diva-160433.

[20] Y. Mansouri, M. Garmehi, M. Sargolzaei, and M. Shadi, "Optimal Number of Replicas in Data Grid Environment", in *First International Conference on Distributed Framework and Applications, 2008, DFmA,* pp. 96-101, 2008.

[21] M. A. Salehi, B. Javadi, and R. Buyya, " Preemption-aware admission control in a virtualized grid federation", In *Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on* pp. 854-861, 2012.

[22] M. Thulin, "Measuring Availability in Telecommunications Networks", Master Thesis: Royal Institute of Technology (KTH) Stockholm, 2004.

[23] B. Meroufel and G. Belalem, "Availability management in the data grid" In *IT Convergence and Services*. Springer Netherlands, pp. 43-53, 2011.

[24] S. Sarra, K. Amar and B. Hafida, "A load balancing strategy for replica consistency maintenance in data grid systems", *Informatica, Vol. 37, No. 3*, 2013.

[25] S. Senhadji, A. Kateb and H. Belbachir, "Increasing Replica Consistency Performances with Load Balancing Strategy in Data Grid Systems", *World Academy of Science, Engineering, and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering, Vol. 7, No. 1*, 153-158, 2013.

[26] M. K. Madi, "Replica Creation Algorithm for Data Grids*", Doctoral dissertation, Universiti Utara Malaysia*, 2012.

[27] Q. Rasool, J. Li and S. Zhang, "Replica Placement in Multi-tier Data Grid", in *Proceedings of 2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing*, pp. 103-108, 2009.

[28] H. Sun, B. Xiao, X. Wang and X. Liu, "Adaptive trade-off between consistency and performance in data replication", *Software: Practice and Experience, Vol. 47, No.* 6, pp. 891-906, 2017.

[29] C. Nicholson, D. G. Cameron, A. T. Doyle, A. P. Millar, and K. Stockinger, "Dynamic data replication in LCG 2008," *Concurrency and Computation: Practice and Experience, Vol. 20*, pp. 1259-1271, 2008.

[30] S. Venugopal, R. Buyya and K. Ramamohanarao, “A taxonomy of data grids for distributed data sharing, management, and processing”, *ACM Computing Surveys (CSUR), Vol. 38, No. 1*, pp. 3, 2006.

[31] T. Amjad, M. Sher and A. Daud, “*A survey of dynamic replication strategies for improving data availability in data grids”,* Future Generation Computer Systems, Vol. 28, pp. 337-349, 2012.

[32] M. Bsoul, I. Phillips and C. Hinde, “MICOSim: A simulator for modelling economic scheduling in Grid computing”, *World Academy of Science, Engineering and Technology, International Science Index,* Vol. *68*, 2012.

[33] C. Nicholson, "The OptorSim Archive of Questions Asked", 2008. [Online]. Available: https://www.scribd.com/document/215874924/OPTORSIM-FAQ. [Accessed: 12- Mar-2015].

[34] R. M. Rahman, K. Barker, and R. Alhajj, "A Predictive Technique for Replica Selection in Grid Environment", In *Seventh IEEE International Symposium on Cluster Computing and the Grid CCGRID 2007,* pp. 163-170, 2007.

[35] R. Ranjan, “Coordinated Resource Provisioning in Federated Grids”, *Unpublished Ph.D. Dissertation, The University of Melbourne, Australia, Department of Computer Science and Software Engineering*, July 2007.

[36] Y. F. Lin, J. J. Wu, and P. Liu, "A List-Based Strategy for Optimal Replica Placement in Data Grid Systems", in *Proceedings of Parallel Processing, 2008. ICPP'08. 37th International Conference on*, pp. 198-205, 2008.

[37] D. G. Cameron, A. P. Millar, C. Nicholson, R. Carvajal-Schiaffino, K. Stockinger and F. Zini, “Analysis of scheduling and replica optimisation strategies for Data Grids using OptorSim”, *Journal of Grid Computing*, Vol. 2, No. 1, pp. 57-69, 2004.

[38] P. Vashisht, R. Kumar and A. Sharma, "Efficient Dynamic Replication Algorithm Using Agent for Data Grid", *The Scientific World Journal*, vol. 2014, pp. 1-10, 2014.

[39] R. M. Rahman, K. Barker, and R. Alhajj, "Replica placement strategies in a Data Grid," *Journal of Grid Computing*, vol. 6, pp. 103-123, 2008.

[40] R. Chang and H. Chang, "A dynamic data replication strategy using access-weights in data grids", *The Journal of Supercomputing*, vol. 45, no. 3, pp. 277-295, 2008.

[41] Z. Zhang, C. Zhang, M. Zuo and Z. Wang, "Dynamic Data Grid Replication Algorithm Based on Weight and Cost of Replica", *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 12, no. 4, 2014.

[42] N. Mostafa, I. Al Ridhawi, and A. Hamza, "An intelligent dynamic replica selection model within grid systems", *IEEE 8th GCC Conference and Exhibition*, Muscat, 2015, pp. (1-6), doi: 10.1109/IEEEGCC.2015.7060061, 2015.

[43] S. Dayyani and M. R. Khayyambashi, “A Novel Replication Strategy in Data Grid Environment with a Dynamic Threshold”, *Databases, Vol 14, No. 17*, 2014.

[44] L. Azari, A. Rahmani, H. Daniel and N. Qader, "A data replication algorithm for groups of files in data grids", *Journal of Parallel and Distributed Computing*, vol. 113, pp. 115-126, 2018.

[45] M. Ciubăncan and M. Dulea, "Implementing advanced data flow and storage management solutions within a multi-VO grid site", *2017 16th RoEduNet Conference: Networking in Education and Research (RoEduNet)*, Targu Mures, pp. 1-4, 2017.

[46] M. A. Mehta, S. Agrawal, and D. C. Jinwala, "Novel algorithms for load balancing using hybrid approach in distributed systems", In *Parallel Distributed and Grid Computing (PDGC), 2012 2nd IEEE International Conference on* pp. 27-32, IEEE, December 2012.

[47] B. Javadi, D. Kondo, J. Vincent and D. Anderson, "Discovering Statistical Models of Availability in Large Distributed Systems: An Empirical Study of SETI@home", *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 11, pp. 1896-1903, 2011.

[48] D. Kondo, B. Javadi, A. Iosup and D. Epema, "The Failure Trace Archive": Enabling Comparative Analysis of Failures in Diverse Distributed Systems", In *10th IEEE/ACM Int'l Symposium on Cluster, Cloud, and Grid Computing (CCGrid)* pp. 398-407, IEEE, 2013.

[49] B. Meroufel and G. Belalem, "Managing Data Replication and Placement based on Availability", *AASRI Procedia*, vol. 5, pp. 147-155, 2013.

[50] R. S. Chang, H. P. Chang and Y. T. Wang, "A dynamic weighted data replication strategy in data grids", *Computer Systems and Applications, AICCSA 2008. IEEE/ACS International Conference on*, pp. 414-421, March 31-April 4 2008.

[51] M. R. Jaju and P. Deshpande, "Dynamic data storage and placement system based on the category and popularity", *International Journal, of Computer Engineering & Technology (IJCET)* Vol. 6, No. 6, pp. 08-15, June 2015.

[52] M. Shorfuzzaman, P. Graham, and R. Eskicioglu, "Popularity-Driven Dynamic Replica Placement in Hierarchical Data Grids", in *Parallel and Distributed Computing, Applications and Technologies, 2008. PDCAT 2008*, pp. 524-531, 2008.

[53] A. Eremeev, G. Korneev, A. Semenov and J. Veijalainen, "The Spanning Tree-based Approach for Solving the Shortest Path Problem in Social Graphs", In *WEBIST 2016: Proceedings of the 12th International conference on web information systems and technologies. Volume 1, ISBN 978-989-758-186-1*, SCITEPRESS, 2016.

[54] O. Almomani and M. Madi, "A GA-Based Replica Placement Mechanism for Data Grid", *International Journal of Advanced Computer Science and Applications (IJACSA), vol. 5, no. 10*, 2014.

[55] Z. Qi, Y. Xiao, B. Shao, and H. Wang, "Toward a distance oracle for billion-node graphs", *Proceedings of the VLDB Endowment, vol. 7, no. 1*, 61-72, 2013.

[56] Q. Xin, T. Schwarz, and E. L. Miller, "Availability in global peer-to-peer storage systems", *Distributed Data and Structures 6, Proceedings in Informatics*, 2004.

[57] J. C. Chu, K. S. Labonte and B. N. Levine, "Availability and locality measurements of peer-to-peer file systems", In *ITCom 2002: The Convergence of Information Technologies and Communications*, pp. 310-321. International Society for Optics and Photonics 2002.

[58] A. Saleh, R. Javidan and M. FatehiKhajeh, "A four-phase data replication algorithm for data grid", *Journal of Advanced Computer Science & Technology*, vol. 4, no. 1, p. 163, 2015.

[59] F. Ben Charrada, H. Ounelli, and H. Chettaoui, "An Efficient Replication Strategy for Dynamic Data Grids", in *Proceedings of International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC),* 2010, pp. 50-54, 2010.

[60] R. Souli-Jbali, M. S. Hidri and R. B. Ayed, "Dynamic Data Replication-Driven Model in Data Grids", In *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, vol. 3, pp. 393-397, July 2015.

[61] G. A. Oliva, F. W. Santana, M. A. Gerosa and C. R. De Souza, "Towards a Classification of Logical Dependencies Origins: A Case Study*", ESEC/FSE, 12th International Workshop on Principles of Software Evolution and the 7th annual*, 2011.

[62] D. Bonacorsi, T. Boccali, D. Giordano, M. Girone, M. Neri, N. Magini and T. Wildish, "Exploiting CMS data popularity to model the evolution of data management for Run-2 and beyond", In *Journal of Physics: Conference Series*, Vol. 664, No. 3, p. 032003, IOP Publishing, 2015.

[63] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling scalable virtual organizations", *International Journal of Supercomputing Applications,* vol. 15, pp. 200-222, 2001.

[64] U. K. Oxon, European DataGrid Project: Experiences of Deploying a Large Scale Testbed for E-science Applications. *Performance Evaluation of Complex Systems: Techniques and Tools: Performance 2002. Tutorial Lectures*, *2459*, pp. 480, 2003.

[65] C. Vázquez, E. Huedo, R. Montero and I. Llorente, "Federation of TeraGrid, EGEE and OSG infrastructures through a metascheduler", *Future Generation Computer Systems*, vol. 26, no. 7, pp. 979-985, 2010.

[66] R. Ranjan, A. Harwood and R. Buyya, "Coordinated load management in Peer-to-Peer coupled federated grid systems", *The Journal of Supercomputing*, vol. 61, no. 2, pp. 292-316, 2010.

[67] M. Petrova-El Sayed, K. Benedyczak, A. Rutkowski, & B. Schuller, "Federated computing on the web: The UNICORE portal", In *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2016 39th IEEE International Convention on* pp. 174-179, May 2016.

[68] Q. H. Vu, M. Lupu and B. C. Ooi, "Peer-to-Peer Computing: Principles and Applications", *Springer, Berlin, Heidelberg,* https://doi.org/10.1007/978-3-642-03514-2, 2010.

[69] R. Ranjan, R. Buyya, , and A. Harwood, "A model for a cooperative federation of distributed clusters", In *HPDC'14: Proceedings of the 14th IEEE International Conference on High-Performance Distributed Computing, Research Triangle Park, North Carolina*, IEEE Computer Society, Los Alamitos, CA, USA, 2005.

[70] J. Basney, T. Fleury and J. Gaynor, "CILogon: A Federated X.509 Certification Authority for CyberInfrastructure Logon", *Concurrency and Computation: Practice and Experience,* Volume 26, Issue 13, pp. 2225-2239, September 2014.

[71] A. Zarochentsev, A. Kiryanov, A. Klimentov, D. Krasnopevtsev, P. Hristov, "Federated data storage and management infrastructure", In *Journal of Physics: Conference Series*, Vol. 762, No. 1, p. 012016, IOP Publishing, 2016.

[72] R. R. Baturin, "*Identity Federation and Its Importance for NASA's Future: The SharePoint Extranet Pilot at Kennedy Space Center (KSC)"*, University of Massachusetts, Amherst, MA 01002, 2013.

[73] M. Mollamotalebi, R. Maghami, and A. S. Ismail, Grid and Cloud Computing Simulation Tools. *International Journal of Networks and Communications*, vol. 3, no. 2, pp. 45-52, 2013.

[74] L. Galli, E. Baracchini, S. Bettarini, F. Bosi, E. Cavallaro, S. Dussoni, M. Minuti, F. Morsani, D. Nicolo, G. Signorelli, F. Tenchini, M. Venturini and J. Walsh, "A Silicon-Based Cosmic Ray Telescope as an External Tracker to Measure Detector Performance", *IEEE Transactions on Nuclear Science*, vol. 62, no. 1, pp. 395-402, 2015.

[75] W. R. Peter, P. Andreas, A. Ali, C. Bi, R. B. Anthony, H. Cole, K. B. Stephen,...and , R. K. Green, "The RCSB protein data bank: integrative view of protein, gene and 3D structural information", *Nucleic Acids Research*, Vol. 45, Issue D1, , pp. D271-D281, 4 January 2017.

[76] C. Grandi, D. Bonacorsi, D. Colling, I. Fisk and M. Girone, "CMS computing model evolution", In *Journal of Physics: Conference Series* Vol. 513, No. 3, p. 032039, IOP Publishing, 2014.

[77] L. Cinquini, D. Crichton, C. Mattmann, J. Harney, G. Shipman, F. Wang,... and Z. Pobre, "The Earth System Grid Federation: An open infrastructure for access to distributed geospatial data," *E-Science (e-Science), 2012 IEEE 8th International Conference on*, vol 1, no. 10, pp. 8-12 Oct., 2012..

[78] C. M. Wang, H. M. Chen, C. C. Hsu and C. C. Huang, "Fedmi: A federation middleware for integrating heterogeneous data grids", In *Parallel and Distributed Processing with Applications (ISPA), 2011 IEEE 9th International Symposium on*, pp. 127-134, 2011.

[79] Y. Murakami, M. Tanaka, D. Lin and T. Ishida, "Service Grid Federation Architecture for Heterogeneous Domains" *2012 IEEE Ninth International Conference on Services Computing*, Honolulu, HI, pp. 539-546, 2012.

[80] M. Tang, B. Lee, C. Yeo and X. Tang, "Dynamic replication algorithms for the multi-tier Data Grid",*Future Generation Computer Systems*, vol. 21, no. 5, pp. 775-790, 2005.

[81] D. Bonacorsi and T. Wildish, "Challenging data management in CMS computing with network-aware systems", *2013 IEEE Nuclear Science Symposium and Medical Imaging Conference (2013 NSS/MIC)*, Seoul, pp. 1-6, 2013.

[82] S. Figueira and T. Trieu, "Data replication and the storage capacity of data grids", In *International Conference on High-Performance Computing for Computational Science* pp. 567-575, Springer, Berlin, Heidelberg, 2008.

[83] H. Zhong, Z. Zhang, and X. Zhang, "A Dynamic Replica Management Strategy Based on Data Grid", in *Proceedings of 2010 Ninth International Conference on Grid and Cloud Computing*, 2010, pp. 18-23, 2010.

[84] K. Sashi and A. Selvadoss Thanamani, "Dynamic Replica Management for Data Grid", *International Journal of Engineering and Technology*, vol. 2, no. 4, pp. 329-333, 2010.

[85] F. Berman, G. Fox, and T. Hey, *The Grid: Past, Present, Future, Grid Computing: Making the Global Infrastructure a Reality*. London, UK: Wiley Press, 2003.

[86] D. Nukarapu, B. Tang, L. Wang and S. Lu, "Data Replication in Data Intensive Scientific Applications with Performance Guarantee "*Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 8, pp. 1299,1306, Aug. 2011.

[87] S. M. Park, J. H. Kim, Y. B. Ko, and W. S. Yoon, “Dynamic data grid replication strategy based on Internet hierarchy”, In *International Conference on Grid and Cooperative Computing* pp. 838-846. Springer, Berlin, Heidelberg, December 2003.

[88] J. Mo, “Performance Modeling of Communication Networks with Markov Chains*”, Morgan & Claypool Publishers*, 2010.

[89] C. T. Yang, C. P. Fu, and C. J. Huang, "A dynamic file replication strategy in data grids," in *TENCON 2007-2007 IEEE Region 10 Conference*, pp. 1-5, 2007.

[90] Y. Mansouri, S. T. Azad and A. Chamkori, “Minimizing cost of K-replica in hierarchical data grid environment”, In *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on* pp. 1073-1080, IEEE, May 2014.

[91] M. Bsoul, A. Al-Khasawneh, Y. Kilani and I. Obeidat, "A threshold-based dynamic data replication strategy", *The Journal of Supercomputing*, vol. 60, no. 3, pp. 301-310, 2010.

[92] M. K. Madi and S. Hassan, “Dynamic replication algorithm in Data Grid: a survey”, In *International conference on network applications, protocols, and services*, November 2008.

[93] S. Naseera and K. V. M. Murthy, “Agent-Based Replica Placement in a Data Grid Environment", in *Proceedings of First International Conference on Computational Intelligence, Communication Systems and Networks. CICSYN'09*, pp. 426-430, 2009.

[94] M. Shorfuzzaman, P. Graham and R. Eskicioglu, “QoS-aware distributed replica placement in hierarchical data grids”, In *Advanced Information Networking and Applications (AINA), 2011 IEEE International Conference on* pp. 291-299. IEEE, March 2011.

[95] Z. Challal and T. Bouabana-Tebibel, "A priori replica placement strategy in data grid", in *Proceedings of 2010 International Conference on Machine and Web Intelligence (ICMWI)*, pp. 402-406, 2010.

[96] J. D. Herbsleb, A. Mockus and J. A. Roberts, “Collaboration in Software Engineering Projects: A Theory of Coordination”, In *In Proceedings of the International Conference on Information Systems (ICIS’06), 2006.*

[97] M. Cataldo, A. Mockus, J. Roberts and J. Herbsleb, "Software Dependencies, Work Dependencies, and Their Impact on Failures", *IEEE Transactions on Software Engineering, vol. 35, no. 6*, pp. 864-878, 2009.

[98] F. Magoules, J. Pan, K. A. Tan, A. Kumar and A. Kumar, “Introduction to Grid Computing”, *London UK CRC Press, Taylor and Francis Group*, pp. 10-14, 2010.

[99] C. Hamdeni, T. Hamrouni and F. B. Charrada, “New evaluation criterion of file replicas placement for replication strategies in data grids”, In *P2P, Parallel, Grid, Cloud, and Internet Computing (3PGCIC), 2014 Ninth International Conference on* pp. 1-8. IEEE., 2014

[100] Z. Mohamad, F. Ahmad, A. N. M. Rose, F. S. Mohamad and M. M. Deris, “Implementation of Sub-Grid-Federation Model for Performance Improvement in Federated Data Grid”, *Malaysian Journal of Applied Sciences, vol. 1, no. 1*, pp. 55-67, 2016.

[101] A. Chamkoori, F. Heidari and N. Parhizgar, “Cost Optimisation of Replicas in Tree Network of Data Grid with QoS and Bandwidth Constraints”, *International Journal of Advanced Computer Science and Applications(IJACSA), vol. 8, no. 6*, 2017.

[102] K. Raganathan, A. Lamnitchi, and I. Foster, “Improving Data Availability through Model-Driven Replication for Large Peer-to-Peer Communities”, In*: Proceedings of Global and Peer-to-Peer Computing on Large-Scale Distributed Systems Workshop,* Berlin, Germany, 2002.

[103] A. Abdullah, M. Othman, H. Ibrahim, M. N. Suleiman, and A. T. Othman, “Decentralized replication strategies for P2P based scientific data grid”, *International Symposium on Information Technology (TSim’08)*, Vol. 3, pp.1-8, 2008.

[104] F. Xhafa, V. Kolici, A. Potlog, E. Spaho, L. Barolli, and M. Takizawa, “Data replication in P2P collaborative systems”, *Proceedings of the 7th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing* (3PGCIC), pp. (49-57), 2012.

[105] M. Gueye, I. Sarr and S. Ndiaye, “Database replication in large-scale systems: optimizing the number of replicas”, In *Proceedings of the* ACM *2009 EDBT/ICDT Workshops* (EDBT/ICDT 2009), New York, NY, USA, 3-9. DOI=http://dx.doi.org/10.1145/1698790.169879, 2009.

[106] U. Tos, R. Mokadem, A. Hameurlain, T. Ayav and S. Bora, "Dynamic replication strategies in data grid systems: a survey", *The Journal of Supercomputing*, vol. 71, no. 11, pp. 4116-4140, 2015.

[107] Q. Rasool, L. Jianzhong, G. S. Oreku, Z. Shuo, and Y. Donghua, "A load balancing replica placement strategy in Data Grid", in *Proceedings of Third International Conference on Digital Information Management, ICDIM*, London, UK, pp. 751-756, 2008.

[108] C. T. Yang, C. J. Huang, and T. C. Hsiao, "A Data Grid File Replication Maintenance Strategy Using Bayesian Networks," in *Intelligent Systems Design and Applications, 2008. ISDA'08,* 2008.

[109] F. B. Megino, M. Cinquilli, D. Giordano, E. Karavakis, , M. N. GironeMagini and D. Spiga, “Implementing data placement strategies for the CMS experiment based on a popularity model”, In *Journal of Physics: Conference Series,* Vol. 396, No. 3, p. 032047. IOP Publishing, 2012.

[110] K. Rajaretnam, M. Rajkumar and R. Venkatesan, “RPLB: A Replica Placement Algorithm in Data Grid with Load Balancing”, *International Arab Journal of Information Technology (IAJIT), vol. 13, no.* 6, 2016.

[111] A. Sulistio, C. S. Yeo and R. Buyya, “A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools”, *Software: Practice and Experience, vol. 34, no.* 7, pp. 653-673, 2004.

[112] M. Mollamotalebi, R. Maghami and A. S. Ismail, “Grid and Cloud Computing Simulation Tools”, *International Journal of Networks and Communications, vol. 3, no. 2,* pp. 45-52, 2013.

[113] C. L. Dumitrescu and I. Foster, “GangSim: a simulator for grid scheduling studies”, In *Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05), Volume 02* (CCGRID '05), IEEE Computer Society, Washington, DC, USA, pp. 1151-1158, 2005.

[114] R. Buyya, R. Ranjan, J. Broberg and M. Dias de Assuncao, "*Gridsim: A grid simulation toolkit for resource modelling and application scheduling for parallel and distributed computing*", 2011.

[115] S. K. Patel, A. K. Sharma and G. Gupta, "State of Simulators in Computational Grid System", *International Journal of Computer Applications, vol. 72, no.* 16, 2013.

[116] S. A. Monsalve, F. G. Carballeira and A. C. Mateos, "Analyzing the performance of volunteer computing for data intensive applications", *2016 International Conference on High Performance Computing & Simulation (HPCS)*, Innsbruck, , pp. 597-604, 2016.

[117] D. H. Manjaiah and A. H. Guroob, "Triple integration optimisation techniques in data grid environment using OptorSim simulator", *2017 International Conference on Data Management, Analytics and Innovation (ICDMAI)*, Pune, pp. 138-144, 2017.

[118] F. Jolfaei and A. T. Haghighat, "The impact of bandwidth and storage space on job scheduling and data replication strategies in data grids", In *Computing technology and information management (ICCM), 2012 8th international conference on*, vol. 1, pp. 283-288, IEEE, 2012.

[119] R. L. Anikode and B. Tang, "Integrating Scheduling and Replication in Data Grids with Performance Guarantee", *IEEE Globecom 2011 proceedings*, 2011.

[120] S. M. Abbasi and M. Noorimehr, "A New Dynamic Data Replication Algorithm to improve execution time in Data Grid", *International Journal of Computer Science and Information Security, vol. 14, no.* 6, pp. 185, 2016.

[121] K. Eng, A. Muhammed, M. A. Mohamed and S. Hasan, "Incorporating the Range-Based method into GridSim for modeling task and resource heterogeneity", *IEEE Access* vol. 5, pp. 19457-19462, 2017.

[122] L.T.M. Blessing and A. Chakrabarti, *DRM: a design research methodology*, Springer Verlag, Heidelberg, (2009).

[123] R. K. Jain, "Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurements, Simulation and Modeling*", John Wiley*, 2015.

[124] A. Habbal, (2014), "*TCP Sintok: Transmission Control Protocol with Delay-based Loss Detection and Contention Avoidance Mechanisms for Mobile Ad hoc Networks*", Ph.D. Thesis, School of Computing, Universiti Utara Malaysia, 2014.

[125] S. Vazhkudai, S. Tuecke, and I. Foster, "Replica selection in the Globus Data Grid", in *Proceedings of International Workshop on Data Models and Databases on Clusters and the Grid (DataGrid 2001)*, pp. 106-113.

[126] M. Guizani, A. Rayes, B. Khan, and A. Al-Fuqaha, "Network Modeling and Simulation: A Practical Perspective*", Wiley-Interscience*, 2010.

[127] R. G. Sargent, "Verification and validation of simulation models", *Journal of Simulation, vol. 7, no.* 1, pp. 12-24, 2013.

[128] J. Y. Le Boudec, "Performance Evaluation of Computer and Communication Systems", *No. LCA-BOOK-2010-001. EPFL Press*, 2010.

[129] M. R. K Grace, S. S. Priya and S Surya, "A Survey on Grid Simulators", *International Journal of Computer Science and Information Technology & Security (IJCSITS)*, ISSN, *Vol. 2, No. 6*, pp. 2249-9555, December 2012.

[130] H. Cordier, C. L’Orphelin, S. Reynaud, O. Lequeux, S. Loikkanen and P. Veyre, “From EGEE Operations Portal towards EGI Operations Portal”, In *Data Driven e-Science* pp. 129-140, Springer, New York, NY, 2011.

[131] C. Mairi and M. Nicholson, "File management for HEP data grids*", Ph.D. thesis, University of Glasgow*, 2006.

[132] M. Tang, B. S. Lee, X. Tang and C. K. Yeo, “The impact of data replication on job scheduling performance in the Data Grid”, *Future Generation Computer Systems*, vol. *22, no.* 3, pp. 254-268, 2006.

[133] F. Gagliardi, B. Jones, F. Grey, M. E. Bégin, and M. Heikkurinen, "Building an infrastructure for scientific Grid computing: status and goals of the EGEE project," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, vol. 363,* pp. 1729, 2005.

[134] G. Gai-Mei and B. Shang-Wang, “Design and Simulation of Dynamic Replication Strategy for the Data Grid”, In *Industrial Control and Electronics Engineering (ICICEE), 2012 International Conference on* pp. 901-903, 2012.

[135] N. Sadashiv and S. D. Kumar, “Cluster, grid and cloud computing: A detailed comparison”, In *Computer Science & Education (ICCSE), 2011 6th International Conference on* (pp. 477-482), IEEE, 2011.

[136] G. Mathieu, and J. Casson, GOCDB4, “A New Architecture for the European Grid Infrastructure”, In *Data Driven e-Science,* pp. 163-174, Springer, New York, NY, 2011.

[137] R. Schollmeier, “A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications”, In *Peer-to-Peer Computing, 2001. Proceedings,* pp. 101-102, August 2001.

[138] B. Schroeder and G. A. Gibson, “The computer failure data repository (CFDR)”, In *Workshop on Reliability Analysis of System Failure Data (RAF'07), MSR Cambridge, UK,* March 2007.

[139] Y. Yusof, “Replication strategy based on data relationship in grid computing”, In *Proceedings of the 2nd International Conference on Advanced Information Technologies and Applications, Dubai, UAE,* pp. 379-386, 2013.

[140] W. H. Bell, D. G. Cameron, A. P. Millar, L. Capozza, K. Stockinger, and F. Zini, "Optorsim: A grid simulator for studying dynamic data replication strategies", *International Journal of High-Performance Computing Applications, vol. 17,* pp. 403-416, 2003.

[141] M. Teng and L. Junzhou, "A prediction-based and cost-based replica replacement algorithm research and simulation", in *Proceedings of 19th International Conference on Advanced Information Networking and Applications, (AINA 2005),* pp. 935-940, 2005.

[142] K. Ranganathan and I. Foster, "Identifying Dynamic Replication Strategies for a High-Performance Data Grid", *International Grid Computing Workshop,* pp. 75-86, 2001.

[143] P. R. Katre and A. Thakare, "A survey on shortest path algorithm for road network in emergency services", *2017 2nd International Conference for Convergence in Technology (I2CT),* Mumbai, 2017, pp. 393-396, 2017.

[144] S. Palúch and T. Majer, "Effective and fast implementation of k-shortest paths algorithms", *2017 18th International Carpathian Control Conference (ICCC),* Sinaia, , pp. 284-289, 2017.

[145] "Calculate node availability", *Solarwinds.com*. [Online]. Available: http://www.solarwinds.com/documentation/en/flarehelp/sam/content/core-calculating-node-availability-sw1184.htm. [Accessed: 20- Feb- 2017].

[146] I. Jurdana, "AVAILABILITY MODEL OF COMMUNICATION NETWORKS IN CONNECTING SHIP SYSTEMS USING OPTICAL FIBER TECHNOLOGY", *Shipbuilding: Theory and Practice of Naval Architecture and Naval Techniques,* Vol. 65 No. 3 September 2014.

[147] I. B. Boneva, A. Rensink, M. E. Kurban and J. Bauer, "Graph abstraction and abstract graph transformation (No. TR-CTI)", *Centre for Telematics and Information Technology, University of Twente,* 2007.

Appendix A: Files Access History Sample Workload Data

	Access Intervals (mins)							
	1:00-1:30	1:30-2:00	2:00-2:30	2:30-3:00	3:00-3:30	3:30-4:00	4:00-4:30	4:30-5:00
File_ID	Number of Access							
1	200	150	120	100	200	150	120	100
2	170	200	240	150	170	200	240	150
3	150	130	200	300	150	130	200	300
4	140	180	210	160	140	180	210	160
5	150	190	170	140	150	90	170	140
6	200	160	140	110	200	160	140	110
7	200	170	150	240	200	70	150	240
8	150	130	200	300	150	130	200	300
9	140	180	210	160	140	180	210	160
10	200	160	140	110	200	160	140	110
11	140	180	210	160	140	180	210	160
12	200	120	150	100	200	120	150	100
13	130	150	300	200	130	150	300	200
14	200	170	150	240	200	170	150	240
15	200	150	120	100	200	150	120	100
16	140	180	210	160	140	180	210	160
17	150	190	170	140	150	190	170	140
18	200	160	140	110	200	160	140	110
19	200	170	150	240	200	170	150	240
20	150	130	20	300	150	130	200	300
21	140	180	210	160	140	180	210	160
22	170	200	240	150	170	200	240	150
23	150	130	200	300	150	130	200	300
24	140	180	210	160	140	180	210	160
25	150	190	170	140	150	190	170	140
26	200	160	140	110	200	160	140	110
27	140	180	210	160	140	80	210	160
28	200	120	150	100	200	120	150	100
29	130	150	300	200	130	150	300	200
30	200	170	150	240	200	170	150	240
31	200	150	120	100	200	150	120	100
32	140	180	50	160	140	180	210	160
33	150	190	170	140	150	190	170	140
34	200	160	140	110	200	160	140	110
35	200	170	150	240	30	170	150	240
36	150	130	200	300	150	130	200	300
37	140	180	210	160	140	180	210	160
38	170	200	240	150	170	200	240	150
39	150	130	200	300	150	130	200	300
40	140	180	210	160	140	180	210	160
41	150	190	170	140	150	190	170	140
42	200	170	150	90	200	170	150	240
43	150	130	200	30	150	130	200	30
44	140	180	210	160	140	180	210	160
45	170	200	240	150	170	200	240	150
46	150	130	200	50	150	130	200	30
47	140	180	210	160	140	180	210	160
48	150	190	170	140	150	190	170	140
49	200	150	120	100	200	150	120	100
50	10	200	24	150	17	20	240	15

Appendix B: Site Connectivity Data from Gnutella file sharing Network (2002)

# Directed graph (each unordered pair of nodes is saved once): #p2p-#Gnutella04.txt
#Directed Gnutella P2P network from August 4 2002
#Nodes: 10876 Edges: 39994

#From_ NodeId	To_ NodeId	#From_ NodeId	To_ NodeId	#From_ NodeId	To_ NodeId	#From_ NodeId	To_ NodeId
0	1	83	223	89	274	94	306
0	2	83	224	90	89	94	307
0	3	83	225	90	92	94	308
0	4	83	226	90	223	96	333
0	5	83	227	90	259	96	334
0	6	83	228	90	260	96	335
0	7	83	229	90	261	96	336
0	8	83	230	90	262	96	337
0	9	83	231	90	263	96	338
0	10	84	408	90	264	96	339
1	2	84	820	90	265	96	340
1	11	84	2619	91	249	96	341
1	12	84	2952	91	250	96	342
1	13	84	4166	91	251	97	27
1	14	84	4657	91	252	97	142
1	15	84	4754	91	253	97	502
1	16	84	5031	91	254	97	742
1	17	84	6337	91	255	97	871
1	18	84	6338	91	256	97	872
1	19	87	208	91	257	97	873
3	20	87	241	91	258	97	874
3	21	87	242	92	164	97	875
3	22	87	243	92	275	97	876
3	23	87	244	92	276	97	877
3	24	87	245	92	277	97	878
3	25	87	246	92	278	97	879
3	26	87	247	92	279	97	880
3	27	87	248	92	280	97	881
3	28	88	84	92	281	97	882
3	29	88	85	92	282	97	883
8	30	88	86	92	283	97	884
8	31	88	292	93	107	97	885
8	32	88	293	93	284	99	4
8	33	88	294	93	285	99	103
8	34	88	295	93	286	99	105
8	35	88	296	93	287	99	320
8	36	88	297	93	288	99	321
8	37	88	298	93	289	99	322
8	38	89	148	93	290	99	323
8	39	89	266	93	291	99	324
10	41	89	267	94	299	99	325
10	136	89	268	94	300	99	326

#From_ NodeId	To_ NodeId	#From_ NodeId	To_ NodeId	#From_ NodeId	To_ NodeId	#From_ NodeId	To_ NodeId
10	137	89	269	94	301	99	327
10	138	89	270	94	302	99	328
10	139	89	271	94	303	99	329
10	140	89	272	94	304	99	330
10	141	89	273	94	305	99	331
99	332	112	359	120	442	125	478
101	139	113	86	120	443	125	479
101	313	113	360	120	444	126	244
101	314	113	361	120	445	126	957
101	315	113	362	121	480	126	958
101	316	113	363	121	481	127	1475
101	317	113	364	121	482	127	2364
101	318	113	365	121	483	127	2799
101	319	113	366	121	484	127	3641
106	166	113	367	121	485	127	3681
106	171	115	395	121	486	127	3682
106	343	115	396	121	487	127	3683
106	344	115	397	121	488	127	3684
106	345	115	398	121	489	127	3685
106	346	115	399	122	149	127	3686
106	347	115	400	122	415	128	18
106	348	115	401	122	416	128	429
106	349	115	402	122	417	128	430
107	2981	115	403	122	418	128	431
107	3307	115	404	123	405	128	432
107	6479	116	512	123	406	128	433
107	7381	118	368	123	407	128	434
107	8612	118	369	123	408	128	435
107	8842	118	370	123	409	128	436
107	8843	118	371	123	410	128	437
107	8844	118	372	123	411	129	2518
107	8845	118	373	123	412	129	3309
108	350	118	374	123	413	129	3310
109	385	118	375	123	414	129	3311
109	386	118	376	124	419	129	3312
109	387	118	377	124	420	129	3313
109	388	119	102	124	421	129	3314
109	389	119	103	124	422	129	3315
109	390	119	322	124	423	129	3316
109	391	119	378	124	424	129	3317
109	392	119	379	124	425	130	446
109	393	119	380	124	426	130	447
109	394	119	381	124	427	130	448
111	102	119	382	124	428	130	449
112	329	119	383	125	470	130	450
112	351	119	384	125	471	130	451
112	352	120	97	125	472	130	452
112	353	120	408	125	473	130	453
112	354	120	438	125	474	130	454
112	355	120	439	125	475	130	455
112	356	120	440	125	476	131	140

#From_ NodeId	To_ NodeId	#From_ NodeId	To_ NodeId	#From_ NodeId	To_ NodeId	#From_ NodeId	To_ NodeId
112	358	120	441	125	477	131	328
131	456	141	504	153	549	165	594
131	457	141	505	153	550	165	595
131	458	141	506	153	551	165	596
131	459	141	507	153	552	53	548
131	460	141	508	153	553	165	597
131	461	141	509	153	554	165	598
131	462	141	510	153	555	166	205
131	463	141	511	154	167	166	599
132	3	144	260	154	580	166	600
132	52	144	377	154	581	166	601
132	219	144	513	154	582	166	602
132	222	144	514	154	583	166	603
132	464	144	515	154	584	166	604
132	465	144	516	154	585	166	605
132	466	144	517	154	586	166	606
132	467	144	518	154	587	166	607
132	468	144	519	154	588	167	608
132	469	144	520	157	155	167	609
133	570	147	402	157	556	167	610
133	1210	147	521	157	557	167	611
133	1755	147	522	157	558	167	612
133	2002	147	523	157	559	167	613
133	2773	147	524	157	560	167	614
133	3680	147	525	157	561	167	615
133	4229	147	526	157	562	167	616
133	4230	147	527	157	563	167	617
133	4231	147	528	157	564	168	155
133	4232	147	529	158	346	168	157
136	730	148	40	158	495	168	540
136	1116	148	121	158	505	168	618
136	1198	148	530	158	565	168	619
136	1454	148	531	158	566	168	620
136	4191	148	532	158	567	168	621
136	4694	148	533	158	568	168	622
136	5062	148	534	158	569	168	623
136	5626	148	535	162	570	168	624
136	8253	148	536	162	571	170	78
136	8958	148	537	162	572	170	715
137	289	150	538	162	573	170	716
137	493	150	539	162	574	170	717
137	494	150	540	162	575	170	718
137	495	150	541	162	576	170	719
137	496	150	542	162	577	170	720
137	497	150	543	162	578	170	721
137	498	150	544	162	579	170	722
137	499	150	545	165	589	170	723
137	500	150	546	165	590	171	543
137	501	150	547	165	591	171	600
141	502	153	197	165	592	171	616
141	503	153	473	165	593	171	625

#From_ NodeId	To_ NodeId	#From_ NodeId	To_ NodeId	#From_ NodeId	To_ NodeId	#From_ NodeId	To_ NodeId
171	626	184	666	194	9	206	328
171	627	184	667	194	290	206	641
171	628	184	668	194	351	206	753
171	629	184	669	194	581	206	754
171	630	184	670	194	836	206	755
171	631	184	671	194	4391	206	756
175	92	184	672	194	4596	206	757
175	190	184	673	194	4609	206	758
175	421	185	3578	194	4870	206	759
175	632	185	8662	194	6699	206	760
175	633	187	167	195	278	207	743
175	634	187	407	195	399	207	744
175	635	187	568	195	605	207	745
175	636	187	693	195	701	207	746
175	637	187	694	195	702	207	747
175	638	187	695	195	703	207	748
176	180	187	696	195	704	207	749
176	561	187	697	195	705	207	750
176	633	187	698	195	706	207	751
176	639	187	699	195	707	207	752
176	640	189	674	198	165	213	172
176	641	189	675	198	178	213	348
176	642	189	676	198	675	213	410
176	643	189	677	198	708	213	761
176	644	189	678	198	709	213	762
176	645	189	679	198	710	213	763
177	4	189	680	198	711	213	764
180	304	189	681	198	712	213	765
180	639	189	682	198	713	213	766
180	646	189	683	198	714	213	767
180	647	190	1091	200	724	218	69
180	648	190	1579	200	725	218	564
180	649	190	2793	200	726	218	768
180	650	190	3004	200	727	218	769
180	651	190	4496	200	728	218	770
180	652	190	4579	200	729	218	771
180	653	190	5173	200	730	218	772
181	359	190	5355	200	731	218	773
181	655	190	9273	200	732	218	774
181	656	190	9357	200	733	220	108
181	657	192	684	201	359	220	775
181	658	192	685	201	734	220	776
181	659	192	686	201	735	220	777
181	660	192	687	201	736	220	778
181	661	192	688	201	737	220	779
181	662	192	689	201	738	220	780
181	663	192	690	201	739	220	781
183	664	192	691	201	740	220	782
184	564	192	692	201	741	220	783
184	665	193	700	201	742	221	784

Appendix C: Site Availability Sample Workload Data

Site status record for availability workload data extracted from Failure Trace Archives (FTA, 2007)

#event _id	compone nt_id	node _id	platform _id	Node _name	event _type	start_time	stop_time	event_end _reason
0	0	927	10	"tg-login1"	1	1.15E+09	1.17E+09	NULL
0	0	928	10	"tg-login2"	1	1.15E+09	1.17E+09	NULL
0	0	929	10	"tg-login3"	1	1.15E+09	1.17E+09	NULL
0	0	930	10	"tg-login4"	1	1.15E+09	1.17E+09	NULL
0	0	739	10	"tg-c740"	1	1.15E+09	1.17E+09	NULL
0	0	748	10	"tg-c749"	1	1.15E+09	1.17E+09	NULL
0	0	962	10	"tg-s148"	1	1.15E+09	1.17E+09	NULL
0	0	234	10	"tg-c235"	1	1.15E+09	1.17E+09	NULL
1	0	234	10	"tg-c235"	0	1.17E+09	1.17E+09	NULL
2	0	234	10	"tg-c235"	1	1.17E+09	1.17E+09	NULL
0	0	233	10	"tg-c234"	1	1.15E+09	1.17E+09	NULL
1	0	233	10	"tg-c234"	0	1.17E+09	1.17E+09	NULL
2	0	233	10	"tg-c234"	1	1.17E+09	1.17E+09	NULL
0	0	236	10	"tg-c237"	1	1.15E+09	1.17E+09	NULL
1	0	236	10	"tg-c237"	0	1.17E+09	1.17E+09	NULL
2	0	236	10	"tg-c237"	1	1.17E+09	1.17E+09	NULL
0	0	235	10	"tg-c236"	1	1.15E+09	1.17E+09	NULL
1	0	235	10	"tg-c236"	0	1.17E+09	1.17E+09	NULL
2	0	235	10	"tg-c236"	1	1.17E+09	1.17E+09	NULL
0	0	230	10	"tg-c231"	1	1.15E+09	1.17E+09	NULL
1	0	230	10	"tg-c231"	0	1.17E+09	1.17E+09	NULL
2	0	230	10	"tg-c231"	1	1.17E+09	1.17E+09	NULL
0	0	229	10	"tg-c230"	1	1.15E+09	1.17E+09	NULL
1	0	229	10	"tg-c230"	0	1.17E+09	1.17E+09	NULL
2	0	229	10	"tg-c230"	1	1.17E+09	1.17E+09	NULL
0	0	232	10	"tg-c233"	1	1.15E+09	1.17E+09	NULL
1	0	232	10	"tg-c233"	0	1.17E+09	1.17E+09	NULL
2	0	232	10	"tg-c233"	1	1.17E+09	1.17E+09	NULL
0	0	231	10	"tg-c232"	1	1.15E+09	1.17E+09	NULL
1	0	231	10	"tg-c232"	0	1.17E+09	1.17E+09	NULL
2	0	231	10	"tg-c232"	1	1.17E+09	1.17E+09	NULL
0	0	310	10	"tg-c311"	1	1.15E+09	1.17E+09	NULL
0	0	238	10	"tg-c239"	1	1.15E+09	1.17E+09	NULL
1	0	238	10	"tg-c239"	0	1.17E+09	1.17E+09	NULL
2	0	238	10	"tg-c239"	1	1.17E+09	1.17E+09	NULL
0	0	237	10	"tg-c238"	1	1.15E+09	1.17E+09	NULL
1	0	237	10	"tg-c238"	0	1.17E+09	1.17E+09	NULL
2	0	237	10	"tg-c238"	1	1.17E+09	1.17E+09	NULL
0	0	790	10	"tg-c791"	1	1.15E+09	1.17E+09	NULL

#event _id	component _id	node _id	platform _id	Node _name	event _type	start_time	stop_time	event_end _reason
0	0	588	10	"tg-c589"	1	1.15E+09	1.16E+09	NULL
1	0	588	10	"tg-c589"	0	1.16E+09	1.16E+09	NULL
2	0	588	10	"tg-c589"	1	1.16E+09	1.17E+09	NULL
0	0	587	10	"tg-c588"	1	1.15E+09	1.17E+09	NULL
0	0	586	10	"tg-c587"	1	1.15E+09	1.17E+09	NULL
0	0	585	10	"tg-c586"	1	1.15E+09	1.16E+09	NULL
1	0	585	10	"tg-c586"	0	1.16E+09	1.16E+09	NULL
2	0	585	10	"tg-c586"	1	1.16E+09	1.17E+09	NULL
0	0	584	10	"tg-c585"	1	1.15E+09	1.17E+09	NULL
0	0	583	10	"tg-c584"	1	1.15E+09	1.17E+09	NULL
0	0	582	10	"tg-c583"	1	1.15E+09	1.16E+09	NULL
1	0	582	10	"tg-c583"	0	1.16E+09	1.16E+09	NULL
2	0	582	10	"tg-c583"	1	1.16E+09	1.17E+09	NULL
0	0	581	10	"tg-c582"	1	1.15E+09	1.17E+09	NULL
0	0	580	10	"tg-c581"	1	1.15E+09	1.17E+09	NULL
0	0	579	10	"tg-c580"	1	1.15E+09	1.17E+09	NULL
0	0	467	10	"tg-c468"	1	1.15E+09	1.17E+09	NULL
0	0	468	10	"tg-c469"	1	1.15E+09	1.17E+09	NULL
0	0	742	10	"tg-c743"	1	1.15E+09	1.16E+09	NULL
1	0	742	10	"tg-c743"	0	1.16E+09	1.16E+09	NULL
2	0	742	10	"tg-c743"	1	1.16E+09	1.17E+09	NULL
0	0	741	10	"tg-c742"	1	1.15E+09	1.16E+09	NULL
1	0	741	10	"tg-c742"	0	1.16E+09	1.16E+09	NULL
2	0	741	10	"tg-c742"	1	1.16E+09	1.17E+09	NULL
0	0	744	10	"tg-c745"	1	1.15E+09	1.17E+09	NULL
0	0	743	10	"tg-c744"	1	1.15E+09	1.17E+09	NULL
0	0	746	10	"tg-c747"	1	1.15E+09	1.17E+09	NULL
0	0	745	10	"tg-c746"	1	1.15E+09	1.17E+09	NULL
0	0	459	10	"tg-c460"	1	1.15E+09	1.17E+09	NULL
0	0	460	10	"tg-c461"	1	1.15E+09	1.15E+09	NULL
1	0	460	10	"tg-c461"	0	1.15E+09	1.15E+09	NULL
2	0	460	10	"tg-c461"	1	1.15E+09	1.17E+09	NULL
0	0	461	10	"tg-c462"	1	1.15E+09	1.17E+09	NULL
0	0	462	10	"tg-c463"	1	1.15E+09	1.17E+09	NULL
0	0	463	10	"tg-c464"	1	1.15E+09	1.17E+09	NULL
0	0	464	10	"tg-c465"	1	1.15E+09	1.17E+09	NULL
1	0	464	10	"tg-c465"	0	1.17E+09	1.17E+09	NULL
0	0	465	10	"tg-c466"	1	1.15E+09	1.17E+09	NULL
0	0	466	10	"tg-c467"	1	1.15E+09	1.16E+09	NULL
1	0	466	10	"tg-c467"	0	1.16E+09	1.16E+09	NULL
2	0	466	10	"tg-c467"	1	1.16E+09	1.17E+09	NULL
3	0	466	10	"tg-c467"	0	1.17E+09	1.17E+09	NULL

#event _id	component _id	node _id	platform _id	Node _name	event _type	start_time	stop_time	event_end _reason
4	0	466	10	"tg-c467"	1	1.17E+09	1.17E+09	NULL
0	0	574	10	"tg-c575"	1	1.15E+09	1.15E+09	NULL
1	0	574	10	"tg-c575"	0	1.15E+09	1.15E+09	NULL
2	0	574	10	"tg-c575"	1	1.15E+09	1.17E+09	NULL
0	0	992	10	"tg-s178"	1	1.15E+09	1.17E+09	NULL
0	0	157	10	"tg-c158"	1	1.15E+09	1.15E+09	NULL
0	0	89	10	"tg-c090"	1	1.15E+09	1.17E+09	NULL
1	0	89	10	"tg-c090"	0	1.17E+09	1.17E+09	NULL
2	0	89	10	"tg-c090"	1	1.17E+09	1.17E+09	NULL
0	0	300	10	"tg-c301"	1	1.15E+09	1.17E+09	NULL
0	0	299	10	"tg-c300"	1	1.15E+09	1.17E+09	NULL
0	0	302	10	"tg-c303"	1	1.15E+09	1.17E+09	NULL
0	0	301	10	"tg-c302"	1	1.15E+09	1.17E+09	NULL
0	0	304	10	"tg-c305"	1	1.15E+09	1.16E+09	NULL
1	0	304	10	"tg-c305"	0	1.16E+09	1.16E+09	NULL
2	0	304	10	"tg-c305"	1	1.16E+09	1.17E+09	NULL
0	0	303	10	"tg-c304"	1	1.15E+09	1.17E+09	NULL
0	0	306	10	"tg-c307"	1	1.15E+09	1.17E+09	NULL
0	0	305	10	"tg-c306"	1	1.15E+09	1.17E+09	NULL
0	0	308	10	"tg-c309"	1	1.15E+09	1.17E+09	NULL
0	0	307	10	"tg-c308"	1	1.15E+09	1.17E+09	NULL
0	0	912	10	"tg-c909"	0	1.15E+09	1.15E+09	NULL
1	0	912	10	"tg-c909"	1	1.15E+09	1.17E+09	NULL
0	0	667	10	"tg-c668"	1	1.15E+09	1.17E+09	NULL
0	0	513	10	"tg-c514"	1	1.15E+09	1.15E+09	NULL
1	0	513	10	"tg-c514"	0	1.15E+09	1.15E+09	NULL
2	0	513	10	"tg-c514"	1	1.15E+09	1.17E+09	NULL
0	0	514	10	"tg-c515"	1	1.15E+09	1.15E+09	NULL
1	0	514	10	"tg-c515"	0	1.15E+09	1.15E+09	NULL
2	0	514	10	"tg-c515"	1	1.15E+09	1.17E+09	NULL
0	0	515	10	"tg-c516"	1	1.15E+09	1.17E+09	NULL
0	0	516	10	"tg-c517"	1	1.15E+09	1.17E+09	NULL
0	0	509	10	"tg-c510"	1	1.15E+09	1.17E+09	NULL
0	0	510	10	"tg-c511"	1	1.15E+09	1.17E+09	NULL
0	0	511	10	"tg-c512"	1	1.15E+09	1.17E+09	NULL
0	0	512	10	"tg-c513"	1	1.15E+09	1.17E+09	NULL
0	0	517	10	"tg-c518"	1	1.15E+09	1.15E+09	NULL
1	0	517	10	"tg-c518"	0	1.15E+09	1.16E+09	NULL
2	0	517	10	"tg-c518"	1	1.16E+09	1.17E+09	NULL
0	0	518	10	"tg-c519"	1	1.15E+09	1.17E+09	NULL
0	0	997	10	"tg-s183"	1	1.15E+09	1.17E+09	NULL
0	0	941	10	"tg-s047"	1	1.15E+09	1.17E+09	NULL

#event _id	component _id	node _id	platform _id	Node _name	event _type	start_time	stop_time	event_end _reason
0	0	98	10	"tg-c099"	1	1.15E+09	1.17E+09	NULL
1	0	98	10	"tg-c099"	0	1.17E+09	1.17E+09	NULL
2	0	98	10	"tg-c099"	1	1.17E+09	1.17E+09	NULL
0	0	97	10	"tg-c098"	1	1.15E+09	1.17E+09	NULL
1	0	97	10	"tg-c098"	0	1.17E+09	1.17E+09	NULL
2	0	97	10	"tg-c098"	1	1.17E+09	1.17E+09	NULL
0	0	45	10	"tg-c046"	1	1.15E+09	1.16E+09	NULL
1	0	45	10	"tg-c046"	0	1.16E+09	1.17E+09	NULL
2	0	45	10	"tg-c046"	1	1.17E+09	1.17E+09	NULL
0	0	46	10	"tg-c047"	1	1.15E+09	1.15E+09	NULL
1	0	46	10	"tg-c047"	0	1.15E+09	1.15E+09	NULL
2	0	46	10	"tg-c047"	1	1.15E+09	1.16E+09	NULL
3	0	46	10	"tg-c047"	0	1.16E+09	1.17E+09	NULL
4	0	46	10	"tg-c047"	1	1.17E+09	1.17E+09	NULL
0	0	43	10	"tg-c044"	1	1.15E+09	1.16E+09	NULL
1	0	43	10	"tg-c044"	0	1.16E+09	1.17E+09	NULL
2	0	43	10	"tg-c044"	1	1.17E+09	1.17E+09	NULL
0	0	44	10	"tg-c045"	1	1.15E+09	1.16E+09	NULL
1	0	44	10	"tg-c045"	0	1.16E+09	1.17E+09	NULL
2	0	44	10	"tg-c045"	1	1.17E+09	1.17E+09	NULL
0	0	41	10	"tg-c042"	1	1.15E+09	1.16E+09	NULL
1	0	41	10	"tg-c042"	0	1.16E+09	1.17E+09	NULL
2	0	41	10	"tg-c042"	1	1.17E+09	1.17E+09	NULL
0	0	42	10	"tg-c043"	1	1.15E+09	1.15E+09	NULL
1	0	42	10	"tg-c043"	0	1.15E+09	1.16E+09	NULL
2	0	42	10	"tg-c043"	1	1.16E+09	1.16E+09	NULL
3	0	42	10	"tg-c043"	0	1.16E+09	1.16E+09	NULL
4	0	42	10	"tg-c043"	1	1.16E+09	1.16E+09	NULL
5	0	42	10	"tg-c043"	0	1.16E+09	1.17E+09	NULL
6	0	42	10	"tg-c043"	1	1.17E+09	1.17E+09	NULL
0	0	39	10	"tg-c040"	1	1.15E+09	1.16E+09	NULL
1	0	39	10	"tg-c040"	0	1.16E+09	1.16E+09	NULL
2	0	39	10	"tg-c040"	1	1.16E+09	1.17E+09	NULL
0	0	40	10	"tg-c041"	1	1.15E+09	1.16E+09	NULL
1	0	40	10	"tg-c041"	0	1.16E+09	1.17E+09	NULL
2	0	40	10	"tg-c041"	1	1.17E+09	1.17E+09	NULL
0	0	124	10	"tg-c125"	1	1.15E+09	1.16E+09	NULL
1	0	124	10	"tg-c125"	0	1.16E+09	1.16E+09	NULL
2	0	124	10	"tg-c125"	1	1.16E+09	1.17E+09	NULL
0	0	123	10	"tg-c124"	1	1.15E+09	1.16E+09	NULL
1	0	123	10	"tg-c124"	0	1.16E+09	1.16E+09	NULL
2	0	123	10	"tg-c124"	1	1.16E+09	1.17E+09	NULL
0	0	126	10	"tg-c127"	1	1.15E+09	1.16E+09	NULL