Performance analysis of data replication and scheduling in data grid

Zhang, Junwei

2009

Zhang, J. W. (2009). Performance analysis of data replication and scheduling in data grid.
Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/38584>

<https://doi.org/10.32657/10356/38584>

NANYANG TECHNOLOGICAL UNIVERSITY

**Performance Analysis of
Data Replication and Scheduling in Data Grid**

Zhang Junwei

**A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of
Doctor of Philosophy**

**School of Computer Engineering
Nanyang Technological University
Singapore**

September 2009

Abstract

The Grid is an infrastructure that enables dynamic sharing and coordinated access of resources among different organizations. As a specialization and extension of the Grid, Data Grid emphasizes on the sharing of large-scale data sets and data storage resources. It has evolved to be the solution for data intensive applications, such as global climate change, High Energy Physics (HEP), astrophysics, and computational genomics. In these research domains, the size of scientific data is measured in terabytes (1024 gigabyte) or even petabytes (1024 terabytes). Such scientific data are stored as large files and replicated across the Data Grid. Scientists geographically located all over the world are able to download these datasets and analyze them for various purposes.

Hierarchical Data Grid is a class of Data Grid that has been adopted by European Organization for Nuclear Research (CERN) to support the distribution of large experimental datasets across the globe. There have been a lot of research works on replication algorithms for the Hierarchical Data Grid. I have developed a probabilistic model of data replication in a Hierarchical Data Grid environment. The model enables us to evaluate the optimality of the replication algorithm in terms of average response time and average bandwidth cost. The accuracy of the model is verified through simulation.

Previous studies in Data Grid have simply assumed that the data is read only and does not change over time; and did not consider the data consistency management. However live data have expiration time as there is a need for relevance in the computation of the data, e.g. earth monitoring data. With the amount of

ABSTRACT

applications supported by the Data Grid constantly increasing, it is essential to take into account the consistency management of the replicas. I have adopted the expiration-based consistency management to keep the data synchronized. The performance of several well-known dynamic replication algorithms for the Hierarchical Data Grid are re-evaluated in the environment where the files have an expiration time, i.e. Time-To-Live (TTL) period. The TTL value does have a major impact on the performance of the algorithms as retrieving a file would depend on not only its availability at a replica server but also its validity due to the time period. I have re-evaluated our previous replication algorithms, and the simulation results show that the Aggregate Bottom-Up (ABU) outperforms the other dynamic replication algorithms in terms of average bandwidth cost and average response time.

As compute power increases, the delay caused by “CPU idle waiting for input data” becomes more pronounced for data intensive applications. In our research, I studied the impact of data transfer on the performance of job scheduling in the Data Grid environment. I have proposed a parallel downloading system that supports replicating data fragments and parallel downloading of replicated data fragments, to improve the job scheduling performance. The performance of the parallel downloading system is compared with non-parallel downloading system, using three scheduling heuristics: Shortest Turnaround Time (STT), Least Relative Load (LRL) and Data Present (DP). Our simulation results show that the proposed parallel download approach greatly improves the Data Grid performance for all three scheduling algorithms, in terms of the geometric mean of job turnaround time. The advantage of parallel downloading system is most evident when the Data Grid has relatively low network bandwidth and relatively high compute power.

Acknowledgements

Firstly, I would like to express my sincerest gratefulness to my supervisor, Associate Professor Francis Lee Bu Sung, for his invaluable advice and suggestions for my research work and also for his kind guidance and encouragement in the aspects of academic study.

Secondly, I would also like to express my whole-hearted gratitude to Associate Professor Tang Xueyan for his invaluable advice, suggestions and comments for my research work. The thanks also go to Associate Professor Yeo Chai Kiat for providing useful comments and suggestions for the work.

Thirdly, I am also grateful to my fellow graduate students and colleagues in Parallel and Distributed Computing Centre (PDCC), especially Alan Tan Yu Shyang for proofreading my thesis and Boom Sivadon Chaisiri for helpful discussions and suggestions on my research work. I also wish to say thanks to the lab technicians of PDCC, especially Iren Ng-Goh Siew Lai for her kind assistance and support.

Last but not least, I am deeply grateful to my family members and friends, for their unconditional love, support and encouragement throughout my Ph.D. study.

TABLE OF CONTENTS

Table of Contents

Abstract	I
Acknowledgements.....	III
Table of Contents	IV
List of Figures	VII
List of Tables.....	IX
List of Abbreviations & Acronyms.....	X
Chapter 1.....	1
Introduction	1
1.1 Background	1
1.2 Motivation.....	3
1.3 Major Contributions	4
1.4 Thesis Organization	6
1.5 Terminology	6
Chapter 2.....	8
Literature Review	8
2.1 Data Grid Architecture	8
2.2 Data Grid Projects	14

TABLE OF CONTENTS

2.3	Replication Strategies for the Data Grid	20
2.4	Data Consistency Management.....	28
2.5	Data Replication and Job Scheduling	32
Chapter 3.....		36
Modeling the Optimal Performance of the Hierarchical Data Grid		36
3.1	Introduction.....	36
3.2	Related Works	37
3.3	Modeling the Hierarchical Data Grid	39
3.4	Performance Analysis	43
3.5	Simulation Results and Model Verification	55
3.6	Summary	66
Chapter 4.....		67
Analysis of Replication Algorithms with		
Expiration-Based Consistency Management		67
4.1	Introduction.....	67
4.2	Replication Algorithms	68
4.3	Expiration-Based Consistency Management	69
4.4	Simulation Environment	71
4.5	Simulation Results	75
4.6	Discussion	80
4.7	Summary	81
Chapter 5.....		82
Impact of Parallel Download on Job Scheduling in Data Grid		
Environment		82
5.1	Introduction.....	82

TABLE OF CONTENTS

5.2	Related Works	84
5.3	System Model and Performance Metric	87
5.4	Theoretical Analysis with M/M/m Queuing Model	90
5.5	Replication Algorithms	92
5.6	Scheduling Algorithms	95
5.7	Simulation Environment	98
5.8	Simulation Results	100
5.9	Discussion	112
5.10	Summary	113
Chapter 6.....		114
Conclusions and Future Work		114
6.1	Conclusions	114
6.2	Directions for Future Work	117
Author's Publications		122
References		123
Appendix A		140
Appendix B.....		142

List of Figures

Figure 2.1: The Layered Grid Architecture [3]	9
Figure 2.2: Major Components of the Data Grid Architecture [3, 19].....	11
Figure 2.3: Major Service Components of the Data Grid Architecture [12].....	14
Figure 2.4: Data Grid Architecture for the LHC Computing Grid [37]	16
Figure 2.5: Earth System Grid (ESG) Collaboration Sites [50]	19
Figure 2.6: Grid Datafarm Architecture [69]	22
Figure 2.7: The Storage Resource Broker (SRB) [74]	24
Figure 2.8: Replication Architecture including the Consistency Service [89]	30
Figure 3.1: Modeling the Hierarchical Data Grid	40
Figure 3.2: A Hierarchical Data Grid Example	42
Figure 3.3: The Multi-Tier Data Grid Model	56
Figure 3.4: An Example of Access Record for SBU and ABU	61
Figure 3.5: Simulation Results for Geometric Distribution	64
Figure 3.6: Simulation Results for Zipf-like Distribution.....	65
Figure 4.1: A Simplified Hierarchical Data Grid with Expiration-Based Consistency Management	70
Figure 4.2: The Simulated Multi-Tier Data Grid	72
Figure 4.3: Average Bandwidth Cost Comparison for $r=3.84\%$	76

LIST OF FIGURES

Figure 4.4: Average Response Time Comparison for $r=3.84\%$	76
Figure 4.5: Average Bandwidth Cost Comparison for $r=38.4\%$	79
Figure 4.6: Average Response Time Comparison for $r=38.4\%$	79
Figure 5.1: System Architecture of the Studied Data Grid.....	87
Figure 5.2: Modeling Job Scheduling Performance with M/M/m Queue [134]	91
Figure 5.3: An Example of the Replica Lists	94
Figure 5.4: Job Turnaround Time Comparison	101
Figure 5.5: CPU Utilization Comparison	104
Figure 5.6: Results for 1 Mbps Network.....	106
Figure 5.7: Results for 10 Mbps Network.....	107
Figure 5.8: Results for 100 Mbps Network.....	108
Figure 5.9: Results for 10 Mbps Network, and High Compute Power of 100.....	111
Figure 6.1: Accessing Replica through Virtual Connections between Peers	118

List of Tables

Table 2.1: Data Grid Projects Categorized by Model [29].....	15
Table 2.2: Replication Strategies Categorized by Replication Method [29]	20
Table 3.1: Hierarchical Data Grid Model Parameters and Notations.....	41
Table 3.2: Notations Used in the Model Analysis.....	47
Table 3.3: Parameters for the Multi-Tier Data Grid with $r=38.4\%$	57
Table 3.4: Affected Parameter Values for the Case of $r=3.84\%$	60
Table 3.5: The Optimal Performance Predicted by the Model.....	60
Table 4.1: Simulation Parameters for $r=38.4\%$	73
Table 4.2: Affected Parameter Values for $r=3.84\%$	74
Table 5.1: Simulation Parameters.....	98

List of Abbreviations & Acronyms

ABU	Aggregate Bottom-Up replication algorithm
AMTT	Arithmetic Mean of Turnaround Time
API	Application Programming Interface
ARCS	Adaptable Replica Consistency Service
BHR	Bandwidth Hierarchy based Replication
BIRN	Biomedical Informatics Research Network
CE	Computing Element
CERN	European Organization for Nuclear Research
CMS	Compact Muon Spectrometer
DP	Data Present scheduling algorithm
DS	Dataset Scheduler
DT	Download Time
Ecogrid	Ecological observatory project
EGEE	Enabling Grids for E-sciencE
ES	External Scheduler
ESG	Earth System Grid
ET	Execution Time
FCFS	First-Come-First-Serve
FSP	Fast SPread replication algorithm
FTP	File Transfer Protocol
GCS	Grid Consistency Service
GDMP	Grid Data Management Pilot

LIST OF ABBREVIATIONS & ACRONYMS

Gfarm	Grid Datafarm
GMTT	Geometric Mean of Turnaround Time
Grid3	Grid2003 production Grid
GridPP	UK Computing for Particle Physics
GriPhyN	Grid Physics Network
GSI	Grid Security Infrastructure
GWA	Grid Workloads Archive
HCS	Hierarchical Cluster Scheduling
HEP	High Energy Physics
HRS	Hierarchical Replication Strategy
LALW	Latest Access Largest Weight
LAN	Local-Area Network
LCG	LHC Computing Grid
LFU	Least Frequently Used
LHC	Large Hadron Collider
LRL	Least Relative Load
LRU	Least Recently Used
LS	Local Scheduler
LSR	Logical Storage Resource
MCAT	Metadata Catalog
MSDT	Multi-Source Data Transfer
NEESgrid	Network for Earthquake Engineering Simulation
NoA	Number of Access
n-PDS	non-Parallel Downloading System
OPT	OPTimal replication algorithm
OU	Overall CPU Utilization
PDS	Parallel Downloading System
PPDG	Particle Physics Data Grid
PRS	Primary Replica Server
PSR	Physical Storage Resource

LIST OF ABBREVIATIONS & ACRONYMS

QT	Queuing Time
RCS	Replica Consistency Service
RSR	Random Static Replication algorithm
SBU	Simple Bottom-Up replication algorithm
SE	Storage Element
SRB	Storage Resource Broker
SRS	Secondary Replica Server
STT	Shortest Turnaround Time
TT	Turnaround Time
TTL	Time-To-Live
VO	Virtual Organization
WAN	Wide-Area Network

Chapter 1

Introduction

1.1 Background

Computers have been used as tools to solve problems in fields such as modeling and simulating complex scientific and engineering problems, diagnosing medical conditions, industrial equipments automation, weather forecasting, stock portfolios management and etc [1]. Despite the advancements made in the compute power, data storage and communication technology areas, computational resources are still deemed insufficient due to the continual emergence of increasingly complex problems [2]. To address this problem, Grid [1-9] is proposed to provide an infrastructure that enables secure sharing, scalable discovering and negotiating access to remote resources via internet; thus making it possible for large scale scientific collaboration among geographically distributed researchers. The new infrastructure provides an aggregation of computational resources, which could be idled low-end personal computers or large clusters, and allows applications to access the resources in a demand-driven manner.

The Grid is a metaphor, comparing the ease in utilizing computational resource from the proposed infrastructure to the ease in accessing electricity from a power grid. As defined by Ian Foster and Carl Kesselman [1] in 1998: “A *computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational*

CHAPTER 1. INTRODUCTION

capabilities". Two years later, to address the social and policy issues, the definition of Grid Computing is refined as "*coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organization*" [3]. The resources shared include computers, software, data and other resources as required by the problem-solving collaboration. The resource providers and consumers then clearly define sharing rules such as what is shared, who is allowed to share and conditions under which sharing occurs [3]. The term "virtual organization (VO)" is then used to refer to a set of individuals and/or institutions defined by such sharing rules.

The major applications that require the use of Computing Grid can be categorized into 5 classes: Distributed Supercomputing, High-Throughput Computing, On-Demand Computing, Data-Intensive Computing and Collaborative Computing [1]. These applications usually require large amount of computational resources that cannot be met by a single system cost-effectively and conveniently. Grid Computing provides standard service protocols to support scalable, secure and high-performance sharing and coordinating of computational resources to meet the requirements of these applications.

The data-intensive applications are both computational and communication intensive. They need to process large set of data that is maintained in geographically distributed repositories, which are continually growing with newly generated data. Over the last few years, the amount of scientific data has increased tremendously due to the increase in the accuracy and growth in the number of digital instruments that have been developed and deployed around the world. Over the next few years, a major jump in the amount of data for the physicists is expected as the European Organization for Nuclear Research (CERN) [10] Large Hadron Collider (LHC) [11] became operational in 2009. As claimed by CERN, they will handle as much information as the entire European telecommunications network does today.

As a specialization and extension of the Grid, Data Grid [12-19] emphasizes on the sharing of large-scale data sets and data storage resources. It has evolved to be the solution for data-intensive applications, such as global climate change, High Energy Physics (HEP), astrophysics, and computational genomics. In these research domains, the size of scientific data is measured in terabytes (1024 gigabytes) or even petabytes (1024 terabytes).

1.2 Motivation

In the Data Grid environment, scientific data are stored as large files and replicated across the Data Grid. Scientists geographically located all over the world are able to download these datasets and analyze them for various purposes. Data replication can effectively reduce access latency and bandwidth consumption in distributed environment. It also helps to achieve load balance, provides fault tolerance, and improves resource availability and reliability. However, due to storage constraint, only a small fraction of the data files can be replicated. Thus, replication algorithms are needed to determine when to perform replication, what files to replicate and where to place the replicas.

The liveness of data is important to these applications. With the amount of applications supported by the Data Grid growing continuously, it is essential to take into account the consistency management of the replicas. When data sets are modified by applications, it is important to maintain consistency among all the copies. Live data have expiration time as there is a need for relevance in the computation of the data, e.g. earth monitoring data [47]. Although relatively new in the area of Data Grid, Expiration-based consistency management is the most widely used mechanism in today's internet [20]. In this approach, each replica is associated with a Time-To-Live (TTL) value. The replica is considered invalid when the TTL expires and the invalid replica will be updated on receiving new access requests. Thus, additional validation cost is introduced in retrieving the data files. The performance of the replication algorithms needs to be re-evaluated with the addition of data consistency management.

Another issue that is observed in a Data Grid environment is that “CPU idle waiting for input data”. The data intensive job cannot start processing until the complete data is available in the compute site. The input data is usually huge in size and thus causes a significant delay to the job turnaround time. As the compute power increases, the delay caused by “CPU idle waiting for input data” becomes more pronounced for data intensive applications. Data replication has proven to be able to reduce such delay in the Data Grid environment. However, previous approaches replicate the complete file on constrained number of replica servers and users can only download the data file from single replica server at one time. A new parallel downloading system is needed to further improve the job scheduling performance in the Data Grid environment. The new system should be able to support replicating data fragments, since the storage is limited, and still preserve the completeness of the file at the same time. It should also allow the users to download replicated data fragments from different servers concurrently and re-assemble them as a complete data file later.

1.3 Major Contributions

The main contributions of this thesis are as follows:

- *Predicting the optimal performance of the Hierarchical Data Grid [138].*
An optimal replication algorithm for the Hierarchical Data Grid is proposed. A probabilistic model is constructed to analyze the performance of the proposed replication algorithm and it gives the upper bound of the performance in terms of both average response time and average bandwidth cost. The model is further verified by simulating the multi-tier Data Grid, which is a specific example of the Hierarchical Data Grid. The performance of the proposed optimal replication algorithm is studied together with three well-known replication algorithms. The simulation results show that the optimal replication algorithm has the best performance and its performance is pretty close to the result obtained

using our probabilistic model.

- *Performance analysis of replication algorithms with consistency management [139].* The generic Hierarchical Data Grid is extended to support the data synchronization with expiration-based consistency management. The performance of several well-known dynamic replication algorithms for the Hierarchical Data Grid are re-evaluated in the environment where the files have an expiration time, i.e. Time-To-Live (TTL) period. Our results show that the TTL value does have a major impact on the performance of the algorithms, as retrieving a file would depend on not only its availability at a replica server but also its validity due to the time period. Furthermore, the simulation results show that the Aggregate Bottom-Up (ABU) outperforms the other dynamic replication algorithms in terms of both average bandwidth cost and average response time.
- *Improving job scheduling performance with parallel downloading system in Data Grid environment [140].* The impact of data transfer on the performance of job scheduling in the Data Grid environment is studied. A parallel downloading system that supports replicating data fragments and parallel downloading of replicated data fragments is proposed. Its performance is compared with non-parallel downloading system, using three scheduling heuristics: Shortest Turnaround Time (STT), Least Relative Load (LRL) and Data Present (DP). The simulation results show that the proposed parallel download approach greatly improves the Data Grid performance for all three scheduling algorithms, in terms of the geometric mean of job turnaround time. The advantage of the proposed parallel downloading system is most evident when the Data Grid has relatively low network bandwidth and relatively high compute power.

1.4 Thesis Organization

The rest of the thesis is organized as follows:

Chapter two reviews the literature that is relevant to our research. The Data Grid Architecture and representative Data Grid projects are reported. Previous work on replication strategies for various types of Data Grid; data consistency management for the Data Grid and the combined performance of job scheduling and data replication algorithms are reviewed.

Chapter three presents a probabilistic model to analyze the optimal performance of the Hierarchical Data Grid. An optimal replication algorithm is proposed and its performance is compared with several heuristic algorithms using simulations.

Chapter four analyzes the performance of replication algorithms with expiration-based consistency management in a Hierarchical Data Grid environment. The impact of consistency management on the performance of replication algorithms is discussed.

Chapter five studies the impact of data transfer on job scheduling performance in the Data Grid environment. A system that supports replicating data fragments and parallel downloading of replicated data fragments is proposed; and its performance is evaluated and compared with previous approach.

Chapter six concludes the thesis by summarizing our research results. Some possible future work is also discussed.

1.5 Terminology

It is to be noted that although *Hierarchical Data Grid* and *multi-tier Data Grid* are two similar terminologies. In this thesis, the term *Hierarchical Data Grid* refers to one class of Data Grid which has a tree topology. The term *multi-tier Data Grid* specially refers to the Data Grid architecture that is proposed and adopted by European Organization for Nuclear Research (CERN) for Large Hadron Collider (LHC) experiments.

CHAPTER 1. INTRODUCTION

In this thesis, the term *client* refers to an entity in the Data Grid that requires service(s) from other entities; while the term *server* refers to an entity in the Data Grid that provides service(s) to other entities. The term *node* refers to an entity in the Data Grid that can be either *client* or *server*.

As for the definition of the *tier number* of the Hierarchical Data Grid, the root of the tree is considered to be at tier 0. Tier 1 refers to the tier that is immediately below the root. And tier 2 is below tier 1. Thus, a node with a lower tier number is nearer to the root. The node with higher tier number is nearer to the bottom of the tree.

Chapter 2

Literature Review

2.1 Data Grid Architecture

With the trend of increasing collaborations among scientists around the world to tackle more complex scientific problems, the Grid concept and technology is developed to provide the underlying support infrastructure. Resources like computational servers, data storage servers and scientific datasets are shared across these scientific communities. Sharing rules are defined by resource providers and consumers to clearly specify who are allowed to share and access resources; and under what conditions. The concept of Virtual Organization (VO) is then formed as “a set of individuals and/or institutions defined by such sharing rules” [3]. To establish, manage and exploit resource sharing dynamically across VO, the Grid architecture is proposed in [3], as shown in Figure 2.1 [3]. The proposed Grid architecture is a *protocol* architecture in the sense that it contains a set of protocols, which defines clearly the basic mechanisms used by VO users to negotiate, establish, manage and exploit sharing relationships. The architecture is designed to be extensible, interoperable, and portable and allows code sharing. The well known open source toolkit used to build Grids: Globus Toolkit [21-22] is one implementation of this architecture.

Figure 2.1: The Layered Grid Architecture [3]

2.1.1 Layered Architecture for Generic Grid

There are five main layers in the proposed Grid architecture: Fabric layer, Connectivity layer, Resource layer, Collective layer and Application layer. Each layer contains a group of standard protocols and serves as foundations to the upper layers. The details of each layer are described in the following:

- **Fabric Layer:** the Grid fabric layer provides mechanisms to allow direct access to the shared resources, such as computational resources, storage systems, network resources and etc. The “resource” can be a physical machine, or a logical entity such as a distributed file system [3]. Standard interfaces are defined to control and access these resources. For computational resource, basic functions to query the current status, monitor running processes, start/stop/restart the resource and etc. are defined here. For storage resource, mechanisms to check storage space, access file information, create new file and etc. are provided. For network resource, tools should be provided for users to query/allocate/reserve

network bandwidth.

- **Connectivity Layer:** the connectivity layer defines communication and authentication protocols that enable secure data exchange between fabric layer resources. The communication protocols define standard interfaces to transfer data across the Grid environment, possibly rely on the existing internet protocols [23] such as TCP/IP. The authentication protocols provide mechanisms to ensure the security of such data transfer and also clearly define access rules for different users/roles. The Grid authentication solutions should also include features like *single sign on*, where people only need to authenticate themselves once to have access to multiple Grid resources.
- **Resource Layer:** the resource layer contains a set of protocols that manage the sharing of single fabric layer resource. Protocols at this layer define standards to negotiate, initiate, monitor and control the sharing of individual resource. They use fabric layer functions to access and control these shared resources; and utilize the connectivity layer protocols to exchange information securely. There are two main types of protocols at this layer. The first type is information protocols, which are used to obtain information about the state of the resource, such as current load for compute server and available storage space for storage server. The second type is management protocols, which provide standard operations to manage the shared resource, such as negotiating access to resource and creating/monitoring sharing processes.
- **Collective Layer:** the collective layer protocols and services coordinate the sharing of a collection of resources and handle their interactions. They are built on top of the resource layer protocols, which provide management for a single fabric layer resource. One typical service provided at this layer is the *directory service*, which lists out the shared resources to the VO users. Users can also search resources by name or by

resource attributes through the *directory service*. Another example is the Grid job scheduling service, which schedules Grid jobs according to scheduling policy and allocates one or more resources to the job.

- **Application Layer:** the application layer sits on top of the other four layers and comprises user applications that operate in a Grid environment. The applications can interact with each other and access the shared resources through well-defined protocols at lower layers. Such applications include drug discovery, economic forecasting, seismic analysis, and back-office data processing [24].

2.1.2 Layered Architecture for Data Grid

Figure 2.2: Major Components of the Data Grid Architecture [3, 19]

Data Grid is a specialization and extension of the Grid [12]; and it also follows the layered architecture shown in Figure 2.1 [3]. The purpose of Data Grid is mainly to provide an essential infrastructure for the emerging data-intensive applications [19]. Such applications usually need to process huge amount of data, which are generated in physics experiments or by supercomputers in scientific simulations. Data Grid provides two main components in addition to the Grid to

support these applications: reliable, high-speed data transport service and replica management service. One popular data transport protocol used in Data Grid is the GridFTP protocol [25-27], which extends the standard File Transfer Protocol (FTP) [28]. GridFTP is optimized for secure data transport in high-bandwidth Wide-Area Networks (WAN) environment; and it supports a rich set of features such as third-party control of data transfer, parallel data transfer and partial file transfer. As for the replica management, Data Grid provides mechanisms to allow users to create, register, locate and manage multiple copies of datasets. With the introduction of replica management, the main components of the Data Grid Architecture are shown in Figure 2.2 [3, 19]. In the following we describe in details the new components introduced by Data Grid at each protocol layer.

At the **Fabric Layer**, in addition to the basic Grid components such as storage systems, network resources and computational resources, two important and fundamental components are introduced by Data Grid: Replica Catalog and Metadata Catalog. The Replica Catalog stores information about the registered replicas and the Metadata Catalog stores descriptive information about the data files.

The **Connectivity Layer** contains the secure, reliable and efficient data transport protocol required by the Data Grid, e.g. the GridFTP protocol. As for protocols supporting replica management, there is no big change in this layer compared to the generic Grid Architecture.

At the **Resource Layer**, the main component related to replica management is the Catalog Management protocol, which provides standards for managing Replica Catalog and Metadata Catalog on one single resource.

At the **Collective Layer**, the Data Grid provides a set of services for managing multiple resources, including Replica Management Services, Replica Selection Services, Metadata Services, Distributed Catalog Services and the basic Grid Information services. The Replica Management Service provides mechanisms to create new copies of partial or complete dataset; register these new copies to

Replica Catalog and allow users to query the Catalog to find replicas of a particular file [19]. The Replica Selection Service provides a general replica selector that can select a particular replica or set of replicas according to the criteria provided by the users. The Metadata Service provides ways to publish and access the metadata, which are descriptive information about the data files [12]. The Distributed Catalog Service provides tools to manage the catalogs distributed across multiple servers. The Information Service is a basic Grid service that enables users to query the status of the shared fabric layer resources, such as workload and network bandwidth.

The **Applications** that will run on Data Grid are mainly data intensive applications, for example Particle Physics application and Climate Modeling application. The applications utilize lower layer functions provided by the Data Grid to process data more efficiently and faster.

2.1.3 Data Grid Services

From the service point of view, the major components of the Data Grid Architecture are summarized in Figure 2.3 [12]. At the bottom of the architecture lies the core Grid services including Resource Management service, Grid Security service and Instrumentation service, which are common services provided in a Generic Grid environment. For Data Grid, the core services also include Storage System and Metadata Repository, which provide low level data management mechanisms. High-level components of the Data Grid include Replica Management service and Replica Selection service, which are described before in the **Collective Layer** services.

Figure 2.3: Major Service Components of the Data Grid Architecture [12]

2.2 Data Grid Projects

A number of Data Grid projects have been developed around the world. Taxonomy of these projects according to various organizational characteristics is given in [29]. One of the interesting characteristics used to categorize Data Grid projects is the Data Grid model, which differentiates the Data Grids according to where the data source is and how the data are distributed in the Data Grid system. There are two popular models used in the existing Data Grid projects: Hierarchical model and Federation model. For the Hierarchical Model, the datasets are from a single source and then distributed to other collaboration sites across the Data Grid. For the Federation model, all the participants are able to share their datasets; and have control over how these datasets can be accessed by other collaborators. In Table 2.1 [29], we categorize the well known Data Grid projects according to the model they used to organize the data.

Table 2.1: Data Grid Projects Categorized by Model [29]

Hierarchical Model	Federation Model	Others
LCG [30-34] EGEE [39-40] GriPhyn [16, 41-42] PPDG [16, 43] Grid3 [44] GridPP [45] Belle Analysis Data Grid [46]	ESG [47-50] BIRN [51] BioGrid Japan [52-54] Virtual Observatories [55-57] eDiaMoND [58]	NEESgrid [59]

2.2.1 Hierarchical Data Grid

One representative Hierarchical Data Grid project is the LHC Computing Grid (LCG) led by CERN, which is a global collaboration of more than 140 computing centers in 34 countries [30]. The LCG project aims to build a storage and analysis infrastructure to distribute the experimental results from the LHC at CERN; and enable the entire High Energy Physics (HEP) community to access and analyze these data. The raw experimental data are to be stored at CERN and parts of the data will be distributed to worldwide regional centers, research institutes and universities. The MONARC [35] architecture group has proposed a multi-tier hierarchical model to provide an efficient and cost-effective method for sharing data, computational and network resources, as shown in Figure 2.4 [37]. The proposed multi-tier model consists of four tiers of Regional Center, which is a “multi-service center which provides significant resources and support” [36], with decreasing levels of complexity and capability. In Figure 2.4 [37], there is one large central computing facility at CERN which sits at tier 0. The raw data generated by the online system are stored here. Tier 1 consists of large regional centers on national scale; and provides expensive compute resource and multi-purpose service. Tier 2 consists of relatively smaller regional centers. Less

expensive compute resources are available here and they are mostly dedicated to data analysis. Below that, there are institute workgroup servers at tier 3. The project of EU DataGrid [38] works on the operations of distributing and transferring data among the tiers. To reduce data access latency and improve network bandwidth utilization, data are replicated from the root center at tier 0 to various regional centers at different tiers.

Figure 2.4: Data Grid Architecture for the LHC Computing Grid [37]

The advantage of the multi-tier Data Grid is evident. The experiment datasets are distributed to the appropriate regional centers and made available for access from multiple sites. Hundreds to thousands of scientists all over the world will be able to access the datasets in a common and efficient way. Physicists worldwide can collaborate and contribute to the analysis of the experimental data, without having to be physically present at CERN. The network bandwidth will be used efficiently because the end-users can download most of the data from regional centers, which are connected with fast local-area network compared to slow wide-area international network. This will improve the user access responsiveness and reduce the network bandwidth cost. With the support of Data

Grid middleware, the resources located in different regional centers, institutes and universities can be utilized as a whole to support data-intensive computing.

The other Data Grid projects which are also using Hierarchical model include:

- Enabling Grids for E-science (EGEE), which is Europe's leading Grid computing project and aims to provide a common Grid computing infrastructure to support various scientific research globally [39];
- Grid Physics Network (GriPhyN) and Particle Physics Data Grid (PPDG), are two major Data Grid projects in the United States, mainly used to support High Energy Physics experiments [16];
- Grid2003 Production Grid (Grid3) project, which aims to provide the infrastructure and platform to support multiple application groups including core participants in GriPhyN [44];
- UK Computing for Particle Physics (GridPP), which is a distributed computing Grid built in UK to collaborate with CERN in particle physics experiments [45];
- Belle Analysis Data Grid, which is a Data Grid testbed set up in Australia, mainly to analyze Belle and ATLAS experiments [46].

2.2.2 Federation Data Grid

For the Federation model, one representative Data Grid project is the Earth System Grid (ESG) project in the United States. ESG aims to use Grid technologies to provide a seamless and powerful environment to analyze the global Earth System models. These models produce outputs in tens of petabytes, which need to be distributed to researchers nationwide to study the global change impacts. The project currently involve several laboratories (ANL, LBNL, LLNL, NCAR, ORNL) and one university (ISI) [47], as shown in Figure 2.5 [50] with their main roles clearly specified. With ESG, climate researchers are able to distribute their model outputs, i.e. climate datasets, and share with other

CHAPTER 2. LITERATURE REVIEW

researchers as community resources [49]. Different from Hierarchical model, which has a single source of data, the Federation model used by ESG enables all climate scientists from the participated research centers to share their datasets.

Other Data Grid projects that are using Federation model include:

- Biomedical Informatics Research Network (BIRN), which aims to build a distributed architecture of shared resources to enable large-scale collaborations in biomedical science [51];
- BioGrid project in Japan, which aims to construct a supercomputer network that integrates observation devices, databases and computational resources specially for biology and medical science [52-54];
- Virtual Observatories project, in which there is an Astronomical Grid Community Research Group working on using Grid technologies to share and analyze astronomical data [57];
- eDiaMoND Grid computing project, which aims to use Grid technologies to enable eHealth by providing fast and reliable sharing and accessing of digitized breast images [58].

Figure 2.5: Earth System Grid (ESG) Collaboration Sites [50]

2.2.3 Other Data Grid Model

In addition to the popular Hierarchical model and Federation model, there are also Data Grid projects using other models. One such example is the NEESgrid (Network for Earthquake Engineering Simulation) [59] or the NEES Cyberinfrastructure [61] in the United States. The project aims to promote collaborations among earthquake scientists, by sharing the earthquake equipments and data for both physical experiments and model-based simulations. Experimental datasets collected from multiple sensors are gathered in a central repository and scientists across the country are able to access the data through a standard interface. The model used by NEESgrid differs from other Data Grid models in that, data from multiple sources are stored in a central repository and there is only a single point for accessing the data [29].

2.3 Replication Strategies for the Data Grid

In Data Grid, scientific experimental datasets are usually replicated to improve the performance of data access; and achieve load balance and fault tolerance. The replication algorithms determine when to do the replication, which data files to replicate and where to replicate them. There are mainly two types of replication strategies in terms of “how frequent the replication process happens”: dynamic or static. For the dynamic replication strategies, the replication process runs after a fixed period of time, or is triggered by changes in the file popularity distribution or storage/bandwidth availability and etc. For the static replication strategies, the replication of data files is done at the beginning and rarely changes over a long period. The advantage of dynamic replication algorithms is that it can adapt to environmental changes and recover from server/network failures fast. They are more suitable to run in a Data Grid environment where users and storage servers may join/leave the Grid on the fly. In an environment where resource conditions are fairly stable over a long time, there is little gain while the network traffic overhead introduced by dynamic replication algorithms is significant. In this case, static replication strategies should be used instead. Some representative replication strategies are listed and compared in [29]. We have extended the list and categorized them according to the replication method used, as shown in Table 2.2 [29].

Table 2.2: Replication Strategies Categorized by Replication Method [29]

Static Replication Strategies	Dynamic Replication Strategies
GDMP [62-65]	K. Ranganathan <i>et al.</i> [76]
Gfarm [68-70]	C. Nicholson <i>et al.</i> [77] and W. Bell <i>et al.</i> [78-79]
SRB [72-74]	M. Tang <i>et al.</i> [81]
	R. Chang <i>et al.</i> [82]
	H. Lamahamedi [83-84]

2.3.1 Static Replication Strategies

The Grid Data Management Pilot (GDMP) project aims to use the Data Grid tools provided by Globus Toolkit to build a replication system that is based on a single middle-ware system [63]. The first production prototype is used by Compact Muon Solenoid (CMS) [66-67] experiment to replicate and transfer object database; and then extended to support file level replication as well [65]. The GDMP is a multi-threaded client-server system which uses a publish-subscription model to manage the data distribution and replication. The GDMP server is a daemon constantly running at the data production site. New data will be added to the replica catalog and published by the server. Clients that have already registered with the server through subscription will receive notification about the new data. They can request to replicate the interested portion of the dataset by setting up a secure connection with the server. The authentication and authorization in GDMP is handled by the Grid Security Infrastructure (GSI) of the Globus. The data transfer is managed by GridFTP at the client side, which can handle failed connections by requesting for a new transfer. In CMS experiments, a single file may contain up to 10^9 objects (events) and that's why GDMP supports both object level and file level replication. The client can download and replicate the complete file or just a partial list of objects. After that, the file will be removed from the source site. In this sense, GDMP is using a static replication strategy as the changing Grid conditions have no effect on the server site; it is up to the client site to decide which files to replicate and when to replicate [29].

Grid Datafarm (Gfarm) architecture is designed to support global petascale data-intensive computing. It targets applications like High Energy Physics, where each data file consists of a set of objects which can be processed independently. There are three major components: *Gfarm filesystem*, *Gfarm process scheduler* and *Gfarm parallel I/O APIs* (Application Programming Interface). In Gfarm, large-scale files are divided into fragments and distributed across server nodes managed by the *Gfarm filesystem*. There are two types of nodes: Gfarm filesystem

CHAPTER 2. LITERATURE REVIEW

nodes and Gfarm metadata server, as shown in Figure 2.6 [69]. Both types of nodes integrate storage space, which is in patascale and with scalable I/O bandwidth, and compute power. The Gfarm filesystem nodes store the physical data files in fragments. The metadata, which consists of mapping from a logical filename to physical file fragments, replica catalog and data information like file size, is stored in the Gfarm metadata nodes. The *Gfarm parallel I/O APIs* provide APIs to access the fragments of a Gfarm file in parallel. The *Gfarm process scheduler* is in charge of scheduling jobs submitted to Gfarm. The built-in scheduling strategy uses the “owner computes” approach or better known as “move the computation to data” approach. For example, in Figure 2.6 [69], “Process A” is scheduled to run on four nodes, where the four fragments of “Data A” are stored. “Process B” is scheduled to run on three nodes, where the three fragments of “Data B” are stored. In the case that the Gfarm filesystem node storing a file fragment is heavily loaded, a new process can be assigned to another node. The new process can either replicate the fragment to the new node or access the fragment remotely. In this way, the replication in Gfarm is not frequently changed and here we categorize it as using static replication strategy. However, it is noted that dynamic replication algorithms are also supported and can be incorporated into the Gfarm system [71].

Figure 2.6: Grid Datafarm Architecture [69]

Storage Resource Broker (SRB) is a middleware that aims to provide a uniform API to access heterogeneous distributed storage resources including file system, databases system and archival storage system [73]. Figure 2.7 [74] shows the three-tier architecture of SRB. At the bottom tier lie the *physical storage resources (PSRs)*, which can be a table in an Oracle [85] database or a directory path in a UNIX [86] filesystem. One or more PSRs can be combined into a single *logical storage resource (LSR)*. Applications typically access LSRs, rather than PSRs. The SRB server lies in the middle-tier, which consists of SRB Master daemon processes and Agent daemon processes. At the top tier are the Application and Metadata Catalog (MCAT). Similar to files organized under directory in a filesystem, data items are organized under collection/sub-collection in SRB. The collections are implemented using LSR, which may contain different types of PSRs. The MCAT contains metadata that describes the collections, data items contained in the collections, PSRs and etc. When the application needs to access LSR, it will send a request to the SRB server. The SRB Master daemon process will check the authentication information and then starts an Agent process to serve the client request. The Agent process interacts with the MCAT and storage resources to finish the request. A federation of SRB servers can be built by interconnecting the servers, where one server acts as a client to another server. SRB is one of the most widely used technologies in Data Grid projects around the world including BIRN, eDiaMoND and IVOA [75]. SRB is categorized here as using Static Replication Strategy, because it focuses on providing a general architecture and relies on additional algorithms to adapt to the changing environment.

Figure 2.7: The Storage Resource Broker (SRB) [74]

2.3.2 Dynamic Replication Strategies

K. Ranganathan *et al.* have studied several simple dynamic replication strategies for the Hierarchical Data Grid in [76]. These strategies include:

- (a). *Best Client*: each node keeps a separate record for every file requested by different clients. At fixed time intervals, the replication process will check the access records on each node and identify the files whose “total number of requests” has exceeded the predefined threshold. For each of these files, a replica will be placed at its best client, which is the client that has generated the most “number of requests” for that file. After a replica is created, the request records for that file will be cleared at the server node and a new recording starts.
- (b). *Cascading Replication*: The replication process checks the “number of requests” for a file tier by tier. If the number exceeds the predefined threshold, a replica will be placed at the next level along the path towards the client.
- (c). *Plain Caching*: The requested file is cached in the client’s local storage after downloading. The performance of caching depends on the storage size

of the client nodes. Small storage size will cause frequent update of the cache contents and thus restricts the performance improvement.

(d). *Caching plus Cascading Replication*: this technique simply combines strategy (b) and (c).

(e). *Fast Spread*: when the requested data file is transferred to client, a replica is placed on each node along the transmission path.

All the above mentioned strategies use the same file replacement strategy: when there is no more space in the storage server, the least popular and oldest file will be deleted. A simulator based on PARSEC [123], which is a basic discrete event simulator used to model events such as file requests and data transfer, is developed to study the performance of these replication strategies. The multi-tier Data Grid environment is simulated with three different kinds of client access patterns. The simulation results show that these replication strategies can significantly reduce both bandwidth consumption and response time. The *Best Client* strategy has the worst performance among all these strategies. The *Cascading* strategy has the best performance when the client access pattern contains a small degree of geographical locality. The *Fast Spread* strategy performs the best when the client access pattern exhibits total randomness.

C. Nicholson *et al.* have studied the performance of several dynamic replication strategies in the simulation environment of LCG 2008. The simulator OptorSim [80] is used to run the simulations, as the current Grids are still not mature enough to run testing of such replication strategies [77]. Two simple aggressive replication strategies [77] and a complex economy-based replication strategy [78-79] are studied. The aggressive replication strategies always try to replicate the requested file. When there is not enough storage, the *Least Recently Used (LRU)* strategy will delete the files that have been used least recently; while the *Least Frequently Used (LFU)* strategy will delete the files that have been used least frequently. The economy-based replication strategy uses an economic model to determine whether replication should occur and which files should be deleted when creating space for

CHAPTER 2. LITERATURE REVIEW

a new replica. In the proposed economic model, data files represent the goods sold in the market. Computing Elements (CEs) purchase them for jobs and Storage Elements (SEs) purchase them for investment. The goal of CEs is to minimize the purchase cost; while SEs try to maximize their profits. When a job running on CE requests a file, the CE will try to buy the cheapest replica from the SEs. SEs who have the replica locally can bid a price indicating the file transfer cost. Those SEs who do not have the replica can start an auction to buy replica from other SEs and then sell to the CE. The simulation results show that all the dynamic replication strategies can improve the Data Grid performance. They are particularly effective when some data locality exists in the data access pattern, such as with a Zipf-like [105-106] distribution. The two simple replication strategies: LRU and LFU are as effective as the complex economy-based strategy in such situations.

M. Tang *et al.* proposed two dynamic replication algorithms: Simple Bottom-Up (SBU) and Aggregate Bottom-Up (ABU) for the Hierarchical Data Grid in [81]. Both algorithms try to replicate the popular files as close as possible to the clients that frequently request them. The replication process checks the data file access history periodically to identify the popular files by comparing their “number of requests” to the predefined threshold. The algorithm tries to replicate the most popular files at server nodes that are nearest to the client. When the storage space is used up, the process will move up to the next level towards the root to replicate the rest of the popular files. In this way, the most popular files are placed closest to the clients, while the not so popular files are placed further from the clients. ABU differs from SBU in the way it handles the file access history. In SBU, the “number of requests” for a file on server node from each client is used to represent the file popularity. While in ABU, the “number of requests” for a file from all the sibling clients of the same server node is aggregated and the merged record is used to represent the file popularity. Due to the tree structure of the Hierarchical Data Grid, the requests from the entire sibling clients will go to the same server node. Thus, ABU can identify the file popularity more accurately. The performance of ABU and SBU are studied through simulations, together with the previously described

CHAPTER 2. LITERATURE REVIEW

Fast Spread (FSP) [76] strategy. The simulated Data Grid environment is constructed according to the estimate for the future Compact Muon Spectrometer (CMS) experiment [66] in 2007. The researchers compared their proposed algorithms with FSP, as well as one static replication algorithm. The performance metrics used include average response time, average bandwidth cost and replication frequency. Various simulations are conducted with different client access patterns. The simulation results show that the dynamic replication algorithms outperform the static replication algorithm in reducing both bandwidth consumption and average response time. ABU has better performance over SBU and FSP in all the simulated scenarios. Although FSP outperforms SBU in terms of average response time, its replication frequency is too high to be applicable in real Data Grid.

R. Chang *et al.* proposed a dynamic replication mechanism called Latest Access Largest Weight (LALW) in [82]. LALW is similar to ABU in the way that it also tries to use replica access records to identify the popular data files and replicates them close to the clients. However, LALW gives different weights to the access records and the latest access records have the largest weight. This is to solve the problem of replicating an old popular file, which has a large “number of total requests” but very few recent requests. LALW adopts the half-life concept to weight the records, where the weight of one access record will be reduced by half after a fixed period. In this way, the weight of an access record will decrease with time and thus achieve the goal of “latest record having largest weight”. The performance of LALW is studied using the Grid simulator OptorSim. The simulation results show that LALW can effectively improve the Data Grid performance without increasing the network burden too much.

H. Lamahmedi *et al.* proposed a dynamic replication strategy that can be used in different Data Grid topologies including hierarchical, ring and hybrid in [83-84]. The proposed strategy tries to perform the replica placement in the way that minimizes the sum of data access cost and replica creation/maintenance cost. It

uses a cost-estimation model to estimate the replica's creation and maintenance cost, as well as the data access gains by creating a new replica. Factors that affect the model include network latency, bandwidth, replica size and etc. The replica manager determines when to create and remove replicas, basing on the statistic data collected from all sites including data access requests, connection bandwidth and storage space availability. A Data Grid simulator called the GridNet is developed to study the performance of the proposed approach. The simulation results show that the data access performance is improved dramatically with the proposed replication strategy and the gain increases with dataset size.

2.4 Data Consistency Management

Data Grid applications, such as High Energy Physics, usually deal with read-only data, which is never updated. The data consistency is not an issue as the replicas are always up-to-date. With the applications supported by the Data Grid keep increasing, there are some applications require the data to be updated from time to time. Thus, it is essential to maintain the consistency of these modifiable data. When one replica is updated, all the other replicas need to be synchronized to have the same contents and to provide a consistent view.

In [87], two categories of replica consistency management schemes are identified: *Eager replication* and *Lazy replication*. Eager replication keeps all the replicas synchronized at all nodes all the time. Whenever the data is updated, all its replicas will be updated immediately, before clients can make any request to them. This is usually implemented using a locking scheme to detect and regulate concurrent operations on the replicas. Eager replication maintains strict synchronization among replicas, at the risk of increasing network burden with huge number of update operations, especially when large data is involved. Lazy replication propagates updates asynchronously to other nodes after the data is modified. It allows replicas to be not synchronized for a period of time. Lazy replication scheme can significantly improve the situation where the network traffic is swamped by Eager replication. However it leaves the clients with the risk of

CHAPTER 2. LITERATURE REVIEW

accessing an outdated replica. As we can see, both Eager replication and Lazy replication have their advantages and disadvantages. Users need to trade-off between network resource efficiency and data accuracy.

Researchers from EU DataGrid project have looked into the replica consistency management issues and proposed a new Grid service, called Replica Consistency Service (RCS) to manage the synchronization of replica updates [89-91]. Data replication issues in distributed database management systems and the Data Grid are compared in details in [88]. In [89], some fundamental differences are identified:

- In Data Grid, data files are distributed and replicated in wide-area network environment. Propagating data updates will introduce a longer delay, compared to database systems, which usually reside in local-area network.
- Most of the data is read-only for Data Grid applications like HEP. A generic mechanism to propagate data updates is not always needed.
- In Data Grid, the replica management service sits on top of heterogeneous storage resources to serve various communities, as shown in Figure 2.3 [12]. The consistency management in Data Grid should also be general enough to support other different storage systems, besides database management systems.

Thus, models used by real-time database applications [92] cannot be simply applied to Data Grid. Instead, the general Grid service RCS is proposed in [89] to sit on top of existing Data Grid services and fit into the Data Grid architecture, as shown in Figure 2.8 [89]. As there are different consistency requirements for various applications, ranging from full synchronization to loose synchronization; RCS supports a set of models for replica synchronization with different consistency levels. The design principles and details of RCS are reported in [93]. One implementation of RCS, that supports replica updates in a single-master scenario with lazy update synchronization, is presented in [91]. There are two

CHAPTER 2. LITERATURE REVIEW

different types of replicas in the proposed system: *master replica* and *secondary replica*. The master replica is the replica that can be modified by the end-users and there is only one master replica for each data file. All the other replicas for this data file are secondary replicas, which are read-only to end-users. When the end-user modifies the master replica, the consistency service will propagate the updates to all the secondary replicas asynchronously with certain delay. The proposed consistency service supports both relational databases and flat files.

Figure 2.8: Replication Architecture including the Consistency Service [89]

Two coherence protocols: lazy-copy and aggressive-copy are proposed in [94] to keep replicas consistent. The single-master scenario is also used here, where the end-users can only update the single master replica of the data, i.e. the original source. The fundamental difference between the two coherence protocols lies at “when to update/synchronize the replicas”. The lazy-copy protocol only propagates the updates from the original source when clients access the replica. On the other hand, the aggressive-copy protocol will propagate the updates to all the replicas once the original source is updated. The aggressive-copy protocol updates the replicas more frequently and rapidly; and the users will not need to wait for the replicas to be updated at the access time. However, the bandwidth resource is wasted if some of the updates are never used by any user. Thus, tradeoff must be made between network resource efficiency and data access delay time. Using High Energy Physics applications and Climate Modeling applications as examples, the

CHAPTER 2. LITERATURE REVIEW

authors suggest that lazy-copy coherence protocol is more suitable for experimental data management, while the aggressive-copy coherence protocol is more suitable for metadata management. The main reason is that experimental data are much larger in size and less frequently accessed, compared to the metadata.

In [95], the Adaptable Replica Consistency Service (ARCS) is proposed to deal with the replica consistency and synchronization problem. ARCS adopts the multi-master model, where multiple master replicas are kept to avoid having a single point of failure, in contrast to the single-master model used in [91, 94]. These master replicas are distributed over different Grid regions; and each region has at most one master replica and unlimited number of secondary replicas. Each region has one region server that manages the consistency within the region and the regions are connected to each other for global consistency communication. For the master replicas, full consistency service is used, i.e. when one master replica is modified, the updates will be propagated immediately to all the master replicas in other regions. Locking mechanism is used to regulate concurrent access to these master replicas. For the secondary replicas, ARCS uses an access-weight based approach to propagate the updates from master replicas. The access weight is associated with the access frequency of a secondary replica within a Grid region, where the region server keeps record of the number of access for each secondary replica within it. Full consistency service is applied to the frequently accessed data, while partial consistency service is applied to the seldom-accessed data. The full consistency service is similar to the aggressive-copy protocol and partial consistency service to the lazy-copy protocol in [91]. The proposed ARCS service is designed to be applicable to the ecological observatory project (Ecogrid) [96] in Taiwan, by providing full consistency service to popular data and partial consistency service to infrequently accessed data.

2.5 Data Replication and Job Scheduling

Data-intensive applications need to process large amount of data, which are replicated across the Data Grid. Scheduling such applications needs to take into consideration the data locations, as fetching large dataset from remote sites will introduce a long delay to the job turnaround time. The data locations, on the other hand, can be changed by the replication strategy. In this way, scheduling algorithms and data replication algorithms are closely related in the Data Grid environment. A survey on the scheduling algorithms for Grid Computing is given in [97]. Two types of scheduling algorithms are identified with consideration of data replication: *decoupled approach* and *integrated approach*. In the decoupled approach, the replication and scheduling algorithms are addressed separately, with the scheduling algorithms focusing on scheduling jobs to compute sites and the replication algorithms focusing on replicating data to storage servers. This approach can simplify the design and implementation of the overall Data Grid System, making it easier to model the system. On the contrary, the integrated approach tries to merge Job Scheduling and Data Replication algorithm to improve the overall performance, emphasizing their interactions.

K. Ranganathan *et al.* pointed out the importance of taking data locations into account while scheduling job placement in [98-99]. A scheduling framework that supports different scheduling and replication algorithms is proposed. The framework composes of three main components: External Scheduler (ES), Local Scheduler (LS) and Dataset Scheduler (DS). ES is responsible for dispatching jobs to different compute sites according to the scheduling algorithm; LS at each compute site handles the execution of jobs assigned to it by ES; and DS is in charge of replicating data files across various storage sites according to the data replication algorithm used. The studied scheduling algorithms include *JobRandom*, which schedules jobs to a randomly selected site; *JobLeastLoaded*, which schedules jobs to the site that has the least load; *JobDataPresent*, which schedules jobs to the site that already has the required data and chooses the least loaded one

CHAPTER 2. LITERATURE REVIEW

when multiple sites have the data; and *JobLocal*, which always runs the jobs locally. The data replication algorithms used by DS include *DataDoNothing*, where no active replication takes place and only caching is performed; *DataRandom*, where popular datasets are replicated to a random site; and *DataLeastLoaded*, where the popular datasets are replicated to the least loaded site. The ChicSim simulator is developed and used to evaluate the performance of various combinations of these replication and scheduling algorithms. The simulation results show that scheduling jobs to sites containing the required input data, i.e. the *JobDataPresent* strategy, and asynchronously replicating popular data across sites, i.e. *DataRandom* or *DataLeastLoaded*, can improve the system performance in terms of both system utilization and response time.

In [71], A. Takefusa *et al.* investigate and compare the combined performance of various scheduling and replication strategies under different Data Grid system models, representatively the central model and the hierarchical model. The studied central model has centralized data storage and processing power, while the multi-tier model proposed by MONARC is studied as a representative hierarchical model. For the central model, jobs are scheduled to run in the central compute server and pull the required input data from the central storage server. For the multi-tier model, the combination of four scheduling algorithms (*OwnerComputes*, *Greedy*, *LoadBound-Read* and *LoadBound-Write*) and three background replication policies (*LoadBound-Replication*, *Aggressive-Replication* and *Do Nothing*) are studied. Different classes of jobs (large, medium and small), as modeled using the observed data from LHC experiments at CERN, are simulated to run on the Grid Datafarm system using the enhanced Bricks [103] Grid simulator. The simulation results show that although the central model is efficient; the multi-tier model can achieve better performance with suitable combination of scheduling and replication algorithm, given that each tier is smaller than the central model.

In [100], D. G. Cameron *et al.* use simulations to study the performances of replica

optimization strategies and job scheduling strategies in the European Datagrid project [38]. The studied model consists of several sites, with each site having a Computing Element (CE) and/or Storage Element (SE). The Resource Broker controls the scheduling of jobs to CEs using scheduling algorithms including *Random*, which schedules jobs to a randomly chosen CE; *Shortest Queue*, which schedules jobs to the CE with the shortest job queue; *Access Cost*, which schedules jobs to the CE that gives the lowest file access cost; and *Queue Access Cost*, which schedules jobs to the CE that gives the lowest sum of access cost for the job itself and all jobs in the queue. The Replica Manager at each site handles the replica selection, creation and deletion. The studied replication strategies include one simple strategy: *Least Frequently Used (LFU)*, which always replicates job data file to the SE and deletes the least frequently used data when there is insufficient space; and two economy-based strategies using different prediction functions: binomial-based [78, 104] and Zipf-based [105-106]. Performances of these algorithms are evaluated using the OptorSim [80] simulator. The simulations are based on the worldwide Grid testbed of the DataGrid project, and realistic factors such as background network traffic are taken into consideration. Performance metrics including job execution time, network usage, node resource utilization and etc. are studied. The simulation results show that scheduling algorithms that consider both the file access cost and the compute resource workload are most effective at improving the resource utilization and the job throughput.

In [101], M. Tang *et al.* propose several replication algorithms and evaluate their impact on the performance of heuristic scheduling algorithms in the Data Grid environment. The studied Data Grid consists of a set of domains, with each domain containing one storage server and zero to many compute servers. The replica server serves as Primary Replica Server (PRS) to the compute sites in the same domain and as Secondary Replica Server (SRS) to the other compute sites. Servers within the same domain are connected with much higher bandwidth compared to the inter-domain connection. The Grid Scheduler assigns jobs to the compute servers according to the scheduling strategy. Heuristic scheduling algorithms are used

including *Shortest Turnaround Time (STT)*, which schedules jobs to the compute server that gives the smallest estimated job turnaround time; *Least Relative Load (LRL)*, which schedules jobs to the compute server that has the least relative workload; *Data Present (DP)*, which tries to schedule jobs to the compute server that have the required data in its cache or in its PRS. Two centralized replication algorithms with different replica placement strategies: *CDR_RTPlace* and *CDR_SMPlace*, and one distributed dynamic replication algorithm: *DDR*, are proposed. The performance of these replication algorithms, together with the three scheduling algorithms, is compared using the XDRepSim simulator. The performance metric used is the Geometric Mean of job Turnaround Time (GMTT). The simulation results show that the proposed dynamic replication algorithms can help to improve the job scheduling performance remarkably. In particular, the combination of STT and CDR_RTPlace performs remarkably in different simulation environments with various job workloads.

In [102], R. Chang *et al.* propose a job scheduling policy, called Hierarchical Cluster Scheduling (HCS) and a dynamic data replication strategy, called Hierarchical Replication Strategy (HRS) to improve the performance of Cluster Grid. Since connections within the same cluster are much faster than the inter-cluster connections, HCS treats the whole cluster as one entity and tries to schedule jobs to the cluster that already have the required input data in the storage servers. In this way, only fast intra-cluster communications are needed and slow inter-cluster communications are avoided. At the same time, HRS tries to replicate popular data files in each cluster by removing duplicate replicas in the same cluster. It uses the concept of “network locality” as in Bandwidth Hierarchy based Replication (BHR) [107] algorithm. The performance of HCS and HRS are studied using both Optorsim simulator and the Taiwan UniGrid [108] simulator, with various access patterns. The simulation results show that HCS and HRS can significantly reduce the data access time, as well as number of inter-cluster communications, compared to other strategies used in Cluster Grid.

Chapter 3

Modeling the Optimal Performance of the Hierarchical Data Grid

3.1 Introduction

Hierarchical Data Grid is a class of Data Grid that has been adopted by CERN [10] to support the distribution of large experimental datasets across the globe. There have been a lot of research works on replication algorithms for the Hierarchical Data Grid. However, the performances of these replication algorithms are only studied using simulations and there is no work on predicting the theoretical optimal performance of the Hierarchical Data Grid. It is important and useful to analyze the optimal performance, as it can be used as a benchmark to compare the performances of different replication algorithms and helps us to understand how close the replication algorithms are approaching the optimal performance.

In this chapter, we construct a mathematical model to predict the optimal performance of the Hierarchical Data Grid. The performance metrics used for the model are average response time and average bandwidth cost. We propose the

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

optimal replication algorithm and present the detailed analysis of its performance. The multi-tier Data Grid is used as a specific example of the Hierarchical Data Grid to verify our model. Three well-known replication algorithms: Simple Bottom-Up (SBU) [81], Aggregate Bottom-Up (ABU) [81] and Fast Spread (FSP) [76] are studied here; and their performances are compared with the optimal performance obtained from the model.

The rest of the chapter is organized as follows. In section 3.2, we report some related research work on replication strategy in Hierarchical Data Grid. Section 3.3 describes how we model the Hierarchical Data Grid with some realistic assumptions and lists the general model parameters. Section 3.4 introduces the performance metrics used in the model and our proposed optimal replication algorithm. Detailed performance analysis of the proposed replication algorithm using the model is also provided. In section 3.5, we use the multi-tier Data Grid as a specific example of the Hierarchical Data Grid to verify our model. Three well-known dynamic replication algorithms are described here; and their performances are simulated and compared with the proposed optimal replication algorithm. Section 3.6 summarizes the chapter.

3.2 Related Works

The European Data Grid project [38] has chosen to use the Hierarchical Data Grid model and done a lot of research work on replica management [77-79, 109-110]. A Grid simulator named OptorSim [80] is developed to study the replica optimization strategies in the Data Grid environment. Various replication strategies are studied with different Data Grid configurations. In particular, an economy-based file replication strategy [78] is proposed to optimize both replica selection for job running and dynamic creation of replicas in Grid sites. Both data locality and network latency are considered for dynamically selecting the best replica. Historical data about file access pattern is analyzed to help make decisions of creating new replicas. The simulations are based on some real experimental data from the CMS testbed and simulated the Grid topology for

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

CMS world wide data production challenge in spring 2002 [111].

As already described in section 2.3.2, other researchers have also proposed various replication algorithms [76, 81-82] to improve the performance of the Hierarchical Data Grid. Simulations are used to compare these algorithms' performance under different Grid environments. However there is no theoretical model to predict the optimal performance of the Hierarchical Data Grid. In our research, we propose to build an analytical model to predict the best performance of the Hierarchical Data Grid, which can serve as the benchmark for performance comparison among different replication algorithms.

The optimal replica placement problem for the Hierarchical Data Grid is studied in [112-113]. The authors address the issues of finding the locations to place the replicas and choosing the optimal number of replicas. Keeping a large number of replicas will reduce the client access cost; but will increase the storage cost of the replica servers in the Data Grid. The authors propose an efficient algorithm for selecting optimal locations to place the replicas, so that the workload among the replicas is balanced. The proposed algorithm can determine the optimal number of replicas required and their optimal locations, given the data usage from each user site and the maximum workload allowed for each replica server.

There are a lot of research works on modeling the job arrival process and workloads in the Grid Environment [114-119]. The authors in [114-115] report their analysis of the job arrival process in the worldwide LHC Computing Grid (LCG) project. Several traffic models are investigated for modeling the inter-arrival time processes. The Grid Workloads Archive (GWA) [116] is proposed to enable storing and exchanging workload data. Real job data is collected and analyzed from several well-known Grid environments. The authors in [117] present a comprehensive statistical analysis of a variety of workloads, which are collected from production Grids, e.g. Enabling Grids for E-science (EGEE) project [39]. The statistical properties of these job workloads are analyzed at different levels, including Virtual Organization (VO) level, as well as

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

user level. The authors in [118] aim to build a probabilistic model for the job arrival process and the delay components introduced at different stages of job processing in a Grid environment. The job arrival process in the EGEE infrastructure is used as an example for the model analysis. These research works lay the foundation for our model; and enable us to build the model to predict the optimal performance of the Data Grid, safely assuming that the job access pattern is given or can be predicted.

The authors in [120] discuss the importance of analytical model to evaluate the system performance and present an analytical model to study the performance of fat-tree based interconnection networks for multi-cluster computing systems in a multi-user environment. The model uses probabilistic analysis and queuing network to evaluate the performance of the interconnection networks for the multi-cluster system. The model is validated through simulations. In this chapter, we present a probabilistic analytical model to study the performance of the Hierarchical Data Grid.

3.3 Modeling the Hierarchical Data Grid

Figure 3.1 shows the architecture of the modeled Hierarchical Data Grid. The model parameters are summarized in Table 3.1.

There are a total of $C+1$ tiers in the modeled Data Grid. Tier 0 is the tier where the root node resides. The root node is the original source of the data files and has enough storage to store all the data files. Tier C is the tier that contains all the client nodes. The client node is designed as compute resource and only has enough storage for one data file. There are T_c client nodes in total. Tiers 1 to $C-1$ are the middle-tiers, where the storage servers reside. The storage servers have limited storage capacity to store portion of the replicas. There are T_s middle-tier storage nodes in total. The total number of nodes in the system is denoted as T_n , where $T_n = 1 + T_c + T_s$.

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

Figure 3.1: Modeling the Hierarchical Data Grid

All the data files are originally generated and stored at the root node. Replicas are created in the middle-tier nodes to improve the system performance. The replication algorithm determines when to create replicas, which data files to replicate and where to store the replicas. The end-users generate requests to download files that they are interested in from time to time. All the end-users are associated with nodes at client tier and thus all the file download requests are generated only from client-tier nodes.

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

Table 3.1: Hierarchical Data Grid Model Parameters and Notations

Parameter	Meaning
C	number of tiers (excluding tier 0)
F	size of the replica file
T_r	total number of different data files in the system
T_n	total number of nodes in the system
T_s	total number of middle-tier storage servers in the system
RS	set of middle-tier storage servers
$T(j)$	tier no. of middle-tier storage server j ($j \in RS$); and $0 < T(j) < C$
T_c	total number of client nodes in the system
CS	set of client nodes
f_c	frequency of download requests from client node c ($c \in CS$)
S	total storage size of the middle-tier servers
D	total size of all files in the system
r	server relative capacity
U_i	bandwidth capacity of link between tier i and tier $i+1$ ($0 \leq i < C$)
W_i	bandwidth cost for link between tier i and tier $i+1$ ($0 \leq i < C$)
B_i	bandwidth cost for network connection between tier i and client tier ($0 \leq i < C$)
M_i	maximum number of files that can be stored at one tier- i server ($0 \leq i < C$)
Q_i	the number of descendant client nodes for each server in tier i ($0 \leq i < C$)
DC_j	the set of descendant client nodes for storage server j ($j \in RS$)
DS_j	The set of direct children nodes of storage server j ($j \in RS$)
$CD_c(x)$	probability distribution function of download requests for file x ($x=1,2,\dots,T_r$) from client node c ($c \in CS$)
N_r	total number of client download requests

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

Due to the tree structure of the Hierarchical Data Grid, each node is only directly connected to its parent node. When a client requests to download a file f_1 , it will send the request to its direct parent node. If file f_1 is found, it will be downloaded to the client from that node. Otherwise, the request will go up another tier to the direct parent of the current node. The process continues until the replica of file f_1 is found and the file will then be downloaded to the client tier. Since the root node has a copy of all the data files and is reachable by all the client nodes, it will ensure that at least one replica of file f_1 is available to serve the download request. For the Hierarchical Data Grid example shown in Figure 3.2, the requested file f_1 is available at root node with id 0 and middle-tier node with id 4. When client node with id 5 sends a download request, the file will be downloaded from root node. While for a request sent by client node with id 7, the file will be downloaded from middle-tier node with id 4.

Figure 3.2: A Hierarchical Data Grid Example

The storage capacity of the middle-tier replica servers has a major impact on the Data Grid performance. With larger storage capacities, the replica servers can replicate more data files in the middle tier resulting in better performance. The relative capacity of the replica servers is defined as:

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

$$r = S/D \times 100\% \quad (3.1)$$

where S is the total storage size of the middle-tier storage servers; D is the total size of all data files in the system.

To simplify the model, all the files are assumed to have the same size F in this model. There are a total of T_r different files in the Data Grid. Each client node c sends download requests at the frequency of f_c , and the file popularity distribution for client node c is modeled as $CD_c(x)$ ($x=1,2,\dots,T_r$). We assume that all the storage servers at the same tier have the same configuration, e.g. have the same storage size and have the same number of children nodes. Each storage server at middle-tier i ($0 \leq i < C$) has the capacity to store M_i number of replicas; and serves Q_i number of descendant client nodes.

For the sake of simplicity, we assume that all the nodes in the same tier have the same network connection capacity with its direct parent/child node. Thus, the network bandwidth capacity for the link between tier i ($0 \leq i < C$) node and tier $i+1$ node can be modeled as U_i . To model the bandwidth cost, a cost is associated with each network connection. The bandwidth cost is W_i for the network link between tier i ($0 \leq i < C$) and tier $i+1$; and B_i between tier i ($0 \leq i < C$) and tier C . Thus,

$$B_i = \sum_{i=0}^{C-1} W_i.$$

3.4 Performance Analysis

3.4.1 Performance Metric:

The performance metrics, used in our research to compare the system performance, are the average response time and the average bandwidth cost. The average response time is the mean value of the response time for all the replica requests from the client nodes. The response time for each replica request is defined as:

$$T_{res} = T_{fin} - T_{req} \quad (3.2)$$

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

where T_{fin} is the time-stamp when the entire file is loaded into the client node; T_{req} is the time-stamp when the client node requests the file. The average response time is related to the user satisfaction, the lower the average response time the higher would be the user satisfaction.

The second performance metric used in our study is the average bandwidth cost, which is the mean value of the bandwidth cost for all data transmission. The bandwidth cost is affected by how far up the tier a client request needs to go to retrieve the replica. As we have put a cost weight on the link at different tiers, the cost of a client request to download a file from tier i ($0 \leq i < C$) is given as follows:

$$\begin{aligned} BC_i &= F \times B_i \\ &= F \times \sum_{i=0}^{C-1} W_i \end{aligned} \quad (3.3)$$

where F is the size of the replica file; B_i is the bandwidth cost for network connection between tier i and client-tier; W_i is the bandwidth cost for link between tier i and tier $i+1$.

For example, as shown in Figure 3.2, if client node with id 5 downloads a file from root node with id 0 at tier 0, the transmission path is node 0 -> node 1 -> node 3 -> node 5. Assume that the file size is 1 GByte. The link cost between tier 0 and tier 1 is 5 per MByte; that between tier 1 and tier 2 is 2 per MByte; and that between tier 2 and tier 3 is 0. Then, the bandwidth cost for this data transmission is $1024 \times (5+2+0) = 7168$.

3.4.2 The OPTimal Replication Algorithm (OPT)

We first show that downloading a replica from the lower tier, which is closer to the clients, will lead to a shorter response time and bandwidth cost. For the bandwidth cost, $BC_i \geq BC_{i+1}$ for $i < C$, since $B_i = B_{i+1} + W_i$. For the response time, downloading from upper tier will introduce an additional delay compared to downloading from lower tier.

By replicating the most popular files near the client nodes, the most frequent

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

download requests will be served by the lower tiers. Thus, the average response time and average bandwidth cost are minimized statistically. We propose the following optimal replication strategy, with the assumption that we know the client access pattern or are able to predict the access pattern in advance.

We replicate the files from middle tier C-1 (closest to the client tier) towards root tier (tier 0) according to the file popularity. The most popular files are replicated to tier C-1 nodes until the node storage is used up. We then move on to the next higher tier C-2 to fill all the node storage with files according to the popularity. The process is continued until the root node is reached. In this manner, the most popular files are replicated near the client nodes. The following of this section shows how the algorithm chooses the set of files to be replicated on each middle-tier storage server.

Due to the tree structure of the Hierarchical Data Grid, each client node c will only be able to download files from its parent node and it only has one parent node in each middle-tier. We use $S(c,i)$ to denote the middle-tier storage node in tier i ($0 < i < C$) that serves download requests from client node c ($c \in CS$). For each middle-tier storage node j ($j \in RS$), the tier that it resides in is denoted as $T(j)$. It serves $Q_{T(j)}$ descending client nodes. Let DC_j denote the set of all these $Q_{T(j)}$ client nodes.

Each client node c ($c \in CS$) sends download requests at frequency of f_c and the file popular follows distribution of $CD_c(x)$ ($x=1,2,\dots,T_r$). Thus, the distribution of aggregated file download requests for storage node j ($j \in RS$) follows:

$$FD_j(x) = \frac{\sum_{c \in DC_j} (CD_c(x) \times F_{c,T(j)}(x) \times f_c)}{\sum_{c \in DC_j} f_c} \quad (j \in RS, \text{ and } x=1,2,\dots,T_r) \quad (3.4)$$

where $F_{c,i}(x)$ is a 0-1 variable indicating whether download requests from client node c for file x reach its parent storage node in tier i .

Let $TF_j(n)$ denote the set of top n most popular files for the file request of

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

distribution $FD_j(x)$. FS_j denotes the set of files that will be replicated on server node j . Since each middle-tier storage node j at tier $T(j)$ can store a maximum of $M_{T(j)}$ files; we will choose $M_{T(j)}$ most popular files from the distribution of $FD_j(x)$. Thus, we have:

$$FS_j = TF_j(M_{T(j)}) \quad (j \in RS) \quad (3.5)$$

The download requests from client nodes will always be able to reach tier $C-1$, which is the tier directly above the client nodes. Thus, we have:

$$F_{c,C-1}(x) = 1 \quad (c \in CS \text{ and } x=1,2,\dots,T_r) \quad (3.6a)$$

For each middle-tier i ($0 < i < C$), client node c will only be able to download file from its parent node $S(c,i)$ in this tier. The download requests will only be able to reach tier i if the requested file x ($x=1,2,\dots,T_r$) is not available in the lower tiers, i.e. tier $i+1$ to tier $C-1$, along the download path of client node c . Thus, we have:

$$F_{c,i}(x) = \begin{cases} 0 & \text{if } x \in \bigcup_{k=i+1}^{C-1} FS_{S(c,k)} \\ 1 & \text{if } x \notin \bigcup_{k=i+1}^{C-1} FS_{S(c,k)} \end{cases} \quad (c \in CS, 0 < i < C-1 \text{ and } x=1,2,\dots,T_r) \quad (3.6b)$$

The replication algorithm runs from tier $C-1$ towards root tier (tier 0) and the set of files to be replicated on tier $C-1$ nodes can be determined using equation (3.4), (3.5) and (3.6a). The set of files to be replicated on the rest of the middle-tier nodes can be decided using equation (3.4), (3.5) and (3.6b) recursively.

Our proposed algorithm is a static replication algorithm. When the system detects changes of client access pattern, it will need to invoke the algorithm again to dynamically adapt to the client changes. However, in our research, we aim to build a model to predict the upper bound performance of the Hierarchical Data Grid. Thus, the cost introduced by dynamic adaptation is not included.

3.4.3 Optimal Performance Analysis

In this section, we use the model to predict the performance of our proposed Optimal Replication Algorithm (OPT), as described in Section 3.4.2. Some of the

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

main notations used in the analysis are summarized in Table 3.2.

Table 3.2: Notations Used in the Model Analysis

$S(c,i)$	the middle-tier storage server in tier i ($0 < i < C$) that serves download requests from client node c ($c \in CS$)
$F_{c,i}(x)$	0-1 variable indicating whether download requests from client node c ($c \in CS$) for file x reach its parent storage server in tier i ($0 < i < C$ and $x=1,2,\dots,T_r$)
$FD_j(x)$	the distribution of aggregated file download requests for middle-tier storage server node j ($j \in RS$ and $x=1,2,\dots,T_r$)
$TF_j(n)$	the set of top n most popular files for the file request of distribution $FD_j(x)$ ($x=1,2,\dots,T_r$ and $n=1,2,\dots,T_r$)
FS_j	the set of files that is replicated on server node j ($j \in RS$)
$P(c,i)$	the probability that client node c ($c \in CS$) download replica from tier i ($0 \leq i < C$)
P_i	the probability that the replica is downloaded from tier i ($0 \leq i < C$)
N_i	average number of concurrent transferring tasks sharing the link from tier i to tier $i+1$ ($0 \leq i < C$)
L_i	average available transmission bandwidth from tier i to tier $i+1$ ($0 \leq i < C$)
H_i	average available bandwidth from client tier C to tier i ($0 \leq i < C$)
R_i	response time to download the replica from tier i ($0 \leq i < C$)
E_{cost}	the average bandwidth cost for the system
E_{delay}	the average response time for the system

For tiers other than root tier, the replica will only be downloaded from tier i ($0 < i \leq C-1$) if it satisfies the following conditions:

- (i). Client node c ($c \in CS$) can only download replicas from its parent storage node $S(c,i)$ in tier i ,

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

- (ii). The client node cannot find the replica at lower tiers, i.e. tier $i+1$ to tier $C-1$, along its download path,
- (iii). The replica exists in tier i .

Since tier $C-1$ is the tier that is directly above the client nodes, it always satisfies condition (ii). Thus, according to condition (i) and (iii), the probability that client node c download requested file from tier $C-1$ is:

$$P(c, C-1) = \sum_{x \in (FS_{S(c, C-1)})} CD_c(x) \quad (c \in CS \text{ and } x=1, 2, \dots, T_f) \quad (3.7a)$$

where $S(c, C-1)$ is the middle-tier storage node in tier $C-1$ that serves download requests from client node c ; $FS_{S(c, C-1)}$ denotes the set of files that is replicated on server node $S(c, C-1)$; and $CD_c(x)$ is the probability distribution function of download requests for file x from client node c .

According to condition (i), (ii) and (iii), the probability that client node c download requested file from tier i ($0 < i < C-1$) is:

$$P(c, i) = \sum_{x \in (FS_{S(c, i)} - \bigcup_{k=i+1}^{C-1} FS_{S(c, k)})} CD_c(x) \quad (c \in CS; 0 < i < C-1 \text{ and } x=1, 2, \dots, T_f) \quad (3.7b)$$

where $S(c, i)$ is the middle-tier storage node in tier i that serves download requests from client node c ; FS_j denotes the set of files that is replicated on server node j ; and $CD_c(x)$ is the probability distribution function of download requests for file x from client node c .

Thus, the probability that the replica is downloaded from tier i for all client nodes is:

$$P_i = \frac{\sum_{c \in CS} (P(c, i) \times f_c)}{\sum_{c \in CS} f_c} \quad (0 < i < C) \quad (3.8)$$

where CS is the set of client nodes; and f_c is the frequency of download requests from client node c .

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

The file will be downloaded from the root node if it is not found in any of the middle-tier nodes:

$$P_0 = 1 - \sum_{i=1}^{C-1} P_i \quad (3.9)$$

The average bandwidth cost as discussed in section 3.4.1 is thus:

$$E_{\text{cost}} = F \times \sum_{i=0}^{C-1} (B_i \times P_i) \quad (3.10)$$

where F is the size of the replica file and B_i is the bandwidth cost for network connection between tier i and client tier.

The network connection between client-tier node and its parent node at tier $C-1$ is not shared, so the average available bandwidth from client tier C to tier $C-1$ is:

$$L_{C-1} = U_{C-1} \quad (3.11)$$

where U_{C-1} denotes the bandwidth capacity of the link between client tier C and tier $C-1$.

For the other tiers, the link capacity between tier i and tier $i+1$ is U_i ($0 \leq i < C-1$) and it is shared by N_i concurrent download requests. As each client node c sends requests at frequency of f_c and contributes $f_c \times E_{\text{delay}}$ requests to the traffic during the period of E_{delay} . Thus, the average number of requests from one client node during period of E_{delay} is:

$$A = \frac{\sum_{c \in CS} f_c}{T_c} \times E_{\text{delay}} \quad (3.12)$$

where CS denotes the set of client nodes and T_c is the total number of client nodes in the system.

Each node at tier $i+1$ serves requests from Q_{i+1} number of client nodes. During the period of E_{delay} , there will be $(A \times Q_{i+1})$ requests from these Q_{i+1} client nodes. The probability of these requests going to tier i or upper tier is $\sum_{k=0}^i P_k$. Thus, the average

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

number of concurrent transferring tasks sharing the link from tier i to tier $i+1$ is:

$$N_i = Q_{i+1} \times A \times \sum_{k=0}^i P_k \quad (0 \leq i < C-1) \quad (3.13)$$

These N_i requests share a single link between tier i and tier $i+1$. Thus, the average available transmission bandwidth from tier i to tier $i+1$ is:

$$L_i = \frac{U_i}{N_i} = \frac{U_i}{Q_{i+1} \times A \times \sum_{k=0}^i P_k} \quad (0 \leq i < C-1) \quad (3.14)$$

When a replica is transferred from tier i to node at client tier C , the available bandwidth is the minimum bandwidth along the transmission path. It is obvious that the average available bandwidth from client tier C to tier i is:

$$\begin{aligned} H_i &= \min (L_{C-1}, L_{C-2} \dots L_{i+1}, L_i) \\ &= \min \left(U_{C-1}, \frac{U_{C-2}}{Q_{C-1} \times A \times \sum_{k=0}^{C-2} P_k} \dots \frac{U_{i+1}}{Q_{i+2} \times A \times \sum_{k=0}^{i+1} P_k}, \frac{U_i}{Q_{i+1} \times A \times \sum_{k=0}^i P_k} \right) \quad (0 \leq i < C) \end{aligned} \quad (3.15)$$

The transmission time from tier i to client tier is:

$$\begin{aligned} R_i &= \frac{F}{H_i} \\ &= F \times \max \left(\frac{1}{U_{C-1}}, \frac{Q_{C-1} \times A \times \sum_{k=0}^{C-2} P_k}{U_{C-2}} \dots \frac{Q_{i+2} \times A \times \sum_{k=0}^{i+1} P_k}{U_{i+1}}, \frac{Q_{i+1} \times A \times \sum_{k=0}^i P_k}{U_i} \right) \quad (0 \leq i < C) \end{aligned} \quad (3.16)$$

The average response time as specified in section 3.4.1 is:

$$\begin{aligned} E_{\text{delay}} &= \sum_{i=0}^{C-1} R_i \times P_i \\ &= R_{C-1} \times P_{C-1} + \dots + R_i \times P_i + \dots + R_0 \times P_0 \end{aligned}$$

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

$$\begin{aligned}
&= [F \times (\frac{1}{U_{C-1}})] \times P_{C-1} \\
&+ \dots \\
&+ \\
&\{F \times \max [\frac{1}{U_{C-1}}, \frac{Q_{C-1} \times A \times \sum_{k=0}^{C-2} P_k}{U_{C-2}}, \dots, \frac{Q_{i+2} \times A \times \sum_{k=0}^{i+1} P_k}{U_{i+1}}, \frac{Q_{i+1} \times A \times \sum_{k=0}^i P_k}{U_i}] \} \times P_i \\
&+ \dots \\
&+ \{F \times \max [\frac{1}{U_{C-1}}, \dots, \frac{Q_{i+1} \times A \times \sum_{k=0}^i P_k}{U_i}, \dots, \frac{Q_1 \times A \times P_0}{U_0}] \} \times P_0 \tag{3.17}
\end{aligned}$$

3.4.3.1 A Simplified Example of the Model

The above analysis is a general model to analyze the optimal performance of the Hierarchical Data Grid. Given that a group of client nodes may have similar access pattern, the general model can be further simplified. One specific example is that all the client nodes have the same request file distribution and the same download frequency; i.e. $CD_c(X) = D(X)$ and $f_c = f$ for any client node c . In this section, we will further simplify the general model with these assumptions.

Using equation (3.4), the distribution of aggregated file download requests for storage node j ($j \in RS$) becomes:

$$\begin{aligned}
FD_j(x) &= \frac{\sum_{c \in DC_j} (D(x) \times F_{c,T(j)}(x) \times f)}{\sum_{c \in DC_j} f} \\
&= \frac{\sum_{c \in DC_j} (D(x) \times F_{c,T(j)}(x))}{Q_{T(j)}} \quad (j \in RS \text{ and } x=1,2,\dots,T_f) \tag{3.18}
\end{aligned}$$

where DC_j denotes the set of descendant client nodes of the middle-tier storage node j ; $T(j)$ denotes the tier that storage node j resides in; $F_{c,i}(x)$ is a 0-1 variable indicating whether download requests from client node c for file x reach its

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

parent storage node in tier i ; and Q_i denotes the number of descendant client nodes of each tier i storage node ($0 < i < C$).

By substituting equation (3.6a) into (3.18), we can find the distribution of aggregated file download requests for storage node j in tier $C-1$ is:

$$\begin{aligned} FD_j(x) &= \frac{\sum_{c \in DC_j} D(x)}{Q_{T(j)}} \\ &= D(x) \quad (j \in RS, T(j)=C-1 \text{ and } x=1,2,\dots,T_r) \end{aligned} \quad (3.19)$$

According to equation (3.5) and (3.19), the files replicated at tier $C-1$ nodes will be:

$$\begin{aligned} FS_j &= TF_j(M_{C-1}) \\ &= TF(M_{C-1}) \quad \text{for } (j \in RS \text{ and } T(j)=C-1) \end{aligned} \quad (3.20)$$

where $TF(n)$ denotes the set of top n most popular files for distribution $D(x)$ and M_i denotes the maximum number of replicas that can be stored at one tier- i node ($0 < i < C$).

This means that each node in tier $C-1$ will replicate the same set of files, which are the top M_{C-1} most popular files for the distribution of $D(x)$. For storage node j in tier $C-2$, which is one tier higher than tier $C-1$, its direct children nodes DS_j are in tier $C-1$. According to equation (3.6b) and (3.20), we have:

$$\begin{aligned} F_{c,C-2}(x) &= \begin{cases} 0 & \text{if } x \in FS_{S(c,C-1)} \\ 1 & \text{if } x \notin FS_{S(c,C-1)} \end{cases} \\ &= \begin{cases} 0 & \text{if } x \in TF(M_{C-1}) \\ 1 & \text{if } x \notin TF(M_{C-1}) \end{cases} \quad (c \in CS \text{ and } x=1,2,\dots,T_r) \end{aligned} \quad (3.21)$$

where $TF(n)$ denotes the set of top n most popular files for distribution $D(x)$ and M_i denotes the maximum number of replicas that can be stored at one tier- i node ($0 < i < C$).

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

By substituting equation (3.21) into (3.18), the distribution of aggregated file download requests for storage node j in tier C-2 becomes:

$$\begin{aligned} FD_j(x) &= \frac{\sum_{c \in DC_j} (D(x) \times F_{c,C-2}(x))}{Q_{C-2}} \\ &= \begin{cases} 0 & \text{if } x \in TF(M_{C-1}) \\ D(x) & \text{if } x \notin TF(M_{C-1}) \end{cases} \quad (j \in RS \text{ and } x=1,2,\dots,T_r) \quad (3.22) \end{aligned}$$

where $TF(n)$ denotes the set of top n most popular files for distribution $D(x)$ and M_i denotes the maximum number of replicas that can be stored at one tier- i node ($0 < i < C$).

Using equations (3.5) and (3.22), the set of files to be replicated on server node j in tier C-2 is:

$$\begin{aligned} FS_j &= TF_j(M_{C-2}) \\ &= TF(M_{C-1} + M_{C-2}) - TF(M_{C-1}) \text{ for } (j \in RS \text{ and } T(j) = C-2) \end{aligned} \quad (3.23)$$

where $TF(n)$ denotes the set of top n most popular files for distribution $D(x)$ and M_i denotes the maximum number of replicas that can be stored at one tier- i node ($0 < i < C$).

This means that each node in tier C-2 will replicate the same set of files, which are the files that have popularity ranking from (M_{C-1}) th to $(M_{C-1} + M_{C-2})$ th. By repeating the above analysis, we can find that the set of files to be replicated on one middle-tier node j can be calculated using equation (3.20) or the following equation, depending on the tier of node j :

$$FS_j = TF\left(\sum_{k=T(j)}^{C-1} M_k\right) - TF\left(\sum_{k=T(j)+1}^{C-1} M_k\right) \text{ for } (j \in RS \text{ and } 0 < T(j) < C-1) \quad (3.24)$$

where $TF(n)$ denotes the set of top n most popular files for distribution $D(x)$ and M_i denotes the maximum number of replicas that can be stored at one tier- i node ($0 < i < C$).

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

Using equation (3.7a) and (3.20), we can find the probability that client node c download requested file from tier $C-1$ is:

$$\begin{aligned}
 P(c, C-1) &= \sum_{x \in FS_{S(c, C-1)}} D(x) \quad (c \in CS; \text{ and } x=1, 2, \dots, Tr) \\
 &= \sum_{x \in TF(M_{C-1})} D(x) \\
 &= \sum_{x=1}^{M_{C-1}} D(x) \tag{3.25}
 \end{aligned}$$

where M_i denotes the maximum number of replicas that can be stored at one tier- i node; and $D(x)$ denotes the client access pattern and $x=1$ represents the most popular file.

Using equation (3.20) and (3.24), we can find out that:

$$\begin{aligned}
 FS_{S(c, i)} &= \bigcup_{k=i+1}^{C-1} FS_{S(c, k)} \\
 &= \left[TF\left(\sum_{k=i}^{C-1} M_k\right) - TF\left(\sum_{k=i+1}^{C-1} M_k\right) \right] - \left\{ \left[TF\left(\sum_{k=i+1}^{C-1} M_k\right) - TF\left(\sum_{k=i+2}^{C-1} M_k\right) \right] + \dots + [TF(M_{C-1})] \right\} \\
 &= \left[TF\left(\sum_{k=i}^{C-1} M_k\right) - TF\left(\sum_{k=i+1}^{C-1} M_k\right) \right] \quad (c \in CS \text{ and } 0 < i < C-1) \tag{3.26}
 \end{aligned}$$

where $TF(n)$ denotes the set of top n most popular files for distribution $D(x)$ and M_i denotes the maximum number of replicas that can be stored at one tier- i node.

Equation (3.26) shows that the set of replica files stored at different tiers has no intersection with each other. By substituting equation (3.26) into equation (3.7b), we can find that the probability that client node c download requested file from tier i ($0 < i < C-1$) is:

$$P(c, i) = \sum_{x \in \left[TF\left(\sum_{k=i}^{C-1} M_k\right) - TF\left(\sum_{k=i+1}^{C-1} M_k\right) \right]} D(x) \quad (c \in CS; 0 < i < C-1 \text{ and } x=1, 2, \dots, Tr)$$

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

$$= \sum_{x=O_i+1}^{M_i+O_i} D(x) \quad \text{where } O_i = \sum_{k=i+1}^{C-1} M_k \quad (3.27)$$

where M_i denotes the maximum number of replicas that can be stored at one tier- i node; O_i represents the total number of files that are replicated at tiers lower than tier i ; $D(x)$ denotes the client access pattern and $x=1$ represents the most popular file.

Since $f_c = f$ for any client node c , using equation (3.8) and (3.25), the probability that the replica is downloaded from tier $C-1$ for all client nodes is:

$$P_{C-1} = \sum_{x=1}^{M_{C-1}} D(x) \quad (3.28)$$

where $D(x)$ is the probability distribution function of the client requests, x denotes the rank of file popularity and the file with rank $x=1$ is the most popular (most frequently accessed) file; and M_{C-1} is the maximum number of replicas that can be stored at one tier $C-1$ node.

Using equation (3.8) and (3.27), the probability that the replica is downloaded from tier i ($0 < i < C-1$) is:

$$P_i = \sum_{x=O_i+1}^{M_i+O_i} D(x) \quad \text{where } O_i = \sum_{j=i+1}^{C-1} M_j \quad (0 < i < C-1) \quad (3.29)$$

where M_i is the maximum number of replicas that can be stored at one tier- i node; and O_i represents the total number of files that are replicated at lower tiers of tier i .

3.5 Simulation Results and Model Verification

In this section, we use the multi-tier Data Grid as an example of the Hierarchical Data Grid. The performance of the optimal replication algorithm is studied in the simulated multi-tier Data Grid with different configurations. The simulation results are compared with the theoretical result obtained from the probabilistic model developed in section 3.4. Three well-known replication algorithms are also

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

studied here and their performances are compared with our proposed optimal replication algorithm.

3.5.1 The Multi-Tier Data Grid Example

Figure 3.3 shows the multi-tier Data Grid architecture as an example of the Hierarchical Data Grid. There are three tiers excluding the root node, i.e. $C=3$. The root node has 5 children nodes at tier 1 and each middle-tier node has 5 children nodes at the lower tier. There are 155 nodes in total, i.e. $T_n = 155$ and 125 of them are client nodes, i.e. $T_c = 125$. Each client node sends one request per 150 seconds i.e. $f_c = 1/150$. All the client nodes are the children nodes of the root node, so $Q_0=125$. Each tier-1 node has 25 children client nodes, i.e. $Q_1 = 25$; and each tier-2 node has 5 children client nodes, i.e. $Q_2 = 5$.

Figure 3.3: The Multi-Tier Data Grid Model

There are 10,000 different data files in total, i.e. $T_f=10,000$, and each file is 2 GBytes in size, i.e. $F = 2\text{GB}$. So the total size for all the files is 20 TB, i.e. $D=20\text{TB}$.

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

There are 30 middle-tier storage nodes, i.e. $T_s=30$. There are 5 tier-1 nodes with storage capacity of 1 TB per node; and 25 tier-2 nodes with storage capacity of 0.1TB per node. So the total storage capacity for middle-tier nodes is $S=7.5$ TB and the server relative capacity $r = S/D = 38.4\%$ (note $1\text{TB}/\text{GB} = 1024$). Each tier-1 node can store 500 replicas, i.e. $M_1 = 500$; and one tier-2 node can store 50 replicas, i.e. $M_2 = 50$.

The network bandwidth is 2.5 Gbps for link between tier 0 and tier 1, i.e. $U_0 = 2.5$ Gbps; 2.5 Gbps for link between tier 1 and tier 2, i.e. $U_1 = 2.5$ Gbps; and 622 Mbps for link between tier 2 and client tier, i.e. $U_2 = 622$ Mbps. The cost associated with these network links are 5, 2 and 0 respectively per Mbyte, i.e. $W_0 = 5$, $W_1 = 2$, $W_2 = 0$. To transfer the replica from root node to client tier, the cost is $B_0=5+2=7$. And obviously $B_1=2$ and $B_2=0$.

In our simulations there are 100,000 client requests for file transfer, i.e. $N_r = 100,000$. The file popularity for all the client nodes are the same and follow Geometric distribution [121] with $p=0.01$ (denoted as Geo-0.01), i.e. $CD_c(x) = p(1-p)^{x-1} = 0.01 \times 0.99^{x-1}$ ($x=1,2,\dots,10000$) for any client node c , where $x=1$ represents the most popular file.

The parameter values of the studied example are summarized in Table 3.3.

Table 3.3: Parameters for the Multi-Tier Data Grid with $r=38.4\%$

Parameter	Meaning	Value
C	number of tiers (excluding tier 0)	3
F	size of the replica file	2 GBytes
T_r	total number of different data files in the system	10,000
T_n	total number of nodes in the system	155
T_s	total number of middle-tier storage nodes in the system	30
T_c	total number of client nodes in the	125

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

	system	
f_c	frequency of download requests from each client	1/150
S	total storage size of the middle-tier servers	7.5 TB
D	total size of all data files in the system	20 TB
r	server relative capacity	38.4%
U_i	bandwidth capacity of link between tier i and tier $i+1$ ($0 \leq i < C$)	$U_0 = 2.5$ Gbps; $U_1 = 2.5$ Gbps; $U_2 = 622$ Mbps
W_i	bandwidth cost for link between tier i and tier $i+1$ ($0 \leq i < C$)	$W_0 = 5$; $W_1 = 2$; $W_2 = 0$ per Mbyte
B_i	bandwidth cost for network connection between tier i and client tier ($0 \leq i < C$)	$B_0 = 7$; $B_1 = 2$; $B_2 = 0$
M_i	maximum number of replicas that can be stored at one tier- i node ($0 \leq i < C$)	$M_0 = 10,000$; $M_1 = 500$; $M_2 = 50$
Q_i	the number of descendant client nodes for each node in tier i ($0 \leq i < C$)	$Q_0 = 125$; $Q_1 = 25$; $Q_2 = 5$
$CD_c(x)$	probability distribution function of download requests for file x ($x=1,2,\dots,T_r$) from each client node c	$p(1-p)^{x-1} = 0.01 \times 0.99^{x-1}$ ($x=1,2,\dots,10000$)
N_r	total number of client download requests	100,000

Using equation (3.28), the probability that the file is downloaded from tier-2 node is:

$$\begin{aligned}
 P_2 &= \sum_{x=1}^{M_2} D(x) \\
 &= \sum_{x=1}^{50} (0.01 \times 0.99^{x-1}) \\
 &= 0.3950
 \end{aligned}$$

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

Using equation (3.29), the probability that the file is downloaded from tier-1 node is:

$$\begin{aligned}
 P_1 &= \sum_{x=O_i+1}^{M_i+O_i} D(x) \text{ where } O_i = \sum_{j=i+1}^{C-1} M_j \quad (0 \leq i < C-1) \\
 &= \sum_{x=50+1}^{500+50} (0.01 \times 0.99^{x-1}) \\
 &= 0.6010
 \end{aligned}$$

Using equation (3.9), the probability that the file is downloaded from tier-0 node is:

$$P_0 = 1 - \sum_{i=1}^{C-1} P_i = 1 - (P_1 + P_2) = 0.004$$

Using equation (3.10), the average bandwidth cost is:

$$\begin{aligned}
 E_{\text{cost}} &= F \times \sum_{i=0}^{C-1} (B_i \times P_i) \\
 &= F \times (B_2 \times P_2 + B_1 \times P_1 + B_0 \times P_0) \\
 &= (2 \times 1024) \text{ Mbytes} \times (0 \times 0.3950 + 2 \times 0.6010 + 7 \times 0.004) / \text{Mbytes} \\
 &= 2519.04
 \end{aligned}$$

Using equation (3.17), the average response time is $E_{\text{delay}} = 26.34$. The detailed calculation is included in Appendix A.

For the case of $r=3.84\%$, the storage capacity for middle-tier nodes is reduced by 10, i.e. for tier-1 node is 0.1 TB and for tier-2 node is 0.01TB. The total storage capacity for middle-tier nodes is $S=0.75$ TB. Now tier-1 node can store up to 50 replicas, i.e. $M_1 = 50$; and one tier-2 node can store up to 5 replicas, i.e. $M_2 = 5$. Values of the other parameters remain the same as in Table 3.3. The affected parameter values are summarized in Table 3.4.

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

Table 3.4: Affected Parameter Values for the Case of $r=3.84\%$

S	total storage size of the middle-tier servers	0.75 TB
r	server relative capacity	3.84%
M_i	maximum number of replicas that can be stored at one tier-i node ($0 \leq i < C$)	$M_0=10,000$; $M_1=50$; $M_2=5$

In addition to Geometric Distribution, we have also analyzed the optimal performance with Zipf-like [122] distribution of client request. We use Zipf-0.8

as an example, i.e. $CD_c(x) = \frac{x^{-0.8}}{\sum_{n=1}^{T_r} n^{-0.8}}$ where T_r is the total number of different

data files in the system; $x \in (1, 2, \dots, T_r)$ and $x=1$ indicates the most popular file.

The optimal performance of the multi-tier Data Grid with different configurations is summarized in Table 3.5.

Table 3.5: The Optimal Performance Predicted by the Model

Configuration	Average Bandwidth Cost	Average Response Time
Geo-0.01; $r=38.4\%$	2519.04	26.34
Geo-0.01; $r=3.84\%$	9787	26.34
Zipf-0.8; $r=38.4\%$	8355.2	26.34
Zipf-0.8; $r=3.84\%$	11404.49	26.34

3.5.2 Replication Algorithms

In this section, we present three well-known dynamic replication algorithms for the multi-tier Data Grid: Simple Bottom-Up (SBU) [81], Aggregate Bottom-Up (ABU) [81] and Fast Spread (FSP) [76].

3.5.2.1 Simple Bottom-Up [81]

The Simple Bottom-Up algorithm processes the client access history to determine the popular replicas, and tries to replicate these popular files as close as possible to the clients that have frequent requests for the file. The replica access history is kept in records in the format of (client ID, file ID, access number). The file ID refers to the filename of the file while the access number refers to the frequency of access of the particular file. A file is categorized as popular when the access number exceeds the predefined threshold. The SBU scans the access records to determine the popular files and sort them in descending order of access number. The most popular files are replicated at the bottom tier, which is nearest to the clients. The algorithm is terminated once all the popular files are replicated or the storage space of all the middle-tier replica servers is filled. The replication algorithm is invoked periodically to adapt to changes in the client access pattern.

Figure 3.4 shows a simplified example of the access record for SBU and ABU. In this simple example, the access record shows that client node 3 has accessed file A once and accessed file B a total of 6 times. Client node 4 has accessed file A a total of 8 times and file B a total of 7 times. After sorting the access record, file A is the most popular file since it is accessed by client 4 a total of 8 times. Using the SBU algorithm, file A will be replicated at node 1 if only one file can be stored.

Figure 3.4: An Example of Access Record for SBU and ABU

3.5.2.2 Aggregate Bottom-Up [81]

The Aggregate Bottom-Up algorithm is an improved version of the Simple Bottom-Up algorithm. The main difference of ABU and SBU lies in the algorithm to process the replica access history. The multi-tier Data Grid has a tree structure and ABU uses this characteristic to aggregate the access requests from the clients that have the same parent node.

SBU processes the historical access records individually, discarding the relations among these records; while ABU aggregates the access records from bottom tier towards root. Using the same example in Figure 3.4, ABU will aggregate the access records of both nodes 3 and 4 as they have the same parent node 1. The resultant records are $\{1, A, 9\}$ and $\{1, B, 13\}$. Now file B is more popular than file A at node 1. If node 1 can only store one replica, replica for file B will be stored. By aggregating the requests from children nodes that have the same parent, ABU replicate the popular files from bottom tier towards root. The same as SBU, ABU is also scheduled to run in fixed intervals.

3.5.2.3 Fast Spread [76]

The idea of Fast Spread is straightforward: to store a replica of the file at each node along the downloading path to the client. It is assumed that all the middle-tier nodes support streaming and can intercept the data stream while it is transferred from the source to the client. In this case, after the client finishes downloading file successfully, all the transit middle-tier nodes will automatically have a copy of the file.

3.5.3 Simulation Results and Discussion

We run the simulations using the DRepSim [81] simulator, using the parameters described in section 3.5.1. The DRepSim simulator was previously developed in [81]. It can simulate the Hierarchical Data Grid environment with different settings and support different replication algorithms. With DRepSim, users can easily simulate the Data Grid environment, by specifying the system topology, replica server storage capacities, network bandwidth, client access pattern and etc,

 CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

in the configuration files. Users can also introduce new replication algorithms to the simulator and study their performance with simulations.

Figure 3.5 shows the simulation results for the four replication algorithms, with Geometric distribution client access pattern. Figure 3.6 shows the simulation results with Zipf-like client request distribution. The studied algorithms are described in section 3.4.2 and 3.5.2. Figure 3.5(a) shows that the optimal replication algorithm (OPT) gives an average bandwidth cost of 9730.53 for server relative capacity $r=3.84\%$ and 2488.04 for $r=38.4\%$, with client requests following a Geometric distribution. As shown in Table 3.5, the average bandwidth cost predicted using our model is 9787 for $r=3.84\%$ and 2519.04 for $r=38.4\%$. The results from the simulation are very close to the analytical results; and this gives us some confidence verifying the accuracy of our model. For the average response time, Figure 3.5(b) shows that the OPT has a performance of 28.7 seconds for $r=3.84\%$ and 27 seconds for $r=38.4\%$. The results are quite close to the prediction in Table 3.5, which is 26.4 seconds for both cases. Comparing Figure 3.6 and Table 3.5 gives us the same conclusion and this shows that our model is correct in predicting the optimal performance of the Hierarchical Data Grid.

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

(a) Average Bandwidth Cost for Geo-0.01

(b) Average Response Time for Geo-0.01

Figure 3.5: Simulation Results for Geometric Distribution

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

(a) Average Bandwidth Cost for Zipf-0.8

(b) Average Response Time for Zipf-0.8

Figure 3.6: Simulation Results for Zipf-like Distribution

CHAPTER 3. MODELING THE HIERARCHICAL DATA GRID

Comparing OPT performance with the three well-known replication algorithms, both Figure 3.5 and 3.6 shows that OPT has the least average bandwidth cost and shortest average response time, i.e. the best performance. FSP performance is the closest to OPT replication algorithm. This is because FSP simply keeps a copy at the middle-tier server along the file transmission path. In a relatively static environment where the file popularity does not change from time to time, FSP has good performance because there is no extra cost introduced by the replication process. On the other hand, the overhead introduced by ABU during the replication process exceeds the benefit it brought in for this simple case. However, FSP has its own problem, such as causing too much I/O operation and the ABU algorithm outperforms FSP in dynamic environment as reported in [81].

3.6 Summary

In this chapter, we have presented a mathematical model for the Hierarchical Data Grid, to predict its optimal performance in terms of average response time and average bandwidth cost. The model parameters and some realistic assumptions are clearly stated. We have also proposed an optimal replication algorithm for the modeled Hierarchical Data Grid, which gives the shortest average response time and least average bandwidth cost. The performance of the proposed replication algorithm is analyzed in detail using the developed probabilistic model.

We have used the multi-tier Data Grid used by CERN, as a specific example of the Hierarchical Data Grid, to verify our model. The performance of our proposed OPTimal replication algorithm (OPT) is studied in a simple Data Grid environment. The simulation results tally with the optimal performance predicted using our mathematical model. Three well-known replication algorithms: Simple Bottom-Up (SBU), Aggregate Bottom-Up (ABU) and Fast Spread (FSP) are also studied here and their performances are compared with the OPT algorithm. The simulation results show that OPT has better performance than all three replication heuristics.

Chapter 4

Analysis of Replication Algorithms with Expiration-Based Consistency Management

4.1 Introduction

Previous researches on replication algorithms have simply assumed that the application data are read-only in Data Grid environment. With the applications supported by the Data Grid constantly increasing, it is essential to take into account the consistency management of the replicas, as described in section 2.4. When the dataset can be modified by applications, it is important to maintain consistency among the existing multiple copies. Over the past few years, researchers from EGEE [89-91] and other Data Grid projects [94-95] have started to work on the Replica Consistency Management problem.

Expiration-based consistency management is the most widely used mechanism in today's internet [20], but relatively new in the area of Data Grid. In this approach,

CHAPTER 4. ANALYSIS OF REPLICATION ALGORITHMS

each replica is associated with a Time-To-Live (TTL) value. The replica is considered invalid when the TTL expires, and the invalid replica will be updated on receiving new access requests. Thus, additional cost/delay in validating and updating the replicas is introduced to the total access cost/time. This may affect the performance of the replication algorithms, which are originally designed with read-only application data.

In this chapter, we aim to analyze the performance of different replication algorithms under the expiration-based consistency management. The simulation environment is according to the architecture of the multi-tier Data Grid, which is a Hierarchical Data Grid designed by MONARC project. We have extended the Data Grid simulator, called DRepSim [81], to support the expiration-based consistency management scheme. The performance of the replication algorithms are re-evaluated under a simulation environment where replicas will expire from time to time. The performance metrics used in our study are the average response time and average bandwidth cost, as described in section 3.4.1.

The rest of the chapter is organized as follows. Section 4.2 presents the replication algorithms, whose performance will be evaluated. Section 4.3 describes how the expiration-based consistency management works in the Hierarchical Data Grid environment. The simulation environment is described in section 4.4. The simulation results are reported in section 4.5. The results are discussed extensively in section 4.6. Finally we conclude this chapter in section 4.7.

4.2 Replication Algorithms

Three dynamic replication algorithms for the Hierarchical Data Grid are already presented in section 3.5.2: Simple Bottom-Up (SBU) [81], Aggregate Bottom-Up (ABU) [81] and Fast SPread (FSP) [76]. In this section, the Random Static Replication (RSR) algorithm [81] is presented as a representation of the static replication algorithms, and its performance is studied to compare with the three dynamic replication algorithms.

4.2.1 Random Static Replication (RSR) [81]

The static replication algorithm allocates replicas at the beginning of the simulation and does not change the allocation over time. As the file popularity changes from time to time, it is impossible to deduce an optimal static replication algorithm without knowing the client access pattern in advance. Thus, in our research we use the Random Static Replication (RSR) algorithm to represent the static replication algorithm and compare its performance with the dynamic replication algorithms.

The RSR algorithm processes the replica servers in the Data Grid one by one. For each replica server, it will randomly choose a set of data files to replicate on the server. This process will stop when the storage space of the replica server is used up or all the data files have been replicated. The algorithm will then move on to process another replica server and it finishes when all the replica servers have been processed.

4.3 Expiration-Based Consistency Management

Expiration-based consistency management has been widely used to ensure that data files are up-to-date on the Internet [20]. In the expiration-based consistency management system, each data file is tagged with a Time-to-Live (TTL) value when it is transferred from the root node. The replica is considered valid only if its TTL value is greater than 0.

A simplified Hierarchical Data Grid, as already described in section 3.3, with expiration- based consistency management is shown in Figure 4.1. We use this example to demonstrate how the expiration-based consistency scheme works in a Hierarchical Data Grid environment.

CHAPTER 4. ANALYSIS OF REPLICATION ALGORITHMS

Figure 4.1: A Simplified Hierarchical Data Grid with Expiration-Based Consistency Management

Node 0 is the root node and it is at tier 0. All the data files are generated and stored at node 0. Hence, the data files are always fresh and valid at node 0. The tier below tier 0 is tier 1, and there are two nodes: node 1 and node 2, at this tier. Node 1 and 2 can download data files from node 0, and the initial TTL value of the data file is assigned with the maximum TTL value. In this example, the maximum TTL value is set to 5 seconds. When time elapses, the TTL value will decrease and the replica will be considered invalid once TTL drops to 0. At the time instance shown in Figure 4.1, Node 1 has replica A with TTL value of 4 seconds; and node 2 has replica A with TTL value of 0, which means the replica expires already. Tier 2 has four nodes: node 3, 4, 5 and 6. These nodes will try to download replicas from their direct parent at tier 1. If a valid replica is available there, it will be downloaded and the initial TTL value remains no change. If the replica at tier 1 already expires, the data will be fetched from tier 0 and the initial TTL value is set to 5 seconds. The TTL value of the data file along the transmission path will be updated. As shown in Figure 4.1, Node 3 has replica A with TTL value of 4 seconds. Node 4 and 5 have

CHAPTER 4. ANALYSIS OF REPLICATION ALGORITHMS

expired replica A. While there is no replica of file A on node 6. At tier 3 are client nodes 7 to 14, which send download requests to their direct parent node at tier 2. In the following, we use four case studies to illustrate the effect of expiration-based scheme.

Case 1: When client node 7 requests to download file A, it finds the replica at node 3 and checks the TTL value = 4 seconds, which means the replica is valid. Node 7 will download replica A from node 3.

Case 2: When client node 9 requests to download file A, it finds that the replica exists at node 4. However, the replica has a TTL value of 0 and is expired. It then searches for replica A upwards to node 1 and finds a valid replica A with TTL = 4 seconds. Node 9 then downloads the replica from node 1, and replica A at node 4 is updated with a new TTL value of 4 seconds. Thus, the TTL value of the replica at both node 1 and node 4 are synchronized. This phenomenon is known as recursive inheritance of expiration time [20].

Case 3: When client node 11 requests to download file A, it searches the tree structure upwards and finds a valid replica only at the root node. Thus, node 11 will download the replica from root node directly; and the replica at node 2 and node 5 are updated to have a TTL value of 5 seconds.

Case 4: When client node 13 requests to download file A, it will download the replica directly from root node, similar to case 3. The replica at node 2 will be updated with TTL value of 5 seconds for all the dynamic replication algorithms. A new replica with TTL value of 5 seconds will be created at node 6 for FSP algorithm, because FSP replicates the files along the transmission path, as described in section 3.5.2.3. Old replicas will be deleted according to the replica replacement policy if there is not enough storage space at node 6.

4.4 Simulation Environment

We have extended the Data Grid simulator, named DRepSim [81], by adding in the

CHAPTER 4. ANALYSIS OF REPLICATION ALGORITHMS

expiration-based consistency management scheme. The simulated environment is according to the multi-tier Data Grid, used by CERN for the LHC experiments, as shown in Figure 4.2. The simulation parameters are similar to the ones specified in section 3.5.1. The only difference is that we have increased the client download requests by 10 times, from 100,000 to 1,000,000, as described later in this section. The essential simulation parameters are summarized in Table 4.1.

Figure 4.2: The Simulated Multi-Tier Data Grid

As shown in Figure 4.2, there are four tiers of compute and storage resources. Tier 0 is the root tier. The CERN offline farm, where all the raw data generated by the online system are stored, is at this tier. Tier 1 has five regional centers, and tier 2 has 25 national centers. There are 125 institute centers at tier 3. Each center in tier 0, 1, and 2 serves five centers in the lower tier. Each center is represented as a node and the infrastructure is a tree structure. Tier 3 is the client tier and all the data file access requests are generated from this tier.

As shown in Figure 4.2, the network bandwidth between any node in tier 3 and its

CHAPTER 4. ANALYSIS OF REPLICATION ALGORITHMS

parent node in tier 2 is 622 Mbps. The bandwidth between any node in tier 2 and its parent node in tier 1 is 2.5 Gbps. Between any node in tier 1 and tier 0, the bandwidth is also 2.5 Gbps. In our research, there is a cost associated with each network link to study the network bandwidth cost. The cost is 5 units per Mbyte for link between tier 0 and 1; 2 units per Mbyte for link between tier 1 and 2; and 0 units for link between tier 2 and 3. Different costs are used due to the fact that international links are more expensive than regional links.

Table 4.1: Simulation Parameters for $r=38.4\%$

Parameter	Value
size of the data file	2 GBytes
total number of different data files in the system	10,000
total number of nodes in the system	155
total number of middle-tier storage servers in the system	30
total number of client nodes in the system	125
frequency of download requests from each client	1/150
storage capacity of one tier-1 node	1 TB
storage capacity of one tier-2 node	0.1 TB
total storage capacity of the middle-tier servers	7.5 TB
total size of all files in the system	20 TB
server relative capacity	38.4%
probability distribution function of download requests for file x from each client node	$p(1-p)^{x-1} = 0.01 \times 0.99^{x-1}$ ($x=1,2,\dots,10000$)
total number of client download requests	1,000,000

As shown in Table 4.1, there are 10,000 different data files in the system and each data file is 2 GBytes. On average, each client node sends one request every 150 seconds and the data access requests follow Poisson distribution. For all the 125 client nodes, there are about 0.8 requests per second. For the file popularity, we adopt the Geometric distribution [121] with $p=0.01$ in our research. In real systems,

CHAPTER 4. ANALYSIS OF REPLICATION ALGORITHMS

the data file popularity may change from time to time. To simulate this scenario, we partition the simulation session into 10 sub-sessions evenly, i.e. 100,000 download requests in each sub-session. The data file popularity in each sub-session still follows Geometric distribution; however the set of popular files are changed from sub-session to sub-session.

There are 30 middle-tier storage nodes. Five of them are tier-1 nodes with storage capacity of 1 TB per node. The other 25 are tier-2 nodes with storage capacity of 0.1TB per node. So the total storage capacity of middle-tier nodes is 7.5 TB. Using equation (3.1) in section 3.3, we have the server relative capacity $r = 38.4\%$ (note $1\text{TB}/\text{GB} = 1024$).

The storage capacities of the middle-tier replica servers are limited and we can only replicate a portion of the files on these servers. With larger storage capacities, more data files can be replicated in the middle-tier servers. This may lead to improved performance of the replication algorithms in the Data Grid environment. Thus, the server relative capacity has a major impact on the performance of replication algorithms. In our research, we have also run the simulations with $r=3.84\%$. Compared to the case of $r=38.4\%$, the only difference is that the storage capacity of the middle-tier nodes is reduced by a factor of 10. For the case of $r=3.84\%$, each tier-1 node has a storage capacity of 0.1 TB and each tier-2 node has a capacity of 0.01TB. The total storage capacity of the middle-tier storage nodes becomes 0.75 TB. The affected parameter values are summarized in Table 4.2.

Table 4.2: Affected Parameter Values for $r=3.84\%$

Parameter	Value
storage capacity of one tier-1 node	0.1 TB
storage capacity of one tier-2 node	0.01 TB
total storage capacity of the middle-tier servers	0.75 TB
server relative capacity	3.84%

4.5 Simulation Results

4.5.1 Simulation Results for $r=3.84\%$

For the server relative capacity $r=3.84\%$, the simulation results are shown in Figure 4.3 and 4.4. Figure 4.3 shows the average bandwidth cost for all the four replication algorithms. The X-axis denotes the maximum TTL value used in each simulation, in seconds in the log scale, i.e. the number of 4 indicates that the maximum TTL value is set to 10^4 seconds. The Y-axis denotes the average bandwidth cost and a higher value indicates worse performance. Figure 4.4 shows the average response time for the three dynamic replication algorithms. The X-axis denotes the maximum TTL value in seconds in the log scale, and the Y-axis denotes the average response time in seconds. The lower the average response time, the better the performance of the replication algorithm is. The average response time of the static replication algorithm is not shown here, because it is too large and not in the same scale with that of the three dynamic replication algorithms.

From Figure 4.3, we can see that the average bandwidth cost decreases when the TTL value is increased, for all the dynamic replication algorithms. This is to be expected, as the new constraint: expiration time of file, will force the client nodes to make more requests towards the root node. And it is expensive to download from the root node because of the long distance network transfer. With a smaller TTL value, the replica will expire more frequently and thus cause more requests directed towards the root node. Thus, the bandwidth cost drops with an increasing TTL value. When the TTL value is large enough, it is likely that the replica will never expire and the TTL value will not be able to affect the performance. That's why we see the performance becomes stable when TTL reaches a value of around $10^{5.5}$ seconds, in both Figure 4.3 and 4.4.

CHAPTER 4. ANALYSIS OF REPLICATION ALGORITHMS

Figure 4.3: Average Bandwidth Cost Comparison for $r=3.84\%$

Figure 4.4: Average Response Time Comparison for $r=3.84\%$

CHAPTER 4. ANALYSIS OF REPLICATION ALGORITHMS

With a TTL value larger than $10^{3.5}$ seconds, ABU has the best performance and RSR has the worst performance in terms of average bandwidth cost, as shown in Figure 4.3. This tallies with the previous research results [81] that ABU has the best performance, without consideration of consistency management. When the TTL value is as small as 10^3 seconds, ABU has the worst performance and FSP has the best performance. This is due to the nature of these replication algorithms. ABU dynamically replicates popular files towards client nodes to improve the performance; however this will introduce additional network traffic to the system. With a small TTL value, the replicas close to client nodes are not useful as they will expire very frequently. Thus, the advantage brought by ABU is not so significant, while the overhead is more obvious. That's why we see the performance of ABU is even worse than RSR, with TTL value of 10^3 seconds in Figure 4.3. While FSP only creates replicas along the transmission path and thus introduce no additional cost. That's why FSP outperforms ABU when TTL value is very small.

The TTL value has little effect on the RSR algorithm. This is because RSR replicates the data file randomly and does not change the replication over time. Thus, the chance that the requests are directed towards root node because of replica expiration is very small.

In terms of average response time, as shown in Figure 4.4, there is little difference in the performance of ABU and FSP, with ABU being slightly better. SBU has an average response time of more than 600 seconds, while the average response time for ABU and FSP is below 100 seconds.

4.5.2 Simulation Results for $r=38.4\%$

For the server relative capacity $r=38.4\%$, the simulation results are shown in Figure 4.5 and 4.6. Figure 4.5 shows the average bandwidth cost for all the four replication algorithms and Figure 4.6 shows the average response time for the three dynamic replication algorithms.

Comparing Figure 4.5 with Figure 4.3, we can find that there is a similar trend for all the algorithms, i.e. the average bandwidth cost decreases when TTL value is increased. Comparing the results from the two figures, the average bandwidth cost drops as the server relative capacity is increased from 3.84% to 38.4%. This is because more replica files are stored in the middle-tier replica servers, and the chance of downloading replicas from the root node is reduced. ABU outperforms FSP, which has the second best performance, by around 30% in most of the cases. In addition, the gap between FSP and SBU also decreases from around 20% to less than 10%.

When the server relative capacity is increased from 3.84% to 38.4%, comparing results in Figure 4.4 and Figure 4.6, the average response time of all the three dynamic replication algorithms decrease. The average response time of ABU decreases from around 80 seconds to 50 seconds, with TTL value of 10^4 seconds. SBU has an even greater reduction, from 750 seconds to 160 seconds. The performance gap between SBU and FSP becomes smaller, when the server relative capacity is increased.

CHAPTER 4. ANALYSIS OF REPLICATION ALGORITHMS

Figure 4.5: Average Bandwidth Cost Comparison for $r=38.4\%$

Figure 4.6: Average Response Time Comparison for $r=38.4\%$

CHAPTER 4. ANALYSIS OF REPLICATION ALGORITHMS

For FSP, it is observed that the average response time increases when TTL is increased from 10^4 to $10^{5.5}$ seconds in Figure 4.6. This is because FSP always keeps a copy of the file on the intermediate replica servers along the transmission path. In this way, the performance is improved, as access to the popular files are served by the intermediate servers, instead of the root. However, the set of popular files will change from sub-session to sub-session in our simulation. Thus, during the sub-session transition period, the previous replicated popular files on the intermediate nodes will become less popular and thus less frequently used. When this happens, the chance of the client nodes having to get the file from the root is increased. Thus, the performance becomes worse as it takes longer time to get the file from the root.

4.6 Discussion

Dynamic replication algorithms improve the system performance by dynamically placing popular replicas close to the client. When the expiration-based consistency management comes to the picture, the price to pay is the overhead to keep the replicas fresh and valid. For the multi-tier Data Grid, when the replica is placed close to the client tier, the response latency is short. However, to keep the replica synchronized with the master copy, the request for expired replica is forced to go further towards the root node. Thus, the replication algorithms must balance the advantage and the overhead when they make decisions of where to place the replicas.

In the case where replicas expire very frequently, i.e. replicas with very small TTL value, it doesn't make sense to create replicas for the data files. Every time when a client node tries to download the replica from the middle-tier nodes, it will find that the replica has expired. This will force the client node to travel up towards the root node to get a fresh copy. The benefits of file replication are lost.

For replicas with a relatively longer and more reasonable TTL value, the advantage brought by ABU far exceeds the overhead. When we increase the TTL value to 10^4

CHAPTER 4. ANALYSIS OF REPLICATION ALGORITHMS

seconds, ABU starts to outperform FSP and the performance gap between FSP and SBU becomes small. With a longer TTL value, the advantage of ABU and SBU is magnified because the replicated popular files can serve more download requests before they expire. Client nodes can download the valid replica from intermediate tier instead of root tier. Downloading file from the root tier is considered slow in terms of response time and expensive in terms of bandwidth cost.

With a higher server relative capacity, the replica servers can store more replicas in the middle tiers to improve the system performance. ABU utilizes these servers better because it tries to replicate the popular files, while FSP simply keeps a copy of the file while transmitting to clients. That is why ABU outperforms FSP significantly when TTL is relatively longer and server relative capacity is higher.

4.7 Summary

In this chapter, we have compared the performance of three dynamic replication algorithms and one static replication algorithm, in a Hierarchical Data Grid environment where the replicas expire from time to time. Our simulation results show that the TTL value has an important effect on the performance of the dynamic replication algorithms. As expected, the performance improves as the TTL value is increased. FSP has the best performance when the TTL value is very small; while ABU has the best performance when TTL value is relatively longer and more reasonable. A higher server relative capacity will also improve the performance of these replication algorithms and especially for ABU and SBU.

Chapter 5

Impact of Parallel Download on Job Scheduling in Data Grid Environment

5.1 Introduction

Data intensive applications, such as High Energy Physics (HEP), computational genomics and etc., usually have large input of scientific data to process. These scientific data are stored as large files and replicated across the Data Grid. Scientists geographically located all over the world are able to download these datasets and analyze them for various purposes. Scheduling such applications in Data Grid needs to take into consideration the location of the job input data, the distribution of the system's compute power and the available network bandwidth. The Grid scheduler decides where to run the job based on the scheduling policy to give the best overall performance. By creating replicas, the data replication algorithm can re-route data requests to certain replica servers and achieve remarkably higher access speed than a single server. At the same time, the widely distributed replicas provide more choices for the Grid scheduler to improve the performance from the perspective of the job. Some researchers have studied the impact of data replication on the job scheduling performance in the Data Grid environment [98-102].

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

Previous approaches only allow complete data files to be replicated at replica servers and transferred across the Data Grid. As the data is usually huge in size and the bandwidth is limited, fetching the required data from a single replica server will introduce a long delay before the data-intensive job can start. As the compute power increases, the delay caused by “CPU idle waiting for input data” will become more pronounced.

In this chapter, we study the impact of parallel download on job scheduling performance in Data Grid environment. Due to the limitation of storage size, it is not feasible for all data file to have multiple complete replicas across the replica servers. Thus, in our proposed parallel download approach, the data file is split into N ($N \in \{2,3,4,\dots\}$) fragments and stored on N different replica servers. These replica fragments are transferred from multiple replica servers concurrently to the compute server and reassembled as a complete file, reducing data access time.

We have extended the framework in [101] to support replicating and parallel downloading of data fragments. The Centralized Dynamic Replication algorithm called Server Merit Oriented Replica Placement (CDR_SMPlace) [101] is adopted as the replication algorithm to dynamically replicate the data files periodically. The algorithm is originally designed to be used in the non-parallel downloading system and we have adapted it to support parallel downloading. Three classic Grid Scheduling algorithms: Shortest Turnaround Time (STT), Least Relative Load (LRL) and Data Present (DP), are employed to schedule the Data Grid jobs. Their performances are discussed for both Parallel Downloading System (PDS) and non-Parallel Downloading System (n-PDS). We also use queuing theory to model the performance without considering the data transfer and obtain the optimal performance theoretically. The optimal performance is also verified by simulation of scheduling pure computing jobs, ignoring the data transfer. It is used as a benchmark to compare with both PDS and n-PDS Systems.

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

The rest of the chapter is organized as follows. Section 5.2 introduces the related works. Section 5.3 presents the Data Grid model that supports parallel downloading and job scheduling. The performance metric, which is used to compare the job scheduling performance, is also discussed. In Section 5.4, we use queuing theory to analyze the optimal performance without considering the data transfer time. Section 5.5 introduces the dynamic replication algorithm, for both PDS and n-PDS system, used in our research. The three classic scheduling algorithms are presented in section 5.6. Section 5.7 describes the simulation environment. The simulation results are reported in section 5.8 and discussed extensively in section 5.9. Finally, we conclude the chapter in section 5.10.

5.2 Related Works

Due to the large size of the input data, the data download time plays a significant part in the Data Grid environment. Various research efforts are made to reduce the data download time, including integration of data replication techniques and parallel download approach. In the parallel download approach, the client can download either from static servers in a stable environment or from dynamic servers in an inconsistent environment. In the case of downloading from dynamic servers, dynamic co-allocation technique is used to dynamically select source servers according to the current network situation and the server loads. The dynamic co-allocation problem has been addressed in many researches [124-127, 156-160]. All these researches focus on finding a perfect dynamic co-allocation algorithm that can achieve high download speed from multiple source servers. The algorithm is designed to work in a dynamically changing environment, where servers may crash or slow down over time. The algorithm also needs to reduce the time that faster server waiting for the slower server to finish the final blocks and thus minimize the overall download time.

The authors in [128] introduce the Multi-Source Data Transfer (MSDT) algorithm to reduce the long latency of data transfer for data-intensive applications. The proposed algorithm downloads a data file in parallel from

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

multiple distributed data sources to achieve large aggregated bandwidth. It is assumed that the source data can be divided into small segments of equal size and each segment is an atomic unit to be transferred. The dynamic scheduling algorithm autonomously assigns segments to each replica server to transfer and it tries to assign more segments to the faster sites. The algorithm is able to obtain higher aggregated bandwidth, reduce connection overheads and achieve superior load balance. The Grid architecture that supports co-allocating data transfer across multiple connections, enabling concurrent downloading from multiple data servers is proposed in [129]. Several co-allocating strategies, such as simple brute-force, history-based and dynamic load balancing are developed to exploit the rate difference among the client-server connections and to address dynamic rate fluctuations. The strategies are evaluated using the GridFTP [25-27] data transfer protocol in wide-area testbed and claimed to be able to greatly improve the network performance.

The authors in [130] propose a parallel download scheme that takes into consideration both the server output bandwidth capacity and client input bandwidth capacity. It is pointed out that the number of servers chosen for the parallel download depends on both the capacity that the server can serve downloads requests and that the client can receive. The proposed scheme has three stages: initial stage, steady stage and end stage. At the initial stage, the parallel download parameters, such as block size and server selection according to server speed and priority, are determined. During the steady stage, the client download speed is monitored continuously. When the download speed is below the maximum of the client input bandwidth, the algorithm will explore the possibility of adding more source servers to increase the download efficiency. The end stage takes special care of the last blocks of the download file. Techniques are employed to prevent slower or crashed servers from slowing down the data transfer. The proposed parallel download scheme is claimed to be able to improve the data download efficiency without overusing the servers.

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

All the above-mentioned researches try to use parallel download approach to speed up the data download process, however the complete data file needs to be replicated over the Data Grid in their approaches. Due to the continuously increasing number of data sets and the large size of data, it will be more efficient to store only the required part of the data file on the replica server, while the client can still download the complete file from different replica servers.

The authors in [131] proposed to store only the required replica fragments instead of the complete replica on the server to save the storage space. A block mapping procedure is proposed to manage the fragmented replica selection and retrieval in Data Grid. The block mapping procedure determines the distribution of the replica blocks in the replica server for later retrieval. The procedure enables the replica server to provide its replica fragments to other members in the Data Grid system. Some co-allocation schemes are employed to retrieve data sets from the servers, after the block mapping-procedure. The download performance is improved using these co-allocation schemes. This research focuses on the implementation of parallel download mechanism and its effect on network performance. However it did not study the effect of parallel download on job scheduling performance, which also needs to take into consideration the availability of compute resources. In our research, we study the impact of parallel download on job scheduling performance in the Data Grid environment, in terms of geometric mean of job turnaround time.

A framework that supports both job scheduling and data replication in the Data Grid is proposed in [101]. Several replication algorithms are put forward and their performance are studied together with some scheduling heuristics. The data replication algorithms proved to be able to improve the scheduling performance from the perspective of job computing time. This framework lays the foundation for our work in this chapter. We have extended the framework to support replicating and parallel downloading data fragments; and compared its performance with the original non-parallel downloading system.

5.3 System Model and Performance Metric

Figure 5.1: System Architecture of the Studied Data Grid

Figure 5.1 shows the system architecture of the Data Grid studied in this chapter. The Data Grid comprises of Grid Scheduler(s), Replica Servers and Compute Servers. The Grid Scheduler is in charge of the job scheduling at the Grid Level, running Grid Scheduling algorithm. The data files, which are input data for Data Grid Jobs, are replicated across the Replica Servers according to the replication algorithm. The Compute Servers provide compute power for Grid Jobs. Each Compute Server has a job queue to store the jobs assigned by the Grid Scheduler and it runs local scheduling algorithm to execute these jobs. The job can only

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

start to run on the Compute Server when its required input data is available in the cache of the Compute Server.

For n-PDS, the replication algorithm places the complete replica of a data file across the replica servers. When a data file is needed, it is downloaded from the replica server that has the highest available bandwidth to the compute server.

For PDS, the data file is split into fragments and the fragments are replicated across multiple replica servers. The replication algorithm will make sure that a complete replica can be re-assembled from the different fragments across the replica servers. When the data file is needed, the replica fragments can be downloaded concurrently from different servers and re-assembled as a complete data file in the compute server's cache before a job can start.

5.3.1 Job Turnaround Time

Job Turnaround Time (TT) measures how long it takes from job submission to job completion. It is the main performance metric for the open online system [132]. Each job is dispatched to one compute server by the Grid Scheduler and queued at the compute server's queue. It has to wait for the input data to be downloaded to the compute server's cache store, before it can start the computation. Thus, the turnaround time can be calculated using:

$$TT_j = \text{Max}\{QT_j, DT_j\} + ET_j \quad (5.1)$$

Where QT_j is the queuing time for job j waiting for all jobs ahead of it to finish; DT_j is the time to download the input data; and ET_j is the time to execute job j on the compute server.

The Arithmetic Mean (AMTT) and Geometric Mean (GMTT) of Turnaround Time are defined as follows:

$$AMTT = \frac{\sum_{j \in J} TT_j}{|J|} \quad (5.2)$$

 CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

$$GMITT = \left(\prod_{j \in J} TT_j \right)^{\frac{1}{|J|}} \quad (5.3)$$

Where J is the job set; $|J|$ is the number of jobs in set J and TT_j is the turnaround time of job j .

Given an example of 10 jobs, among which 9 jobs have a TT of 1 second each and 1 job has a TT of 10,000 seconds; the AMTT can be calculated as about 1000 seconds through equation (5.2). Using equation (5.3), the GMITT is calculated as about 1.096 second. As we can see from the results, the AMTT is dominated by long jobs, while the GMITT treats all jobs equally [133]. Thus, the GMITT is used as the main performance metric in our research and a lower GMITT implies better performance.

5.3.2 CPU Utilization

CPU utilization measures the percentage that the CPU is busy over a period of time. It is used to determine whether the compute resource is used efficiently. The CPU utilization for compute server i can be calculated using:

$$CU_i = \frac{T_{b,i}}{T_s} \quad (5.4)$$

Where $T_{b,i}$ denotes the busy time for server i and T_s denotes the total simulation time.

The Overall CPU Utilization (OU) is the average CPU utilization of all the compute servers in the system and it is defined as:

$$OU = \frac{T_b}{|CS|} \times \frac{1}{T_s} = \frac{\sum_{i \in CS} T_{b,i}}{|CS| \times T_s} \quad (5.5)$$

Where T_b denotes the total server busy time; T_s denotes the simulation time; CS denotes the set of all the compute servers; and $|CS|$ denotes the number of compute servers.

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

In a simulation with fixed job workload, the estimation of total server busy time can be calculated using:

$$\tilde{T}_b \approx \frac{\sum_{j \in J} T_j}{C} \quad (5.6)$$

Where J is the set of all the jobs; T_j is the computing time for job j running on one server with unit compute power; and C is the average compute power of all the servers.

In our research, we simulate the system with 100,000 jobs and T_j has a mean of 5 hours, i.e. 18,000 seconds. Each compute server has a compute power of 10. Thus, the total server busy time T_b is about 180,000,000 seconds using equation (5.6). The job arrival rate is 1 job per 10 seconds following Poisson distribution, so the last job will arrive about 1,000,000 seconds after the simulation begins. Thus, T_s is greater than 1,000,000 seconds. There are 200 compute servers in the system. Using equation (5.5), the maximum OU expected is about 90% with efficient job scheduling.

If we run the same workload with same number of compute servers, but each compute server has a compute power of 100. The total server busy time T_b will be about 18,000,000 seconds. Thus, the maximum OU expected is only about 9% with efficient job scheduling.

5.4 Theoretical Analysis with M/M/m Queuing Model

The M/M/m queuing model can be used to analyze the system performance where the incoming jobs have Poisson arrival rate, system has multiple servers in parallel and the service rates of these servers follow exponential distribution. Figure 5.2 shows a simple representation of an M/M/m queue [134]. The jobs arrive with rate of λ and are dispatched to one of the m compute servers. Each compute server serves the jobs with service rate of μ .

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

Figure 5.2: Modeling Job Scheduling Performance with M/M/m Queue [134]

In our research, we have studied a system where the jobs arrive following Poisson distribution; the job computing time is exponentially distributed; and each compute server has the same processing capacity. Thus, assuming the delay due to data transfer is negligible, the scheduling performance can be analyzed using M/M/m queue.

For the system with the following key parameters:

Job arrival rate: $\lambda = 1/10$ (job per second)

Server service rate: $\mu = 1/1,800$ (job per second)

Number of servers: $m = 200$

Using the M/M/m model, we can derive the following results:

Average job queuing time: $T_q = 8.57$ seconds

Average job turnaround time: $T = 1808.57$ seconds

The detailed calculation can be found in Appendix B.

5.5 Replication Algorithms

The data files are replicated across the Data Grid to improve the performance. However the storage size is limited, we cannot replicate all of the data files. Replication algorithm is used to determine which data files to replicate, where to replicate them and when to do the replication. Our proposed parallel downloading system further enables replicating only part of the complete data file on the data servers to save storage space, while preserving the file completeness in the Data Grid environment. The technique used is a general approach and can be applied to any existing complete-file replication algorithms.

Several replication algorithms are proposed in [101]. In our research, we have simply chosen the Centralized Dynamic Replication algorithm: Server Merit Oriented Replica Placement (CDR_SMPlace) as an example to compare the performance of non-Parallel downloading system and parallel downloading system.

5.5.1 The non-Parallel Downloading System (n-PDS)

For the non-Parallel Downloading System, the complete data file is replicated. CDR_SMPlace is invoked periodically to replicate the frequently accessed data files. It tries to place popular replicas to the replica server which has a smaller location merit value. The location merit [101] of replica server 's' is defined as:

$$Merit_s = \sum_{i \in CS} \frac{C_i}{B_{i,s}} \quad (5.7)$$

Where C_i is the compute power of server i; CS is the set of all the compute servers; and $B_{i,s}$ is the bandwidth between compute server i and replica server 's'.

CDR_SMPlace consists of two main steps:

Step 1: Process the historical data access records to find the popular data files to be replicated. There is a central replication master in the system, which oversees all the replica servers. Each replica server keeps a record of how many

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

times each file is accessed. All the data access records from each replica server are collected and sent to the central replication master. The replication master aggregates these historical access records and sums the number of access (NoA) for the same file. The file with a larger NoA means that it is more popular than other files and will have a higher priority to be replicated. The algorithm also ensures that the most popular files are replicated more than once if necessary.

Step 2: For each selected data file in step 1, find the replica server with the minimum location merit, calculated using equation (5.7), to store it. A smaller value of location merit implies that the replica server is in a more crucial location. The replication algorithm terminates when all the selected popular data files are replicated or when the storage space is used up.

5.5.2 The Parallel Downloading System (PDS)

The CDR_SMPlace algorithm is modified to support replicating data fragments and downloading data in parallel, while preserving the file completeness. A data file is split into fragments and stored on several replica servers. They can be downloaded concurrently from multiple replica servers when needed. The modified algorithm is described below.

Initially the replica servers are randomly partitioned into groups. Each group consists of N replica servers, where $N \in \{2, 3, 4, \dots\}$. In the case where the total number of replica servers ' n ' is not divisible by N , the last group will consist of $(n \bmod N)$ replica servers.

Step 1: Run the CDR_SMPlace as described in section 5.5.1 to get the list of files to be replicated on each replica server.

Step 2: For each group of the replica servers, merge the replica lists of all the group members to get the group replica list. For every replica in the group replica list, divide it into N fragments and replicate the N fragments across the N replica servers in this group.

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

The portion of the replica to be replicated on server ‘s’ can be calculated using:

$$p_s = \frac{S_s}{\sum_{i \in G} S_i} \quad (5.8)$$

Where S_s denotes the storage capacity of replica server ‘s’; and G denotes the set of all the replica servers (including server ‘s’) in the same group as server ‘s’.

In our research, we have implemented the algorithm with $N=2$. The following example demonstrates how the algorithm works with $N=2$. In this example, replica server A and B are grouped together. Server A has a storage capacity of 30GB and Server B has a storage capacity of 20GB. After running CDR_SMPlace in step 1, the list of files to be replicated on server A is {file 1, file3, file 9}, i.e. *List A* in Figure 5.3. And the list of files to be replicated on server B is {file 8, file 11, file 13}, i.e. *List B* in Figure 5.3. In step 2, we combine the two lists and get the *Group List* to be replicated on both server A and B, i.e. {file 1, file 3, file 9, file 8, file 11, file 13}. Using equation (5.8), the portion to be replicated on server A is 60%. So we can replicate the first 60% of all the files in the *Group List* on replica server A and replicate the remaining 40% of these files on replica server B.

Figure 5.3: An Example of the Replica Lists

For example, when file 9 is needed, the first 60% will be downloaded from replica server A and the remaining 40% will be downloaded from server B. The two parts can be downloaded concurrently and re-assembled as file 9 when both

parts are downloaded to the compute server. In this manner, file completeness is preserved.

5.6 Scheduling Algorithms

As shown in Figure 5.1, all jobs are submitted to the Grid Scheduler, which will dispatch the jobs to the compute servers according to some scheduling algorithms. The jobs are then queued or executed on the compute server, following the local scheduling policy. The local scheduling policy adopted in our research is First-Come-First-Serve (FCFS). The Grid Scheduling algorithms used in our research are presented in the following sub-sections.

5.6.1 Shortest Turnaround Time (STT)

The Shortest Turnaround Time (STT) algorithm schedules a new job to the compute server that leads to the shortest turnaround time for the job. The turnaround time of job j , if scheduled to compute server i , is estimated using:

$$\tilde{T}_{j,i} = \max\{\tilde{Q}T_{j,i}, \tilde{D}T(f_j, i)\} + \tilde{E}T_{j,i} \quad (5.9)$$

Where $\tilde{Q}T_{j,i}$ is the estimated queuing time for job j on server i ; f_j denotes the input file of job j ; $\tilde{D}T(f_j, i)$ is the estimated time to download file f_j to server i and $\tilde{E}T_{j,i}$ is the estimated execution time for job j running on server i .

When job j is submitted, the user needs to specify its estimated execution time BT_j , running on a compute server with unit compute power. Thus, the estimated execution time of job j running on server i is:

$$\tilde{E}T_{j,i} = \frac{BT_j}{C_i} \quad (5.10)$$

Where C_i denotes the compute power of server i .

To estimate the queuing time, we assume that the transmission of input data does not delay the job start time, i.e. every job in the queue will start to run

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

immediately after the previous job finishes. Thus, the estimated queue time for job j on server i is:

$$\tilde{Q}T_{j,i} = \frac{\sum_{b \in Q(i)} BT_b}{C_i} + \tilde{R}T_{k,i} \quad (5.11)$$

Where $Q(i)$ denotes the set of jobs that are already in the queue of compute server i ; $\tilde{R}T_{k,i}$ denotes the remaining time of job k , which is currently running on server i .

If there is no job running on server i , $\tilde{R}T_{k,i} = 0$. Otherwise, it can be calculated as follow:

$$\tilde{R}T_{k,i} = \max(0, \frac{BT_k}{C_i} - PT_{k,i}) \quad (5.12)$$

Where $PT_{k,i}$ is the time that job k has already been running in compute server i . In the case BT_k is under-estimated by user, $\tilde{R}T_{k,i}$ will be ensured not to be smaller than 0.

As for the estimated data download time, $\tilde{D}T(f_j, i)$ is always 0 for the case of scheduling pure computing jobs without input data. For scheduling jobs with input file, if the input file f_j is already available in the compute server i 's cache store, $\tilde{D}T(f_j, i)$ is 0. Otherwise, f_j will be downloaded from one replica server for n-PDS; and downloaded from N replica servers for PDS.

For n-PDS, it will download the data from replica server m , which contains f_j and has the highest bandwidth to compute server i among all the replica servers that contain replica f_j . The estimated data download time is:

$$\tilde{D}T(f_j, i) = \frac{F_j}{B_{i,m}} \quad (5.13)$$

Where F_j is the size of file f_j ; and $B_{i,m}$ is the network bandwidth between compute server i and replica server m .

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

Let M denotes the set of replica servers which belong to the same group with server m . For PDS, it will download the data from all the N replica servers in set M . The estimated data download time is:

$$\tilde{DT}(f_j, i) = \max \left\{ \frac{F_j \times P_n}{B_{i,n}} \text{ for } \forall n \in M \right\} \quad (5.14)$$

Where p_n is the portion of file f_j that resides in replica server n ($0 \leq p_n \leq 1$ and $\sum_{n \in M} P_n = 1$).

For each new job, the STT algorithm estimates its turnaround time on every compute server using equation (5.9). The compute server which provides the minimum estimated turnaround time will be chosen to run the job.

5.6.2 Least Relative Load (LRL)

The Least Relative Load (LRL) algorithm tries to achieve load balance among compute servers, by scheduling new jobs to the compute server that has the least relative workload. The relative load is simply defined as:

$$RL_i = \frac{N_i}{C_i} \quad (5.15)$$

Where N_i denotes the total number of jobs in compute server i , including both running job and queued jobs; and C_i denotes the compute power of server i .

5.6.3 Data Present (DP)

The input files of Data Grid jobs are usually huge in size and it takes a long time to stage the large data to the compute server. The Data Present (DP) algorithm is inspired by the idea of moving the Data Grid job towards the required input data. The algorithm first tries to find a compute server which already has the required input file in its cache store. If there is only one such compute server, the job will be assigned to this server. In the case of multiple compute servers having the data file, the one with the least relative load will be chosen. However, if no compute server has the data file in its cache store, the algorithm will estimate the time to

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

download the input data for each compute server. As described in section 5.6.1, the data download time can be estimated using equation (5.13) for n-PDS and equation (5.14) for PDS. The compute server with the shortest download time will be chosen to run this job.

For the case of scheduling pure computing jobs (i.e. no data transfer involved), it can be considered as a special case that all the data files are already cached in the compute server. In this case, the DP algorithm is the same as the LRL algorithm.

5.7 Simulation Environment

5.7.1 The Data Grid Environment

As shown in Table 5.1, there are 25 replica servers and 200 compute servers in the simulated Data Grid environment. Each compute server has a compute power of 10, and a data cache to store the data files required by the jobs submitted to this server. The data file in the cache will be deleted if it is not referred to by any running job or jobs in the queue.

Table 5.1: Simulation Parameters

Parameters	Values
number of replica servers	25
number of compute servers	200
number of different data files	10,000
data file size	U(500MB,5GB,100MB)
file popularity distribution	Zipf-like ($\alpha=1.0$)
server relative capacity	50%
number of jobs	100,000
job inter-arrival time	10 second

There are 10,000 different data files in the system and the file size is set to U(500MB,5GB,100MB), where U(a,b, Δ) denotes a random number with uniform

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

distribution, ranging from a to b with granularity of Δ . Each data file has a primary copy. The primary copies are randomly distributed among the replica servers and will not change over the whole simulation time.

The data file popularity follows Zipf-like distribution [122], i.e. the total number of requests for the n^{th} popular file is proportional to n^{-a} , where a is a constant. In our research, a is set to 1.0. The data file popularity will change from time to time in reality. To simulate this scenario, the simulation time is divided into 10 sub-sessions evenly; and in each sub-session, the set of popular files are different from the previous sub-sessions.

The server relative capacity is defined in section 3.3 and can be calculated using equation (3.1). In our research, the relative capacity is set to 50% and it implies that on average half of the data files can be replicated on the replica servers, in addition to their primary copies.

All the replica servers are directly connected to each other. Each compute server has direct access to all the replica servers. For the simplicity reason, we assume that all these network links are dedicated lines and they have the same network bandwidth. We have investigated the system performance with network bandwidth of 1 Mbps, 10 Mbps and 100 Mbps.

5.7.2 The Job Workload

Researchers in [118] have tried to analyze and model the job arrival process in a Grid environment and they find out that “a Poisson process is quite likely to accurately model the arrival process behavior” [118]. Thus, in our research we have chosen to use the Poisson distribution to model the job workload.

In our simulation, there are 100,000 jobs in total. New jobs arrive following a Poisson process with a rate of 0.1 per second. The workload format follows the standard defined in [135]. In real life, different jobs will have different computing time even on the same compute server. We use exponential distribution to model the different computing time of the jobs. The actual computing time of each job,

 CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

running on one compute server with unit compute power, is exponentially distributed with mean of 5 hours, i.e. 18,000 seconds. The Grid scheduler requires the users to give an estimated job computing time when they submit the jobs to the Data Grid system. Since it is difficult for the users to estimate the actual computing time of the job, it is very likely that there will be some difference between the user estimated time and the actual job computing time [136]. The following method, which is derived from the model in [137], is used to generate the estimated job computing time. For a given job with actual computing time of t_a , its estimated computing time t_e is a random variable follows Gaussian distribution with mean of t_a and standard deviation of $3t_a$.

5.8 Simulation Results

In this section, we report the simulation results, which are based on the performance metrics described in section 5.3. The results are labeled “optimal” for scheduling pure computing jobs without data transfer; “n-PDS” for non-parallel downloading system; and “PDS” for parallel downloading system.

Figure 5.4 shows the simulation results in terms of job turnaround time. For the case of scheduling pure computing jobs without data transfer (Category “optimal”), all the three scheduling algorithms have similar performance in terms of both Arithmetic Mean of Turnaround Time (AMTT) and Geometric Mean of Turnaround Time (GMTT). In terms of AMTT, the simulation results show that it is around 1800 seconds for all three scheduling algorithms and this tallies with the result derived theoretically using M/M/m queuing model in section 5.4. The scheduling algorithms have the best performance, i.e. lowest AMTT and GMTT, when there is no data transfer required. We use this best performance as the benchmark to compare with the performance of both Parallel Downloading System (PDS) and non-Parallel Downloading System (n-PDS). This will help us understand how the data transfer affects the job scheduling performance in terms of GMTT and overall CPU utilization.

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

(a). AMTT Comparison

(b). GMTT Comparison

Figure 5.4: Job Turnaround Time Comparison

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

From Figure 5.4, we can see that PDS has smaller AMTT and GMTT, i.e. better performance, than n-PDS for all three scheduling algorithms. The only exception is for DP scheduling algorithm, where both PDS and n-PDS have similar AMTT and PDS outperforms n-PDS in terms of GMTT. This is because AMTT is dominated by long jobs and does not reflect the performance fairly, as described in section 5.3.1. In the following, we will mainly use GMTT as the performance metric.

Figure 5.5(a) shows the overall CPU utilization. Figure 5.5(b), 5.5(c) and 5.5(d) show more details on the CPU utilization distribution by reporting the number of compute servers falling into different ranges. The X-axis indicates CPU utilization range in percentage that is equal to or greater than the indicated value, but less than the value to its right. For example, a value of 20 indicates a CPU utilization range of 20% (including) to 30% (excluding). The Y-axis represents the number of compute servers that fall into each CPU utilization range. For example, Figure 5.5(b) shows that for STT the CPU utilization of 191 compute servers is within the range of 80%-90% and 9 within the range of 90%-100%.

From Figure 5.5(a), we can see that category “optimal” has the highest overall CPU utilization of about 89.5%. This is because the CPU does not need to wait for the data transfer and it tallies with the result that category “optimal” has the best performance in Figure 5.4. Figure 5.5(d) shows that all the three algorithms have similar distribution of CPU utilization for the 200 compute servers when there is no data transfer involved. This tallies with the result that all the three scheduling algorithms have similar performance in terms of both AMTT and GMTT in Figure 5.4.

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

(a). Overall CPU Utilization

(b). CPU Utilization Distribution for n-PDS

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

(c). CPU Utilization Distribution for PDS

(d). CPU Utilization Distribution without Data Transfer

Figure 5.5: CPU Utilization Comparison

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

For STT and LRL, the overall CPU utilization of PDS is less than 1% higher than n-PDS as shown in Figure 5.5(a). The CPU utilization distribution for both algorithms are quite similar. However, Figure 5.4(b) shows PDS has achieved 44% decrease for STT and 30% decrease for LRL in terms of GMTT comparing to n-PDS. Since the job turnaround time is determined by both computing time (queuing time and execution time) and data transfer time as shown in equation (5.1), the performance improvement in this case is mainly coming from reduction of data transfer time.

Among the three schedulers, STT has the best performance and DP has the worst performance in terms of both AMTT and GMTT as shown in Figure 5.4. This tallies with the fact that STT has the highest CPU utilization and DP has the lowest CPU utilization in Figure 5.5(a). As shown in Figure 5.5(b) and 5.5(c), the CPU utilization of most of the compute servers is less than 70% for DP. While for STT and LRL, CPU utilization of most of the servers is higher than 80%. This shows that STT and LRL can achieve much better load balance than DP in this case. It is because DP algorithm favors the compute servers that already have the input data in their cache and this will sometimes lead to unbalanced load. The inefficient use of compute power has resulted in the bad performance of algorithm DP comparing to the other two algorithms.

5.8.1 Impact of Network Bandwidth

We have studied the scheduling performance for both PDS and n-PDS systems with different network bandwidth of 1 Mbps, 10 Mbps and 100 Mbps and the results are shown in Figure 5.6, 5.7 and 5.8 respectively.

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

(a). GMTT

(b). Overall CPU Utilization

Figure 5.6: Results for 1 Mbps Network

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

(a). GMTT

(b). Overall CPU Utilization

Figure 5.7: Results for 10 Mbps Network

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

(a). GMTT

(b). Overall CPU Utilization

Figure 5.8: Results for 100 Mbps Network

 CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

Comparing Figure 5.6(a) with 5.7(a), we can find that GMTT for both systems decreases when the network bandwidth is increased. This is as expected since high network bandwidth will result in reducing Download Time (DT) and possibly Turnaround Time (TT). The major improvements are for jobs scheduled using algorithm STT and LRL, where the GMTT is decreased by a factor of close to 10. This shows that DT is the major component of TT when the network bandwidth is relatively small. This phenomenally tallies with equation (5.1), when the download time DT has a major impact on TT. The performance improvement for both algorithms is mainly from reducing DT.

Comparing PDS with n-PDS, both Figure 5.6(a) and 5.7(a) show that PDS has a shorter GMTT, i.e. better performance over n-PDS for all the three scheduling algorithms. This is because PDS can greatly reduce DT, which is a major component of TT especially when network bandwidth is constrained. The figures also show that PDS results in higher overall CPU utilization, as the chance of “CPU waiting for data” is reduced for STT and LRL. As for scheduling algorithm DP, the overall CPU utilization is doubled as shown in Figure 5.6(b). This is because with faster data transfer by PDS, the compute server will have higher chance to remove the cache file before it is otherwise used and pinned (reserved) by new jobs. In this way, the cache is refreshed more frequently instead of always staying in the compute server. This will prevent DP from keep on scheduling jobs to one “hot pot” server and thus improve the situation of unbalanced load.

When we increase the network bandwidth to 100 Mbps as shown in Figure 5.8, the performance improvement using PDS is not so evident and there is not much change in terms of overall CPU utilization. This is because the network bandwidth is already high enough to bring the GMTT quite close to the optimal performance shown in Figure 5.4(b). Thus, the reduction of data transfer time by PDS does not have much impact on the performance. The bad performance of DP is once again caused by its unbalanced load as indicated by the low overall CPU utilization shown in Figure 5.8(b).

5.8.2 Impact of Compute power

We have also run the simulations for both n-PDS and PDS with the compute power of each server increased by a factor of 10 (i.e. each compute server power is increased from 10 to 100), and with 10 Mbps network bandwidth for example. The new results are shown in Figure 5.9.

Comparing Figure 5.9(a) with Figure 5.7(a), we can find that GMTT for both systems decreases, i.e. the performance becomes better. This is as expected since high compute power means Execution Time (ET), which directly contributes to TT, is shortened. The performance improvement is most significant for scheduling algorithm DP, which has about 80% decreases in GMTT. For STT and LRL, the GMTT is decreased by about 60%. With high compute power, the server loads become more balanced for DP, as shown in Figure 5.9(b).

Figure 5.9(b) shows that category “optimal” has an overall CPU utilization of about 9% and it tallies with the expected result as analyzed in section 5.3.2. Both n-PDS and PDS system also have almost the same overall CPU utilization as category “optimal”. The relatively low figure of 9% means that there is more than enough compute power in the system. In this case, compute power is not the bottleneck of performance and the data download time plays a major part in the job turnaround time as shown in Figure 5.9(a).

Figure 5.9(a) shows that n-PDS has a GMTT of almost 20 times that of the system without data transfer. This is because the job turnaround time is dominated by the data transfer time, provided that the compute power is high enough. In this case, the 10 Mbps network becomes the performance bottleneck. PDS can significantly reduce the data transfer time and that is why it can reduce GMTT by 35%-45% compared to n-PDS as shown in Figure 5.9(a).

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

(a). GM TT

(b). Overall CPU Utilization

Figure 5.9: Results for 10 Mbps Network, and High Compute Power of 100

5.9 Discussion

For n-PDS, the data file is downloaded from one single replica server. The time to download file f_j to the compute server i from replica server m is:

$$DT(f_j, i) = \frac{F_j}{B_{i,m}} \quad (5.16)$$

Where F_j is the size of file f_j ; and $B_{i,m}$ is the bandwidth between server i and server m .

For PDS, the data file is concurrently downloaded from a group of N replica servers. The time to download file f_j to compute server i from the group M where replica server m belongs to is:

$$DT(f_j, i) = \text{Max}\left\{\frac{F_j \times P_n}{B_{i,n}} \text{ for } \forall n \in M\right\} \quad (5.17)$$

Where p_n is the portion of file f_j that resides in replica server n ($0 \leq p_n \leq 1$) and $\sum_{n \in M} P_n = 1$.

For the case where $B_{i,n} (\forall n \in M)$ has similar values, it is very clear that PDS has a smaller download time, since $0 \leq p_n \leq 1$. In our research, we have simulated this scenario for simplicity. For the case where $B_{i,n} (\forall n \in M)$ have a wide range of different values, it becomes more complex. The system must be smart enough to utilize the high network bandwidth, i.e. assign large portion of the data file to fast replica server and p_n should take into consideration the network bandwidth.

Both n-PDS and PDS have better performance with higher network bandwidth and higher compute power. This is reasonable as high network bandwidth will decrease the download time DT in equation (5.1); and high compute power will reduce the queuing time QT and the job execution time ET .

PDS can provide much better performance, compared to n-PDS, especially when the Data Grid has relatively low network bandwidth and relatively high compute

CHAPTER 5. IMPACT OF PARALLEL DOWNLOAD ON JOB SCHEDULING

power. When the compute power is large, the job queuing time QT and job execution time ET are relatively small compared to data download time DT. Thus, DT plays the major role in the job turnaround time. The parallel downloading system improves the Data Grid performance by greatly reducing the data download time.

5.10 Summary

In this chapter, we have studied the impact of parallel download on job scheduling performance in Data Grid environment. In our proposed parallel download approach, the data file is split into fragments and stored on different replica servers. The replica fragments are transferred concurrently from multiple replica servers when needed. The parallel download approach can better utilize the network bandwidth and speed up the data access. We have compared the performance of the proposed parallel downloading system with the traditional non-parallel downloading system using three different Grid scheduling algorithms. The simulation results show that the parallel downloading system has better performance, in terms of geometric mean of job turnaround time, for all the three scheduling algorithms. Parallel downloading of data files is extremely critical when the network bandwidth is limited.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Data Grid, as a specialization and extension of the Grid, has been adopted as the solution to the distribution and management of large amount of data, which are geographically distributed in collaborative computing environment, in areas such as high energy physics, life science and etc. One representative Data Grid architecture is the Hierarchical Data Grid, which is adopted by CERN to support and analyze LHC experiments.

In this thesis, we have developed a probabilistic model to study the optimal performance of the Hierarchical Data Grid [138]. The performance metrics used include both average response time and average bandwidth cost. The model parameters include the network topology, configurations of the Data Grid storage resources, distribution of the client nodes, the replica access pattern and etc. An optimal replication algorithm for the modeled Hierarchical Data Grid is proposed and presented. The proposed algorithm aims to replicate popular files near the client nodes to save bandwidth cost and reduce data transfer time. Its performance is analyzed in details using our mathematical model, which gives the shortest average response time and least average bandwidth cost of the modeled Hierarchical Data Grid. The multi-tier Data Grid used by CERN for

CHAPTER 6. CONCLUSIONS AND FUTURE WORK

LHC experiments, as a specific example of the Hierarchical Data Grid, is used to verify our model. The performance of our proposed OPTimal replication algorithm (OPT) is studied in a simple Data Grid environment with different client access patterns. The simulation results tally with the optimal performance predicted using our mathematical model. Three well-known replication algorithms: Simple Bottom-Up (SBU), Aggregate Bottom-Up (ABU) and Fast Spread (FSP) are also studied and their performances are compared with the OPT algorithm. The simulation results show that OPT performs better than all three replication heuristics. In the simple case where the replica access pattern follows geometric distribution and does not change from time to time, the FSP algorithm has the best results, approaching the theoretical results obtained using our model.

We have also studied the performance of replication algorithms with expiration-based consistency management in the Hierarchical Data Grid environment. Previous studies on replication algorithms have simply assumed that the replica files are read-only and do not consider the replica consistency management problem. As stated in section 2.4, more and more researchers have realized the importance of the replica consistency management. Applications like the Earth Monitoring project [47] produces data that are frequently updated. Thus, we need to extend the Data Grid to support consistency management. Other than that, meta-data, which are used to describe the actual data produced by the real applications, also change from time to time. It is of interest to most of the users, to identify the actual application data. These meta-data will be updated frequently and we need to keep replicas of these meta-data consistent across the Data Grid.

In our study [139], every replica file is associated with a Time-To-Live (TTL) value and considered invalid after the TTL value becomes 0. The expiration-based consistency management is used to ensure the replicas are synchronized. Our simulation results show that ABU algorithm has the best performance with reasonable TTL value and the performance is further improved with higher relative replica server capacity. In the case where replica expires very frequently, i.e.

CHAPTER 6. CONCLUSIONS AND FUTURE WORK

replica with very small TTL value, it doesn't make sense to replicate these replicas. Every time when a client node tries to download the replica it will find that the replica TTL has expired. This will force the client node to travel upwards to the root tier to get a fresh copy. The benefits of file replication are lost. For replicas with a relatively longer and more reasonable TTL value, the advantage brought by ABU far exceeds the overhead. With a longer TTL value, the advantage of ABU is magnified because the replicated popular files can serve more download requests before they expire. Client nodes can download the valid replica from intermediate tiers instead of the root tier. Downloading files from the root tier is considered slow in terms of response time and expensive in terms of bandwidth cost. With a relatively higher replica server capacity, the replica servers can store more replicas in the middle tier to improve the system performance. ABU utilizes these servers better because it tries to replicate the popular files, while FSP simply keeps a copy of the file while transmitting to clients. That is why ABU outperforms FSP significantly with a relatively longer TTL and higher relative replica server capacity.

Data replication has proved to be a method to improve the job scheduling performance in the Data Grid environment. However traditional replication approaches only support replicating complete data files on the replica servers. Due to the limited size of the storage servers, it is not feasible for all the data files to have multiple complete replicas across the Data Grid. As the data is usually large in size and the network bandwidth is limited, downloading the input data from a single replica server will introduce a long delay before the data-intensive compute job can be started. With the compute power constantly increasing, as indicated by Moore's Law [141], the delay caused by "CPU idle waiting for input data" will become more pronounced. In our research, we have proposed a general technique to extend the existing replication algorithms to support replicating data fragments and preserve file completeness at the same time. This enables the system to download data fragments concurrently from different replica servers to the compute server and assemble them locally as a complete data file after all

CHAPTER 6. CONCLUSIONS AND FUTURE WORK

fragments are downloaded. The parallel download approach can utilize the network bandwidth better and reduce the data download time, which is an essential part of the job turnaround time especially when the network bandwidth is slow. In [140], we have compared the performance of the proposed parallel download system with traditional non-parallel download system using three different Grid scheduling algorithms. The simulation results show that the parallel download approach can greatly reduce the geometric mean of job turnaround time in all the studied cases, for all the three scheduling algorithms. The improvement is most significant when the system has limited network bandwidth and sufficient compute power.

6.2 Directions for Future Work

In this section, we present some possible future works.

6.2.1 Utilizing Virtual Connections

Traditional replication strategies for the Hierarchical Data Grid did not consider the virtual connection between peer nodes (children nodes belong to the same parent node). For example, as shown in Figure 6.1, there is no direct connection between node 3 and node 4. However the virtual connection is there since node 3 and node 4 are connected through the same parent node 1. In the scenario shown in Figure 6.1, the replica is not available at node 1 or node 3, but available at node 0 and node 4. In the case that clients from node 3 request to access the replica, previous approaches will download the replica from node 0 to node 1 and then node 3. However, given the virtual connection between node 3 and node 4, it is possible that the replica can be downloaded from node 4 also. This new choice is likely to be preferable if the virtual connection between node 3 and 4 is faster and cheaper than the direct link between node 0 and node 1. With this new choice, the replica management system needs to make decisions whether to use direct connection or virtual connection to gain better performance. Furthermore it will affect the replication algorithms, because now the replication algorithms have the

choice to place the replica at the parent node or peer node. It is interesting to study the effects of the virtual connections between peer nodes on the performance of the replication algorithms. The performance of data replication strategies can be further improved by utilizing these virtual connections.

Figure 6.1: Accessing Replica through Virtual Connections between Peers

6.2.2 Integrating Data Replication and Job scheduling algorithm

Data Replication is a commonly used technique to reduce data access cost in Data Grid environment. Researchers have also studied its impact on the job scheduling performance for data intensive applications, as reported in section 2.5. Some replication algorithms are proposed to replicate popular files to replica servers, which are near fast compute sites or are close enough to serve more compute sites. These replication algorithms are studied in combination with different heuristic scheduling algorithms and have been proved to be able to improve the scheduling performance in terms of turnaround time. However, in these studies, replication algorithms and scheduling algorithms are simply put together and treated as separate algorithms. There is no direct interaction between scheduling algorithm

CHAPTER 6. CONCLUSIONS AND FUTURE WORK

and the data replication algorithm. We believe such interaction is necessary to get the best overall performance, because the decisions to schedule jobs and replicate data are closely related to each other.

The problem will become more complex when we combine both job scheduling and replication strategies. But still it is interesting to propose an integrated scheduling and replication strategy for the Data Grid and compare its performance with the previous decoupling approaches.

6.2.3 Data Replication in Cloud Computing

Cloud Computing [142-146] has become a popular research topic recently. One fundamental difference between Cloud Computing and Grid Computing is that in Cloud Computing, physical resources are virtualized using virtualization technology. Virtual machines are used to run applications, which sit on top of the physical resources. The management of these virtual machines is more flexible as they can be dynamically started, stopped or migrated. Traditional Grid job will be submitted by the Grid scheduler to one compute server and finish running there; or resubmitted if any error occurs. In cloud computing, the job will be submitted to virtual machines, which can be migrated to another physical server dynamically. When it comes to data intensive applications, this movement of virtual machines will need to take into consideration the input data. If the input data is available at the target physical server, the migration of virtual machine will only need to involve the running job processes. Otherwise the input data will also need to be moved together with the running job every time the virtual machine is migrated. The migration of virtual machines may be due to server malfunction, or server scheduled shutdown, or to migrate virtual machines to a single server and shutdown others to save energy cost, and etc. The data replication algorithm will need to take into consideration the virtual machine migration policies, for example replicating data files near high priority physical servers where virtual machines will be assigned to frequently. It is interesting to study the impact of migration of virtual machine on the data replication strategy;

and propose new data replication algorithms that serve the needs of cloud computing applications.

6.2.4 Job Scheduling and Data Replication in Hadoop

MapReduce [147] is a programming model proposed by Google [148] researchers to simplify the processing of large scale data sets. With this new programming style, programmers need to write their applications using *map* and *reduce* [147] functions, which will be automatically able to run in parallel on a large cluster of commodity machines. The runtime system will take care of the partitioning of input data, scheduling map/reduce jobs, handling failed jobs and etc. One popular implementation of the MapReduce model is the Apache Hadoop [149], which provides the software framework and runtime system for data-intensive applications written in MapReduce style. In Hadoop, the default job scheduling algorithm is First-Come-First-Serve (FCFS) [150], and currently two schedulers are developed by the community to allow fair sharing of compute resources among multi-user workloads: Fair Scheduler [151] developed at Facebook [152] and Capacity Scheduler [153] developed at Yahoo [154].

The data replication in Hadoop is mainly for the purpose of fault tolerance, i.e. in the case of hardware failure, remaining replicas of the lost data sets can be used. For the example of replication factor of 3, each data block is replicated in three nodes: two nodes in the same rack and one node in a different rack [155]. At runtime, the job can select the closest replica of its input data, where the network connection within the same rack is much faster than inter-rack connection. This replication strategy treats all the data files the same, without differentiating the popularity of the files. In case of a sudden increase in demand for one popular file, the data access time will slow down the system performance dramatically using the current scheduling algorithm and replication strategy. It is of our interest that the system performance can be improved if we schedule the jobs close to the nodes that are storing the input data; and at the same time, actively replicate the data files, especially the popular ones, to the nodes with more available compute

CHAPTER 6. CONCLUSIONS AND FUTURE WORK

power to balance the workload. In this way, the scheduling algorithm and data replication algorithm can interact more actively to gain better performance.

Author's Publications

Journal Papers:

- 1) **J.W. Zhang**, B.S. Lee, X.Y. Tang and C.K. Yeo, "A Model to Predict the Optimal Performance of the Hierarchical Data Grid", Future Generation Computer Systems (FGCS): The International Journal of Grid Computing: Theory, Methods and Applications, Issue 26, Volume 1, Jan 2010, pp. 1-11 (Published)
- 2) **J.W. Zhang**, B.S. Lee, X.Y. Tang and C.K. Yeo, "Improving Job Scheduling Performance with Parallel Access to Replicas in Data Grid Environment", The Journal of Supercomputing: An International Journal of High-Performance Computer Design, Analysis, and Use (**Accepted**)

Conference Papers:

- 3) **J.W. Zhang**, B.S. Lee, X.Y. Tang and C.K. Yeo, "Impact of Parallel Download on Job Scheduling in Data Grid Environment", In Proceedings of the Seventh International Conference on Grid and Cooperative Computing (GCC'08), Oct. 2008, Shenzhen, China (**Best Paper Award**)
- 4) **J.W. Zhang**, B.S. Lee, X.Y. Tang, M. Tang and C.K. Yeo, "Analysis of Dynamic Replication Algorithms with Expiration-based Consistency Management for Multi-Tier Data Grid", In Proceedings of the 9th International Conference on High Performance Computing, Grid and e-Science in Asia Pacific Region (HPCAsia'07), Sep. 2007, Seoul, Korea

References

- [1] I. Foster and C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, Los Altos, CA, 1999
- [2] I. Foster, The Grid: A New Infrastructure for 21st Century Science, Grid Computing: Making the Global Infrastructure a Reality, John Wiley & Sons, Inc, 2003, pp. 51-63
- [3] I. Foster, C. Kesselman and S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations, International Journal of High Performance Computing Applications, Volume 15, Issue 3, 2001, pp. 200-222
- [4] J. Nick, I. Foster, C. Kesselman and S. Tuecke, The Physiology of the Grid: an Open Grid Services Architecture for Distributed Systems Integration, Open Grid Service Infrastructure WG, Global Grid Forum, 22 June, 2002
- [5] A. J. G. Hey and A. E. Trefethen, The UK e-Science Core Programme and the Grid, Future Generation Computer Systems, Volume 18, Issue 8, 2002, pp. 1017-1031
- [6] L. Ferreira *et al.*, Introduction to Grid Computing with Globus, IBM Corporation, 2002. <http://www.ibm.com/redbooks>
- [7] F. Berman, G. Fox and A. J. Hey, Grid Computing: Making the Global Infrastructure a Reality, John Wiley & Sons, Inc, 2003

REFERENCES

- [8] I. Foster and A. Iamnitchi, On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing, in Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS 2003), 2003, pp. 118-128
- [9] J. Joseph and C. Fellenstein, Grid Computing, IBM Press On Demand Series, 2004
- [10] European Organization for Nuclear Research (CERN)
<http://public.web.cern.ch/Public/Content/Chapters/AboutCERN/AboutCERN-en.html>
- [11] The Large Hadron Collider (LHC) <http://lhc.web.cern.ch/lhc>
- [12] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury and S. Tuecke, The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets, Journal of Network and Computer Applications, Volume 23, Issue 3, 2000, pp. 187-200
- [13] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger and K. Stockinger, Data Management in an International Data Grid Project, in Proceedings of the First IEEE/ACM International Workshop on Grid Computing, 2000, pp. 77-90
- [14] W. Allcock, I. Foster, S. Tuecke, A. Chervenak and C. Kesselman, Protocols and Services for Distributed Data-Intensive Science, in Proceedings of ACAT2000, 2000, pp. 161-163
- [15] I. Foster and C. Kesselman, A Data Grid Reference Architecture - Draft of February 1, 2001, GriPhyN Technical Report, GRIPHYN 2001-12, 2001
- [16] I. Foster and C. Kesselman, GriPhyN/PPDG Data Grid Architecture, Toolkit, and Roadmap: The GriPhyN and PPDG Collaborations, GriPhyN Technical Report, GRIPHYN-2001-15, 2001

REFERENCES

- [17] W. E. Johnston, The Computing and Data Grid Approach: Infrastructure for Distributed Science Applications, Computing and Informatics, Special Issue on Grid Computing, Winter 2002
- [18] P. Avery, Data Grids: a New Computational Infrastructure for Data-intensive Science, Philosophical Transactions of the Royal Society Series A - Mathematical Physical and Engineering Sciences, Volume 360, No. 1795, 2002, pp.1191-1209
- [19] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel and S. Tuecke, Data Management and Transfer in High-performance Computational Grid Environments, Parallel Computing, Volume 28, Number 5, May 2002, pp.749-771
- [20] X. Y. Tang and S. T. Chanson, Analysis of Replica Placement under Expiration-Based Consistency Management, IEEE Transactions on Parallel and Distributed Systems, Volume 17, Issue 11, Nov. 2006, pp. 1253-1263
- [21] I. Foster and C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, International Journal of Supercomputer Applications, Volume 11, Issue 2, 1997, pp. 115-128
- [22] The Globus Toolkit <http://www.globus.org/toolkit/>
- [23] Internet Protocols <http://tools.ietf.org/html/rfc791>
- [24] Grid Computing on Wikipedia http://en.wikipedia.org/wiki/Grid_computing
- [25] W. Allcock, GridFTP: Protocol Extensions to FTP for the Grid, Global Grid Forum GFD-R-P. 020, 2003
- [26] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, The Globus Striped GridFTP Framework and Server,

REFERENCES

- in Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, Conference on High Performance Networking and Computing, November 12-18, 2005
- [27] GridFTP Protocol http://www.globus.org/grid_software/data/gridftp.php
- [28] File Transfer Protocol (FTP) <http://www.w3.org/Protocols/rfc959>
- [29] S. Venugopal, R. Buyya and K. Ramamohanarao, A Taxonomy of Data Grids for Distributed Data Sharing, Management, and Processing, ACM Computing Surveys, Volume 38, Issue 1, Jun. 2006
- [30] Worldwide LHC Computing Grid (LCG) <http://lcg.web.cern.ch/LCG>
- [31] The LHC Computing Grid Technical Design Report
<http://lcg.web.cern.ch/LCG/tdr/>
- [32] M. Lamanna, The LHC Computing Grid Project at CERN, in Proceedings of the IXth International Workshop on Advanced Computing and Analysis Techniques in Physics Research, November 2004, pp. 1-6
- [33] J. Shiers, The Worldwide LHC Computing Grid (Worldwide LCG), in Proceedings of the Conference on Computational Physics 2006 (CCP 2006), 2006
- [34] I. Bird, B. Jones and K. F. Kee, The Organization and Management of Grid Infrastructures, Computer, Volume 42, Issue 1, Jan. 2009, pp. 36-46
- [35] The MONARC Project <http://monarc.web.cern.ch/MONARC/>
- [36] Regional Centers for LHC computing
<http://barone.web.cern.ch/barone/monarc/RCArchitecture.html>
- [37] P. Capiluppi, World Wide Computing, Reported at the LHCC Comprehensive Review of CMS Software and Computing, October 2000
http://cmsdoc.cern.ch/cms/software/reviews/LHCC_review_oct_00/
- [38] The DataGrid Project <http://eu-datagrid.web.cern.ch/eu-datagrid/>

REFERENCES

- [39] EGEE Portal: Enabling Grids for E-science <http://www.eu-egee.org/>
- [40] F. Gagliardi and M. E. Begin, EGEE - Providing a Production Quality Grid for e-science, Local to Global Data Interoperability - Challenges and Technologies, 2005, pp. 88-92
- [41] GriPhyN - Grid Physics Network in ATLAS
<http://www.usatlas.bnl.gov/computing/grid/griphyn/>
- [42] P. Avery and I. Foster, The GriPhyN Project: Towards Petascale Virtual-Data Grids, GriPhyN Report 2000-1
- [43] Particle Physics Data Grid <http://ppdg.net/>
- [44] I. Foster *et al.*, The Grid2003 Production Grid: Principles and Practice, 13th IEEE International Symposium on High Performance Distributed Computing (HPDC-13'04), 2004, pp. 236-245
- [45] GridPP – UK Computing for Particle Physics <http://www.gridpp.ac.uk/>
- [46] G. Moloney, Belle Data Grid and the Australian Grid Program, Presented at HEP Data Grid Workshop at KEK, Tsukuba, Japan, Nov. 21 - 22, 2005
- [47] Earth System Grid (ESG) <http://www.earthsystemgrid.org/>
- [48] B. Allcock, I. Foster, V. Nefedova, A. Chervenak, E. Deelman, C. Kesselman, J. Leigh, A. Sim, A. Shoshani, B. Drach and D. Williams, High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data Grid Technologies, SC 2001, November 2001
- [49] I. Foster, E. Alpert, A. Chervenak, B. Drach, C. Kesselman, V. Nefedova, D. Middleton, A. Shoshani, A. Sim and D. Williams, The Earth System Grid II: Turning Climate Datasets Into Community Resources, Proceedings of the American Meteorological Society Conference, 2001

REFERENCES

- [50] I. Foster, D. Middleton and D. Williams, The Earth System Grid (ESG), Presentation at the DOE SciDAC PI Meeting, Napa, CA, 10-11 March, 2003 <http://www.earthsystemgrid.org/about/documentsPage.do>
- [51] BIRN - Biomedical Informatics Research Network <http://www.nbirn.net/>
- [52] BioGrid Japn <http://www.biogrid.jp/project/>
- [53] S. Date, BioGrid Project in Japan - For Accelerating Science and Industry, Presented at GGF 6, Chicago, Oct. 17, 2002
- [54] S. Date, K. Fujikawa, H. Matsuda, H. Nakamura and S. Shimojo, An Empirical Study of Grid Applications in Life Science: Lesson Learnt from BioGrid Project in Japan, International Journal of Information Technology, 2005
- [55] International Virtual Observatory Alliance (IVOA) <http://www.ivoa.net/>
- [56] EURO-VO: The European Virtual Observatory <http://www.euro-vo.org/pub/>
- [57] Astronomical Grid Community Research Group
<http://www.ivoa.net/cgi-bin/twiki/bin/view/IVOA/AstroRG>
- [58] eDiaMoND - Diagnostic Mammography National Database Project
<http://www.ediamond.ox.ac.uk>
- [59] NEESgrid: Virtual Collaboratory for Earthquake Engineering
<http://www.neesgrid.org/>
- [60] B. Spencer *et al.*, NEESgrid: A Distributed Virtual Laboratory for Advanced Earthquake Experimentation and Simulation, 13th World Conference on Earthquake Engineering, Vancouver, Canada, August 2004
- [61] NEES Cyberinfrastructure <http://www.nees.org/it/>
- [62] GDMP Home <http://project-gdmp.web.cern.ch/project-gdmp/>

REFERENCES

- [63] M. Hafeez, A. Samar and H. Stockinger, A Data Grid Prototype for Distributed Data Production in CMS, VII International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT2000), October 2000
- [64] A. Samar and H. Stockinger, Grid Data Management Pilot (GDMP): A Tool for Wide Area Replication, IASTED International Conference on Applied Informatics (AI2001), Innsbruck, Austria, February 2001
- [65] H. Stockinger, A. Samar, B. Allcock, I. Foster, K. Holtman and B. Tierney, File and Object Replication in Data Grids, Proceedings of the tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001
- [66] The Compact Muon Solenoid (CMS) Experiment <http://cms.cern.ch/>
- [67] K. Holtman, CMS Data Grid System Overview and Requirements, The Compact Muon Solenoid (CMS) Experiment Note 2001/037, CERN, Switzerland, 2001
- [68] Grid Datafarm - Gfarm File System <http://datafarm.apgrid.org/>
- [69] O. Tatebe, Y. Morita, S. Matsuoka, N. Soda and S. Sekiguchi, Grid Datafarm Architecture for Petascale Data Intensive Computing, Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2002), pp.102-110
- [70] O. Tatebe, S. Sekiguchi, Y. Morita, S. Matsuoka and N. Soda, Worldwide Fast File Replication on Grid Datafarm, Proceedings of the 2003 Computing in High Energy and Nuclear Physics (CHEP03), La Jolla, Ca, USA, March 2003
- [71] A. Takefusa, O. Tatebe, S. Matsuoka and Y. Morita, Performance Analysis of Scheduling and Replication Algorithms on Grid Datafarm Architecture for High-Energy Physics Applications, Proceedings of the 12th IEEE

REFERENCES

- International Symposium on High Performance Distributed Computing (HPDC'03)
- [72] SRB – Storage Resource Broker
http://www.sdsc.edu/srb/index.php/Main_Page
- [73] C. Baru, R. Moore, A. Rajasekar and M. Wan, The SDSC Storage Resource Broker, in Proceedings of CASCON'98 Conference , Toronto, Canada, Nov. 30 - Dec. 3, 1998
- [74] A. Rajasekar, Replicated Data Management using SRB, GGF 4, Toronto, Feb 2002
- [75] R. Moore, A. Rajasekar and M. Wan, Data Grids, Digital Libraries and Persistent Archives: An Integrated Approach to Publishing, Sharing and Archiving Data, in Proceedings of the IEEE, Volume 93, Issue 3, March 2005, pp. 578-588
- [76] K. Ranganathan and I. Foster, Identifying Dynamic Replication Strategies for High Performance Data Grids, in Proceedings of the Second International Workshop on Grid Computing, Denver, CO, November 2001, pp. 75-86
- [77] C. Nicholson, D. G. Cameron, A. T. Doyle, A. P. Millar and K. Stockinger, Dynamic Data Replication in LCG 2008, Concurrency and Computation: Practice & Experience, Volume 20, Issue 11, Aug. 2008, pp. 1259-1271
- [78] W. Bell, D. Cameron, R. Carvajal-Schiaffino, P. Millar, K. Stockinger and F. Zini, Evaluation of an Economy-Based Replication Strategy for a Data Grid, International Workshop on Agent Based Cluster and Grid Computing, Tokyo, 2003
- [79] W. H. Bell, D. G. Cameron, L. Capozza, P. Millar, K. Stockinger and F. Zini, Simulation of Dynamic Grid Replication Strategies in OptorSim,

REFERENCES

- International Journal of High Performance Computing Applications, Volume 17, Issue 4, 2003
- [80] W. Bell, D. Cameron, L. Capozza, P. Millar, K. Stockinger and F. Zini, OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies, International Journal of High Performance Computing Applications, Volume 17, 2003
- [81] M. Tang, B. S. Lee, C. K. Yeo and X. Y. Tang, Dynamic Replication Algorithms for the Multi-Tier Data Grid, Future Generation Computer Systems, Special Issue: Parallel Computing Technologies, Volume 21, Issue 5, May 2005, pp. 775-790
- [82] R. Chang and H. Chang, A Dynamic Data Replication Strategy using Access-Weights in Data Grids, The Journal of Supercomputing, Volume 45, Issue 3, Sep. 2008, pp. 277-295
- [83] H. Lamahamedi, B. Szymanski, Z. Shentu and E. Deelman, Data Replication Strategies in Grid Environments, in Proceedings of the fifth International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP), Oct. 23 - 25, 2002
- [84] H. Lamahamedi, Z. Shentu and B. Szymanski, Simulation of Dynamic Data Replication Strategies in Data Grids, Proceedings of 12th Heterogeneous Computing Workshop (HCW'03), Nice, France, April 2003
- [85] Oracle <http://www.oracle.com/>
- [86] The UNIX System <http://www.unix.org/>
- [87] J. Gray, P. Helland, P. O'Neil and D. Shasha, The Dangers of Replication and a Solution, In Proceedings of the 1996 ACM SIGMOD, International Conference on Management of Data, Montreal, Quebec, Canada, June 04 - 06, 1996

REFERENCES

- [88] H. Stockinger, Distributed Database Management Systems and the Data Grid, In Proceedings of the eighteenth IEEE Symposium on Mass Storage Systems and Technologies, April 17 - 20, 2001
- [89] D. Dullmann, W. Hoschek, J. Jaen-Martinez, B. Segal, A. Samar, H. Stockinger and K. Stockinger, Models for Replica Synchronization and Consistency in a Data Grid, in Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'), San Francisco, CA, 2001
- [90] A. Domenici, F. Donno, G. Pucciani, H. Stockinger and K. Stockinger, Replica Consistency in a Data Grid, IX International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT03), Tsukuba, Japan, December 2003
- [91] A. Domenici, F. Donno, G. Pucciani and H. Stockinger, Relaxed Data Consistency with CONStanza, in Proceedings of the Sixth IEEE international Symposium on Cluster Computing and the Grid (CCGRID'06), Washington DC, May 16 - 19, 2006
- [92] M. Nicola and M. Jarke, Increasing the Expressiveness of Analytical Performance Models for Replicated Databases, in Proceedings of the 7th International Conference on Database Theory, Jan. 10 - 12, 1999
- [93] H. Stockinger, A. Domenici, F. Donno, G. Pucciani and K. Stockinger, Replica Consistency Service (RCS) - Design Principles and Basic Architecture v0.3, Technical Report - Draft, August 21, 2003
- [94] Y. Sun and Z. Xu, Grid Replication Coherence Protocol, The 18th International Parallel and Distributed Processing Symposium, Santa Fe, USA, April 2004, pp.232-239
- [95] R. Chang and J. Chang, Adaptable Replica Consistency Service for Data Grids, In Proceedings of the third International Conference on

REFERENCES

- Information Technology: New Generations (ITNG'06), Washington, DC, April 10 - 12, 2006, pp. 646-651
- [96] The Ecogrid project <http://ecogrid.nchc.org.tw/>
- [97] F. Dong and S. G. Akl, Scheduling Algorithms for Grid Computing: State of the Art and Open Problems, Technical Report No. 2006-504, School of Computing, Queen's University
- [98] K. Ranganathan and I. Foster, Decoupling Computation and Data Scheduling in Distributed Data-intensive Applications, In Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing, Edinburgh, Scotland, July 2002
- [99] K. Rangmathan and I. Foster, Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids, Journal of Grid Computing, Volume 1, 2003, pp53-62
- [100] D. G. Cameron, A. P. Millar, C. Nicholson, R. Carvajal-Schiaffino, F. Zini and K. Stockinger, Analysis of Scheduling and Replica Optimisation Strategies for Data Grids using OptorSim, Journal of Grid Computing, 2005
- [101] M. Tang, B.S. Lee, X.Y. Tang and C.K. Yeo, The Impact of Data Replication on Job Scheduling Performance in the Data Grid, Future Generation Computing System, Volume 22, Issue 3, 2006, pp. 254-268
- [102] R. Chang, J. Chang and S. Lin, Job Scheduling and Data Replication on Data Grids, Future Generation Computer Systems, Volume 23, Issue 7, Aug. 2007, pp. 846-860
- [103] Bricks - A Performance Evaluation System for Grid Computing Scheduling Algorithms <http://ninf.apgrid.org/bricks/>
- [104] M. Carman, F. Zini, L. Serafini and K. Stockinger, Towards an Economy-Based Optimisation of File Access and Replication on a Data

REFERENCES

- Grid, In Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID), May 21 - 24, 2002
- [105] G. K. Zipf, Selected Studies of the Principle of Relative Frequency in Language, Harvard University Press, Cambridge, Mass, 1932
- [106] G. K. Zipf. Human Behavior and the Principles of Least Effort, Addison-Wesley, 1949
- [107] S.M. Park, J.H. Kim, Y.B. Ko and W.S. Yoon, Dynamic Data Grid Replication Strategy based on Internet Hierarchy, In second International Workshop on Grid and Cooperative Computing (GCC'2003)
- [108] P. Shih, H. Chen, Y. Chung, C. Wang, R. Chang, C. Hsu, K. Huang and C. Yang, Middleware of Taiwan UniGrid, In Proceedings of the 2008 ACM Symposium on Applied Computing (SAC'08), Fortaleza, Ceara, Brazil, March 16 - 20, 2008
- [109] L. Guy, P. Kunszt, E. Laure, H. Stockinger and K. Stockinger, Replica Management in Data Grids, Technical Report, GGF5 Working Draft, July 2002
- [110] P. Kunszt, E. Laure, H. Stockinger and K. Stockinger, File-based Replica Management, Future Generation Computer Systems, Volume 21, Issue 1, 2005, pp. 115-123
- [111] V. Lefebure and T. Wildish, The Spring 2002 DAQ TDR Production, CMS NOTE 2002/034, CMS CERN, Switzerland, September 2002
- [112] P. Liu and J. Wu, Optimal Replica Placement Strategy for Hierarchical Data Grid Systems, in Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06), 2006, pp. 417-420
- [113] C. Wang, C. Hsu, P. Liu, H. Chen and J. Wu, Optimizing Server Placement in Hierarchical Grid Environments, Journal of Supercomputing, Volume 42, Issue 3, 2007, pp. 267-282

REFERENCES

- [114] H. Li, M. Muskulus and L. Wolters, Modeling Job Arrivals in a Data-Intensive Grid, Job Scheduling Strategies for Parallel Processing, 2007, pp. 210-231
- [115] H. Li, and M. Muskulus, Analysis and Modeling of Job Arrivals in a Production Grid, ACM SIGMETRICS Performance Evaluation Review, Volume 34, Issue 4, Mar. 2007, pp. 59-70
- [116] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters and D. H. Epema, The Grid Workloads Archive, Future Generation Computer Systems, Volume 24, Issue 7, Jul. 2008, pp. 672-686
- [117] H. Li, Workload Dynamics on Clusters and Grids, The Journal of Supercomputing, Volume 47, Issue 1, Jan. 2009, pp. 1-20
- [118] K. Christodoulopoulos, V. Gkamas and E. A. Varvarigos, Statistical Analysis and Modeling of Jobs in a Grid Environment, Journal of Grid Computing, Volume 6, Nov. 2007, pp. 77-101
- [119] R. M. Piro, A. Guarise, G. Patania and A. Werbrouck, Using Historical Accounting Information to Predict the Resource Usage of Grid Jobs, Future Generation Computer Systems, Volume 25, Issue 5, May. 2009, pp. 499-510
- [120] B. Javadi, M. K. Akbari and J. H. Abawajy, Multi-cluster Computing Interconnection Network Performance Modeling and Analysis, Future Generation Computer Systems, Jul. 2008
- [121] Geometric Distribution
http://en.wikipedia.org/wiki/Geometric_distribution
- [122] Zipf's Law http://en.wikipedia.org/wiki/Zipf's_law
- [123] PARSEC: Parallel Simulation Environment for Complex Systems
<http://pcl.cs.ucla.edu/projects/parsec/>

REFERENCES

- [124] C. Yang, I. Yang, K. Li and S. Wang, Improvements on Dynamic Adjustment Mechanism in Co-allocation Data Grid Environments, *Journal of Super Computing*, Jun. 2007, pp. 269-280
- [125] K. Czajkowski, I. Foster and C. Kesselman, Resource Co-allocation in Computational Grids, In *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-98'99)*, August 1999
- [126] R. S. Bhuvaneswaran, Y. Katayama and N. Takahashi, Dynamic Co-allocation Scheme for Parallel Data Transfer in Grid Environment, In *Proceedings of 1st International Conference on Semantics, Knowledge and Grid (SKG2005)*, pp. 178-188
- [127] R. S. Bhuvaneswaran, Y. Katayama and N. Takahashi, Redundant Parallel Data Transfer Schemes for the Grid Environment, In *Proceedings of the 2006 Australasian Workshops on Grid Computing and E-Research - Volume 54*, Hobart, Tasmania, Australia, January 16 - 19, 2006
- [128] C. Wang, C. Hsu, H. Chen and J. Wu, Efficient Multi-Source Data Transfer in Data Grids, In *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, May 2006, pp. 421-424
- [129] S. Vazhkudai, Enabling the Co-Allocation of Grid Data Transfers, In *Proceedings of the Fourth International Workshop on Grid Computing*, Nov. 2003, pp 41-51
- [130] R. Chang, M. Guo and H. Lin, A Multiple Parallel Download Scheme with Server Throughput and Client Bandwidth Considerations for Data Grids, *Future Generation Computer System*, Oct. 2008, pp. 798-805
- [131] R. Chang and P. Chen, Complete and Fragmented Replica Selection and Retrieval in Data Grids, *Future Generation Computer System*, May 2007, pp. 536-546

REFERENCES

- [132] D. G. Feitelson and L. Rudolph, Metrics and Benchmarking for Parallel Job Scheduling, In Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing
- [133] W. Cirne and F. Berman, When the Herd is Smart: Aggregate Behavior in the Selection of Job Request, IEEE Transactions on Parallel and Distributed Systems, 2003, pp.181-192
- [134] G. E. Keiser, Local Area Networks, McGraw-Hill, Inc, pp.185-18, 1989
- [135] S. J. Chapin, W. Cirne, D. G. Feitelson, J. P. Jones, S. T. Leutenegger, U. Schwiegelshohn, W. Smith and D. Talby, Benchmarks and Standards for the Evaluation of Parallel Job Schedulers, In Proceedings of 5th Job Scheduling Strategies for Parallel Processing, Apr. 1999
- [136] C. B. Lee, Y. Schwartzman, J. Hardy and A. Snaveley, Are User Runtime Estimates Inherently Inaccurate?, In Proceedings of 10th Job Scheduling Strategies for Parallel Processing, Jun. 2004
- [137] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen and R. F. Freund, Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems, Journal of Parallel and Distributed Computing, Nov. 1999, pp.107-131
- [138] J.W. Zhang, B.S. Lee, X.Y. Tang and C.K. Yeo, A Model to Predict the Optimal Performance of the Hierarchical Data Grid, Future Generation Computer Systems (FGCS), Volume 26, Issue 1, Jan 2010, pp. 1-11
- [139] J.W. Zhang, B.S. Lee, X.Y. Tang, M. Tang and C.K. Yeo, Analysis of Dynamic Replication Algorithms with Expiration-based Consistency Management for Multi-Tier Data Grid, The 9th International Conference on High Performance Computing, Grid and e-Science in Asia Pacific Region (HPCAsia), Seoul, Korea, Sep 2007
- [140] J.W. Zhang, B.S. Lee, X.Y. Tang and C.K. Yeo, Impact of Parallel Download on Job Scheduling in Data Grid Environment, Seventh

REFERENCES

- International Conference on Grid and Cooperative Computing (GCC'08), Oct. 2008, Shenzhen, China
- [141] Moore's Law http://en.wikipedia.org/wiki/Moore's_law
- [142] G. Boss, P. Malladi, D. Quan, L. Legregni and H. Hall, Cloud Computing, IBM Whitepaper, Oct. 2007
- [143] B. Hayes, Cloud Computing, Communications of the ACM, Volume 51, Issue 7, Jul. 2008, pp. 9-11
- [144] I. Foster, Y. Zhao, I. Raicu and S. Lu, Cloud Computing and Grid Computing 360-Degree Compared, Grid Computing Environments Workshop 2008 (GCE'08), 12-16 Nov. 2008, pp. 1-10
- [145] M. Armhurst *et al.*, Above the Clouds: A Berkeley View of Cloud Computing, Technical Report No. UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009
- [146] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg and I. Brandic, Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility, Future Generation Computer Systems, Volume 25, Issue 6, Jun. 2009, pp. 599-616
- [147] J. Dean and S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, Communications of the ACM, Volume 51, Issue 1, Jan. 2008, pp. 107-113
- [148] Google Home <http://www.google.com/>
- [149] Apache Hadoop Project <http://hadoop.apache.org/>
- [150] T. White, Hadoop: The Definitive Guide, O'Reilly & Associates Inc, Jun. 2009
- [151] Fair Scheduler for Hadoop
http://hadoop.apache.org/common/docs/r0.20.0/fair_scheduler.html
- [152] Facebook Home <http://www.facebook.com/>

REFERENCES

- [153] Capacity Scheduler for Hadoop
http://hadoop.apache.org/common/docs/r0.20.0/capacity_scheduler.html
- [154] Yahoo Home <http://www.yahoo.com/>
- [155] D. Borthakur, The Hadoop Distributed File System (HDFS): Architecture and Design, The Apache Software Foundation, 2007.
- [156] C. Yang, M. Yang, W. Chiang, Implementation of a Cyber Transformer for Parallel Download in Co-Allocation Data Grid Environments, Seventh International Conference on Grid and Cooperative Computing(GCC'08), Shenzhen, 24-26 Oct. 2008, pp. 242-253
- [157] C. Yang, I. Yang, W. Chiang, Enhancement of anticipative recursively adjusting mechanism for redundant parallel file transfer in data grids, Journal of Network and Computer Applications, Volume 32, Issue 4 (Jul. 2009), pp. 834-845
- [158] C. Yang, S. Wang and C. Chu, Implementation of a dynamic adjustment strategy for parallel file transfer in co-allocation data grids, The Journal of Supercomputing, online first, 17 June 2009
- [159] R. Chang, C. Lin, J. Ruey, S. Hsi, An efficient and bandwidth sensitive parallel download scheme in data grids, 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE 2008), 6-10 Jan. 2008, pp. 296 - 301
- [160] R. Chang, C. Lin, and S. Hsi, Accessing data from many servers simultaneously and adaptively in data grids, Future Generation Computer System , Volume 26, Issue 1 (Jan. 2010), pp. 63-71

Appendix A

Using equation (3.17), the average response time is:

$$\begin{aligned}
 E_{\text{delay}} &= F \times \left(\frac{1}{U_2} \right) \times P_2 \\
 &\quad + F \times \max \left[\frac{1}{U_2}, \frac{Q_2 \times A \times (P_1 + P_0)}{U_1} \right] \times P_1 \\
 &\quad + F \times \max \left[\frac{1}{U_2}, \frac{Q_2 \times A \times (P_1 + P_0)}{U_1}, \frac{Q_1 \times A \times P_0}{U_0} \right] \times P_0 \\
 &= 2 \text{ GBytes} \times \left(\frac{1}{622 \text{ Mbps}} \right) \times 0.3950 \\
 &\quad + 2 \text{ GBytes} \times \max \left(\frac{1}{622 \text{ Mbps}}, \frac{5 \times A \times (0.6010 + 0.004)}{2.5 \text{ Gbps}} \right) \times 0.6010 \\
 &\quad + 2 \text{ GBytes} \times \max \left(\frac{1}{622 \text{ Mbps}}, \frac{5 \times A \times (0.6010 + 0.004)}{2.5 \text{ Gbps}}, \frac{25 \times A \times 0.004}{2.5 \text{ Gbps}} \right) \times 0.004 \\
 &= 2 \times 1024 \times 8 \text{ Mbits} \times \left(\frac{1}{622 \text{ Mbps}} \right) \times 0.3950 \\
 &\quad + 2 \times 1024 \times 8 \text{ Mbits} \times \max \left(\frac{1}{622 \text{ Mbps}}, \frac{5 \times A \times 0.6014}{2.5 \times 1024 \text{ Mbps}} \right) \times 0.6010 \\
 &\quad + 2 \times 1024 \times 8 \text{ Mbits} \times \max \left(\frac{1}{622 \text{ Mbps}}, \frac{5 \times A \times 0.6014}{2.5 \times 1024 \text{ Mbps}}, \frac{25 \times A \times 0.004}{2.5 \times 1024 \text{ Mbps}} \right) \times 0.004 \\
 &= 10.40 + 2 \times 1024 \times 8 \text{ Mbits} \times \max \left(\frac{1}{622 \text{ Mbps}}, \frac{5 \times A \times 0.6014}{2.5 \times 1024 \text{ Mbps}} \right) \times 0.605 \\
 &= 10.40 + \max (15.936, 11.635A) \tag{A.1}
 \end{aligned}$$

According to equation (3.12), the average number of requests from one client node during period of E_{delay} is:

$$A = \frac{\sum_{c \in CS} f_c}{T_c} \times E_{\text{delay}} = \frac{125 \times (1/150)}{125} \times E_{\text{delay}} = E_{\text{delay}} / 150 \tag{A.2}$$

APPENDIX

By substituting equation (A.2) into (A.1), we have:

$$E_{\text{delay}} = 10.40 + \max [15.936, 0.0776 \times E_{\text{delay}}] \quad (\text{A.3})$$

If $0.0776 \times E_{\text{delay}} \geq 15.936$, i.e. $E_{\text{delay}} \geq 205.36$, equation (A.3) becomes:

$$E_{\text{delay}} = 10.40 + 0.0776 \times E_{\text{delay}} \quad (\text{A.4})$$

Solving equation (A.4), we have $E_{\text{delay}} = 11.27$. However, it is contradictory to the condition that $E_{\text{delay}} \geq 205.36$. So the result is not valid for the case $0.0776 \times E_{\text{delay}} \geq 15.936$. Thus, $E_{\text{delay}} < 205.36$ and equation (A.3) becomes:

$$E_{\text{delay}} = 10.40 + 15.936 \quad (\text{A.5})$$

Solving equation (A.5), we have $E_{\text{delay}} = \mathbf{26.34}$ and it tallies with the condition that $E_{\text{delay}} < 205.36$.

Appendix B

The M/M/m model parameters are as follow:

Job arrival rate: $\lambda = 1/10$ (job per second)

Server service rate: $\mu = 1/1,800$ (job per second)

Number of servers: $m = 200$

Using these parameters, we can calculate that:

$$P_0 = \left[\sum_{n=0}^{m-1} \frac{\rho^n}{n!} + \frac{\rho^m}{m!(1-\rho/m)} \right]^{-1}$$

$$\text{Where } \rho = \frac{\lambda}{\mu} = \frac{1/10}{1/1800} = 180$$

$$P_0 = \left[\sum_{n=0}^{200-1} \frac{180^n}{n!} + \frac{180^{200}}{200!(1-180/200)} \right]^{-1} = \left[105 \times \frac{180^{200}}{200!} \right]^{-1}$$

Average number of jobs in the queue is:

$$\begin{aligned} N_q &= \frac{\rho^m(\rho/m)}{m!(1-\rho/m)^2} P_0 \\ &= \frac{180^{200}(180/200)}{200!(1-180/200)^2} P_0 \\ &= 90 \times \frac{180^{200}}{200!} P_0 = 0.8571 \end{aligned}$$

Average job queuing time is:

$$T_q = \frac{N_q}{\lambda} = \frac{0.8571}{1/10} = 8.571$$

Average job turnaround time is:

$$T = T_q + \frac{1}{\mu} = 8.571 + \frac{1}{1/1800} = 1808.571$$