RELATIONSHIP BASED REPLICATION ALGORITHM FOR DATA GRID

YUHANIS YUSOF

SCHOOL OF COMPUTING
UNIVERSITI UTARA MALAYSIA
2014

DISCLAIMER

We are responsible for the accuracy of all opinions, technical comment, factual report, data, figure, illustration in this report. We bear full responsibility for the checking whether material submitted is subjected to copyright or ownership right. UUM does not accept any liability for the accuracy of such comment, report and other technical and factual information and the copyright or ownership right claims.

RESEARCHER

____

YUHANIS YUSOF

ACKNOWLEDGEMENT

The author wish to express her gratitude to Universiti Utara Malaysia for the financial support under the *‘Penyelidikan Individu Berimpak Tinggi’* research grant and for facilitating the management of the research.

Abstract

Data Grid is an infrastructure that manages huge amount of data files and provides intensive computational resources across geographically distributed systems. To increase resource availability and to ease resource sharing in such environment, there is a need for replication services. This research proposes a replication algorithm, termed as Relationship based Replication (RBR) that integrates users, grid and system perspective. In particular, the RBR includes information of three different relationships in identifying file(s) that requires replication; file-to-user, file-to-file and file-to-grid. Such an approach overcomes existing algorithms that is based either on users request or resource capabilities as an individual. The Relationship based Replication algorithm aims to improve the Data Grid performance by reducing the job execution time, bandwidth and storage usage. The RBR was realized using a network simulation (OptorSim) and experiment results revealed that it offers better performance than existing replication algorithms.

Keywords: grid computing, data replication, data grid, replica creation, replica quantity.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

List of Abbreviations

AC	Access Cost
ASU	Average Storage Usage
CE	Computing Element
CE Usage	Computing Element Usage
CDN	Content Delivery Network
ENU	Efficient Network Usage
EU DataGrid	European data grid
FL	File Lifetime
FV	File Value
FW	File Weight
ISP	Information Service Provider
LALW	Last Access Largest Weight
LFU	Least Frequently Used
LRU	Least Recently Used
MJET	Mean Job Execution Time
MONARC	Model Of Networked Analysis At Regional Centers
NoA	Number Of Access
PBRP	Popularity Based Replica Placement
QAC	Queue Access Cost
QL	Queue Length
RBR	Relationship based Replication
RC	Read Cost
RLS	Replica Location Services

RB	Resource Broker
SBU	Simple Bottom-Up
SC	Storage Cost
SE	Storage Element
SE Usage	Storage Element Usage
TCP/IP	Transmission Control Protocol / Internet Protocol
TH	Threshold Value

CHAPTER ONE
INTRODUCTION

1.1 Introduction

With rapid advances in scientific instrumentation and simulation, scientific data are growing fast in both data size and data analysis complexity. The next generation of scientific applications in domains as diverse as high energy physics, climate modeling, and earth sciences involve the production of large datasets from simulations or large-scale experiments. Analysis of these datasets and their dissemination among researchers located over a wide geographic area requires high capacity resources such as supercomputers, high bandwidth networks, and mass storage systems.

The grid computing (Foster & Kesselman, 1999; G.A.Gravvanis, 2009) paradigm unites geographically-distributed and heterogeneous computing, storage, and network resources and provide unified, secure, and pervasive access to their combined capabilities. Therefore, grid platforms enable sharing, exchange, discovery, selection, and aggregation of distributed heterogeneous resources such as computers, databases, visualization devices, and scientific instruments (Venugopal, Buyya, & Winton, 2006). Hence leading to the creation of virtual organizations (Foster, 2002a; Foster, Kesselman, & Tuecke, 2001; Wasson & Humphrey, 2003) by allowing geographically-distributed communities to pool resources in order to achieve common objectives. These resources can be divided into computing or storage units that can be accessed or shared by large numbers of remote users. Computing unit or Computational Grid (Frederic Magoulès, 2010) focuses on supplying computing power, while storage unit or data grid focuses on enabling and facilitating reliable access and sharing of data management resources in widely distributed locations.

A data grid (Chervenak et al., 2003; Foster, Alpert, et al., 2002) is an infrastructure that deals with huge amounts of data to enable grid applications to share data files in a coordinated manner. Such an approach is seen to provide fast, reliable and transparent data access. Nevertheless, data grid creates a challenging problem in a grid environment because the volume of data to be shared is large despite the limited storage space and network bandwidth (Nicholson, Cameron, Doyle, Millar, & Stockinger, 2008; Wilkinson, 2009). Furthermore, resources involved are heterogeneous as they belong to different administrative domains in a distributed environment. It is unfeasible for all users to access a single instance of data (e.g. a data file) from one single organization (e.g. site). This would lead to the increase of data access latency. Furthermore, one single organization may not be able to handle such a huge volume of data by itself.

Motivated by these considerations, a common strategy is used in data grids as well as in distributed systems, and this strategy is known as replication. Replication vouches efficient access without large bandwidth consumption and access latency (A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, & Tuecke., 2001; Chervenak et al., 2002; Guy, Kunszt, Laure, Stockinger, & Stockinger, 2002; Lamehamedi, Shentu, Szymanski, & Deelman, 2003; Lamehamedi, Szymanski, Shentu, & Deelman, 2002; Otoo, Olken, & Shoshani, 2002; Ranganathan & Foster, 2001b). The replication technique is one of the major factors affecting the performance of data grids (You, Chang, Chen, Tian, & Zhu, 2006). Creating replicas can reroute client requests to certain replica sites and offer higher access speeds. Hence, well-defined replication strategies will smooth data access, and reduce job execution cost (Tang, Lee, Tang, & Yeo, 2006). Such a strategy should also be able to deal with dynamic changes in the grid environment, such as dynamic resource availability and access patterns.

Figure 1.1 shows a high-level view example of a worldwide data grid, consisting of computational and storage resources in different countries that are connected by high speed networks. The thick lines show high bandwidth networks linking the major centers and the thinner lines are lower capacity networks that connect the latter to their subsidiary centers. The data which were generated from an instrument, experiment, or a network of sensors is stored in its principal storage site and is transferred to the other storage sites around the world on request through the data replication mechanism. Users query their local replica catalog to locate datasets that they require. The data may be transmitted to a computational site such as a cluster or a supercomputer facility for processing. After processing, the results may be sent to a visualization facility, a shared repository, or to the desktops of the individual users.

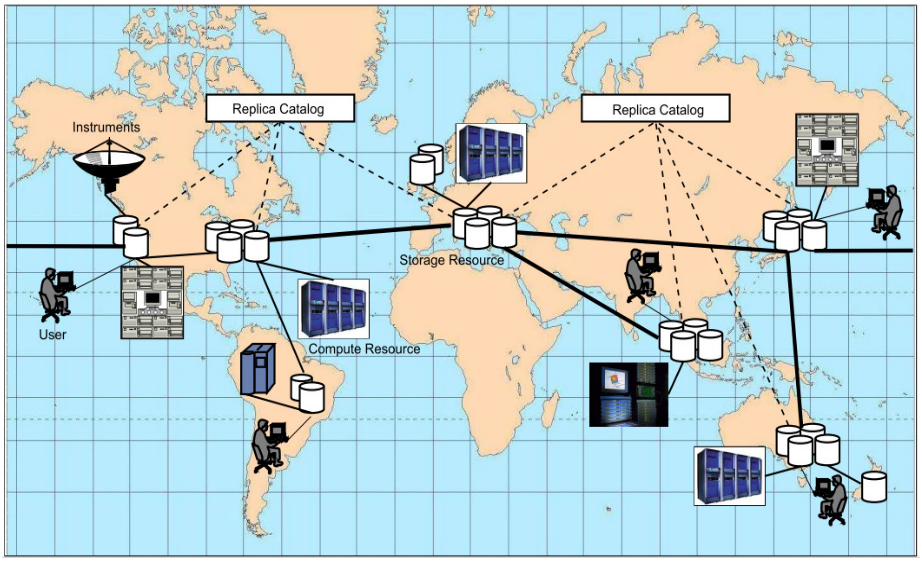


Figure 1.1 A High-Level View of Data Grid (Srikummar Venugopal, 1996)

.

1.2 Research Motivation

Replication can be motivated by two issues, availability of data (fault tolerance) and system performance (Abdelsalam A. Helal, Abdelsalam A. Heddaya, & Bharat B. Bhargava, 1996; Caitriana M. Nicholson, 2006). In a data grid, the high level of reliability (Caitriana M. Nicholson, 2006; Xie, Dai, & Poh, 2004) of the main data storage sites makes fault tolerance less of an issue, while the large file sizes increase the file access times of grid jobs. Therefore, performance becomes the main motivation for replication in data grids.

In the context of the data grid, increasing the performance of the system can be achieved by improving the overall resource usage, which includes network and storage resources (Lamehamedi & Szymanski, 2007). Improving network resource usage is achieved by good utilization of network bandwidth that is considered as an important factor affecting job execution time (Yang, Huang, & Hsiao, 2008). Meanwhile, improving storage resource usage is achieved by good utilization of storage space usage (Al Mistarihi & Yong, 2008).

Performing data replication introduces additional problems: the decision of replication must be wisely made (identifying the appropriate data file to be replicated), replicas must be properly located, their numbers must be properly determined, their lifetime must be managed properly, and the related storage and resources must be utilized efficiently. To sum up, data replication process has to take into account both users' and system's perspectives. Even though these problems can be solved by existing replication algorithms (Chang, 2006; Mansouri, Garmehi, Sargolzaei, & Shadi, 2008; Pangfeng & Jan-Jan, 2006; Ranganathan & Foster, 2001a; Ranganathan, Iamnitchi, & Foster, 2002; Rasool, Jianzhong, Oreku, Shuo, & Donghua, 2008; Ruay-Shiung, Hui-Ping, & Yun-Ting, 2008; Shorfuzzaman, Graham, & Eskicioglu, 2008; Tang, Lee, Tang, & Yeo, 2005; Tang et al., 2006; Tang, Lee, Yeo, & Tang, 2005; Wang, Yang, & Chiang, 2007; Yang, Fu, & Huang, 2007; Yi-Fang, Pangfeng, & Jan-Jan, 2006), existing work require enhancements due to the absence of system's perspective in terms of replication decision making**.**

The replication is performed (deciding the file to be replicated and the required number of replicas) based on users' perspective, i.e. according to number of access of a file. Therefore, the number of times a system makes replication has a possibility to be increased. As a result, the network usage would be affected, this is because each replication consumes network bandwidth and increases network traffic. Moreover, the replication decision of current works does not involve the deletion process of unwanted replicas in their decision. Thus the storage cost will be increased. In this context, storage cost is the space used to store data. Therefore, increasing the storage cost would lead to less storage availability. According to (David G. Cameron, 2005) less storage availability would lead to longer job execution time and larger network usage because only fewer replicas can be accommodated in the data grid, and most files will be read remotely.

1.3 Objectives of the Research

The main goal of this research is to develop a replication algorithm aimed at improving the performance of the data grid system. In order to achieve this goal, the following research objectives were formulated:

a. To formulate a resource selection function to identify which data file requires replication.
b. To formulate a replica quantity function that determines the required number of replicas for the identified data file.
c. To design a replication algorithm that integrates the proposed resource selection and replica quantity functions.
d. To evaluate the proposed algorithm in a simulation environment.

1.4 Significance of the Research

The proposed replication algorithm can be considered as a long-term strategy that aims at best utilization of grid resources usage, namely reducing storage use and reducing network bandwidth consumption. In other words, this proposed algorithm gives a bird's eye view on all components; in a grid environment, the system designers or system administrators would be interested in this view in order to determine the overall resource requirements and to configure, to monitor, and to control the overall system components.

The proposed replication algorithm is also beneficial for grid users as job execution time is reduced. Users' jobs which are under execution would require data files and the grid system in turn would place the required files (i.e. replicas) as close as possible to the users (i.e. requesting sites).

1.5 Scope of the Research

This research focused on replica creation in a data grid system. Details of the scope is as below:

i. Data used in this research is of read-only type. Thus, this research has not considered the consistency of write types and overheads of update propagation costs in this research.
ii. This research focuses on a tree-like-structured grid model, which reflects the hierarchical structure in grid systems (David, 2003; Hoschek, Jaen-Martinez, Samar, Stockinger, & Stockinger, 2000; Ranganathan & Foster, 2001b). The hierarchical data grid model is a common architecture used in various research

works (Abawajy, 2004; David, 2003; Hoschek et al., 2000; Ranganathan & Foster, 2001b).

iii. The modality of data that is used in this work is in the form of structured data, specifically source code data.

1.6 Report Layout

The remainder of this report is organized as follows:

Chapter 2 provides an overview of the background material and establishes the concepts and issues covered in the thesis. In this chapter, a brief critical study and survey of the relevant existing studies are presented.

Chapter 3 describes the steps taken in achieving the defined aim and objectives. The chapter also presents the brief information on the grid architecture and the utilized simulator. The performance evaluation metrics that were used as benchmarks to evaluate the proposed algorithm are also presented in this chapter.

Chapter 4 describes the research solution that is encapsulated in a replica creation algorithm for solving the research problem. The algorithm requirements and design are explained in detail using appropriate examples. This chapter also covers the implementation of the proposed algorithm, which includes the integration of the proposed algorithm into the simulation environment.

Chapter 5 presents the results obtained in the simulation experiments. Additionally, comparison is made with exiting replication algorithms.

Chapter 6 summarizes the research work, highlights research contributions, and gives direction for future work related to this research.

CHAPTER TWO
RELATED WORKS

This chapter first explores: data grid and the challenges of data grid by illustrating some examples of the growth of data requirements for the scientific applications. Then related data-intensive studies are explored in order to provide an overview of the area and domain of this research. Then data replication strategies of Replica Creation Stage are discussed in details, each strategy discussed with the corresponding related works. The analysis of the features and limitations on the state of the art of replica creation stage strategies is performed.

2.1 Data Grid

The term data grid (Allcock et al., 2002; Allcock et al., 2001; Foster, 2002b) refers to an infrastructure that provides data management services for users in order to access, store, transfer, and replicate data files located within distributed storage media. Moreover, a data grid connects a collection of hundreds of geographically distributed computers and storage resources to facilitate sharing of data, storage resources, and computational power (Chervenak et al., 2003; Johnston, 2002).

Through the linking of all these equipment, the Grid can provide a platform through which users can access aggregated computational, storage, and networking resources to execute their data-intensive applications using remote data (Avery, 2002; Foster, Kesselman, Nick, & Tuecke, 2002). It promotes a rich environment for users to analyze data and share the results with their collaborators across institutional and geographical boundaries (Magoulès, Pan, Tan, & Kumar, 2009; Shen, 2008; Srikummar Venugopal, 2006).

2.2 The Challenges in Scientific Grid

The first Grid was conceived by computing science (Gagliardi, Jones, Grey, Bégin, & Heikkurinen, 2005; Hey & Trefethen, 2005). The scale of scientific experiments has grown so fast that traditional methods of computing used to solve associated problems are now quite inadequate. Scientific experiments such as high-energy physics (F. Berman, G. Fox, & Hey., 2003; The LHCb Collaboration. LHCb Computing Model. Technical Report CERN-LHCC-2004-036/G-084, CERN, January 2005), climate modeling, earthquake engineering (Foster, 2000; Fox et al., 2002), bioinformatics (Kelly et al., 2004), and astronomy are generating huge volumes of data which are measured in terabytes and rising to petabytes within just a couple of years (Magoulès & Yu, 2009). There are many examples that illustrate the spectacular growth of data requirements for scientific applications (Yu & Buyya, 2005), as will be described in the following sections.

2.2.1 High Energy Physics

The most cited example of massive data generation in the field of High Energy Physics (HEP) (High Energy Physics Experiment Website) is the Large Hadron Collider (LHC), which is the most powerful giant particle accelerator at CERN (the European Organization for Nuclear Research) (European Organization for Nuclear Research (CERN)). HEP consists of four main experiments namely ALICE (The ALICE Collaboration. ALICE Computing Model. Technical Report CERN-

LHCC-2004-038/G-086, CERN, January 2005.), ATLAS (The ATLAS Collaboration. The ATLAS Computing Model. Technical Report CERN-LHCC-2004-037/G-085, CERN, January 2005.), CMS (CMS Data Challenge 2004; Holtman, 2001; The CMS Collaboration. The CMS Computing Model. Technical Report CERN-LHCC-2004-035/G-083, CERN, January 2005.), and LHCb (The LHCb Collaboration. LHCb Computing Model. Technical Report CERN-LHCC-2004-036/G-084, CERN, January 2005), which are designed to understand the fundamental particles of matter and the forces acting between them. HEP experiments will produce several petabytes of raw and derived data that will be accessed from different centers around the world through very heterogeneous computational resources. The raw data are generated at a single location (CERN) where the accelerator and experiment are hosted, but the computational capacity required to analyze them implies that the analysis must be performed at geographically distributed centers. In practice, CERN's experiments are collaborations among thousands of physicists from about 300 universities and institutes in 50 countries.

2.2.2 Climate Modeling

Another example of science that faces large quantities of data is climate model computations (Chervenak et al., 2003). Climate modeling requires long duration simulations and generates very large files that are needed to analyze the simulated climate (Bernholdt et al., 2005). These simulations, however, will produce tens of petabytes of output in future and if this output is to be useful it must be distributed to climate researchers at various institutions.

2.2.3 Bioinformatics

Genomics require programs such as genome sequencing projects, which produce huge amounts of data. The analysis of these raw biological data requires very large computing resources. Bioinformatics (Kelly et al., 2004) involve the integration of computers, software tools, and databases in an effort to address these biological applications, since genome sequences provide copious information about species from microorganisms to human beings. The analysis and comparison of genome sequences are necessary for the investigation of genome structures which is useful for the prediction about the functions and activities of organisms.

2.2.4 Astronomy

Another data-intensive application in the astronomy field is the Sloan Digital Sky Survey (SDSS) (Sloan Digital Sky Survey website. Available online at: http://www.sdss.org/) which aims to map in detail one quarter of the entire sky and determines the positions and absolute brightness of more than 100 million celestial objects. It will also measure the distances to more than a million galaxies. SDSS and other astronomy applications are performed in several regions of the electromagnetic spectrum and produce an enormous amount of data.

There are many other examples which could be drawn from chemistry (Dooley, Milfeld, Guiang, Pamidighantam, & Allen, 2006), engineering (Farooq, Majumdar, & Parsons, 2007), and earth science (Foster, Alpert, et al., 2002). Suffice to say that science in general is facing a flood of data as technology develops and that in many cases, grids are seen as a viable solution to address these problems.

2.3 Data Grid Layered Architecture

The applications layer provides services and access interfaces for a specific community. These services invoke services provided by the layers below and customize them to suit the target domains, such as high energy physics, biology, and climate modeling.

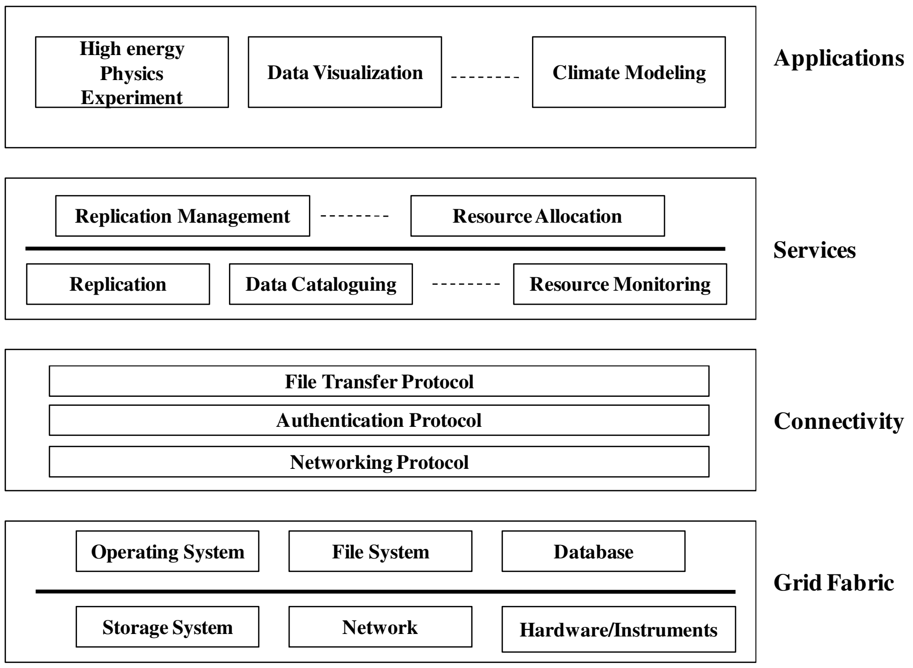


Figure 2.1 Overview of Data Grid Architecture

The services layer is divided into two sub-layers: the high-level sub-layer and the low-level sub-layer. The high-level sub-layers are the services located in the upper layer such as replication management, replica selection optimization, and resource allocation. The high-level sub-layers make use of the low-level sub-layers in order to improve the service quality for users. Replication management service manages the number of replicas and their locations in the grid sites in order to optimize the grid resource usage. However, the replica selection service provides the best replica location for the users or the jobs under execution. The low-level services at the same layer provide services to the upper level such as replication, data cataloguing, and resource monitoring. The data catalogue service provides a number of services such as record all replicas and their physical locations on the grid sites, register the newly created replicas, and delete the replicas from the registry that has been decided to be deleted by the replication management service. The replication service is different from the replication management service. The replication management service decides, and the replication service executes what has been decided by the replication management service. Once the replication management decides to create a new replica, the replication service creates a new copy of the specified file and uses data transfer service to move the copy (replica) to the underlying site location that is determined by the replication management service.

The connectivity layer consists of protocols used to query resources in the grid fabric layer and to conduct data transfers between them. These protocols are built on core protocols for communication such as TCP/IP and file transfer protocols (for example GridFTP). The grid fabric consists of software and physical hardware components such as computing and storage resources.

2.4 Existing Grid System

This section explores the current grid systems and middleware architecture and features by highlighting the replication mechanism.

2.4.1 Storage Resource Broker

SRB (Mathew J. Wyatt, Nigel G.D. Sim, Dianna L. Hardy, & Atkinson, 2007) is a client- server middleware that provides a management system for data replica and a uniform single interface. SRB manages heterogeneous distributed data storage to allow users to access files and database seamlessly. The unified view of the data files stored in disparate media and locations are provided, and transparent to the users so that the dispersed data appears to the user as stored locally (Krishnamurthy, Sanders,

& Cukier, 2002). Data replication in SRB is applicable if the data is required to be much closer to the user (Rajasekar et al., 2003). Replicas can be created using SRB or from outside the system and several forms of data replication are possible.

2.4.2 Grid Data Farm

Grid data farm (Othman, O'Ryan, & Schmidt, 2001) is defined as a group of physical files that distributed across grid sites and appear to the user as a single logical file system that stored in the form of fragments. Individual fragments can be replicated and managed in order to provide service to the data-intensive applications. While executing a program, the process scheduler dispatches it to the site that has the segment of data that is required by the program. If the sites that house the required data are overloaded, the file system creates a replica of the required fragment on another site.

2.4.3 Globus Toolkit

As defined and explained by Ian Foster (Vazhkudai, Tuecke, & Foster, 2001) Globus is:

- A *community* of users who collaborate on sharing of grid resources across cooperate, institutional, and geographic boundaries. Globus also is a community of developers for the development of open source software, and related documentation for building grids and grid based applications for distributed computing and resource federation.
- The *infrastructure* that supports this community such as: code repositories, interface, protocols, email lists, and problem tracking systems.

- The *software* itself, which consists of a set of libraries and programs for solving common problems that occur when building distributed system services and applications.

The Globus data grid architecture (Karl et al., 1998; The Globus Alliance) is divided into two main layers: high-level services and core services, as shown in Figure 2.3. The hierarchical organization explains the possibilities for using the core services to build the high-level service, so that many data management services and complex storage management systems such as Storage Resource Broker (SRB), can share common low level mechanisms. The services that Globus offers are: Security, Information Services, Resource Management, and Data Management. The Information Services provide information about the status of grid resources. The Resource Management uses information from Information Services to enable users to access available resources and to allow the system to schedule resource allocations. The Data Management provides the ability to access and manage data and data resources on the grid [13]. The Globus toolkit provides several components to move, copy, and locate data.

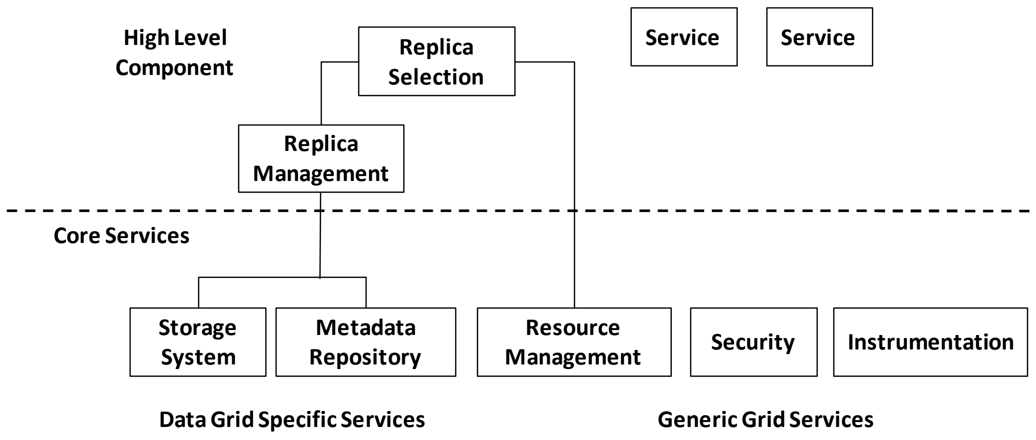


Figure 2.2: Globus Data Grid Architecture

Typical usage scenarios of Globus and hence the proposed replication algorithm is in High Energy Physics (HEP) applications. High Energy Physics (HEP) data management requires very large amounts of both processing power and data storage.

The four experiments of the Large Hadron Collider (LHC) will accumulate of the order of 5-8 petabytes of raw data per year. In addition, during the preparation phase prior to the start of LHC data taking, a similar order of magnitude of simulated data will be required to design and optimize the detectors. Each LHC experiment will form a single Virtual Organization (VO), comprising of the order of 2000 scientists from over 50 countries. HEP community seeks to take advantage of the distributed nature of computing grids to provide physicists with the best possible access to both simulated and real LHC data, from their home institutes. Data replication and management is hence considered to be one of the most important aspects of HEP computing grids. The task of replicating LHC data to the various collaborating institutes within a VO will be handled by Data Management services of Globus, such

2.5 Replication in Data Grids

One of the principle goals of data grids is to improve transparent access to globally distributed data, making data access and location as easy as if it is occurring on a local computer (Guy et al., 2002). Optimization of data access can be achieved via data replication (Carman, Zini, Serafini, & Stockinger, 2002; Dutka, Slota, Nikolow,

& Kitowski, 2004), whereby identical copies or replicas of data are generated and stored at distributed sites. Data replication increases the data availability and reliability for the users and decreases the job execution time, but on the other side the replication increases the storage cost, and affects the network bandwidth consumption either positively or negatively. The replication strategies influence the network bandwidth positively when the number of replicas are balanced and distributed across grid sites efficiently. However, the replication strategies affect the network bandwidth negatively when the numbers of replicas are not proportional to the appropriate replica demand.
as RBR.

2.5.1 Resource Identification

In order to perform a replication, a suitable resource must be identified. In general there are two types of triggers that can be considered:

Trigger on file request

When the Storage Element of a site is requested for a file which it does not store, this could trigger a replication strategy. This kind of strategy is also called an unconditional strategy where replication is performed for every request. The most well-known replacement policies used commonly in operating systems are: Least Recently Used (LRU) and Least Frequently Used (LFU) (Silberschatz, Galvin, & Gagne, 2006), which are used in page replacement to free the storage space for more important data. LRU and LFU are examples for this kind of replication strategy that is deployed in data grids (Ranganathan & Foster, 2001b). In the LRU strategy, the requested site caches the required replica, and if the local storage is full or the current free space is insufficient for the required replica, the least important (victim) replica should be determined and deleted in order to free storage. The victim replica in LRU is the replica that has the maximum period of time between the current time and the last time the replica was requested. However in LFU, the victim replica is the replica that has the least number of requests, or also known as the least popular replica.

Trigger on popularity conditions

Another possible trigger could be a file on some other Storage Element of the site reaching a certain level of popularity. This would require monitoring of all file popularities, perhaps in a central database or by a publish/subscribe method (one Storage Element could subscribe to another one to receive regular updates of its top ten most popular files, for example), and this kind of strategy is also called a conditional strategy.

The process of determining the popularity of a file (identifying which file is to be replicated), may vary from one mechanism to another. The most common characteristic that is widely used to define popularity is the Number of Access (NoA) to the file (Ruay-Shiung et al., 2008; Tang, Lee, Tang, et al., 2005; Tang et al., 2006). NoA stands for the access rate of the file within a certain time interval. However, determining the certain time interval differs from one mechanism to another. File access pattern analysis has always been employed as a powerful tool to design efficient replication decision (Ko, Morales, & Gupta, 2007; Meyer, Annis, Wilde, Mattoso, & Foster, 2006).

For example, in (Ranganathan & Foster, 2001a) the authors consider NoA only in the current time interval. The performance of five distinct strategies had been evaluated using simulation framework; 1) Best Client: replica is created for the client who accesses the file the most; 2) Cascading: a replica is created on the path to the best client; 3) Plain Caching: a local copy is stored upon initial request; 4) Caching + Cascading: combines plain caching and cascading; and 5) Fast Spread: file copies are stored at each node on the path to the best client. The evaluation was done using three different kinds of access patterns. Similar to the work undertaken in this study, the research does not include consistency issues and the data used in the work was read-only data. The three different access patterns are:

i. Random access pattern, which has no locality in patterns;

ii. Data contain a small amount of temporal locality—temporal locality means that the potential access to the popular file in the past is more than others—where some accessed files are likely to be accessed again; and

iii. Data contain small amount of geographical and temporal locality—the files recently accessed by client are likely to be accessed by nearby clients.

On the other hand, there are work (Rasool et al., 2008; Shorfuzzaman et al., 2008; Tang et al., 2006; Tang, Lee, Yeo, et al., 2005; Wang et al., 2007; Yang et al., 2007) that consider NOA in the present and past time intervals, which means that the popularity of the file is determined by analyzing the access history of different time intervals. It has been acknowledged the fact that files that are requested in the present are apt to be requested in the near future. Therefore, popularity of a file depends on the number of access made to the file by the users. And yet, different calculations are used to determine the popularity of the file. In (Tang, Lee, Yeo, et al., 2005), two replication mechanisms were proposed in the multi-tier architecture for data grids, including Simple Bottom-Up (SBU) and Aggregate Bottom-Up (ABU). The SBU algorithm replicates any data file that exceeds a pre-defined threshold. The main shortcoming of SBU is the lack of consideration to the relationship with historical access records. For the sake of addressing the problem, ABU is designed to aggregate historical records to the upper tier until it reaches the root.

The authors in (Tang et al., 2006) determined the popularity of the file by analyzing data access history, the average number of access, and computed NoA. Files with NoA values that are greater than the computer average NoA will be replicated. Hence, the order of which files to be replicated depends on the NoA. The larger the NoA, the more popular the file is, and it will be given a higher priority during the replication process. In (Rasool et al., 2008; Rasool, Li, & Zhang, 2009), the average access frequency ($\mathrm{freq}_{\mathrm{avg}}$) is calculated as a ratio of the sum of all access frequencies to the total number of files, then the files which have access frequency greater than or equal to $\mathrm{freq}_{\mathrm{avg}}$ are marked for replication.

Nevertheless, these replication strategies do not consider the time period of when the files were accessed. If a file was accessed for a number of times in the past, while none was made recently, the file would still be considered popular and hence it will be replicated. Some economical model-based replica schemes have been proposed. The authors in (Bell, Cameron, Capozza, et al., 2003; Cameron et al., March 2004) use an auction protocol to make the replication decision where the files are evaluated using two prediction functions, namely a binomial-based function and a Zipf-based function. In (Ben Charrada, Ounelli, & Chettaoui), the evaluation of the files is performed based on number of requests and the existing number of replicas. In (Ben Charrada, Ounelli, & Chettaoui, 2010), the authors suggested that the file must be replicated if it has been requested too many times and there are not enough copies. In other words, the file will be replicated if its average weight exceeds the average weight of the entire grid. Average weight of a file is calculated by dividing number of requests of the underlying file by the number of existing copies, while the average weight of the entire grid is calculated by dividing the total number of requests of the files by the number of existing copies.

Meanwhile, an optimal replication strategy (DORS) has been proposed by (Wuqing, Xianbin, Zhuowei, Yuping, & Shuibing, 2010), where the authors empirically inferred a threshold to decide whether to replicate the file or not. The threshold is represented by the storage system's relative capacity, which is defined as the ratio of the storage size to the total data set sizes (R). When the number of the file's replicas is greater than R, the file will not be replicated, but when the number of the file's replicas is less than R, the file will be replicated.

The work represented in (Zhong, Zhang, & Zhang, 2010) proposed a replication strategy where replicas are automatically increased according to file access. Once the number of accesses of a certain replica is higher than a threshold, it is labeled as "hot data" and replicated.

The algorithms proposed in (Ruay-Shiung et al., 2008) and (Sashi & Thanamani, 2010) are called Last Access Largest Weight (LALW) and Dynamic Replica Creation and Placement (DRCP) respectively, and both of which tried to solve this problem. The key point of these two algorithms is to give different weights to files having different ages.

The LALW and DRCP algorithms are similar to other algorithms (Rasool et al., 2008; Tang et al., 2006; Tang, Lee, Yeo, et al., 2005) by means of using information on access history to determine the popularity of a file. However, an innovation is included by adding a tag to each access history record of a file. The weight of the record decays to half of its previous weight after a constant time interval. Older access history records have smaller weights; it means that a more recent historical record is more important. An Access Frequency is calculated to represent the importance of access histories in different time intervals.

However, the above approaches (i.e. LALW and DRCP) assume that the decay rate is constant and equals ½, and this means all files decay at the same rate, regardless of the access rate of each one. As a result, the decay rate of weight will be slower. Subsequently the storage element will take time to delete the unwanted files (i.e. the less popular files).

The popularity of the file or the file value is used in two directions: the first direction is to trigger replica creation/deletion strategy as mentioned before. The second direction is to trigger replica replacement strategy, as the less valuable file is replaced by the most valuable file. The difference between replica deletion and replica replacement is that replica deletion is invoked before the replica replacement strategy where the files that have the minimum values are deleted. Meanwhile, the replica replacement strategy is invoked when there is no space for newly created replica in the underlying storage element, and given such a situation, the replica of low value would be replaced by the replica of higher value. The most well-known replacement policies used commonly in operating systems are: Least Recently Used (LRU) and Least Frequently Used (LFU) (Silberschatz et al., 2006), which are used in page replacement to free the storage space for more important data. LRU and LFU are examples for this kind of replication strategy that is deployed in data grids (Ranganathan & Foster, 2001b). In (Teng & Junzhou, 2005; Tian & Luo, 2007, 2010), the authors proposed a prediction-based replica replacement algorithm using a two-stage process to evaluate the popularity of a replica. They considered some features such as bandwidth and replica size. The simulation results demonstrated that their algorithm contributed to better grid performance. The work in (Zhao, Xu, Xiong, & Wang, 2009) suggested a replica replacement algorithm based on economic model and opportunity cost, the files have been evaluated using zipf-like distribution prediction model and then weighted using the file transfer cost model. If the needed replica has a higher weight than the replica with the lowest weight in local storage, that file will be deleted and the new replica will be transferred into the local site. In (Wuqing et al., 2010), the authors proposed a replacement policy that determines the victim file using two kinds of evaluations.

Firstly by evaluating the replica's access frequency using the half-life principle that is used in (Ruay-Shiung et al., 2008; Sashi & Thanamani, 2010), and secondly by evaluating the replica's access cost that is affected by replica size and network bandwidth. Both evaluations are combined together, and the replica with minimum value will be replaced by the newly created replica.

Data replication has two direct improvements on the performance of the data grid. One is to speed up data access, which leads to a shorter execution time of grid jobs; and the other one is to save bandwidth between sites, which can avoid network congestion with the sudden frequently required data. However, replication is also bounded by two factors: the size of storage available at different sites within the data grid and the bandwidth between these sites (Venugopal, Buyya, & Ramamohanarao, 2006). Furthermore, the files in a data grid are mostly large (Rahman, Barker, & Alhajj, 2008, 2009); so, replication to every site and hosting unlimited number of replicas would be unfeasible. Therefore deciding the optimal number of is needed.
The common cost functions that are used in the literature (Al Mistarihi & Yong, 2008; Garmehi & Mansouri, 2007; Kalpakis, Dasgupta, & Wolfson, 2001; Mansouri et al., 2008; Pangfeng & Jan-Jan, 2006; Ranganathan et al., 2002; Yi-Fang et al., 2006) are listed below:

Communication Cost (Read Cost): a lot of research studies considered allocating replicas to sites that minimize the read cost (Garmehi & Mansouri, 2007; Kalpakis et al., 2001; Mansouri et al., 2008). Read cost is usually defined as the cost of transferring a file over the data grid system to the end user.

Storage Cost (Replication Cost): the cost of storing a file at a certain site (Al Mistarihi & Yong, 2008; Mansouri et al., 2008; Pangfeng & Jan-Jan, 2006; Ranganathan et al., 2002; Yi-Fang et al., 2006). The storage cost might reflect the size of the file, the throughput of the site, or the fact that a copy of the file is residing at a specific site, which is also called replication cost.

Access Cost: the time taken to access the data files in replica sites (Caitriana M. Nicholson, 2006).

2.5.2 Number of Replicas

The denser the distribution of replicas is, the shorter the distance a client needs to travel in obtaining a copy of the replica (Pangfeng & Jan-Jan, 2006). In other words, increasing number of replicas would lead to higher data availability. However, given the size of resources included within a data grid, the cost of maintaining multiple copies of resources (i.e. data files) and storing them in the data grid system would be expensive; therefore, the number of replicas should be bounded. A mechanism for creating replicas that allows the achievement of availability and performance goals without consuming undue amounts of storage and bandwidth is thus needed.

The work in (Ranganathan et al., 2002) suggested a algorithm that helps to determine number of replicas needed to maintain the desired availability in P2P communities so that each site within the data grid is authorized to create replicas for the files. The availability of a file depends on the failure rate of peers in the network. A function has been developed to calculate the number of replicas needed for a certain availability threshold. However this algorithm has disadvantages: firstly, the exact number of replicas is not determined; rather it depends on the location service accuracy which depends on the existing number of replicas. The accuracy of the replica location service determines the percentage of accessible files, and thus if the location service is ineffective, more replicas are created to ensure data availability. Secondly, the replica deletion mechanism is not considered, thus the storage cost may be increased.

Meanwhile in (Pangfeng & Jan-Jan, 2006; Yi-Fang et al., 2006), the authors had not taken into account the issue of availability to determine the number of replicas. The problem of determining number of replica has been formulated as follows: given the amount of workload a replica server can handle (D), find the minimum number of replica so that the maximum workload is not more than (D).

Furthermore, (Mansouri et al., 2008) proposed an algorithm formulated by using a dynamic programming-based algorithm. The purpose of their proposed algorithm is to find the optimal number of data file replica over data grid systems, so that the read cost (transferring file over the data grid system to the end-user) and the cost of storage (site building cost) can be minimized. The drawback of those approaches (Mansouri et al., 2008; Pangfeng & Jan-Jan, 2006; Yi-Fang et al., 2006) is that storage capacity has been neglected. As a result, if the site has insufficient space, it will not be chosen to host the replica even if it offers low overall cost.

Another variable was investigated by (Ruay-Shiung et al., 2008) who identified the number of replicas that need to be created, based on the access frequencies of each file that has been requested. By calculating the quotient of average access frequency of popular file divided by average access frequency of all files, the number of replica can be determined.

Meanwhile (Al Mistarihi & Yong, 2008) proposed a replication strategy that makes replication decisions whether to increase the number of replicas to face the high volume of requests, or to reduce the number of replicas to save more storage space. Evidently, increasing the number of replicas will decrease the response time, but the storage cost will be increased accordingly (Al Mistarihi & Yong, 2008).

Table 2.1: Summary of Work in Replica Quantity

Authors	Technique	Variables	Methodology
(Ranganathan et al., 2002)	Dynamic Placement Algorithm	▪ availability	Determine how many replicas are needed to maintain the desired availability
(Pangfeng & Jan-Jan, 2006; Yi-Fang et al., 2006)	Optimal Number of Replica	▪ workload (sum of data requests	Given the total amount of workload a server can handle, then decide the minimum number of replica
(Mansouri et al., 2008)	Optimal Number of	▪ read cost	Find number of replica so that the overall cost is

	Replica (ONR)	▪ storage cost	minimized
(Ruay-Shiung et al., 2008)	Latest Access Largest Weight (LALW)	▪ access frequency	By calculating the quotient of average access frequency of popular file divided by average access frequency of all files, the number of replica is determined
(Al Mistarihi & Yong, 2008)	Replica Management in Grid (RmGrid)	▪ replica request ▪ storage cost	By increasing the number of replicas of the most valuable files and decreasing the number of replicas of the less valuable files

2.5.3 Replica Placement

Replica placement is the process of identifying where to place copies of replicated data files within a data grid system. Transferring a data file from a site to a client consumes an amount of bandwidth. One challenge that is raised from this is to locate candidate sites where the replica could be hosted (Rahman et al., 2008) so as to minimize the amount of bandwidth used.

In (Ranganathan & Foster, 2001a), Rangthan and Foster introduced six replication strategies. They compared those six strategies by measuring average response time and the total bandwidth consumed for each strategy. The lower the response time and the lower the bandwidth consumption, the better the replication strategy is. However, there is a trade-off between response time and bandwidth consumption. The authors concluded that if users are focused on lower response time, then the Cascading strategy would be the best option. On the other hand, if users prefer the consumption of bandwidth to be the most important issue, then Fast Spread is the better choice of all the six strategies. Nevertheless, these two strategies also do not consider storage cost. If a particular file is no longer popular, it will still be stored by the storage element. That will therefore be a waste of free storage. In the Fast Spread replication strategy, the replica is copied to every node it visits when it is brought backward to the requesting node. In contrast to Fast Spread, Modified Fast Spread (MFS) (Bsoul, Al-Khasawneh, Kilani, & Obeidat, 2010) does not necessarily copy the replica to every node it visits when it is brought backward. It is copied to the visited node in two cases. The first case is if the visited node has sufficient free storage space to store the requested replica. The second case is if the node's free storage space is less than the size of the requested replica, and this replica was found more important than a group of existing replicas that their sizes are greater than or equal to the size still needed to make the node's storage able to store it.

In a different approach, the authors of (Yang et al., 2007) proposed a dynamic maintenance strategy called Dynamic Maintenance Service (DMS) to improve the performance of the grid environment. DMS decides where to place the replicas based on two main parameters: request frequency and free storage space. However, the replica deletion mechanism is not considered; rather the system does not locate the replica at a site unless there is enough space even if it brings benefit to system performance.

Meanwhile, (Wang et al., 2007) proposed a replica placement scheme that tries to overcome the bottleneck caused by increasing the downlinks, which are occurring at the same time. The proposed strategy chooses the best site to host the replica according to the evaluation result based on the number of user request and transmission cost. The purpose of the strategy is to replicate the file to a site that provides minimum average transmission cost. Transmission cost is defined to be inversely proportional to bandwidth, and the site that provides the minimum average transmission cost is selected.

Following the bandwidth aspect, (Park, Kim, Ko, & Yoon, 2004) proposed a replication strategy, called Bandwidth Hierarchy based Replication (BHR) to reduce access time by avoiding network congestion. BHR reduces the time taken to access and transfer the file. It places a replica at a high bandwidth location. However, such an approach only considers transmission cost and does not guarantee to minimize the overall cost.

A load balancing replication strategy has been proposed by (Rasool et al., 2008), where the most frequently accessed file is placed closed to the users and the decision of replica placement is made based on the access load and the storage load of the candidate replica servers and their sibling nodes. In relation to this, (Rahman, Barker, & Alhajj, 2005) discussed various replication strategies namely; MinimizeExpectedUtil, MaximizeTimeDiffUtil, MinimizeMaxRisk, and MinimizeMaxAvgRisk while considering the utility and risk indexes, and making the replica placement decision by optimizing the average response time. They concluded that considering both current network state and file requests are better than considering the file requests alone.

Meanwhile, (Rahman et al., 2008) proposed a static replica placement algorithm to place replica files in best p candidate nodes that minimizes the total response of each site by using Lagrangean Relaxation, which is a heuristic approach (Fisher, 2004) to measure the response time of each client node to its nearest server node. The algorithm is most likely the p-median problem. They also used the user requests and network latency as parameters to decide when to maintain replica dynamically.

Work by (Garmehi & Mansouri, 2007) suggested an algorithm that is formulated by using a dynamic programming method to find optimal placement k replicas of an object, such that the overall cost (i.e. storage cost plus read cost) is minimized. Read cost is defined as the data transfer cost and storage cost is the cost of placing replicas at the sites. However, the algorithm does not guarantee the availability of the file, as the priority choice of location is given to those who provide cheaper services regardless of its availability.

The authors in (Lin, Wu, & Liu, 2008; Yi-Fang et al., 2006) proposed a placement algorithm so that the workload of user requests among the replicas is balanced. The workload is defined as number of requests that a server satisfies. Given the data usage and maximum workload allowed for each replica server, they suggested algorithm can efficiently determine the minimum number of replicas required. On the other hand, the authors in (Ranganathan et al., 2002) suggested a algorithm that provides a function that evaluates the placement of replica. The objective of this function is to maximize the difference between the replication benefits and replication cost (storage cost and transfer time). The benefit is the reduction in transfer time to the potential users, the storage cost is the storage cost at the remote site, and the transfer time is the duration from the current location to the new location. Yet again, the replica deletion mechanism was still not considered, thus the storage space cost may be increased.

Then (Abawajy, 2004) proposed an improvement, namely in the form of the Proportional Share Replication policy. The method places replicas on the optimal locations when the number of sites and total replicas to be distributed is known.
Meanwhile, the work on replication algorithm by (Shorfuzzaman et al., 2008) had resulted in a Popularity Based Replica Placement (PBRP) algorithm for hierarchical data grids. The idea behind PBRP is to place replicas as close as possible to those clients that frequently request data files. Further work by (Rasool et al., 2009) presented a replica placement strategy in multi-tier data grid that categorized the files based on their access frequency into two groups: 1) Most Frequent Files (MFF) that are replicated and placed at the parent node of their respective best clients, where the best client for a file is a client which generates the maximum request for that file, and 2) Least Frequent Files (LFF) that are placed at one tier below the root of the data grid along the path of their best client. In (Ben Charrada et al., 2010), a dynamic placement algorithm was proposed that takes into account the dynamicity of sites in the data grid, since a site can at any time leave the grid and possibly join again later. Thus, two parameters were investigated: the request number for each file by each

site, and utility of each site that involves the number of times the site did not answer to a file request due to its absence from the grid.

Then, (Ruay-Shiung et al., 2008) proposed a replication mechanism that replicates the popular file to a suitable site according to the access frequencies for each file that has been requested. Access frequency is an essential parameter that should be taken into account when determining replica placement. However, some important parameters such as overall cost (i.e. storage cost and read cost), distance and availability should not be neglected; otherwise the overall system performance is degraded.

The work presented in (Naseera & Murthy, 2009) suggested a replica placement mechanism that deploys an agent at every site that holds the master copy of files for which replicas are to be created. The main function of each agent is to select the candidate site for placing the replica based on response time and job execution time. The replica is placed at the site that minimizes the time taken for obtaining all the files required by the job. However, storage capacity is ignored as a result if the site is full and provides the minimum response time; it will not be selected to host the replica. On the other hand, a priori replica placement was proposed in (Challal & Bouabana-Tebibel, 2010), where the replicas are created and placed before starting jobs and launching any work on the grid. The replication is performed at once after the original copies are created and before any file request has been made. The main objectives are to maximize the distance between identical replicas and to minimize the distance between different replicas, so that each site can find the different replicas faster within its vicinity. However, this approach does not cope with the dynamicity of grid environment, and moreover, the storage capacity is not taken into consideration.

Table 2.2: Summary of Work in Replica Placement

Authors	**Technique**	**Variable**	**Methodology**
(Wang et al., 2007)	Fair Replica Placement	▪ Bandwidth ▪ Number of user request	Duplicate the file to a node that provides minimum average transmission cost (maximum bandwidth)
(Ranganathan et al., 2002)	Dynamic Placement Algorithm	▪ Storage Cost ▪ Transfer Time	Maximize the deference between replication benefit and replication cost
(Garmehi & Mansouri, 2007)	Optimal Placement of Replicas	▪ Communication/read Cost ▪ Storage Cost	Place the replica so that the overall cost (read and storage cost) is minimized
(Yang et al., 2007)	Dynamic Maintenance Services (DMS)	▪ Request Frequency ▪ Storage Capacity	If the request frequency of file is more than the maximum request rate, and there is free space in the site then DMS will duplicate file to that location
Ruay-Shing et al. (Ruay-Shiung et al., 2008)	Latest Access Largest Weight (LALW)	▪ Access Frequency	According to the access frequencies for each file that has been requested, a popular file is replicated to a suitable site

Through the literature review on this subject, it was concluded that there are many drawbacks of the current replica creation strategies, and there is a need for enhancing these strategies even further. Obviously, in order to get benefits from replication strategies, the storage cost and the read cost must be minimized. From the literature, it was observed that there is a lack of suitable current replication algorithms for the management of data grid resource usage (i.e. network and storage resources), and thus more enhancement is needed.

2.6 Summary of Chapter

In this chapter, the background on the issues that are covered in this study has been provided. This chapter was divided into two main parts; first part presented a brief description and characteristics of data grids and some challenges of the applications running in such environment. The second part presented the need for a replication algorithm in a data grid environment. Related works of data replication was analytically investigated and presented. This included the related research and progress in replication algorithms, and recent works in this research domain. The second aspect is concerned with the individual functions of the replication algorithm, where some parameters have been neglected by other replication algorithms, which should be considered. For example, in evaluating the files to determine which file is to be replicated and deleted, the implementer needs to consider the dependency relationships between files, period of time it has been in the system and the decay or growth rate of the file request.

In the response to the literature survey presented in this study, it is proposed that there is a need for an enhanced replication algorithm that embodies all the core functions listed above, and moreover, it should include the neglected parameters by other works as discussed in this chapter. In the next chapter, the methodology and simulation setting employed in this research work will be detailed.

CHAPTER THREE
RESEARCH METHODOLOGY

This chapter presents the undertaken steps for this research. This research starts by formulating a Resource Selection function that determines the importance of a file to the users and data grid system. This is followed by a second stage that focuses on formulating a replica number function that utilizes the developed resource selection function. The third stage of the research is to integrate the two functions in a replication algorithm (proposed as the Relationship-based Replication algorithm) that is later evaluated in a simulation environment. Figure 3.1 illustrates the undertaken steps for research.

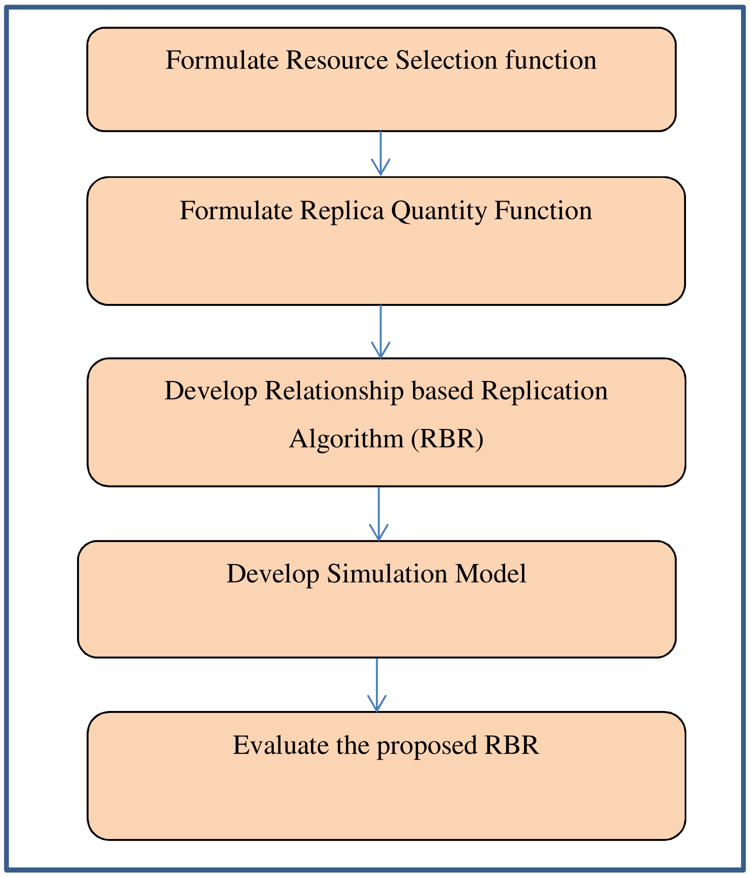

Formulate Resource Selection function
Formulate Replica Quantity Function
Develop Relationship based Replication Algorithm (RBR)
Develop Simulation Model
Evaluate the proposed RBR

Figure 3.1: Research Steps

3.1 Formulate Resource Selection Function

In order to formulate the resource selection function, information on number of access of a resource (e.g data file) is combined with information on resource dependencies and age. Hence, the function is designed by utilizing three types of relationships:

1) File-to-users (F2U) (Madi, 2012) - Relationship that describes the behavior of a file being requested by users, and notes the change to this request(whether is a growth or decay change). The relationship is represented using the exponential model. If the change is seen to be in the form of growth, then the particular resource is assumed to be important, and vice versa.
2) File-to-file relationship (F2F) (Madi, 2012) - Relationship that describes the behavior of a file requesting other files, and notes the level of dependency of the file. Resources that are highly dependent on is likely to be more important than the others.
3) File-to-grid (F2G) - Relationship that describes lifetime of a file in the grid system.

Details of the resource selection formulation along an example of utilization of the function are presented in Chapter 4.

3.2 Formulate Replica Quantity Function

The second step of the research is to formulate the a function that determines the appropriate number of copies for the identified resources. In designing this function, we adapt the work presented in 2012 (Madi, 2012) that integrates the importance of the file to the users and the grid system. The produced function (described in detail in section 4.2) presents users with the estimated number of required replicas.

3.3 Develop Relation-based Replication (RBR) algorithm

The third step of this research was to formulate an algorithm that integrates the proposed functions and this is illustrated in the upcoming chapter, specifically in Figure 4.4.

3.4 Develop Simulation Model

In this research, the OptorSim (Bell, Cameron, Millar, et al., 2003; Cameron et al., March 2004; Cameron et al., 2004) simulator was utilized to simulate the proposed replication algorithm. The main idea of OptorSim is when given a grid topology, resources, and a set of jobs and optimization strategy, it can simulate data movement around these job runs and supply information on various factors that could be used to evaluate the performance of the optimization strategy. The key advantage of OptorSim is that it is much closer to reality since it is based on the EU DataGrid architecture (Cameron et al., March 2004), which is widely used by grid computing communities (Bell, Cameron, Millar, et al., 2003; David G. Cameron, 2005; The European Data Grid Project). Furthermore, OptorSim is capable of providing a testbed similar to the original data grid environment by providing multiple grid sites with storage elements that can be used to create and store replicas. Users can also set the parameters of OptorSim according to their requirements to run the simulation. A more detailed architecture and implementation of OptorSim can be found in (Al-Mistarihi & Yong, 2008; David, 2003; Hong, Xue-dong, Xia, Zhen, & Wen-xing, 2008; Lei, Vrbsky, & Hong, 2007; Ruay-Shiung et al., 2008; Shorfuzzaman et al., 2008).

The RBR algorithm works in the background of the data grid system in such a way that there is no direct connection with users. RBR relies on other existing data grid core services, such as Replica Location Services (RLS) that provides information related to the physical file locations, and Information Service Provider (ISP) (Vazhkudai et al., 2001) such as Network Weather Services (Wolski, 1997) to provide the network availability and status. As shown in Figure 3.2, RBR offers the following functionalities:

1. gathers replica locations information from RLS;
2. gathers network bandwidth information from the NWS;
3. gathers job information from the history file; and
4. makes decisions on replica creation and replica quantity.

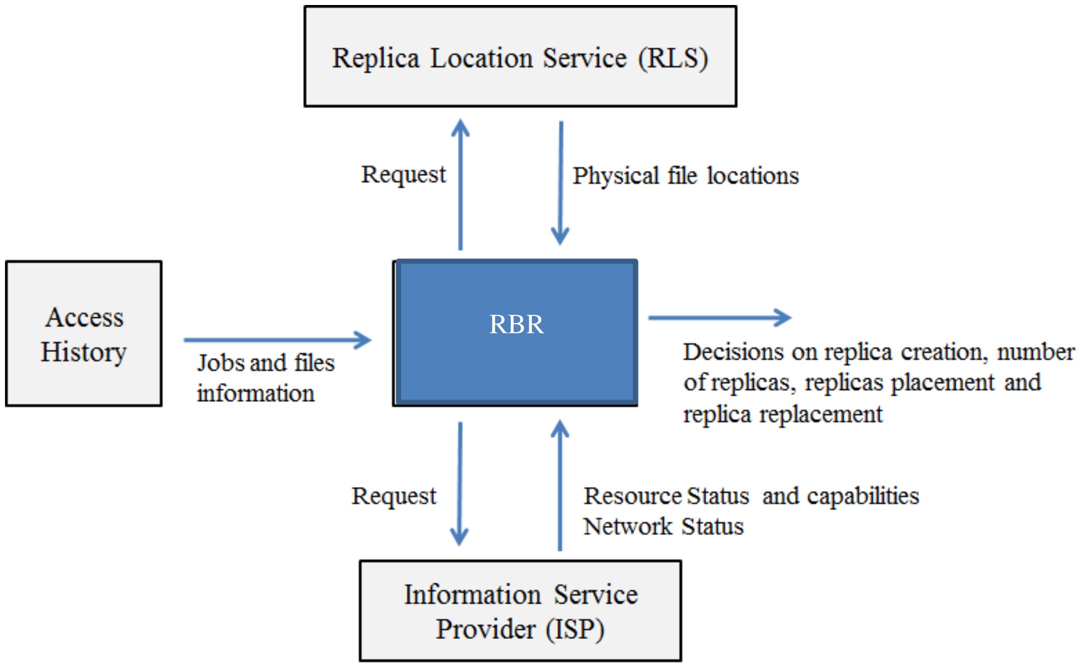

Replica Location Service (RLS)
Request
Physical file locations
Access History
Jobs and files information
RBR
Decisions on replica creation, number of replicas, replicas placement and replica replacement
Request
Resource Status and capabilities
Network Status
Information Service Provider (ISP)

Figure 3.2: The Proposed RBR Component in OptorSim Architecture

In EU DataGrid, a set of high energy physics analysis jobs was generated from the Compact Muon Solenoid (CMS) (Ruay-Shiung et al., 2008) experiments in the European Organization for Nuclear Research (CERN) (CMS Data Challenge 2004; Holtman, 2001) project. Jobs were based on the CDF use-case as described in (European Organization for Nuclear Research (CERN)).

The EU DataGrid topology includes 20 sites in USA and Europe as shown in Figure 3.3. Within this model, each site, excluding CERN and FNAL, was assigned with a Computing and Storage Element. CERN and FNAL were allocated with Storage Elements only, since they produce the original files and store them.

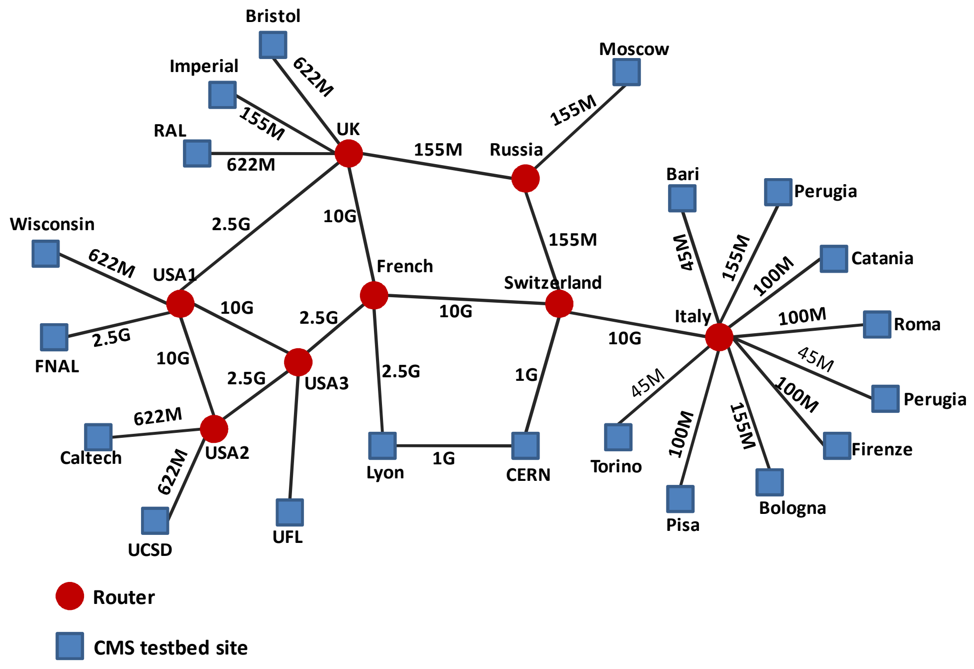


Figure 3.3: The EU Data Grid Testbed Sites and Their Associated Network Geometry

3.5 Evaluate the Proposed Relation-based Replication Algorithm (RBR)

The proposed RBR is compared against existing algorithms that includes the LALW (Ruay-Shiung et al., 2008) and DRCM (Madi, 2012). The evaluation is based on the performance metrics and parameter settings.

3.5.1 Performance Metrics

3.5.1.1 Mean Job Execution Time

This is defined as the average time required to execute a job starting from the time it is scheduled to the Computing Element until it has finished processing all of the required files. It is calculated by accumulating the time taken by each job and divided by the number of jobs (Bell, Cameron, Capozza, et al., 2003; Bell, Cameron, Millar, et al., 2003; Ben Charrada et al., 2010; Cameron et al., 2003; Cameron et al., 2004; Ruay-Shiung et al., 2008), as shown in the following formula:

$$\mathrm{MJET} = \frac{\sum \mathrm{T}_{\mathrm{Departure}} - \mathrm{T}_{\mathrm{Arrive}}}{\mathrm{n}} \tag{3.1}$$

where,

T_{Arrive}: start time of job execution,
$T_{Departure}$: completion time of job execution, and
n: total number of processed jobs in the simulation.

3.5.1.2 Efficient Network Usage (ENU)

ENU is defined as a measure of how well the replication strategy uses the network (Bell, Cameron, Millar, et al., 2003). It is computed as:

$$ENU = \frac{N_{\text{remote file access}} + N_{replications}}{N_{\text{remote file access}} + N_{local\ file\ access}} \tag{3.2}$$

where $N_{\text{remote file access}}$ is the number of accesses that Computing Element reads a file from a remote site, $N_{replications}$ is the total number of file replication that occurs, and $(N_{\text{remote file access}} + N_{local\ file\ access})$ is the number of times that Computing Element reads a file from a remote site or reads a file locally.

A lower value would indicate that the utilization of network bandwidth is more efficient. In order to get a low ENU, the numerator, $N_{replications}$, should be small.

3.5.1.3 Storage Element Usage

The average of all storage reserve capacity in the data grid can reflect the total system storage cost (Bell, Cameron, Millar, et al., 2003; Cameron et al., 2004). The

Average Storage Usage (ASU) metric is computed by the following equation (Bell, Cameron, Millar, et al., 2003):

$$\mathrm{ASU} = \frac{\sum_{i=1}^{n} \frac{U}{C}(\mathrm{site}_i)}{N} \times 100\% \tag{3.3}$$

where,
U: storage usage space that is reserved by the data files,
N: number of sites in the data grid, and
C: total capacity of the storage medium.

3.5.1.4 Computing Element Usage (CE Usage)

This is defined as the percentage of time that a CE is active (transferring or processing data) during the simulation. The CE usage of the whole grid is computed by aggregating the CE usage of each individual CE. 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 grid (Bell, Cameron, Millar, et al., 2003). Low CE usage, on the other hand, would mean that some CEs have long queues while others are underused.

3.5.2 Parameter Settings

In order to expose the throughput and system performance, the simulation was executed on different scenarios that employ different parameter settings. The parameters that may influence replication algorithms includes (Ruay-Shiung et al., 2008): number of submitted jobs, access history length, storage metric (D) and job delay time.

3.5.2.1 Number of submitted jobs (Workload Test)

System scalability can be tested by the number of jobs running during the simulation. In this research, to simulate different number of jobs, the maximum number of submitted jobs was increased by a factor of four and the minimum was decreased by a factor of four, i.e. number of jobs that is considered in our evaluation varied between 200 and 4000 jobs.

3.5.2.2 Access History Length

This is defined as the period of time for which the information on file access is kept. The history of file access is used by replication algorithms to identify the most popular file in the next time window. Therefore the length of access history used in the calculations must be carefully chosen to produce accurate prediction. If the history does not go back in time far enough, the statistics of file access may not be accurate, but if the history goes back too far, it may provide overdue and useless information. Thus the length of access history considered for evaluation varies between 10^3 seconds and 10^6 seconds, where the reasons for which are detailed out in Chapter 5.

3.5.2.3 Storage Metric (D)

It is defined as the ratio of the Storage Element size to the total dataset size (Tang, Lee, Tang, et al., 2005; Tang et al., 2006)

$$D = \frac{Storage\ Element\ Size}{Total\ dataset\ Size} \tag{3.4}$$

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 deletion and the replication strategy will have little effect on the performance of the grid. If $D < 1$, than the storage element is not capable of storing all required files so deletion must take place and choices have to be made on which replicas to keep. In order to study the effect of storage metric, different file sizes that vary between 200 to 2000 MB were considered and used in the experiments.

3.5.2.4 Job Delay

This is defined as the rate at which jobs are submitted to the data grid. The job delay was fixed at 25 seconds in all of the experiments.

3.6 Summary of Chapter

This chapter describes the steps taken in achieving the aim of the research. Five stages undertaken; formulation of resource selection, formulation of replica quantity, development of RBR algorithm, development of OptorSim simulator and the performance evaluation of the proposed RBR. In the upcoming chapter, details of the RBR algorithm and its evaluation are presented.

CHAPTER FOUR
RELATIONSHIP BASED REPLICATION ALGORITHM

In this chapter, the implementation of Relationship-based Replication (RBR) algorithm to improve performance of a grid system is described. To that end, a detail design of RBR that includes the resource selection function and replica quantity function is presented.

4.1 Resource Selection

In a data grid, when a resource (e.g a data file) is required by a job and is not available on a local storage, it may either be replicated or read remotely. If a file has been replicated, in the future, when it is requested, any job can accessed it quickly and the job execution time can be reduced. However, if replicating a resource file requires the deletion of other resources such as data file(s), future jobs that require the deleted resources may consume additional computational time. Therefore, a decision must be made whereby only the most resource files are replicated and the least ones are deleted. The replication decision includes two issues: 1) which file should be created/deleted and 2) how many copies to be created/deleted. The proposed algorithm (RBR) includes the perspectives of two parties: users and system.

Due to the limited storage capacity, replication decision should be made to conform users' needs so that high demanded files (popular replicas) are efficiently maintain and files that are rarely utilized are removed.

4.1.1 File-To-User Relationship (F2U)

Popularity of a file depends on the number of access made to the file by users (Tang, Lee, Yeo, et al., 2005). With this, popular data files can be identified by analyzing the file access history. Many real world phenomena can be modeled by functions that describe how things grow or decay as time passes. Examples of such phenomena include the studies of populations and bacteria (Ranganathan & Foster, 2001a, 2001c; Ranganathan et al., 2002). The work presented by (Madi, 2012) adopts the exponential growth/decay model in determining popularity of a file. This is due to the fact that each file has its own number of access and the value increases by the increase of access rate and vice versa. If the access rate increases, so does the growth/decay rate.

If we use N_f^t to represent the number of accesses for file f at time t, and N_f^{t+1} to represent the number of accesses at time $\mathrm{t}+1$, the exponential growth/decay model would be given by:

$$N_f^{t+1} = N_f^t \times (1 + r) \tag{4.1}$$

where r is the growth or decay rate in number of accesses of a file in one time interval. Therefore, the value of r using the following formula can be calculated:

$$r = \left(N_f^{t+1}/N_f^t\right) - 1 \tag{4.1.1}$$

Assume t is the number of passed intervals, and N_f^t indicates the number of access for the file f at time interval t, then we get the sequence of access numbers:

$N_f^0\ N_f^1\ N_f^2\ N_f^3 \ldots\ldots\ N_f^{t-1}\ N_f^t$

Therefore, there are $t-1$ time intervals, and each time interval has a growth or decay rate in number of accesses of a file. So according to the exponential growth/decay model, the equation can be written as in the following:

$r_0 = \left(N_f^1/N_f^0\right) - 1,$

$r_1 = \left(N_f^2/N_f^1\right) - 1,$

$r_2 = \left(N_f^3/N_f^2\right) - 1,$

$$r_{t-1} = \left(N_f^t/N_f^{t-1}\right) - 1 \qquad (4.1.2)$$

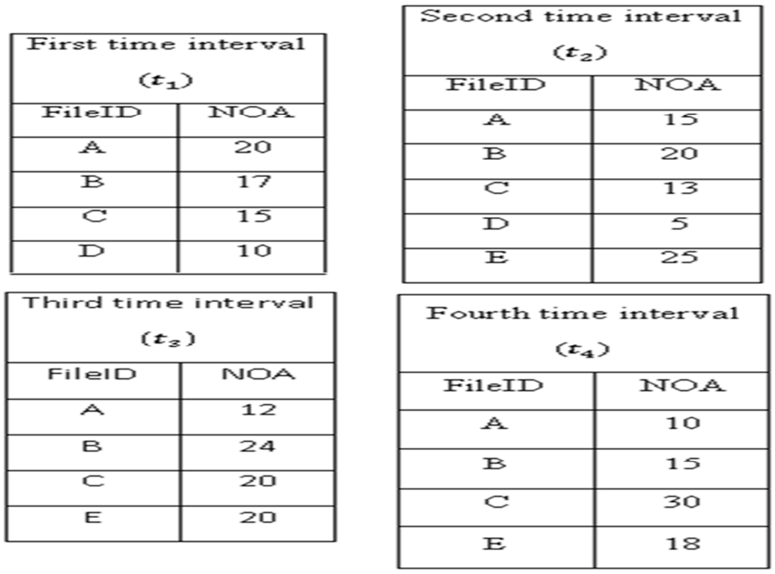
Therefore the average rate for all intervals is:

$$r = \sum_0^{t-1} r_i/t - 1 \qquad (4.1.3)$$

Having known the average accessed rate (growth or decay) for a file during the past intervals, the number of access for the upcoming time interval can be estimated, which is termed as the *File Lifetime* (Madi, 2012) :

$$File\ Lifetime = N_f^t \times (1 + r) \qquad (4.1.4)$$

In order to avoid extreme cases where the growth or decay rate is equal to infinity, it is assumed that all files have been accessed for at least once. Using the data that is provided in Figure 4.2, an example to explain the concept of the strategy is presented in the following paragraph. In the example, there are four time intervals (t1, t2, t3, t4) with different number of accesses (NOA) of five data files.



First time interval (t_1)	
FileID	NOA
A	20
B	17
C	15
D	10

Second time interval (t_2)	
FileID	NOA
A	15
B	20
C	13
D	5
E	25

Third time interval (t_3)	
FileID	NOA
A	12
B	24
C	20
E	20

Fourth time interval (t_4)	
FileID	NOA
A	10
B	15
C	30
E	18

Figure 4.1: File Request in Time Interval

There are five different files (A, B, C, D, and E) accessed during the four time intervals (t_1, t_2, t_3, and t_4) In order to calculate the *File Lifetime* value of each file, the average growth/decay rate of the file during four time intervals is calculated and substituted into equations (4.1.3) and (4.1). Figure 4.3 shows the process of calculating the values for files A, B, and C. In the same way, the value of 1.76 was obtained as the number of access for file D, and 13.1 for file E.

	t_1	t_2	t_3	t_4	t_5
A	20	15	12	10	N_A^5

- The average growth/decay rate of file A is:

$$r = \frac{(-0.25) + (-0.2) + (-0.16)}{3} = -0.21$$

- The estimated number of access of file A is:

$$N_A^5 = 10 * (-0.21 + 1) = 7.9$$

	t_1	t_2	t_3	t_4	t_5
B	17	20	24	15	N_B^5

- The average growth/decay rate of file B is:

$$r = \frac{0.18 + 0.20 + (-0.38)}{3} = 0.001$$

- The estimated number of access of file B is:

$$N_B^5 = 15 * (0.001 + 1) = 15.0$$

	t_1	t_2	t_3	t_4	t_5
C	15	13	20	30	N_C^5

- The average growth/decay rate of file C is:

$$r = \frac{(-0.13) + 0.54 + 0.50}{3} = -0.30$$

- The estimated number of access of file C is:

$$N_C^5 = 30 * (0.30 + 1) = 39.1$$

Figure 4.2: Calculation for *FileLifetime*

4.1.2 File-to-File Relationship (F2F)

As mentioned in Chapter one, the data files that are used in this research are in the form of source code modality. Thus, there is a possibility of having files that require other files in order to be executed or compiled. In other words, there may exist dependency relationship between files (Kapitza, 2003; Kreft, Booth, & Wimpenny, 1998; Kremer, 1993). We utilized such relationship as additional factor that contributes in identifying resources that are to be replicated. Such an approach is seen to contribute in determining importance of a file to the resource management system, and, is represented as *File Weight*. This research employs the calculation of *File Weight* as described in (Madi, 2012) :

$$File\ Weight = \sum_{i=1}^{n} FL_i \times DL_i \tag{4.2}$$

where,
n: total number of files in a grid system,
FL: File Lifetime as $\boldsymbol{File\ Lifetime = N_f^t \times (1 + r)}$ and
DL: dependency level of other files on the underlying file, and if there is no dependency, DL is assumed to be zero. This is counted as number of files that are dependent on the resource file.

In order to understand how to calculate file weight, we make use of the previous example. Suppose that files, as shown in Figure 4.1 have some dependencies among them, as depicted in Figure 4.3. The present dependency relationships in Figure 4.4 would suggest that file B is more important than file A as there are three files (A, C, and E) that depend on file B while none exist for file A. Hence, the File Weight of files A, B, C, D, and E are obtained as follows:

$$\text{File Weight}(t, A) = 0$$

$$\text{File Weight}(t, \text{FileB}) = (8 \text{ x } 0.35) + (39 \text{ x } 0.33) + (8 \text{ x } 0.15) = 16.86$$

$$\text{File Weight}(t, C) = 0$$

$$\text{File Weight}(t, D) = 0$$

$$\text{File Weight}(t, \text{FileB}) = (8 \text{ x } 0.15) = 1.2$$

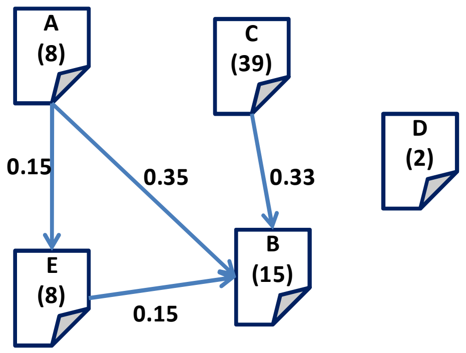


Figure 4.3: Dependencies between Files

4.1.3 File-To-Grid Relationship (F2G)

The third relationship that is incorporated in this research is the File-To-Grid (F2G) relationship. Such relationship refers to the time period of existence for a particular resource, in other words, the age of a data file. Such a relationship is important as it shows the vitality of the file. For example, if there are two files having the same number of access, but of different age, then the older file is considered to be less popular than the younger one. This is because the younger file seems to be more valuable as it receives the same amount of request but in a shorter time period. The age of the file can be calculated as the time file being included in the grid until the current time.

$$File\ Age = Time_{current} - Time_{attach} \tag{4.3}$$

The work presented in this research evaluates a resource file by combining information from users, file management and the grid itself. With this, the F2G relationship along with the F2U and F2F are taken into consideration when determining the importance of a resource. Hence, the File Value is computed as the following equation:

$$File\ Value(t, f) = \frac{FileLifetime(t,f) + FileWeight(t,f)}{File\ age(t,f)} \tag{4.4}$$

$FileLifetime$ (FL) , $FileWeight$ (FW) and $File\ Age$ (FA) are used to compute the $File\ Value$ (FV) that is used as an indicator of the volume of demand for a file in a grid system, and the proposed replication algorithm will decide which file to be replicated. The larger the value of $File\ Value$ (FV), the more important the file is to the grid system.

4.2 Replica Quantity

In determining the number of replication or deletion, we adapt the replica quantity strategy implemented in (Madi, 2012). Nevertheless, we are employing the proposed $File\ Value$ (FV) in the strategy. In this strategy, RBR triggers the resource selection function (equation 4.4) and use it to calculate file power of users' perspective (FP_{users}). This is computed as follows:

$$FP_{users} = \frac{FV}{\sum_{\forall files} FV} \tag{4.5}$$

where,
FP: file power from user's perspective, and
FV: File Value.

Additionally, there is also information from the view point of the system that is represented by the availability of a resource in the system (Madi, 2012). This depends on the current number of replicas of the underlying file and is computed as:

$$FP_{system} = \frac{NoC}{\sum_{\forall files} NoC} \tag{4.6}$$

where,
FP: file power from system's perspective, and
NoC: number of copies of the underlying file.

Later, a balance between users' perspective and system's perspective (Madi, 2012) is determined and the utilized function is as follows:

$$\boldsymbol{FP_{users} = TH * FP_{system}} \tag{4.7}$$

where, TH is the threshold value that determines how many percent the number of copies that are supposed to exist to meet the users request of the underlying file. The threshold value is specified in the form of percentage, which varies according to the grid situation, such as the current bandwidth, the type of the running applications and jobs, and the workload of the system (number of jobs and number of files).

With this, the replica quantity function for this research will be as follows:

$$\mathbf{ENoR} = \frac{(\mathbf{FP_{users}} - (\mathbf{TH} \times \mathbf{FP_{system}})) \times \sum_{\forall files} \boldsymbol{NoC}}{\mathbf{TH}} \tag{4.9}$$

where, ENoR: the estimated number of replicas.

There are three cases that may occur:

Case 1: if the $ENoR > 0$, then the system will replicate ENoR replicas of the underlying file,

Case2: if the $ENoR < 0$, then the system will delete ENoR of existing replicas, and

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

In order to illustrate how the strategy works, consider the following example:

Assume a grid system has 15 files and their corresponding values and number of existing copies exists as shown in Table 4.2. Assume that the threshold value (TH) used is 50%, that means the $\mathrm{FP_{users}}$ should double the value of $\mathrm{FP_{system}}$.

Table 4.1: Examples of Files and its Related Information

File name	File value	Number of copies	File name	File value	Number of copies
File1	26	1	File9	25	1
File2	30	2	File10	22	4
File3	32	1	File11	13	1
File4	31	3	File12	9	2
File5	28	4	File13	11	3
File6	20	2	File14	8	1
File7	10	3	File15	17	1
File8	15	5	**Total**	**297**	**34**

The main concern here is to determine which file needs to be replicated and which file needs to be deleted. The first step is to calculate the power of each file in terms of users' perspective, and system perspective according to formulas (4.5), (4.6), and (4.9). For example, the power of File1 from users' perspective and system's perspective, and ENoR for File1 are computed as follows:

$$FP(File1)_{users} = \frac{26}{297} = 0.088$$

$$FP(File1)_{system} = \frac{1}{34} = 0.029$$

$$\text{ENoR} = \frac{(0.088 - 2 \times 0.029) \times 34}{2} = 0.488 \cong 0.5 \cong 1$$

Due to the fact that number of replica values must be in the form of integer number, so the ENoR value is rounded up to the nearest integer. Therefore, the estimated number of replicas is 1, which means File1 needs to be replicated once. In the same way, all FP values and ENoR for each file are computed as shown in Table 4.3.

Table 4.2: Examples of ENoR

File name	Users power	System power	ENoR
File1	0.088	0.029	0.5
File2	0.101	0.059	-0.3
File3	0.108	0.029	0.8
File4	0.104	0.088	-1.2
File5	0.094	0.118	-2.4
File6	0.067	0.059	-0.9
File7	0.034	0.088	-2.4
File8	0.051	0.147	-4.1

File9	0.084	0.029	0.4
File10	0.074	0.118	-2.7
File11	0.044	0.029	-0.3
File12	0.030	0.059	-1.5
File13	0.037	0.088	-2.4
File14	0.027	0.029	-0.5
File15	0.057	0.029	0.0

The results from Table 4.2 show that File1 needs to be replicated by one copy as ENoR approximately equals to one, while three copies of File10 need to be deleted where its ENoR values approximately equal to three. Meanwhile, the ENoR for File2 and File9 approximately equal to 0, and therefore no action will occur as they are considered to be stable files. The rest of the files are in the same manner. To this end, there will be three lists of files, where the first list contains files that need to be replicated, the second list contains files that need to be deleted, and the third list contains files that require no further action.

4.3 RBR Algorithm

This section presents the algorithm of the proposed replication strategy. It integrates both of the proposed resource selection and replica quantity functions. The algorithm is as the one illustrated in Figure 4.4.

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: Number of Replicas for the Identified Resource

Procedures:

```
/* Resource Identification */
1: for each files in the data grid
2: Calculate $r \leftarrow \sum_0^{t-1} r_i / t - 1$
3: Calculate $FileLifetime(t, f)\ using\ N_f^t \times (1 + r)$
4: Calculate $FileWeight(t, f)$ using $\sum_{i=1}^{n} FL_i \times DL_i$
5: Calculate $FileAge$ using $Time_{current} - Time_{attach}$
6: Calculate $File\ Value(t, f)$ using $\frac{FileLifetime(t,f)+FileWeight(t,f)}{File\ age(t,f)}$

/*Replica Quantity */
7: Calculate $FP_{users}$ using $\frac{FileValue}{\sum_{\forall files} FileValue}$
8: Calculate $FP_{system}$ using $\frac{NoC}{\sum_{\forall files} NoC}$
9: Calculate ENoR using $\frac{(\mathbf{FP_{users}} - (\mathbf{TH} \times \mathbf{FP_{system}})) \times \sum_{\forall files} \boldsymbol{NoC}}{\mathbf{TH}}$
10:     if ($ENoR$>0) then  Add $file_i$ to Popular_List
11:     else if ($ENoR$<0) then  Add $file_i$ to Unwanted_List
12:     else if ($ENoR$=0) then  Add $file_i$ to Stable_List
```

Figure 4.4: The Relationship-based Replication Algorithm

4.4 RBR Implementation

As stated previously in Chapter three, the RBR is realized via the OptorSim simulator. The following sub sections discuss the integration of RBR into OptorSim.

4.4.1 Integration of RBR into OptorSim

OptorSim is capable of simulating many areas of the grid and these areas can be divided into packages, where each package contains a collection of related classes. The package diagram shown in Figure 3.4 describes those within OptorSim and their relations. Starting at the lowest level, the *optorsim.time* package deals with how time is measured within the simulation, while *optorsim.infrastructure* simulates the underlying grid infrastructure including the network, grid sites, and basic components of the site: computing Element and Storage Element. The P2P network and messaging system along with the auctioning process is included in the *optorsim.auctions* package. The functionality of replica management components including Replica Location Service is implemented in the *optorsim.reptorsim* package, while the replica optimization strategies are in the *optorsim.optor* package. *Optorsim* is the highest level package that simulates the resource broker and users, and also controls the GUI.

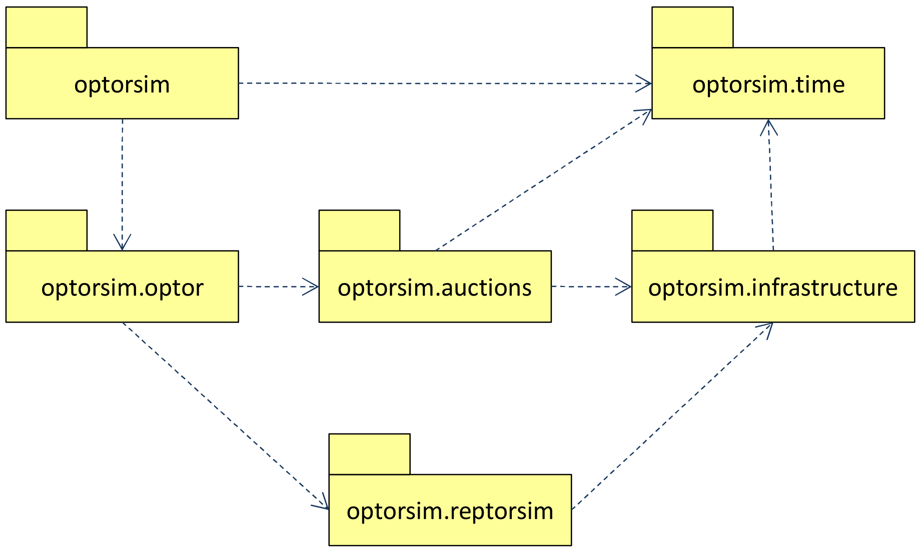


Figure 4.5: UML Package Diagram of OptorSim

There exist three replication algorithms employed in OptorSim, namely, LFU, LRU, and Economic algorithm. In this work, we include three additional algorithms namely, the LALW (Ruay-Shiung et al., 2008), RBR (Madi, 2012) and RBR which is the proposed algorithm. The RBR and LALW along with other replication algorithms that have already been implemented in OptorSim, are written in Java and integrated into the *optorsim.optor* package of the simulator where it is termed as *RBROptimiser, DRCMOptimiser and LALWOptimiser*. These Java classes directly extend the skelOptor class that exists in *optorsim.optor* package as shown in Figure 4.5. The implementing classes and subclasses are shown as a UML class diagram in Figure 4.6. In general, the simulation works as follows: the process starts when users submit a job to the RB, which in turn searches for appropriate CE, and schedules the job to any CE by following one of the scheduling algorithms defined in the parameter file. When the CE is ready to execute a job, it starts to process the files that are needed for the job.

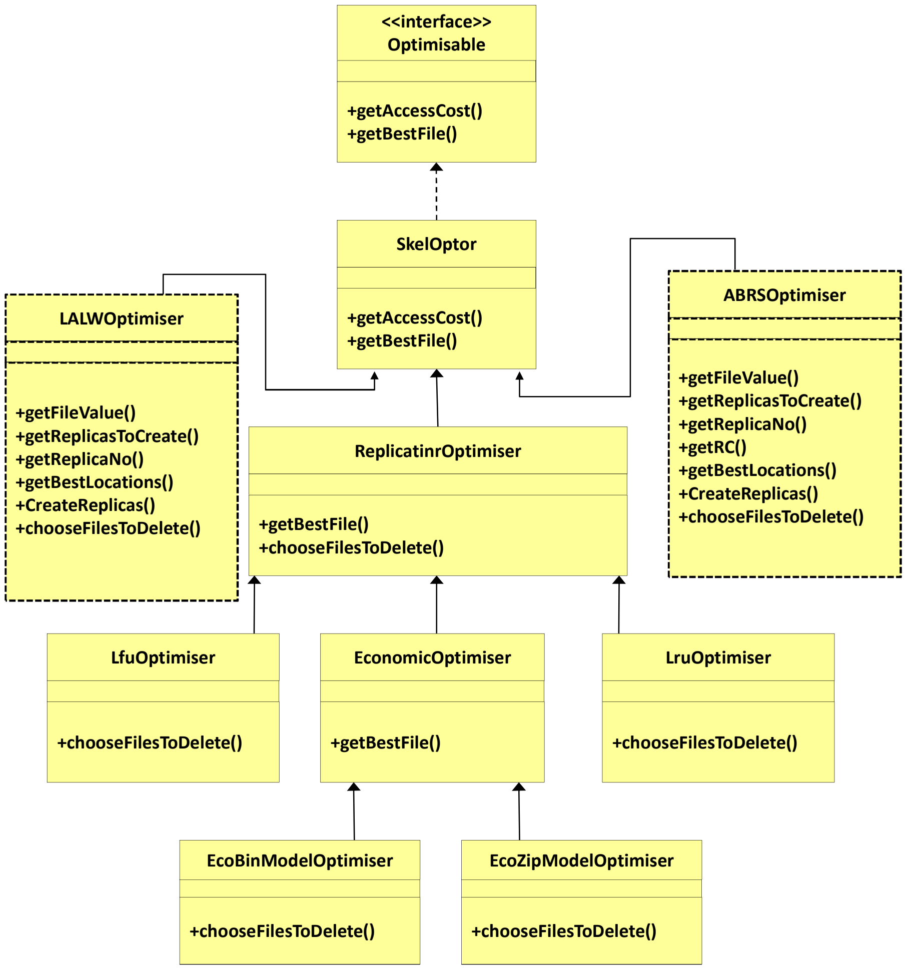


Figure 4.6: UML Class Diagram of OptorSim

The order of processing the files is according to the access pattern defined in the parameter file. The CE then calls the local optimizer 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 files for the job have been processed. In the OptorSim, each site has its own replica optimizer termed as local optimizer, and its main role is to find the best replica and replicate it in the local SE according to the chosen strategy. In this work, the Simple Optimizer is used as a local optimizer that finds the "best" replica of the required file but never replicates, as all files are read by remote I/O. The replication decision is made by the proposed algorithm, RBR. In constant time interval, RBR gets information of the files from Replica Catalogue (RC). RC holds the mappings of logical file to physical file names (Silberschatz et al., 2006), evaluates the files in the system, and makes the replication decision if it is necessary. When the replication process has been performed, the RBR registers the new replica into the RC as shown in Figure 4.7.

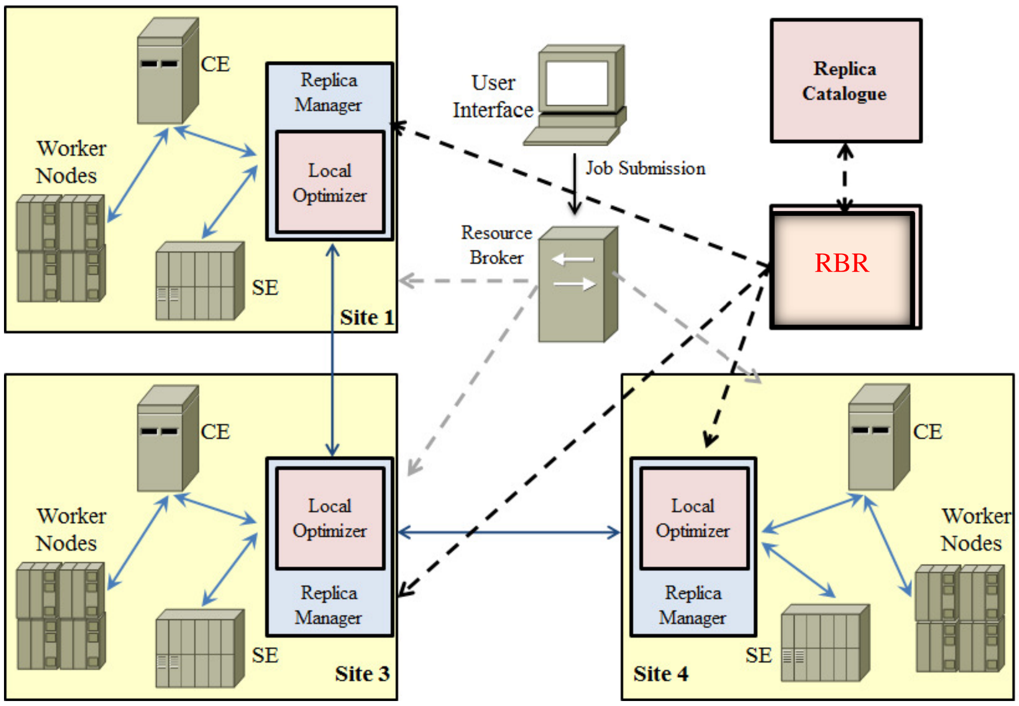


Figure 4.7: RBR in OptorSim

4.5 Summary of Chapter

This chapter presents the relationship-based replication algorithm termed as RBR. The RBR include three viewpoints in deciding files that requires replication: the file-to-user (F2U), file-to-file (F2F) and file-to-grid relationships (F2G). Such an approach is hoped to minimize the job execution time, network bandwidth consumption, and storage element usage. The performance evaluation of this algorithm is discussed in the next chapter where the RBR is also compared against existing replication algorithms.

CHAPTER FIVE
RESULTS AND DISCUSSIONS

In order to evaluate the proposed RBR, we conducted a comparative evaluation against the DRCM (Madi, 2012), LALW (Ruay-Shiung et al., 2008) and other existing algorithms (LFU and LRU) that are built-in the utilized simulator A series of tests with their results are presented based on the parameters discussed in the previous Chapter.

5.1 Number of Jobs

It is important to understand how replication algorithms perform with the increase of numbers of jobs on the grid (Ruay-Shiung et al., 2008). Using the Queue Access Cost scheduler, we undertake the workload test by conducting various number of jobs, ranging from 200 to 4000. The basic parameter settings used in this experiment is shown in Table 5.1, and result of the workload test is shown in Table 5.2.

Table 5.1: Parameter Settings for Workload Test

Parameter	Value
Number of Jobs	200, 500, 1000, 2000, 4000
Scheduler	QAC scheduler
Site Policy	All Job Types
Access history length	1000000 ms
Storage metric (D)	0.67
Max. Queue Size	200
Job Delay	2500 ms

Table 5.2: Simulation Results for Workload Test

Number of Jobs	Metrics	LRU	LFU	LALW	DRCM	RBR
200	MJET	4582	4398	3931	3792	3545
	ENU	56.22	55.23	37.87	35.16	31.92
	ASU	34.58	33.96	34.13	29.91	27.73
	CEU	21.83	19.53	22.15	23.54	23.41
500	MJET	10911	8994	7839	7791	7566
	ENU	46.19	47.46	36.88	31.17	30.06
	ASU	36.17	37.45	35.71	32.42	28.78
	CEU	18.87	20.31	25.91	26.38	27.15
1000	MJET	17108	17030	16241	14522	12311
	ENU	44.42	43.21	34.25	28.94	26.88
	ASU	39.49	39.64	37.12	35.46	29.97
	CEU	24.34	25.6	30.25	32.62	34.27
2000	MJET	56567	55948	54133	52689	50361
	ENU	45.76	46.42	32.45	27.19	24.36
	ASU	40.63	40.64	38.63	36.11	30.54
	CEU	21.5	20.43	25.74	31.75	33.83
4000	MJET	114652	106979	104129	103771	103396
	ENU	45.83	47.53	30.89	25.37	22.73
	ASU	40.62	40.64	40.11	37.63	31.54
	CEU	23.96	23.88	28.11	31.91	33.27

In order to show the efficiency of the DRCM over the existing algorithms, the efficiency values are calculated. For example, the RBR outperformed DRCM by 14.15% in ENU metric, LRU by 5.12% in MJET metric, and 14.51% in ASU metric.

Table 5.4 shows the efficiency of the RBR -as percentage values- over other existing algorithms.

Table 5.3: Efficiency Result for Workload Test

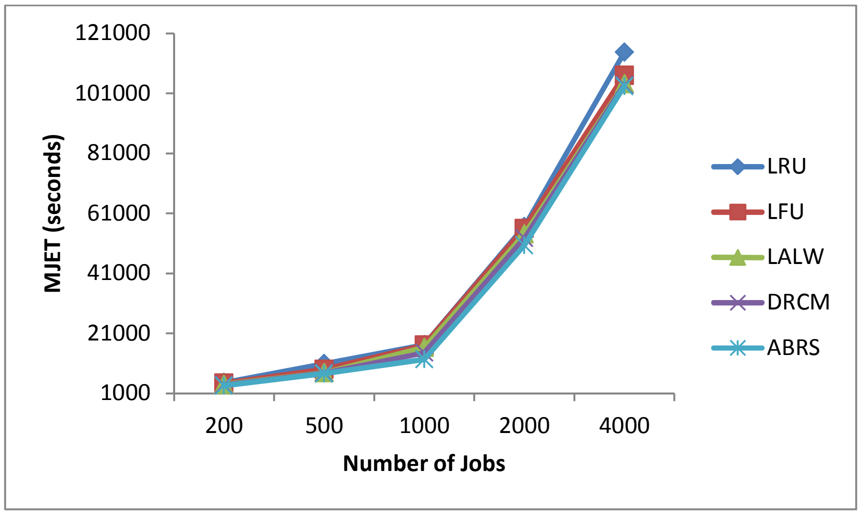
Metrics	LRU	LFU	LALW	DRCM
MJET	7.87%	7.43%	24.25%	5.12%
ENU	41.12%	40.97%	23.09%	14.15%
ASU	17.89%	18.02%	19.55%	14.51%
CEU	30.22%	32.30%	15.30%	9.04%

In what follows, we discuss and analyze the result that is presented in Table 5.2. The results show a linear increase in the MJET as the number of jobs on the grid increases. This is because, as more jobs are submitted, the queue at the sites increases. If the job submission rate is higher than the grid's job processing rate, this build-up of queues is inevitable. Hence, a preferred algorithm is an algorithm that has less MJET. As shown in Figure 5.1, for MJET, the RBR is the best among existing algorithms. Utilizing the RBR, the mean job execution time is reduced and is noted to better by 5.12% over DRCM, 24.25% over LALW, and about 7 % over LRU and LFU.

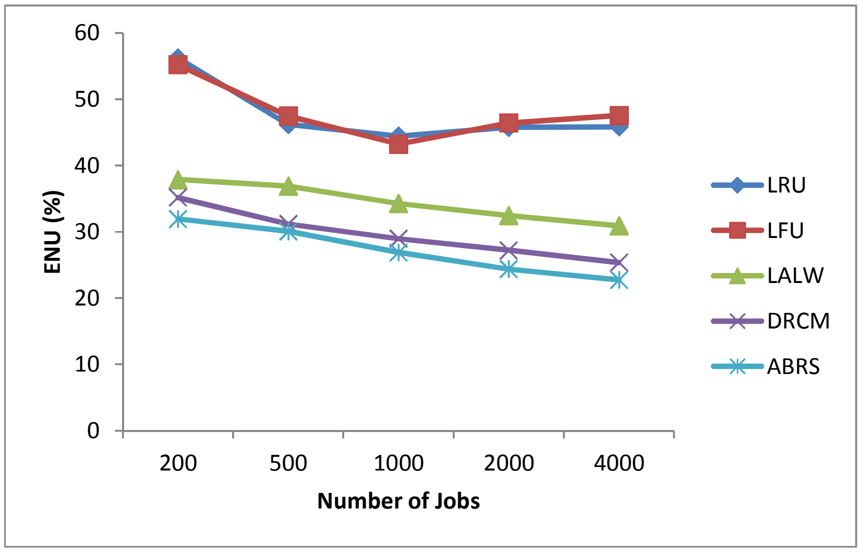
Referring to the Average Storage Usage (ASU), the LFU and LRU algorithms are noted to utilize more storage as they replicate files to the local storage. This is followed by the LALW and DRCM. However, by using RBR, the storage usage is reduced by outperforming LRU, LFU, LALW, and DRCM by 17.89%, 18.02%, 19.55%, 14.51% respectively.

On the other hand, results of Efficient Network Usage (ENU) show a slight linear decrease as number of jobs on the grid increases. This is because at the start of the simulation the queues are small, but they build up quickly while the files are replicated in the grid. Once the replication process has established, the execution time are reduced and the queue is shorten. The ENU gradually decreases with the increment in number of jobs because the amount of replication decreases over time. The LRU and LFU have the highest effective network usage, showing that they are poor at making replication decisions. The RBR uses the lowest amount of network resources for the tested number of jobs because it is able to make better decision in deciding file that requires replication.

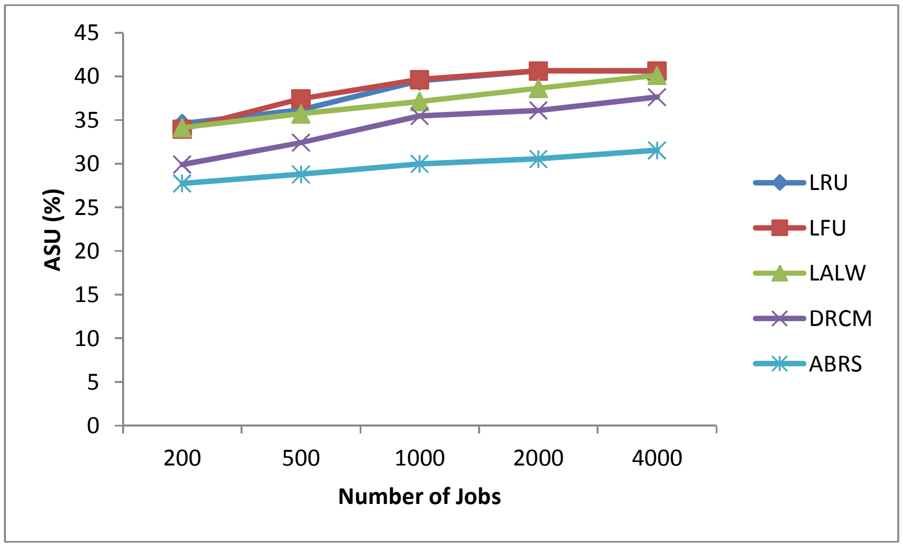
Looking at the Computing Element Usage (CEU) metric, it can be seen that the CEU generally grows as the number of jobs increases, reflecting the heavy workload. However, there is an obvious drop between 1000 and 2000, this is because with the higher number of jobs, the scheduling algorithm is sending most of the extra jobs to a few sites from where the data are easily accessible, leading to more uneven distribution of jobs around the grid. The same trend, although less marked, there is a slight drop seen with RBR. This indicates that RBR leads to make a good balance in the grid, this is because RBR distribute the replicas among the sites taking into account the workload of the sites in the grid and places of the existing replicas, which in turn drive the scheduling algorithm to make a balance while submitting the jobs, as they send the job to the computing elements that are close to the data


121000
101000
81000
61000
41000
21000
1000
MJET (seconds)
200
500
1000
2000
4000
Number of Jobs
LRU
LFU
LALW
DRCM
ABRS

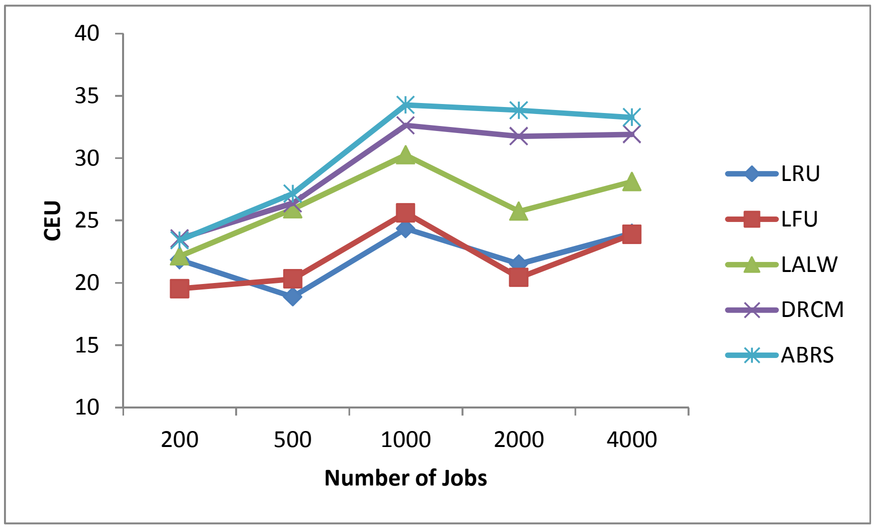
(a)


60
50
40
30
20
10
0
ENU (%)
200
500
1000
2000
4000
Number of Jobs
LRU
LFU
LALW
DRCM
ABRS

(b)



(c)



(d)

Figure 5.1: Workload Test Results of (a) MJET, (b) ENU, (c) ASU and (d) CEU

5.2 Length of Access History

In this experiment, the effect of access history length on the performance of RBR and other existing algorithms is investigated. Using QAC scheduling algorithm and submitting 500 jobs to the grid, the access history length varying between 10^3 ms and 10^6 ms. In order to test the behavior of RBR in different cases, namely when the access history has a poor information on file accesses, and when the access history has enough information on file accesses. Thus, we consider in this experiment that job submission rate (job delay) varying between 1000 ms and 2500 ms. The MJET, ENU, and ASU are measured. The basic parameter setting is shown in Table 5.4 while the result is provided in Table 5.5.

Table 5.4 Parameter Settings for Access History Test

Parameter	Value
Number of Jobs	500
Scheduler	QAC scheduler
Site Policy	All Job Types
Access history length	10^3 ms, 10^4 ms, 10^5 ms, 10^6 ms
Storage metric (D)	0.67
Max. Queue Size	200
Job Delay	1000 ms, 1500 ms, 2000 ms, 2500 ms

The first test in this experiment considers access history length of 1000 ms and job delay of 1000 ms. This means that the access history contains information on only one job files, i.e. the access history length may not be adequate as we have 500 jobs. However, we also include experiment that considers an access history length of 10^6 ms while the job delay is defined at 2500 ms, hence indicating that the access history would have a view of the overall access patterns.

Table 5.5: Simulation Results for Access History Test

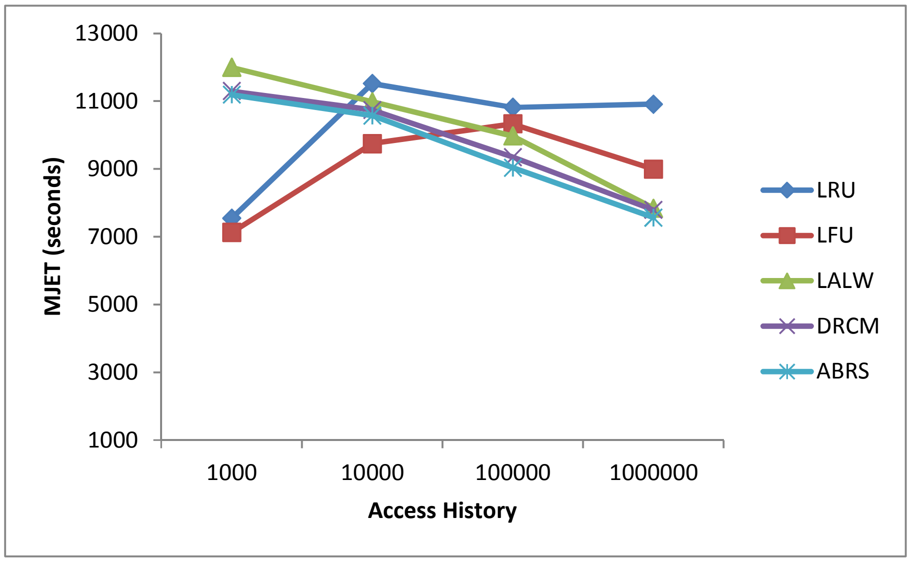
Access History Length	Job Delay	Metrics	LRU	LFU	LALW	DRCM	RBR
10^3	1000	MJET	7543	7127	11995	11295	11189
		ENU	30.96	30.85	47.26	46.26	40.29
		ASU	35.83	34.4	37.27	28.91	27.13
		CEU	21.49	20.25	23.53	25.31	27.41
10^4	1500	MJET	11518	9745	10980	10740	10570
		ENU	43.18	43.13	44.26	43.26	41.56
		ASU	38.19	36.5	37.36	30.11	28.98
		CEU	23.3	18.35	24.28	26.35	28.15
10^5	2000	MJET	10816	10327	9972	9346	9039
		ENU	43.74	47.87	40.26	37.26	33.88
		ASU	35.98	37.67	37.91	32.98	29.37
		CEU	22.75	19.4	24.98	26.49	28.47
10^6	2500	MJET	10911	8994	7839	7791	7566
		ENU	46.19	47.46	36.88	31.17	30.06
		ASU	36.17	37.45	35.71	32.42	28.78
		CEU	18.87	20.31	25.91	26.38	27.15

Based on data tabulated in table 5.5, the efficiency of the RBR -as percentage values- over other existing algorithms is shown in Table 5.6.

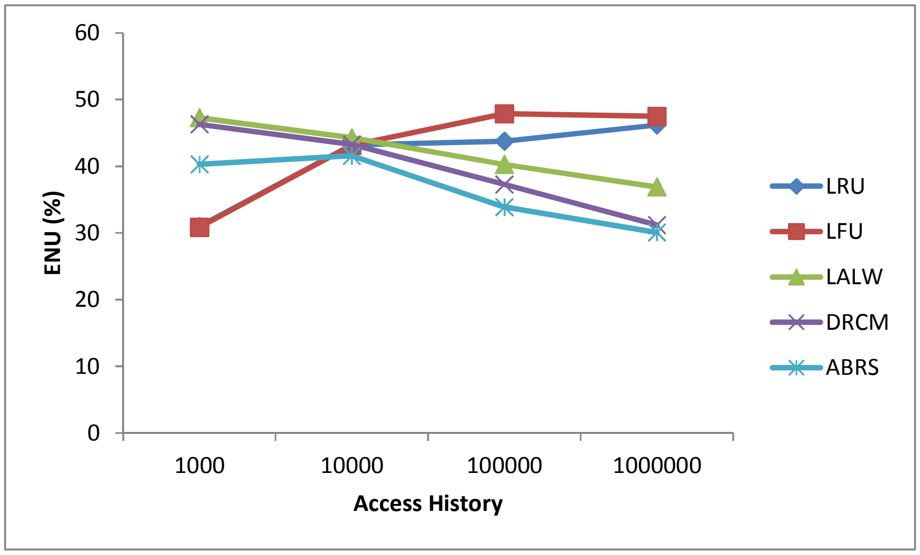
Table 5.6: Efficiency Result for Access History Test

	LRU	LFU	LALW	DRCM
MJET	3.98%	-8.21%	3.98%	2.06%
ENU	11.14%	13.89%	13.56%	7.70%
ASU	21.83%	21.75%	22.93%	8.17%
CEU	28.67%	41.97%	12.64%	6.36%

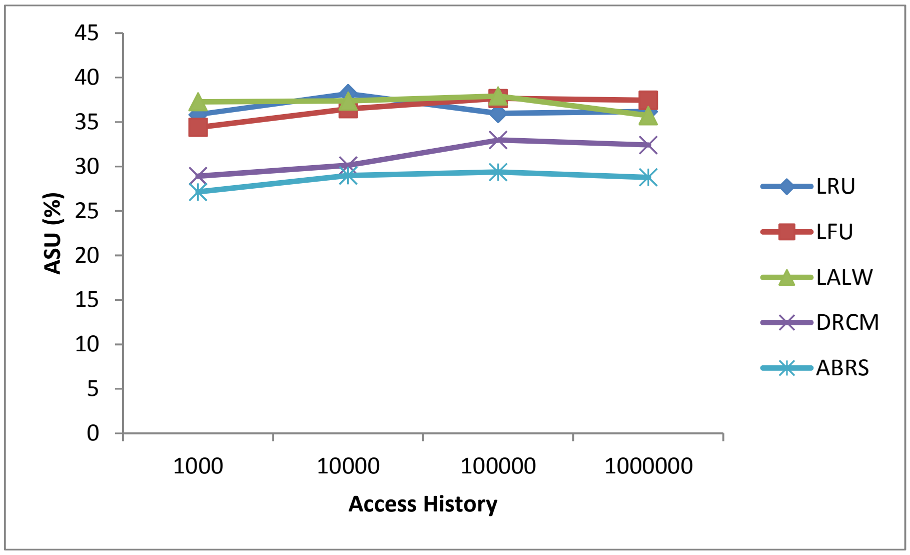
Figure 5.2 clearly shown that the performance of all algorithms get worse until the access history contains enough information on the files, and there is not a large variation in mean job time of each algorithm. The poor performance of LALW, DRCM, and RBR with small access histories, however, LFU and LRU are the best performer. LALW, DRCM and RBR perform badly with small access history because the files values changes rapidly that seemingly worthless files will be deleted when they are likely to be requested in the near future. Those strategies namely LALW, DRCM and RBR require a large access history to be able to assess well which files are worth keeping. There is no noticeable effect of the length of access history on both of storage element usage and computing element usage.


13000
11000
9000
7000
5000
3000
1000
MJET (seconds)
1000
10000
100000
1000000
Access History
LRU
LFU
LALW
DRCM
ABRS

(a)


60
50
40
30
20
10
0
ENU (%)
1000
10000
100000
1000000
Access History
LRU
LFU
LALW
DRCM
ABRS

(b)



(c)

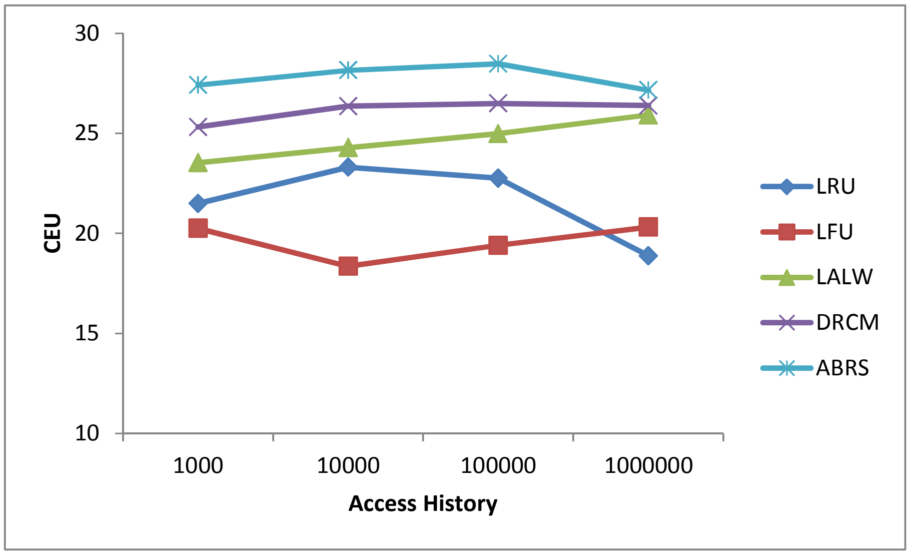


Figure 5.2: Access History Test Result for (a) MJET, (b) ENU, (c) ASU and (d) CEU

5.3 Storage Size

The sizes of the files and in turn the value of D may affect the performance of each replication algorithm. The less storage space available at a site would lead to a longer job execution time and higher network usage as fewer replicas can be accommodated in the grid. In this experiment, we investigate the performance of RBR with different storage size. The settings used in this experiment is shown in Table 5.7 and the results is shown in Table 5.8.

Table 5.7: Parameter Settings for Storage Size Test

Parameter	Value
Number of Jobs	500
Scheduler	QAC scheduler
Site Policy	All Job Types
Access history length	1000000 ms
Storage metric (D)	0.05, 0.37, 0.66, 1.31
Max. Queue Size	200
Job Delay	2500 ms

Table 5.8: Simulation results for Storage Size Test

Storage Metric	Metrics	LRU	LFU	LALW	DRCM	RBR
0.05	MJET	12004	11221	10820	10821	10932
	ENU	89.21	88.91	89.26	88.94	88.46
	ASU	94.66	93.94	91.11	91.24	90.47
	CEU	25.25	24.22	25.29	26.38	27.21
0.37	MJET	10529	10281	9734	9486	9272
	ENU	55.19	53.22	53.68	52.98	50.75
	ASU	57.21	56.36	53.98	53.91	47.35
	CEU	24.31	24.93	23.32	25.84	26.71
0.66	MJET	10011	9484	9250	8123	6985
	ENU	46.49	47.56	35.34	33.88	29.57
	ASU	37.47	38.11	36.97	32.71	27.52
	CEU	24.53	25.78	25.44	25.52	26.19
1.31	MJET	6712	7101	8174	8210	8117
	ENU	13.79	14.13	15.92	15.88	16.42
	ASU	21.92	18.95	23.34	19.21	18.83
	CEU	27.52	21.91	29.62	30.02	32.59

Data in Table 5.9 shows the efficiency of the RBR -as percentage values- over other existing algorithms.

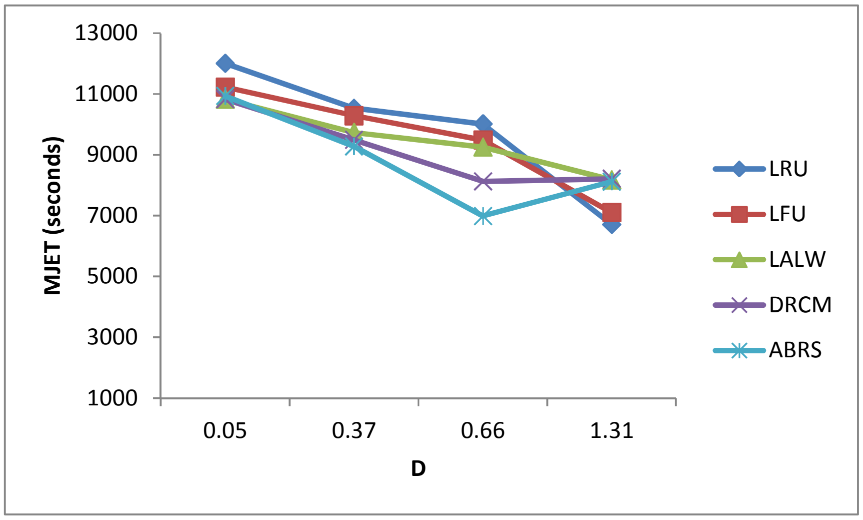
Table 5.9: Efficiency Result for Storage Size Test

	LRU	**LFU**	**LALW**	**DRCM**
MJET	10.06%	7.30%	7.04%	3.64%
ENU	9.52%	9.14%	4.63%	3.38%
ASU	12.82%	11.18%	10.34%	6.55%
CEU	10.91%	16.38%	8.71%	4.58%

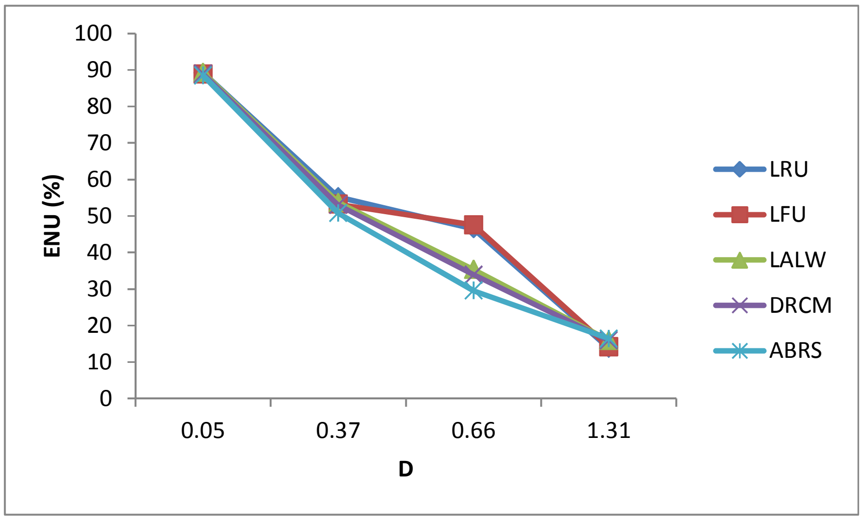
Based on the data depicted in Table 5.8, for the smallest value of D (i.e. D=0.05), the MJET all replication algorithms is high, as replication lose its advantage compared to remote access and each new job is more likely to request files which have not been requested before, because the available space in the storage elements is very limited. The job scheduler submits the jobs evenly among the sites, even if the sites have a heavy workload, and thus the increasing number of jobs that are waiting in the queue in the sites are increased as well as the mean job time. For the highest value of D (i.e. D = 1.31) as shown in Figure 5.3, the LRU and LFU are slightly faster than other strategies, because the files are likely to be read locally as LRU and LFU always replicate the files. Looking at ENU metric in Figure 5.3, it is noticeable that ENU falls as D decreases, for all replication algorithms due to the same reason, as there is enough space in storage elements to the extent that all of the replicas can be accommodated and read locally. The RBR perform the best when (D= 0.37) and D = 0.66 as it gives the shortest job execution time and smallest value of ENU. This is because LALW and DRCM ignore the age of the file, which in turn will give the recently created files small value and then later will be deleted because of their low value. However, RBR evaluates the files taking into account their ages with the aim of keeping the potential popular file available.

Moreover, RBR outperforms LALW by 3.38% in improving ENU. This is because number of replications required by RBR is less DRCM – RBR adds more restrictions in evaluating the files that depends on three criteria to determine files that require replication as compared to only two by DRCM.

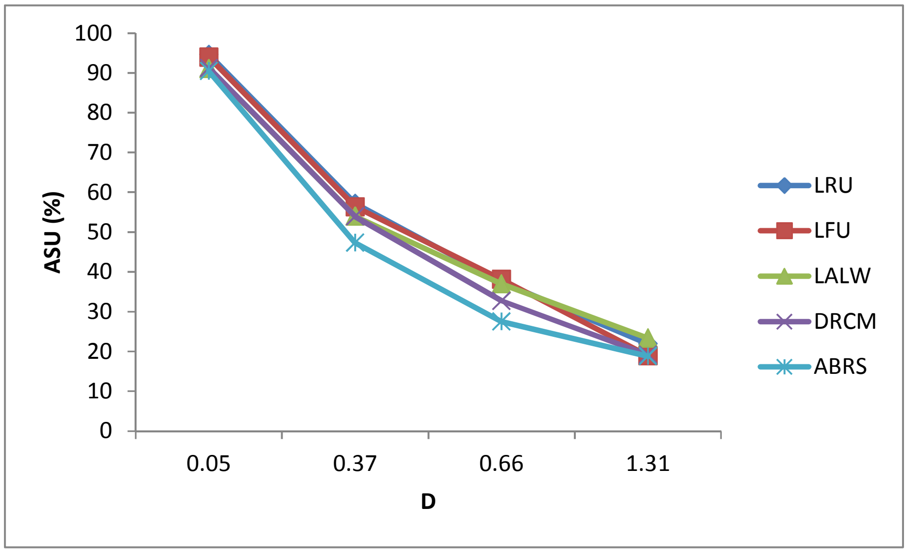
Looking at CE metric in Figure 5.3, there is a little variation in the computing element usage when value of D < 1. This is because the files are spread around the sites, in this case RBR outperform other replication algorithms. When D > 1 (i.e. D=1.3) there is a noticeable increase in computing element usage meaning that replication algorithms lead to make balance in grid system. Due to large storage space that allows the sites to store all the files in grid system, therefore every site in grid is likely to be a candidate that chosen by job scheduler to run the job. As a result, job scheduler has more choices when submitting the jobs to the grid sites and can balance number of jobs at each site.



(a)



(b)



(c)

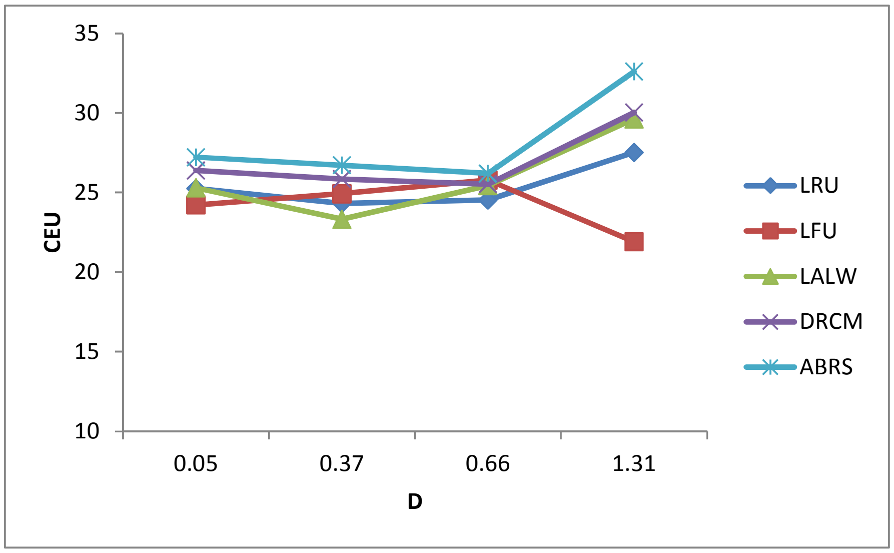


Figure 5.3: Storage Size Test Result on (a) MJET, (b) ENU, (c) ASU and (d) CEU

5.4 Summary of Chapter

In this chapter, we presented the result of our simulation experiments. In the simulation experiments, different scenarios were employed to evaluate the RBR and other relevant replication algorithms. The simulation results showed an overall improvement of the performance of data grid when RBR was employed. As a result the overall bandwidth consumption decreases, and RBR is a better algorithm for storage usage. In addition, the RBR greatly affects the work of job scheduler and in turn the overall computing element usage.

In the next chapter, the conclusion and contribution of the research work presented in this report will be described. It will also include some suggestion on how the work can be continued in the future.

CHAPTER SIX
CONCLUSION

This chapter presents a conclusion of the research work as explored and described in the report. The research contributions are supported by the experimental results which are highlighted. The applicability of the proposed algorithm in the real world is also presented, followed by a discussion of the research limitations. Eventually, several possible future research directions to realize and extend the work are also identified and recommended.

Data replication is a technique to move and cache data close to users. By replication, data access performance can be improved dynamically. The general idea of replication is to store copies of data in different locations so that data can be easily recovered if one copy at one location is lost or unavailable. Therefore, the proposed algorithm (RBR) has been designed and implemented as a response to the need of an alternative replication algorithm in the established domain, where data proliferation and limited resources in data grids are common. The main problem that is addressed by this research is how to make a decision on replica creation in order to satisfy both the grid resources and grid users.

Resource satisfaction is achieved by reducing the overall cost which includes reducing storage cost and network bandwidth. On the other hand, user satisfaction is achieved by reducing job execution time. The proposed RBR allows for greater user satisfaction and resource satisfaction simultaneously because it complies with grid resource limitations and the requirements of the users' job. Making a decision on replication and deletion is not an easy task. It was observed that considering all grid request patterns in evaluating the files which will influence the decision, is better than considering only the most well-known request. In this context, the most well-known request is the request which is made directly by the user for a specific data file. Additionally, it was learned that considering the distribution of the replication along with other parameters such as the transfer time of data file among sites, and workload of each site have a significant effect on the overall system performance, specially the job execution time, because grid sites expose geographical localities in the data grid environment.

6.1 Contribution of the Research

The contribution of this research work is related to the proposing of a new replica creation algorithm that enhances the performance of the data grid by reducing job execution time and reducing the overall grid resource cost. This has been achieved by proposing the followings:

i. **a resource selection function**

This research proposes a new function (as in equation 4.4) to be used in determining the suitability or urgency of a resource (i.e data file) to be replicated. The larger the value of the proposed $File\,Value$ (FV), the more important the resource is to the grid system. The utilized exponential model evaluates the resource in terms of user demand behavior and notes the importance of the resource to the users. This is complemented by utilizing information on the resource relationships with existing resources. Additionally, the time period of a resource is also considered in the formulation of the function.

ii. a replica quantity function

The proposed RBR also consist a function in determining the number of replication that is suitable or required by the identified resource. The proposed function 9as in equation 4.6) adapts the one presented in (Madi, 2012) by employing the proposed $File\,Value$ (FV). The proposed function determines the number of replica by considering existing volume of demand and storage space.

iii. a replication algorithm that is based on relationships

Based on the two proposed functions, this research integrates them into an algorithm that contributes in determining which resource that requires replication and how many copies of it would be beneficial to the data grid system. The proposed Relationship-based Replication (RBR) is shown in Figure 4.4.

iv. the implementation of RBR in a data grid simulator (OptorSim)

This implementation can be used by other researchers for comparison or modification purposes. It presents the proposed RBR and existing replication algorithms.

6.2 Future Work

The work reported in this research has opened up several avenues for exploration and one of the main extensions is in the area of replica management. Once the resource and number of replication has been identified, research can also be undertaken to determine the optimized location of the newly created replica. Storing a replica in the most suitable site would contribute in reducing job execution time.

The second strategy that could be included in replica maintenance is the re-location of existing resource (and its replicas) in the data grid system. This is required as the volume and pattern of demand can changed dynamically. Hence, a function or algorithm that relocates resource file and their replicas to sites that provide better services in the context of the current situation and network conditions would beneficial.

REFERENCES

A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, & Tuecke., S. (2001). The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. *Journal of Network and Computer Applications, 23*.

Abawajy, J. H. (2004). Placement of file replicas in data grid environments. *Lecture Notes in Computer Science*, 66-73.

Abdelsalam A. Helal, Abdelsalam A. Heddaya, & Bharat B. Bhargava. (1996). *Replication techniques in distributed systems*: Kluwer Academic Publishers.

Al-Mistarihi, H. H. E., & Yong, C. H. (2008). Response Time Optimization for Replica Selection Service in Data Grids. *Journal of Computer Science, 4*(6), 487-493.

Al Mistarihi, H. H. E., & Yong, C. H. (2008). Replica management in data grid. *International Journal of Computer Science and Network Security IJCSNS, 8*(6), 22.

Allcock, B., Bester, J., Bresnahan, J., Chervenak, A. L., Foster, I., Kesselman, C., . . . Tuecke, S. (2002). Data management and transfer in high-performance computational grid environments. *Parallel Computing, 28*(5), 749-771.

Allcock, B., Bester, J., Bresnahan, J., Chervenak, A. L., Kesselman, C., Meder, S., . . . Foster, I. (2001, 17-20 April 2001). *Secure, efficient data transport and replica management for high-performance data-intensive computing.* Paper presented at the IEEE Mass Storage Systems and Technologies.

Avery, P. (2002). Data Grids: a new computational infrastructure for data-intensive science. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences, 360*(1795), 1191.

Bell, W. H., Cameron, D. G., Capozza, L., Millar, P., Stockinger, K., & Zini, F. (2003). Simulation of Dynamic Grid Replication Strategies in OptorSim. *Journal of High Performance Computing Applications, 17*(4).

Bell, W. H., Cameron, D. G., Millar, A. P., Capozza, L., Stockinger, K., & Zini, F. (2003). Optorsim: A grid simulator for studying dynamic data replication strategies. *International Journal of High Performance Computing Applications, 17*(4), 403-416.

Ben Charrada, F., Ounelli, H., & Chettaoui, H. (4-6 Nov. 2010). *An Efficient Replication Strategy for Dynamic Data Grids.* Paper presented at the P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2010 International Conference on.

Ben Charrada, F., Ounelli, H., & Chettaoui, H. (2010, 4-6 Nov. 2010). *An Efficient Replication Strategy for Dynamic Data Grids.* Paper presented at the *Proceedings of* International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC),.

Bernholdt, D., Bharathi, S., Brown, D., Chanchio, K., Chen, M., Chervenak, A., . . . Fox, P. (2005). The earth system grid: Supporting the next generation of climate modeling research. *Proceedings of the IEEE, 93*(3), 485-495.

Bsoul, M., Al-Khasawneh, A., Kilani, Y., & Obeidat, I. (2010). A threshold-based dynamic data replication strategy. *The Journal of Supercomputing*, 1-10.

Caitriana M. Nicholson. (2006). *File management for HEP data grids.* PhD thesis, University of Glasgow.

Cameron, D. G., Carvajal-Schiaffino, R., Millar, A. P., Nicholson, C., Stockinger, K., & Zini, F. (2003). *UK grid simulation with OptorSim.* Paper presented at the *Proceedings of* UK e-Science All Hands Meeting, Nottingham, UK.

Cameron, D. G., Carvajal-Schiaffino, R., Millar, A. P., Nicholson, C., Stockinger, K., & Zini, F. (March 2004). Evaluating scheduling and replica optimisation strategies in OptorSim. *Journal of Grid Computing*, 57-69.

Cameron, D. G., Millar, A. P., Nicholson, C., Carvajal-Schiaffino, R., Stockinger, K., & Zini, F. (2004). Analysis of scheduling and replica optimisation strategies for data grids using OptorSim. *Journal of Grid Computing, 2*(1), 57-69.

Carman, M., Zini, F., Serafini, L., & Stockinger, K. (2002). *Towards an economy-based optimisation of file access and replication on a data grid.* Paper presented at the *Proceedings of* Second IEEE International Symposium on Cluster Computing and the Grid (CCGRID'02).

Challal, Z., & Bouabana-Tebibel, T. (2010). *A priori replica placement strategy in data grid.* Paper presented at the *Proceedings of* 2010 International Conference on Machine and Web Intelligence (ICMWI), .

Chang, H. P. (2006). A Dynamic Data Replication Strategy Using Access-Weights in Data Grids.

Chervenak, A., Deelman, E., Foster, I., Hoschek, W., Iamnitchi, A., Kesselman, C., . . . Tierney, B. (2002). *Giggle: A framework for constructing scalable replica location services*. Paper presented at the International IEEE Supercomputing Conference (SC 2002), Baltimore, USA.

Chervenak, A., Deelman, E., Kesselman, C., Allcock, B., Foster, I., Nefedova, V., . . . Drach, B. (2003). *High-performance remote access to climate simulation data: A challenge problem for data grid technologies.* Paper presented at the Super Computing.

CMS Data Challenge 2004. http://www.uscms.org/s&c/dc04/.

David G. Cameron. (2005). *Replica management and optimisation for data grids.* PhD. Thesis, University of Glasgow.

David, W. B. (2003). *Evaluation of an economy-based file replication strategy for a data grid.* Paper presented at the International Workshop on Agent based Cluster and Grid Computing.

Dooley, R., Milfeld, K., Guiang, C., Pamidighantam, S., & Allen, G. (2006). From proposal to production: Lessons learned developing the computational chemistry grid cyberinfrastructure. *Journal of Grid Computing, 4*(2), 195-208.

Dutka, L., Slota, R., Nikolow, D., & Kitowski, J. (2004). *Optimization of Data Access for Grid Environment.* Paper presented at the Grid Computing.

European Organization for Nuclear Research (CERN). http://public.web.cern.ch/Public/Welcome.html.

F. Berman, G. Fox, & Hey., T. (2003). *The Grid: Past, Present, Future, Grid Computing: Making the Global Infrastructure a Reality*. London, UK: Wiley Press.

Farooq, U., Majumdar, S., & Parsons, E. W. (2007). *Engineering grid applications and middleware for high performance.* Paper presented at the Proceedings of the 6th international workshop on Software and performance.

Fisher, M. L. (2004). The Lagrangian relaxation method for solving integer programming problems. *Management science*, 1861-1871.

Foster, I. (2000). Internet computing and the emerging grid. *Nature Web Matters, 7*.

Foster, I. (2002a). *The grid enabling resource sharing within virtual organizations.* Paper presented at the WWW 2000 Conference.

Foster, I. (2002b). The Grid: A New Infrastructure for 21st Century Science. *PHYSICS TODAY, 55*(2), 42-47.

Foster, I., Alpert, E., Chervenak, A., Drach, B., Kesselman, C., Nefedova, V., . . . Williams, D. (2002). *The Earth System Grid II: Turning climate datasets into community resources*. Paper presented at the Annual Meeting of the American Meteorological Society.

Foster, I., & Kesselman, C. (1999). The Grid: Blueprint for a New Computing Infrastructure. *San Francisco: Morgan Kaufmann Publishers, 24*(677), 8.

Foster, I., Kesselman, C., Nick, J. M., & Tuecke, S. (2002). Grid services for distributed system integration. *Computer, 35*, 37-46.

Foster, I., Kesselman, C., & Tuecke, S. (2001). The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of Supercomputing Applications, 15*(3), 200-222.

Fox, G., Ko, S. H., Pierce, M., Balsoy, O., Kim, J., Lee, S., . . . Varank, M. (2002). Grid services for earthquake science. *Concurrency and Computation: Practice and Experience, 14*(6 7), 371-393.

Frederic Magoulès. (2010). *Fundamentals of grid computing: theory, algorithms and technologies*. USA: Chapman & Hall/CRC Numerical Analysis & Scientific Computing.

G.A.Gravvanis, J. P. M., H.R. Arabina, D.A. Power (Ed.). (2009). *Grid Technology and Applications: Recent Developments*. New York: Nova Science Publishers, Inc.

Gagliardi, F., Jones, B., Grey, F., Bégin, M. E., & Heikkurinen, M. (2005). 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, 363*(1833), 1729.

Garmehi, M., & Mansouri, Y. (2007, 17-20 Dec. 2007). *Optimal Placement Replication on Data Grid Envirments.* Paper presented at the *Proceedings of* Information Technology, (ICIT 2007). 10th International Conference on.

Guy, L., Kunszt, P., Laure, E., Stockinger, H., & Stockinger, K. (2002). *Replica management in data grids*. Paper presented at the Global Grid Forum.

Hey, T., & Trefethen, A. E. (2005). Cyberinfrastructure for e-Science. *Science, 308*(5723), 817.

High Energy Physics Experiment Website. http://www.hep.net.

Holtman, K. (2001). CMS data grid system overview and requirements. *CMS Note, 37*.

Hong, L., Xue-dong, Q., Xia, L., Zhen, L., & Wen-xing, W. (2008). *Fast Cascading Replication Strategy for Data Grid.* Paper presented at the Proceedings of the 2008 International Conference on Computer Science and Software Engineering-Volume 03.

Hoschek, W., Jaen-Martinez, J., Samar, A., Stockinger, H., & Stockinger, K. (2000). Data management in an international data grid project. *Lecture Notes in Computer Science*, 77-90.

Johnston, W. E. (2002). Computational and data Grids in large-scale science and engineering. *Future Generation Computer Systems, 18*(8), 1085-1100.

Kalpakis, K., Dasgupta, K., & Wolfson, O. (2001). Optimal placement of replicas in trees with read, write, and storage costs. *IEEE Transactions on Parallel and Distributed Systems*, 628-637.

Kapitza, S. P. (2003). *The statistical theory of global population growth.* Paper presented at the Formal descriptions of developing systems.

Karl, S. B., Czajkowski, K., Fitzgerald, S., Foster, I., Johnson, A., Kesselman, C., . . . Tuecke, S. (1998). *Application Experiences with the Globus Toolkit.* Paper presented at the *Proceedings of* Eighth IEEE Symposium on High Performance Distributed Computing.

Kelly, N., Jithesh, P. V., Simpson, D. R., Donachy, P., Harmer, T. J., Perrott, R., . . . McKee, S. (2004). *Bioinformatics data and the grid: The GeneGrid data manager.* Paper presented at the UK e-Science All Hands Meeting 2004 (AHM04).

Ko, S. Y., Morales, R., & Gupta, I. (2007). *New worker-centric scheduling strategies for data-intensive grid applications.* Paper presented at the Proceedings of the 8th ACM/IFIP/USENIX international conference on Middleware.

Kreft, J. U., Booth, G., & Wimpenny, J. W. T. (1998). BacSim, a simulator for individual-based modelling of bacterial colony growth. *Microbiology, 144*(12), 3275.

Kremer, M. (1993). Population growth and technological change: one million BC to 1990. *The Quarterly Journal of Economics, 108*(3), 681-716.

Krishnamurthy, S., Sanders, W. H., & Cukier, M. (2002). *Performance evaluation of a probabilistic replica selection algorithm.* Paper presented at the Proceedings of the Seventh International Workshop on Object-Oriented Real-Time Dependable Systems, (WORDS 2002). .

Lamehamedi, H., Shentu, Z., Szymanski, B., & Deelman, E. (2003, April 2003). *Simulation of dynamic data replication strategies in data grids.* Paper presented at the *Proceedings of* 12th Heterogeneous Computing Workshop (HCW2003), Nice, France, .

Lamehamedi, H., Szymanski, B., Shentu, Z., & Deelman, E. (2002). *Data Replication Strategies in Grid Environments.* Paper presented at the Fifth International Conference on Algorithms and Architectures for Parallel Processing.

Lamehamedi, H., & Szymanski, B. K. (2007). Decentralized data management framework for data grids. *Future Generation Computer Systems, 23*(1), 109-115.

Lei, M., Vrbsky, S. V., & Hong, X. (2007). *A dynamic data grid replication strategy to minimize the data missed.* Paper presented at the *Proceedings of* 3rd International Conference on Broadband Communications, Networks and Systems. BROADNETS. .

Lin, Y. F., Wu, J. J., & Liu, P. (2008). *A List-Based Strategy for Optimal Replica Placement in Data Grid Systems.* Paper presented at the *Proceedings of* Parallel Processing, 2008. ICPP'08. 37th International Conference on.

Madi, M. (2012). *Replica Creation Algorithm for Data Grid.* (PhD), Universiti Utara Malaysia, Sintok.

Magoulès, F., Pan, J., Tan, K. A., & Kumar, A. (2009). *Introduction to grid computing* (Vol. 6): CRC.

Magoulès, F., & Yu, L. (2009). *Grid resource management: towards virtual and services compliant grid computing*: CRC Press.

Mansouri, Y., Garmehi, M., Sargolzaei, M., & Shadi, M. (2008). *Optimal Number of Replicas in Data Grid Environment.* Paper presented at the First International Conference on Distributed Framework and Applications, 2008. DFmA 2008. .

Mathew J. Wyatt, Nigel G.D. Sim, Dianna L. Hardy, & Atkinson, I. M. (2007). *YourSRB: A cross platform interface for SRB and Digital Libraries.* Paper presented at the Proceedings of the fifth Australasian symposium on ACSW frontiers-Volume 68.

Meyer, L., Annis, J., Wilde, M., Mattoso, M., & Foster, I. (2006). *Planning spatial workflows to optimize grid performance.* Paper presented at the Proceedings of the 2006 ACM symposium on Applied computing.

Naseera, S., & Murthy, K. V. M. (2009). *Agent Based Replica Placement in a Data Grid Environement.* Paper presented at the *Proceedings of* First International Conference on Computational Intelligence, Communication Systems and Networks. CICSYN'09.

Nicholson, C., Cameron, D. G., Doyle, A. T., Millar, A. P., & Stockinger, K. (2008). Dynamic data replication in lcg 2008. *Concurrency and Computation: Practice and Experience, 20*(11), 1259-1271.

Othman, O., O'Ryan, C., & Schmidt, D. (2001). The Design and Performance of an Adaptive CORBA Load Balancing Service. *IEEE Distributed Systems Online, 2*(4), 48-60.

Otoo, E., Olken, F., & Shoshani, A. (2002). *Disk cache replacement algorithm for storage resource managers in data grids.* Paper presented at the 2002 ACM/IEEE conference on Supercomputing, Baltimore, Maryland

Pangfeng, L., & Jan-Jan, W. (2006, 16-19 May 2006). *Optimal replica placement strategy for hierarchical data grid systems.* Paper presented at the Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on.

Park, S. M., Kim, J. H., Ko, Y. B., & Yoon, W. S. (2004). Dynamic data grid replication strategy based on Internet hierarchy. *International Workshop on Grid and Cooperative Computing, 1001*, 1324–1331.

Rahman, R. M., Barker, K., & Alhajj, R. (2005). *Replica placement in data grid: considering utility and risk.* Paper presented at the *Proceedings of* Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on.

Rahman, R. M., Barker, K., & Alhajj, R. (2008). Replica placement strategies in data grid. *Journal of Grid Computing, 6*(1), 103-123.

Rahman, R. M., Barker, K., & Alhajj, R. (2009). Performance evaluation of different replica placement algorithms. *International Journal of Grid and Utility Computing, 1*(2), 121-133.

Rajasekar, A., Wan, M., Moore, R., Schroeder, W., Kremenek, G., Jagatheesan, A., . . . Olschanowsky, R. (2003). Storage resource broker-managing distributed data in a grid. *Computer Society of India Journal, special issue on SAN, 33*(4), 42-54.

Ranganathan, K., & Foster, I. (2001a). *Design and Evaluation of Dynamic Replication Strategies for a High Performance Data Grid.* Paper presented at the International Conference on Computing in High Energy and Nuclear Physics, Beijing.

Ranganathan, K., & Foster, I. (2001b). Identifying Dynamic Replication Strategies for a High-Performance Data Grid. *International Grid Computing Workshop*, 75-86.

Ranganathan, K., & Foster, I. (2001c). Identifying dynamic replication strategies for a high-performance data grid. *Grid Computing—GRID 2001*, 75-86.

Ranganathan, K., Iamnitchi, A., & Foster, I. (2002). *Improving data availability through dynamic model-driven replication in large peer-to-peer communities.* Paper presented at the Global and Peer-to-Peer Computing on Large Scale Distributed Systems Workshop.

Rasool, Q., Jianzhong, L., Oreku, G. S., Shuo, Z., & Donghua, Y. (2008). *A load balancing replica placement strategy in Data Grid.* Paper presented at the *Proceedings of* Third International Conference on Digital Information Management, ICDIM, London, UK.

Rasool, Q., Li, J., & Zhang, S. (2009). *Replica Placement in Multi-tier Data Grid.* Paper presented at the *Proceedings of* 2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing.

Ruay-Shiung, C., Hui-Ping, C., & Yun-Ting, W. (2008). *A dynamic weighted data replication strategy in data grids.* Paper presented at the AICCSA 2008: *Proceedings of* IEEE/ACS International Conference on computer systems and applications.

Sashi, K., & Thanamani, A. S. (2010). *A New Replica Creation and Placement Algorithm for Data Grid Environment.* Paper presented at the *Proceedings of* 2010 International Conference on Data Storage and Data Engineering.

Shen, S. (2008). *Grid computing: International Symposium on Grid Computing*: Springer-Verlag New York Inc.

Shorfuzzaman, M., Graham, P., & Eskicioglu, R. (2008). *Popularity-Driven Dynamic Replica Placement in Hierarchical Data Grids.* Paper presented at the Parallel and Distributed Computing, Applications and Technologies, 2008. PDCAT 2008.

Silberschatz, A., Galvin, P. B., & Gagne, G. (2006). *Operating System Principles*: Wiley India Pvt. Ltd.

Sloan Digital Sky Survey website. Available online at: http://www.sdss.org/

Srikummar Venugopal. (2006). *Scheduling Distributed Data-Intensive Applications on Global Grids.* (PhD thesis), PhD thesis, University of Melbourne, Australia.

Tang, M., Lee, B., Tang, X., & Yeo, C. (2005). Combining data replication algorithms and job scheduling heuristics in the data grid. *Lecture notes in computer science, 3648*, 381.

Tang, M., Lee, B. S., Tang, X., & Yeo, C. K. (2006). The impact of data replication on job scheduling performance in the Data Grid. *Future Generation Computer Systems, 22*(3), 254-268.

Tang, M., Lee, B. S., Yeo, C. K., & Tang, X. (2005). Dynamic replication algorithms for the multi-tier Data Grid. *Future Generation Computer Systems, 21*(5), 775-790.

Teng, M., & Junzhou, L. (2005). *A prediction-based and cost-based replica replacement algorithm research and simulation.* Paper presented at the *Proceedings of* 19th International Conference on Advanced Information Networking and Applications, (AINA 2005). .

The ALICE Collaboration. ALICE Computing Model. Technical Report CERN-LHCC-2004-038/G-086. (CERN, January 2005.).

The CMS Collaboration. The CMS Computing Model. Technical Report CERN-LHCC-2004-035/G-083. (CERN, January 2005.).

The ATLAS Collaboration. The ATLAS Computing Model. Technical Report CERN-LHCC-2004-037/G-085. (CERN, January 2005.).

The European Data Grid Project. http://eudatagrid.web.cern.ch/eu-datagrid/. from http://eudatagrid.web.cern.ch/eu-datagrid/

The Globus Alliance. http://www.globus.org/.

The LHCb Collaboration. LHCb Computing Model. Technical Report CERN-LHCC-2004-036/G-084. (CERN, January 2005).

Tian, T., & Luo, J. (2007). *A Prediction-based Two-Stage Replica Replacement Algorithm.* Paper presented at the *Proceedings of* 11th International Conference on Computer Supported Cooperative Work in Design, (CSCWD 2007).

Tian, T., & Luo, J. (2010). A VO-Based Two-Stage Replica Replacement Algorithm. *Network and Parallel Computing*, 41-50.

Vazhkudai, S., Tuecke, S., & Foster, I. (2001). *Replica selection in the globus data grid.* Paper presented at the *Proceedings of* International Workshop on Data Models and Databases on Clusters and the Grid (DataGrid 2001).

Venugopal, S., Buyya, R., & Ramamohanarao, K. (2006). A taxonomy of data grids for distributed data sharing, management, and processing. *ACM Computing Surveys (CSUR), 38*(1), 3.

Venugopal, S., Buyya, R., & Winton, L. (2006). A Grid service broker for scheduling e Science applications on global data Grids. *Concurrency and Computation: Practice and Experience, 18*(6), 685-699.

Wang, C., Yang, C., & Chiang, M. (2007). A Fair Replica Placement for Parallel Download on Cluster Grid. *Lecture Notes in Computer Science, 4658*, 268.

Wasson, G., & Humphrey, M. (2003). *Policy and enforcement in virtual organizations.* Paper presented at the Proceedings of the 4th International Workshop on Grid Computing.

Wilkinson, B. (2009). *Grid computing: techniques and applications*: Chapman & Hall/CRC.

Wolski, R. (1997). *Forecasting network performance to support dynamic scheduling using the network weather service.* Paper presented at the *Proceedings of* The Sixth IEEE International Symposium on High Performance Distributed Computing.

Wuqing, Z., Xianbin, X., Zhuowei, W., Yuping, Z., & Shuibing, H. (2010, 20-22 Aug. 2010). *A Dynamic Optimal Replication Strategy in Data Grid Environment.* Paper presented at the *Proceedings of* International Conference on Internet Technology and Applications, .

Xie, M., Dai, Y. S., & Poh, K. L. (2004). *Computing systems reliability: models and analysis*: Springer Us.

Yang, C. T., Fu, C. P., & Huang, C. J. (2007). *A dynamic file replication strategy in data grids.* Paper presented at the TENCON 2007-2007 IEEE Region 10 Conference.

Yang, C. T., Huang, C. J., & Hsiao, T. C. (2008). *A Data Grid File Replication Maintenance Strategy Using Bayesian Networks.* Paper presented at the Intelligent Systems Design and Applications, 2008. ISDA'08.

Yi-Fang, L., Pangfeng, L., & Jan-Jan, W. (2006). *Optimal placement of replicas in data grid environments with locality assurance.* Paper presented at the Parallel and Distributed Systems, 2006. ICPADS 2006. 12th International Conference on.

You, X., Chang, G., Chen, X., Tian, C., & Zhu, C. (2006). *Utility-Based Replication Strategies in Data Grids.* Paper presented at the Fifth International Conference on Grid and Cooperative Computing.

Yu, J., & Buyya, R. (2005). A taxonomy of scientific workflow systems for grid computing. *ACM Sigmod Record, 34*(3), 44-49.

Zhao, W., Xu, X., Xiong, N., & Wang, Z. (2009). *A Weight-Based Dynamic Replica Replacement Strategy in Data Grids.* Paper presented at the *Proceedings of* Asia-Pacific Services Computing Conference, .

Zhong, H., Zhang, Z., & Zhang, X. (2010). *A Dynamic Replica Management Strategy Based on Data Grid.* Paper presented at the *Proceedings of* 2010 Ninth International Conference on Grid and Cloud Computing.