Data replication and management for data grid

Tang, Ming

2006

Tang, M. (2006). Data replication and management for data grid. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/2456>

<https://doi.org/10.32657/10356/2456>

Nanyang Technological University

Downloaded on 26 Apr 2023 09:31:24 SGT

Nanyang Technological University

**Data Replication and Management for
Data Grid**

Tang Ming

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 639798**

2006

Statement of Originality

I hereby certify that the content of this thesis is the result of work done by myself and has not been submitted for a higher degree to any other University or Institution.

.....

Date

.....

Signature

Abstract

The Data Grid is an infrastructure that manages large-scale data and computational resources across widely distributed communities to support data-intensive computing. As a specialization and extension of the Grid, the Data Grid puts additional emphasis on the sharing of large amounts of data and storage resources. Since data-intensive computing brings tremendous data movements, remote data access optimization is an essential part of the Data Grid. This thesis focuses on the techniques of data replication and management to improve the data access performance for the Data Grid.

Replication is a common and effective approach used in distributed systems to reduce response time, transfer time, and bandwidth consumption. The replication strategy determines which files are to be replicated, when to create new replicas, and the locations of the new replicas. In this thesis, some replica placement algorithms are evaluated with extensive simulations under various conditions of network topologies and client distributions. The performances of the algorithms are analyzed and it is found that the number of replicas and the total access cost fit the logarithmic relationship tightly. A logarithmic function is proposed and applied to address the minimum facility location problem.

The Internet Backplane Protocol (IBP) is a middleware for sharing storage resources on the network. Now tens of Terabyte of publicly shared storage resources are supported by IBP servers around the world. We develop a data management system to enable data sharing in IBP. The system is also a solution to integrate IBP with existent Data Grids.

ABSTRACT

Dynamic data replication strategies are studied for the multi-tier Data Grid, which is a typical architecture adopted in some leading international scientific projects for data-intensive computing. Two dynamic data replication algorithms that are adaptable to a variety of data access patterns are proposed. The simulation results show that both algorithms can significantly reduce the average response time of data access. Furthermore, the performances in terms of average bandwidth cost and replication frequency indicate that both replication algorithms are cost-effective and feasible.

Data replication is not an isolated problem in the Data Grid. In particular, Grid job scheduling is tightly correlated with data replication. A Data Grid architecture that supports efficient data access for the Grid job is introduced, and two centralized and one distributed dynamic data replication algorithms are proposed. The policies combining different Grid scheduling heuristics and replication algorithms are evaluated with various simulations. The results demonstrate that the proposed dynamic replication algorithms can reduce the job turnaround time.

Acknowledgements

It is my pleasure to thank all the people who supported me to make this thesis possible. In particular, I would like to express my deep and sincere gratitude to my supervisor, Assoc. Prof. Lee Bu-Sung, for his remarkable guidance and constant encouragement during the course of my Ph.D. study at the Nanyang Technological University. His stimulating suggestions helped me in all the time of research for and writing of this thesis.

I would give my thankfulness to Dr. Cai Wentong and Dr. Stephen John Turner for their patience of listening and discussion with me in the group meeting. Their valuable advice inspired me. I would also like to thank Ms. Yeo Chai-Kiat and Dr. Tang Xueyan for their comments in improving the quality and presentation on the works of dynamic data replication. I appreciate Dr. Micah Beck and the Logistical Networking team of University of Tennessee for their great support to me on IBP related works. And I also wish to say thanks to Mr. Zeng Yi, Mr. Zhu Huabing, Mr. Wang Lizhe, Ms. Feng Yuhong, Mr. Xavier Percival, Mr. Zhu Ning and all the other people in the *Parallel & Distributed Computing Center* (PDCC) who had supported my research work in one way or another.

Finally, I would like to dedicate this thesis to my wife and my family for their love and support.

Contents

Abstract	i
Acknowledgements	iii
Contents	v
List of Figures	xi
List of Tables	xv
1 Introduction	1
1.1 Background	1
1.2 Motivation	4
1.3 Contributions	6
1.4 Thesis Organization	7
2 Literature Review	9
2.1 Data Grid	9
2.1.1 Design Principles	9

CONTENTS

2.1.2	Architecture	10
2.2	Data Replication	12
2.2.1	Replication System	12
2.2.2	Caching vs. Replication	14
2.2.3	Replication for the Data Grid	19
2.3	Distributed Data Management	21
2.3.1	Replica Location Service	21
2.3.2	Storage Resource Broker	23
2.3.3	Stork	24
2.3.4	Grid Datafarm	25
2.4	Summary	26
3	Replica Placement	27
3.1	Introduction	27
3.2	Related Work	28
3.3	Replica Placement Algorithms	30
3.3.1	Model of Replica Placement Problem	30
3.3.2	Replica Placement Algorithms	32
3.4	Simulation Environment	35
3.4.1	Network Topologies	35
3.4.2	Client Distributions	36
3.5	Performance Evaluation	38

CONTENTS

3.5.1	Waxman Random	38
3.5.2	Internet AS-level	40
3.5.3	Degree Power-Law	41
3.5.4	Transit-Stub	42
3.5.5	Discussion	43
3.6	Summary	44
4	Logarithmic Characteristic of Replica Placement Algorithms	45
4.1	Introduction	45
4.2	Analysis	46
4.2.1	Impact of Client Distribution	46
4.2.2	Impact of Network Topology	48
4.3	Discussion	50
4.4	Summary	51
5	Data Management for the IBP-based Data Grid	53
5.1	Introduction	53
5.2	Logistical Networking	55
5.2.1	Internet Backplane Protocol	55
5.2.2	exNode	57
5.2.3	Logistical Backbone	58
5.2.4	Logistical Runtime System	59
5.3	Globus Replica Catalog	60

CONTENTS

5.4	Data Management	62
5.4.1	System Architecture	63
5.4.2	ExNode File Warehouse	64
5.4.3	IBP Data Management	65
5.5	Support for Grid Computing	70
5.6	Summary	72
6	Dynamic Replication Algorithms for the Multi-Tier Data Grid	75
6.1	Introduction	75
6.2	The Multi-Tier Data Grid	76
6.2.1	The System Architecture Supporting Dynamic Replication	78
6.3	Dynamic Replication Algorithms	80
6.3.1	Problem Definition	80
6.3.2	Simple Bottom-Up	82
6.3.3	Aggregate Bottom-Up	85
6.3.4	Fast Spread	88
6.4	Simulation Method	89
6.4.1	Topology of the Simulated Data Grid	90
6.4.2	Data File Access Pattern	91
6.4.3	Replica Server Capacity	93
6.5	Performances	95
6.5.1	Average Response Time	96

CONTENTS

6.5.2	Average Bandwidth Cost	99
6.5.3	Replication Frequency	101
6.5.4	Discussion	102
6.6	Summary	103
7	Combining Data Replication and Job Scheduling in the Data Grid	105
7.1	Introduction	105
7.2	System Model and Performance Metric	107
7.2.1	Performance Metric	109
7.3	Dynamic Data Replication	110
7.3.1	Centralized Dynamic Replication (CDR)	111
7.3.2	Distributed Dynamic Replication (DDR)	115
7.4	Grid Scheduling	116
7.4.1	Shortest Turnaround Time (STT)	118
7.4.2	Least Relative Load (LRL)	119
7.4.3	Data Present (DP)	119
7.5	Simulation Method	120
7.5.1	Simulated Environment of the Data Grid	120
7.5.2	Workload	124
7.6	Results	125
7.6.1	Impact of Data Grid Environment	126
7.6.2	Impact of Job Workload	130

CONTENTS

7.6.3	Discussion	134
7.7	Summary	136
8	Conclusion	137
8.1	Summary	137
8.2	Future Work	140
	Acronyms	143
	Author's Publications	145
	Bibliography	147

List of Figures

2.1	Data Grid architecture [44, 4, 42].	11
2.2	An example architecture of RLS [26].	22
3.1	The calculation algorithm for the load of every replica.	31
3.2	An example application of the set-cover algorithm.	34
3.3	Waxman random topology. Random client distribution.	39
3.4	Waxman random topology. Affinity client distribution.	39
3.5	Waxman random topology. Disaffinity client distribution.	40
3.6	AS-level topology on Nov. 8, 1997. Random client distribution.	41
3.7	Degree power-law topology. Random client distribution.	42
3.8	Transit-Stub topology. Random client distribution in stub domains. . . .	43
4.1	1997 AS-level. Random client distribution.	47
4.2	1997 AS-level. Affinity client distribution.	47
4.3	1997 AS-level. Disaffinity client distribution.	47
4.4	AS-level on Jan. 2, 2000.	49
4.5	Degree power-law.	49

LIST OF FIGURES

4.6	Transit-Stub.	49
4.7	Waxman random.	49
5.1	The network storage stack of the logistical networking [6].	55
5.2	A sample of exNode file.	58
5.3	L-Bone servers and IBP depots.	59
5.4	An example catalog using Globus Replica Catalog [4].	62
5.5	System architecture of data management for IBP.	63
5.6	Architecture of ExNode File Warehouse (EFW).	65
5.7	Uploading a file with IDM.	67
5.8	Downloading a file with IDM.	68
5.9	Refreshing a file with IDM.	69
5.10	Deleting a file with IDM.	70
5.11	Support for Grid computing.	71
6.1	The multi-tier Data Grid architecture for the LHC computing [22].	77
6.2	System architecture supporting dynamic replication.	78
6.3	The Simple Bottom-Up replication algorithm.	83
6.4	An example of the history and the node relations.	85
6.5	The Aggregate Bottom-Up replication algorithm.	86
6.6	The function of AGGREGATE.	87
6.7	The topology of the modeled multi-tier Data Grid.	91
6.8	Distributions of data file requests in the whole simulation session (Zipf-1.0).	93

LIST OF FIGURES

6.9	Distributions of data file requests in the fifth sub-session (Zipf-1.0).	94
6.10	Performance comparison of RSR, SBU and ABU.	97
6.11	Average response time vs. replica server capacity for Geo-.01.	97
6.12	Average response time vs. replica server capacity for Zipf-1.0.	98
6.13	Average response time vs. replica server capacity for Zipf-0.8.	98
6.14	Average bandwidth cost vs. replica server capacity.	100
6.15	Replication frequency of the dynamic replication methods.	102
7.1	System architecture of the Data Grid.	107
7.2	Policy performances under the base simulation configuration.	126
7.3	Policy performances when the network capacity is scaled up.	127
7.4	Geometric mean of turnaround time vs. replica server capacity for STT.	128
7.5	Geometric mean of turnaround time vs. replica server capacity for LRL.	129
7.6	Geometric mean of turnaround time vs. replica server capacity for DP.	130
7.7	Policy performances when $\bar{e} = 0$	132
7.8	Geometric mean of turnaround time vs. average relative estimation error for STT.	133
7.9	Policy performances when the job arrival rate is decreased.	133
7.10	Policy performances when the job computational cost is halved.	134

List of Tables

3.1	Complexities of the replica placement algorithms.	35
3.2	Parameters of the network topologies used for simulation.	37
4.1	Correlation coefficients.	48
4.2	Summary of correlation coefficients.	50
6.1	Configurations of replica server capacities.	95

Chapter 1

Introduction

1.1 Background

Computer technology experiences dramatic growth over the past decade. The computer power, data storage, and communication keep improving exponentially. However, the computational resources have always been limited compared to the resource demands from modern scientific and engineering applications [40]. One natural means addressing this problem is to share and coordinate computational resources via the network, so that the computational resources can be provided to the applications on demand without prior knowledge where the underlying resources come from.

As one of the most attractive and active research fields, Grid computing has been studied since the middle of 1990s. The Grid was proposed as a distributed computing infrastructure for advanced science and engineering applications. In 1998, Ian Foster and Carl Kesselman [41] defined *computational Grid* or just *Grid* as “*a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities*”. Two years later in 2000, the definition of the Grid computing was refined as: the Grid is concerned with “*coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations*” [44]. The *virtual*

Chapter 1. Introduction

organization (VO) is a set of individuals and/or institutions defined by specific resource sharing rules [44], such as what is shared, who is allowed to share, and the conditions under which sharing occurs. This new definition addresses social and policy issues in the Grid and emphasizes the highly controlled resource sharing.

Grid applications are categorized as distributed supercomputing, high throughput computing, on demand computing, data-intensive computing, and collaborative computing [41]. All these applications have a similar characteristic, that is, their enormous resource requirements can hardly be met by a single system cost-effectively. By providing scalable, secure and high-performance mechanisms for sharing and coordinating computation, data and communication resources across distributed organizations, Grid computing is promising to play a significant role in various disciplines.

Nowadays the efficient management and processing of the huge distributed and shared data resources around the world become a significant topic for both scientific research and commercial applications. In data-intensive computing, new information is obtained by processing the data that is maintained in geographically distributed repositories. Data movement represents a large fraction of turnaround time for data-intensive computing. Such application is often intensive in computation and communication.

The *Data Grid* was put forward to support data-intensive computing [27]. It is an infrastructure that manages large-scale data files and provides intensive computational resources across widely distributed communities. As a specialization and extension of the Grid, the Data Grid puts additional emphasis on the sharing of large amounts of data and data storage resources. The Data Grid provides the fundamental management mechanisms for distributed data, including support for managing data on remote storage systems, a uniform namespace for referring the data, a catalog for managing information about the data, and a data virtualization mechanism enabling data location transparency and data materialization transparency [80, 7].

1.1 Background

In domains of high energy and nuclear physics, astronomy, biology, and meteorology, the volume of data is already measured in Terabytes and will soon be in Petabytes [27]. Many scientific projects are dealing with large-scale data, various computational resources, and collaboration management. For example, in the next twenty years, the major high energy physics (HEP) experiments will lead us into a new age of understanding for the fundamental interactions, structures and symmetries that govern the nature of matter and space-time [61]. A new particle accelerator, the Large Hadron Collider (LHC), will be in operation at European Organization for Nuclear Research (CERN) in 2007, and four HEP experiments will produce several Petabytes of data per year over 15 to 20 years [56]. The data comprises raw measurement data and metadata describing the properties of the raw data. The experiment collaboration will involve 2000 physicists from 150 institutes in more than 30 countries. The experimental data will be stored at CERN and parts of the data will be stored at world-wide distributed regional centers, institutes and universities. Although the globally distributed computing and storage resources are large as a whole, they are inadequate for the physicists' requirement without a coordinated management. Hence it is crucial to manage global resources effectively to maximize the potential of the experiments for breakthrough discoveries.

Another example is the meteorology. Climate models are essential for understanding climate changes and they are improved by exhaustively analyzing today's models [61]. Supercomputers generate enormous model data from long-term simulations of climate systems with different parameters that can be compared with one another and with remote satellite sensing data [8]. Climate models need large computing resources and there are only a few sites around the world that are suitable for executing these models. Meteorologists who examine the model data are scattered all over the world. To accomplish model analysis, the data of interest are transferred from the computer modeling site to the meteorologists' institutions for various post simulation analysis tasks. The technologies

that enable geographically distributed communities to perform analysis on large scale of data efficiently and cost-effectively are required.

Some research projects on Data Grid have been carried out to deal with data-intensive computing, such as European DataGrid [36], GriPhyN [49], PPDG [87], and e-Science [35], and the advantages of the Data Grid are being recognized widely.

1.2 Motivation

In the Data Grid, data-intensive applications cause tremendous data movements between data repositories and computing resources. Data access must be optimized to meet the application requirements.

Data replication can effectively reduce access latency, response time and bandwidth consumption in distributed environment. At the same time, replication can improve system availability and fault tolerance. However, the cost of replica consistency management could be expensive. In the Internet and distributed database, large amounts of data are writable and the updates must be propagated to all replicas to ensure replica consistency. The benefits brought by data replication will be diminished if the cost of consistency management is too high. Fortunately, in the data-intensive applications, most of the managed data are published by experiments and they are seldom changed. For instance, in the CERN high energy physics applications, about 90% data are read-only [56, 104]. Hence replication is a feasible technique to optimize data access in the Data Grid.

Data replication has demonstrated its advantages in the areas of Internet, database and distributed computing. Nevertheless, there is no universal replication strategy applicable to all distributed environments. Although the existing replication methods provide helpful references, they cannot be used in the Data Grid as the problem context is drastically different. New challenges are faced in the Data Grid in comparison with the

1.2 Motivation

traditional distributed systems. For example, the numbers of involved facilities and users are large, the data size is huge, the resources change dynamically, and the system belongs to multiple owners. The replication related research topics in the Data Grid include replication strategy, replica placement, replica selection and consistency management. This thesis focuses on the techniques of replica placement and dynamic replication strategy.

Data replication is not an isolated problem in the Data Grid. In particular, Grid job scheduling is tightly correlated with data replication. On the one hand, the data required by the job impact the Grid scheduling decision. For the data-intensive computing, data transmission time may be a major portion of the job turnaround time. In addition to the job computing attributes, the Grid scheduling should take into account the positions of the data required by the job. On the other hand, Grid job scheduling can change the data access pattern and affect the replication status consequently. It is important to investigate the interactions between data replication algorithms and job scheduling heuristics, and find suitable policies that combine these two processes to improve the performance from the perspective of the jobs.

In the Data Grid, data management encapsulates the underlying network storage systems and provides a uniform operation interface to applications. With the data management service, users can easily publish, locate and access shared data files in the distributed environment. The Internet Backplane Protocol (IBP) is a middleware for sharing storage resources on the network, and tens of Terabyte of publicly shared storage resources supported by IBP are available around the world now. As IBP lacks a data management system, the data in IBP is not sharable to the public. To harness the IBP storage and data resources and enable IBP to act as a basic infrastructure for the Data Grid, a data management system is required.

1.3 Contributions

The main contributions of this thesis are:

- *Performance evaluation for the replica placement algorithms.* The set-cover replica placement algorithm, which attempts to place replicas in the centers of clustered clients, is put forward and evaluated together with the well-known greedy and vertex degree based algorithms. The performances in terms of total access cost and replica load balance are evaluated for the replica placement algorithms under various network topologies and client distributions. It is found that the greedy algorithm shows the best performance in total access cost among all algorithms, and set-cover is very close to greedy. On the Transit-Stub topology, the set-cover algorithm performs the best for load balance. From numerous simulations of the replica placement algorithms, we find that the number of replicas and the total access cost fit the logarithmic relationship tightly. A logarithmic function is proposed and applied to solve the *minimum facility location* problem.
- *A data management system for the Internet Backplane Protocol (IBP).* A data management system for the IBP-based Data Grid has been developed to support the data sharing among distributed communities. With the data management support, the IBP depots can be used as a world-wide data cache to improve data access performance for Grid computing.
- *Dynamic replication algorithms for the multi-tier Data Grid.* Two dynamic replication algorithms, Simple Bottom-Up (SBU) and Aggregate Bottom-Up (ABU), are put forward for the multi-tier Data Grid, which is a typical architecture adopted in some scientific projects to support large-scale distributed computing. The performances of SBU and ABU are compared with the Fast Spread replication strategy.

1.4 Thesis Organization

In all our studied situations of various data access patterns and replica server capacities, ABU exhibits significant performance advantage in terms of average response time and average bandwidth cost for the data access, and its performances are better than SBU and Fast Spread in most cases. The low replication frequencies of SBU and ABU indicate that both algorithms are more practical and cost-effective than Fast Spread.

- *Policies combining data replication and job scheduling to shorten Grid job turnaround time.* A generic Data Grid architecture supporting efficient data replication and job scheduling is proposed. Two centralized dynamic replication algorithms, CDR_RTPlace and CDR_SMPlace, and a distributed replication algorithm, DDR, are put forward. In addition, the Grid scheduling heuristics of Shortest Turnaround Time (STT), Least Relative Load (LRL) and Data Present (DP) are proposed. The policies that combine different data replication algorithms and job scheduling heuristics are evaluated with diverse simulations by varying the settings of the Data Grid environment and job workload. The results demonstrate that the dynamic replication algorithms can shorten the job turnaround time significantly. In particular, the policy of STT+CDR_RTPlace performs soundly under various conditions of system environment and workload.

1.4 Thesis Organization

The rest of the thesis is organized as follows:

- **Chapter 2** reviews the literature that is relevant to our research. The previous works of data replication for the Internet and for the Data Grid are reviewed. Several representative systems supporting data management for the Grid are described.

Chapter 1. Introduction

- **Chapter 3** analyzes some replica placement algorithms. Various simulations are carried out to verify and compare the performances of the algorithms.
- **Chapter 4** reports on the logarithmic characteristic of the replica placement algorithms. The observed logarithmic characteristic is applied to solve the *minimum facility location* problem.
- **Chapter 5** presents a data management system for the Internet Backplane Protocol (IBP). The system components and primary operation protocols are introduced.
- **Chapter 6** describes the background and advantages of the multi-tier Data Grid and presents our proposed dynamic replication algorithms. The performances of the algorithms are evaluated and compared with the previous approach.
- **Chapter 7** presents a generic Data Grid architecture that supports efficient data replication and job scheduling. The policies combining different data replication algorithms and job scheduling heuristics are proposed and evaluated by running simulations under various situations of system configuration and workload.
- **Chapter 8** summarizes the results of the research and discusses its future work.

Chapter 2

Literature Review

2.1 Data Grid

Today, how to manage and process the huge distributed and shared data resources efficiently around the wide-area networks becomes a significant topic for both scientific research and commercial application. Scientific data volumes are approximately doubling each year, and the demand for computational resources to perform scientific data analysis is rising even faster than data volumes [51]. The Data Grid [27] was put forward to support data-intensive computing. It is an infrastructure that manages large-scale data files and provides intensive computational resources across widely distributed communities. As a specialization and extension of the Grid, the Data Grid puts additional emphasis on the management of large amount of data and storage resources.

2.1.1 Design Principles

Many requirements should be satisfied when developing the Data Grid. In particular, high-speed, reliable and transparent access to remote data are required. Design principles of the Data Grid were discussed in [27, 62].

The Data Grid must integrate multiple distributed, heterogeneous, and independently

Chapter 2. Literature Review

managed data sources. Although large and centralized data storage works in many situations, to deal with data sources at multiple sites is a prerequisite for the Data Grid. The Data Grid architecture should be independent of the low level mechanisms used to store data and transfer data. Interfaces should be defined to encapsulate the specific storage system and data transfer method. The Data Grid should allow the user to find data based on characteristics, where the physical location of the data is transparent to the user.

The Data Grid must support data replication to minimize network traffic and ensure that data is close to where it is needed for computation. It should automatically stage input data files at the hosts where the computation will take place, and migrate the resultant output to the specified storage locations.

To facilitate the operation on the widely distributed resources, the Data Grid will be built upon the Grid middleware (e.g., Globus), which already provides resource management, security, information and basic data management services. Hence, the specialized Data Grid services are compatible with the underlying Grid infrastructure.

2.1.2 Architecture

The Grid architecture consists of five layers: fabric, connectivity, resource, collective, and application [44]. Regarding the specific requirements and design principles for the Data Grid, its architecture is shown in Figure 2.1 [44, 4, 42], and the details of each layer are as follows:

Fabric The Grid fabric layer provides the resources that can be shared and accessed with Grid protocols. The resources include storage systems, computing systems, networks and databases. Interfaces to local control are provided to the higher layer.

Connectivity The connectivity layer aims at communicating easily and securely in the

2.1 Data Grid

Figure 2.1: Data Grid architecture [44, 4, 42].

Grid. It defines standard protocols for communication and authentication. Communication protocols include transport, routing and naming to enable the data exchange between fabric layer resources, and they are mainly derived from TCP/IP protocol stack. Authentication protocols provide secure mechanisms for verifying users and resources. Security functions of access control (authorization), data confidentiality, delegation and single sign-on are required.

Resource The resource layer enables fabric resource sharing in the Grid. Protocols of access, information and management for individual resources are provided in this layer. Access protocols support remote resource access. Information protocols are used to publish and obtain information of the resource, including its configuration, load and usage policy. Management protocols are used to negotiate resource requirements and operations for the shared resource.

Collective The collective layer supports the interaction and coordinated management of multiple resources. Services of replica management, replica selection, information, job management are required for the Data Grid system. The replica management

service is responsible for keeping track of replicas, creating or removing replicas when required, and providing access to replicas. Replica selection service finds the replica that offers the best performance among replicas of the desired data content in the Data Grid. Information service collects and provides access to information about the Grid resources. Examples of information are machine type, operating system version, number of processors, available disk space, network load, CPU load and job queue status. Job management allows the scheduling and control of jobs that run on multiple resources.

Application Discipline-specific applications, such as high energy physics and climate modeling, can be constructed by calling services defined at any layer.

2.2 Data Replication

Data replication is a technique that can improve the service accessibility in distributed systems. Replication can shorten data access time perceived by the end-user and reduce network traffic by redirecting client requests to a replica that is closest to the client. Replication can also improve the system availability, as the failure of a replica does not result in entire service outage. At the same time, replication introduces costs as it consumes storage and network resources. The goal of data replication is to provide the client with the best data access performance while consuming the minimal resources.

2.2.1 Replication System

Sivasubramanian *et al.* [102] modeled the replication problem as an abstract objective function, which takes uncontrollable or controllable parameters from the perspective of the replication system. Examples of uncontrollable parameters are client request rate, data popularity, and available network bandwidth. Examples of controllable parameters

2.2 Data Replication

are number of replicas, replica location, and replication time. To achieve the optimal performance, only the controllable parameters can be manipulated. Continuous feedback is necessary, i.e., the system status should be monitored in real-time in order to adjust the replication decisions properly.

For the replication systems that are scattered across a large geographic area, the primary research topics include:

- Replication strategy. The replication strategy determines which data to be replicated, when to create new replicas, and where the new replicas to be placed. Issues of metric determination, adaptation triggering and replica placement need to be addressed [102]. The metric that the system aims to optimize should be identified. Examples of client-perceived metrics are data access time and network distance, while the system-perceived metric examples include network usage and replication cost.

The performance of a replication system is impacted by various uncontrollable parameters such as data access pattern and network condition. The adaptation triggering function identifies the changes of the system and triggers replication in accordance with the detected system situations. The adaptation can be triggered at fixed time intervals or triggered by the event indicating the changes of the uncontrollable parameters in the system.

The problem of replica placement includes two aspects [102]: replica server placement and replica content placement. Replica server placement determines the locations for the replica servers, whereas replica content placement chooses the replica servers to which the data will be replicated. Server placement is used by the system operators in the system installation or upgrading phases, and the locations are chosen from the perspective of the whole set of data files in the system. Content

Chapter 2. Literature Review

placement is called more frequently to adapt to the system changes, and the servers are chosen from the perspective of individual data files to be replicated.

- Replica selection. The replica selection service finds the replica that provides the best performance for the end-user. Usually the replica that is at the server offering the highest data access speed will be selected. The other metrics impacting replica selection include replica server load, network distance and service stability. Replica selection should be able to discover the current system conditions and adapt the decisions accordingly.
- Replica consistency management. For the writable data files, the changes should be traced and propagated to all replicas. Consistency management adds overhead to the system, particularly when the data are frequently overwritten, the number of replicas is large, and strict consistency is required. There are some well-known replica consistency algorithms described in [53, 107, 47], including Simple Read One Write All (ROWA), Read One Write All Available (ROWA-A), primary copy, and voting (also called *quorum consensus*). Consistency models with different consistency levels were proposed in [33] for database files in the Data Grid. However, there is very little research on replica consistency for general data files in the Data Grid as most of the managed data are read-only [56].

2.2.2 Caching vs. Replication

Caching and replication are two closely related areas, and they are both used to improve data access performance in distributed environment. In caching, if the required data is not in the cache, the data will be fetched from the data server and stored in the cache before being passed to the client. The client can fetch the data from the cache directly if the cache contains the required data. The typical examples of caching include memory cache, Web browser cache and proxy cache. In replication, the system pro-actively creates copies

2.2 Data Replication

of data at various replica servers. Examples of replication include Web site mirroring and distributed database replication.

Rabinovich *et al.* [89] distinguished the two techniques as “caching is storing an object at a place that sees the object anyway in the course of request processing”, while “replication is storing an object at a place that would not otherwise see the object”. In other words, a cache encounters both hit and miss requests, which flow to the cache regardless of the cache content. In contrast, a replica server normally sees only hit requests because the request is directed to the server only if the server has the replica of the desired object.

2.2.2.1 Caching

The major research topic of caching is the caching replacement policy [3, 112, 9], which is used to decide which document to evict from the cache to accommodate the new data object when the cache is full. Classical caching replacement policies include the least-recently used (LRU) and the least-frequently used (LFU). LRU evicts the object that has not been accessed for the longest period of time, on the basis that the requests follow temporal locality. LFU evicts the object that was requested least frequently, on the basis that the data popularity keeps consistent in a long-term. Some replacement policies are put forward for Web caching and they are classified into two groups [3]: key-based and function-based policies. Key-based replacement policies evict objects based on a primary key, and ties are broken based on secondary key and even tertiary key. The commonly used keys include last access time, number of previous accesses, average retrieval time and document size. Policies of SIZE, Hyper-G [114], LRU-threshold and LRU-MIN [2] belong to the key-based replacement type. Function-based replacement policies employ a cost function that combines multiple keys, and the cost values are used in the replacement decisions. Examples of function-based replacement policies are GreedyDualSize [21], Hybrid [117], and Size-Adjusted LRU (SLRU) [3].

Chapter 2. Literature Review

In contrast to traditional passive caching, pro-active caching creates copies of data objects in advance of the requests. Korupolu *et al.* [65] proposed two cooperative placement algorithms, GreedyPlace and AmortizingPlace, and a cooperative replacement algorithm, hierarchical-greedy-dual (HGD), for a hierarchical large-scale caching architecture. All these caching algorithms create copies pro-actively. Given the data access patterns and the network distances between servers, the placement algorithms are run at intervals to determine where to place each object in order to minimize the average access cost. For HGD, each cache runs the local greedy-dual caching replacement algorithm [119]. Upon a cache miss, a data object is evicted from the cache and then is passed up to the parent cluster for possible inclusion in one of its caches.

Research [111] and [60] studied prefetching caching for the Web, which pro-actively fetches data objects to the caches in advance of user requests. The prefetched data objects are chosen by analyzing the data access patterns. Both research attempts to improve the performances in terms of cache hit rate and bandwidth consumption. The prefetching algorithms are threshold-based, that is, only the objects whose certain criteria exceeds a specified threshold will be prefetched.

2.2.2.2 Replication

Data replication has been extensively applied to the Internet. Most of the replication research aims to reduce data access cost. Rabinovich *et al.* [91] put forward a protocol for dynamic replication and migration of Internet objects in response to client demand changes. The protocol attempts to place data objects close to a majority of requests and ensure that no servers become overloaded. It consists of two algorithms, one of which decides the number and placement of replicas and the other distributes requests among replicas. The protocol is used in their RaDaR system [90]. In RaDaR, every replica server runs the replication algorithm autonomously. Two thresholds of client request rate are

2.2 Data Replication

defined: deletion threshold R_d and replication threshold R_r , where $R_d < R_r$. The replica is deleted if its request rate is below R_d . If the replica request rate is above R_d , the algorithm attempts to migrate the replica to a server located closer to the clients that send more than a half of requests. If the migration fails and the replica's request rate is greater than R_r , the algorithm attempts to replicate the data. The request distribution algorithm directs the request to the closest replica if it is not overloaded. Otherwise, the request is directed to the least loaded replica.

Pierre *et al.* [85] developed Globule as a replication platform for the Web. Globule integrates the services of dynamic creation and removal of replicas, consistency management, and client direction. The main novelty of Globule is that each Web document is associated with the most suitable replication strategy. Replication issues are encapsulated inside each document in order to enable the per-document replication policy. To select the best replication strategy, the trace-based simulation is run internally for each document. A number of replication heuristics are run individually with the collected trace logs for the document, and the heuristic with the best performance is chosen to associate with the document. Hence each document's replication strategy varies adaptively depending on its access pattern.

Kangasharju *et al.* [64] studied replication strategies for Content Distribution Network (CDN). In their model, every Internet Autonomous System (AS) is a node with finite storage for object replicas. The objective of the replication problem is to minimize the average number of ASs traversed when clients access the required objects. Among their proposed replication heuristics, *Greedy-Global*, which takes into consideration all replica servers when making the replication decisions, works the best.

Kalpakis *et al.* [63] studied replica placement in a tree network in order to minimize the aggregated costs of read, write and storage. A $O(n^3k^2)$ -time dynamic programming algorithm was put forward for placing k replicas of a data object on a tree with n nodes.

Chapter 2. Literature Review

The algorithm was extended to take into consideration the load and service capacity constraints on the nodes. A $O(n^3 k^2 C_{max})$ -time replica placement algorithm was proposed in order to minimize the costs and satisfy the load and capacity constraints on the nodes, where C_{max} is the maximal capacity of each node in the tree.

In order to speed up the data access in the Internet, Rodriguez *et al.* [98] proposed a dynamic parallel data access scheme. With this scheme, a client connects multiple replica servers at the same time and downloads different portions of the data file from different servers. When the downloading is finished, the client reassembles the data blocks into the original file. Experiments were carried out using mirror servers in the Internet. For every experiment, the number of used parallel connections was less than five. The questions of how many and which replica servers should be chosen for parallel access if there are a large number of replicas in the system are not answered. Nevertheless, the experimental results demonstrate that dynamic parallel access can increase data download speed.

Some research attempts to improve other performance metrics instead of data access time. G. On *et al.* [84] studied replica placement problem by modeling the widely distributed system as a graph where all nodes and edges have the known values of availability and failure probability. The goal is to choose the number and location of replicas to satisfy the quality of availability (QoA) requirement.

Cherkasova *et al.* [25] proposed the FastReplica algorithm to efficiently and reliably replicate large files in the Internet. FastReplica exploits network paths among the replica servers to speed up the overall replication time. To replicate a large file among n servers, where n is in the range of 10 to 30, the file is partitioned into n sub-files evenly in size and each sub-file is transferred to a different server. After the distribution step, each server opens $n - 1$ concurrent connections to the remaining servers and sends its received sub-file to them. FastReplica also provides solutions to replicate large files to a large number of servers, and the issue of reliability is addressed.

2.2 Data Replication

2.2.3 Replication for the Data Grid

Several recent studies have examined the problem of data replication in the Grid. Ranganathan and Foster [95] put forward several straightforward dynamic replication strategies, including the Fast Spread and Cascading methods, for the hierarchical Data Grid architecture. Three kinds of localities of request patterns are defined: temporal locality, where recently accessed files are likely to be accessed again; geographical locality, where files recently accessed by a client are likely to be accessed by the neighborhood clients; and spatial locality, where files near a recently accessed file are likely to be accessed. The spatial locality is applicable only if the files are associated and ordered. In their simulator, the network links are not shared and each replica server can only serve one data request at any time. The file popularity follows the geometric distribution. From their simulation results, Fast Spread has the best performance when the data access pattern contains no locality, while Cascading works better when there is a small degree of geographical and temporal localities.

A different cost model was proposed by Lamahamedi *et al.* [68] to decide the dynamic replication. This model evaluates the benefits by creating a replica and the costs of creation and maintenance for the replica, and it is applied by the Replica Manager in each intermediate storage site in a decentralized manner. The Data Grid architecture is a hybrid of tree and ring topologies, and data access among same tier nodes is allowed. The simulation results show that the dynamic replication method can improve the data access performance and decrease data transfer cost.

Some works have been carried out to study the problems of data replication and job scheduling together [96, 13, 78, 32]. Ranganathan and Foster [96] further extended their research to consider the data replication and job scheduling working together for the Data Grid. The External Scheduler is modeled to assign the jobs to specific computing sites,

Chapter 2. Literature Review

and the Data Scheduler running at each site is responsible for dynamically creating replicas for popular data files. Various combinations of scheduling and replication strategies are evaluated through simulations. Their results show that data locality is an important factor when scheduling the jobs. The simple scheduling policy of always assigning jobs to the sites that contain the required data works very well if the popular datasets are replicated dynamically. Takefusa *et al.* [106] also reported similar conclusions using the Grid Datafarm [45] architecture and the Bricks Grid simulator.

OptorSim [20, 13] is a Data Grid simulator for studying data replication and job scheduling strategies. The simulated Grid architecture is similar to [96] in that a global Resource Broker schedules the jobs to the computing sites and a Replica Optimiser in each site performs local replica optimization. The replication operation is independently determined by each site. For every data access required by the locally running job, the Replica Optimiser will determine whether the data should be replicated to local storage and which replicas should be removed if there is not enough space. The dynamic replication strategies used in their study [20] evolve from traditional cache replacement methods. The economic replication strategies were put forward and they attempt to improve the profits brought by the replicas and decrease the cost of data management at the same time. The detailed information of the economy-based file replication strategies was reported in [23, 14]. Their simulation results show that the scheduling strategy considering both the data file access cost for the jobs and the workload of computing resources produces the shortest mean job execution time, and the economic replication strategies can improve the Grid performance.

Mohamed and Epema [78] proposed the Close-to-Files (CF) scheduling policy that attempts to assign jobs to computer clusters that already contain the input files. For the input files, two cases are considered in the experiments: one without replication and another with each file replicated in three different sites. In both cases, the files are

2.3 Distributed Data Management

randomly distributed among the file servers. Their results indicate that the combination of CF scheduling policy and file replication is beneficial in terms of system utilization.

Desprez and Vernois [32] presented a method that simultaneously determines job scheduling and replica placement for a set of jobs. Their method requires full knowledge on the set of jobs to be scheduled and relies on the assumption that the complexity of the job is linear in time with the size of the data to be processed. The objective of the job scheduling and replica placement is to maximize system throughput, which is the number of jobs that can be executed per unit time on the system. The problem is solved with a linear program. The experimental results show that their approach can improve the system throughput when the storage space is not large in comparison with the size of all data in the Grid.

2.3 Distributed Data Management

In the Data Grid, distributed data management consists of a set of protocols and services for coordinated management across administrative domains. In this section a set of representative systems supporting data management for the Grid are introduced.

2.3.1 Replica Location Service

In the geographically distributed Data Grid, the data files are often replicated to multiple remote sites. The service to locate one or more replicas of the desired data file is required. The Replica Location Service (RLS) [26, 28] maintains and provides access to information about the physical locations of the data replicas. RLS was developed by the Globus project and Work Package 2 of the European DataGrid project [36] cooperatively, and it is part of Globus Toolkit Version 3.0.

In RLS, the term Logical File Name (LFN) is used to denote a unique logical identifier

Chapter 2. Literature Review

for data content that may have one or more physical replicas. The Physical File Name (PFN) specifies the physical location of a copy of the file on a storage system. The function of RLS is to identify the PFNs that are associated with the specified LFNs.

An example of RLS architecture is shown in Figure 2.2 [26]. The RLS architecture consists of a set of Local Replica Catalog (LRC) and Replica Location Index (RLI) servers organized in hierarchy order. Located in the lowest level of RLS, each LRC maintains mappings between LFNs and the associated PFNs. Clients can query LRC to discover the replicas for a logical file name. Located in the higher level of RLS, each RLI contains a set of mappings from LFNs to LRCs. Clients can query RLI about mappings stored in one or more LRCs.

Figure 2.2: An example architecture of RLS [26].

The RLS index structure can be constructed with different performance and reliability characteristics by varying the number of RLIs and the degrees of partitioning and redundancy. The partition can be performed on either the LFN name space or replica site name space. The redundancy indicates whether each replica is indexed at multiple RLIs.

Strong consistency is not required in RLS, so soft state update protocol is used. *Soft state* is the information that times out and must be updated at intervals. Stale information is removed implicitly via time outs. With soft state update, LRCs send summaries of their state to relevant RLIs, which then update their indices with the

2.3 Distributed Data Management

received information. To reduce network traffic, the soft state update can be compressed before transmission.

RLS differs from the Globus Replica Catalog (GRC) that is available as part of the Globus Toolkit Version 2.2 in several ways. First, as a distributed system, RLS is designed to provide greater reliability by avoiding single point of failure as well as providing better load balancing, performance and scalability. In contrast, GRC is a centralized catalog. Second, RLS implementation is based on open source relational database technology, while GRC is based on LDAP. Finally, RLS strictly separates replication information from other types of metadata assuming such information is stored in a separate metadata service, but GRC can contain metadata about logical files.

2.3.2 Storage Resource Broker

The Storage Resource Broker (SRB) [103] is developed by San Diego Supercomputer Center (SDSC). SRB is a middleware providing a uniform storage interface to access heterogeneous distributed storage resources including file systems, database systems, and archival storage systems. A Metadata Catalog (MCAT) service is used in SRB to provide attribute-based access to data collections and items. MCAT manages descriptive and system metadata associated with data collections and system resources. The descriptive metadata describes the contents of entire data collections and individual data items, and system metadata provides location and access control information for collections and data items.

Together, SRB and MCAT provide a scalable information discovery and data access system for publishing and computing with scientific data and metadata. With MCAT, SRB supports location transparency by accessing datasets and resources based on their attributes rather than their names or physical locations.

Chapter 2. Literature Review

Each SRB server controls a distinct set of physical storage resources, and a federation of SRB servers allows access to distributed storage resources [10]. The federated SRB scheme allows one SRB server to act as a client to another SRB server. This enables a client application to access data stored anywhere in the distributed system, even if the application is not directly connected to the controlling SRB server.

SRB provides the following Data Grid capabilities [93]: handle multiplicity of platforms, resources and data types; provide transparent access to resources and attribute-based access to data and collections; handle replication of data and provide replicated data management; handle access control and provide auditing facilities; and provide remote operations for data movement, metadata extraction and indexing.

2.3.3 Stork

Stork [66] is a scheduler for data placement in the Grid, and it is developed by the Condor project [30]. The data placement activities are treated as important as the computational jobs. They need to be queued, scheduled, monitored, managed, and even check-pointed. Most of the Grid applications require transferring the input data required by the job from a remote site to the execution site, running the job, and then transferring the output data from the execution site to the users. All of these computational and data placement activities are regarded as jobs and are represented as nodes in a Directed Acyclic Graph (DAG). The dependencies between the activities are represented as directed arcs. The job description language ClassAd [94] is used to describe the data placement jobs. The meta-scheduler for Condor, DAGMan [31], can interact with Stork and submit the data placement jobs to Stork.

Stork supports multiple data transfer protocols such as FTP, GridFTP and HTTP, and it can decide to use which protocol at the run-time. Functions of failure recovery and efficient resource utilization are included in Stork.

2.3 Distributed Data Management

2.3.4 Grid Datafarm

Grid Datafarm (Gfarm) [45] architecture is designed for Petascale data-intensive computing in the Grid. It consists of Gfarm file system, process scheduler, and parallel I/O APIs [108].

The Gfarm file system is a parallel file system. Local disks of the computers in the Grid compose the Gfarm file system. The nodes of Gfarm file system are the computers running with the daemon process *gfsd*, which facilitates remote file operations with access control, user authentication, and node status monitoring. Each Gfarm file system node provides both storage and computing capabilities. A *Gfarm file* is a large-scale file that is divided into fragments and then distributed across the Gfarm nodes, so that it can be accessed in parallel. The fragments of the Gfarm files can be replicated in the system for fault-redundancy and load balance. The consistency management is not necessary because all Gfarm files are read-only. Gfarm metadata database manages the information such as the mapping from a logical Gfarm file name to its fragment file names, replica catalog, and platform information about the Gfarm nodes.

As data-intensive computing involves intensive data access, Gfarm emphasizes the tight coupling of storage and computation [106]. The scheduling policy of moving the computation to the data, or called “owner computes”, is adopted by Gfarm.

Gfarm parallel I/O APIs support multiple streams to access the fragments of a Gfarm file in parallel [118]. Features of TCP tuning techniques for fast file transfer, file recovery and regeneration are provided by Gfarm.

2.4 Summary

In this chapter, the design principles and architecture of the Data Grid are introduced. As data replication is the main concern of this study, its primary research aspects are outlined. The caching technique is reviewed for comparison with replication. From the survey of the replication approaches existing in the areas of Internet and Grid, it can be known that replication is an effective and practical method to improve data access performance in a large-scale distributed system. However, the replication strategies used in the Internet cannot be directly used for the Data Grid due to the different environments. The growing number of data-intensive applications motivate our research in replication for the Data Grid.

Distributed data management is another important research topic for the Data Grid and it is closely relevant to replication. Some systems supporting data management in the Grid are reviewed.

Chapter 3

Replica Placement

3.1 Introduction

Replica placement algorithm determines where to place replicas. Although the models and approaches studied in this chapter are mainly from the perspective of replica server placement, they can also be applied to replica content placement. Hence in this chapter the term *replica* will refer synonymously to replica server and replica content.

The goal of this study is to evaluate and conclude the performances of some replica placement algorithms, so that the results can guide us toward their effective applications in different environments. In this chapter, we define a model for replica placement problem, in which the network connections of the Data Grid are modeled as a graph, and the clients, replicas and routers are vertices of the graph. It is assumed that all clients have the same request rate and each client only accesses its closest replica. The optimal replica placement is a typical *minimum k -median* problem, which is NP-hard [74]. Algorithms studied in this chapter are sub-optimal solutions for the problem of replica placement to get the lowest overall access costs on the network. The well-known greedy and vertex degree based replica placement algorithms are described, and we put forward our set-cover algorithm that attempts to put replicas close to clustered clients.

Chapter 3. Replica Placement

The load balance among replicas is a significant performance metric for replica placement, but it is seldom considered in existing research. In this chapter, an evaluation model of replica load balance is proposed.

The performances in terms of total access cost and load balance are analyzed for the replica placement algorithms. The replica placement algorithms are evaluated with simulations on diverse network topologies, including Waxman random, Internet AS-level, degree power-law, and Transit-Stub. The client distribution may affect the performances of replica placement algorithms, so different client distribution methods are used in the simulations. The performances of replica placement algorithms are discussed according to the simulation results.

The rest of this chapter is organized as follows. The next section reviews the related works on replica placement. Section 3.3 presents the model of replica placement problem and some replica placement algorithms. Simulation approaches and results are presented in Section 3.4 and Section 3.5 respectively. Section 3.6 summarizes the chapter.

3.2 Related Work

Currently, most research of replica placement is for the Internet and Content Distribution Networks (CDN). The replica placement methods have been studied in the areas of Web server placement [88], Web cache placement [67] and mirror server placement [59].

The frequently used optimal models of replica placement can be classified into minimum k -median and minimum k -center. For example, Qiu *et al.* [88] and Radoslavov *et al.* [92] used k -median as the optimal model to place the replicas. Krishnan *et al.* [67] also used models deduced from k -median to resolve the cache location problem. In contrast, Jamin *et al.* used k -center for the placement of Internet mirror [59] and instrumentation [58]. The objective of *minimum k -median* is to minimize the sum of access costs for

3.2 Related Work

all clients, while the objective of *minimum k -center* is to minimize the maximum access cost among all clients. The detailed explanation of minimum k -median problem is given in Section 3.3.1.1.

Some replica placement algorithms proposed by previous research are well recognized now. Qiu *et al.* [88] proposed a greedy algorithm that is simple but can offer high performance. The greedy algorithm places replicas on the network in a greedy fashion without replacing already placed replicas. In each step to place a replica, the greedy algorithm tries to minimize the sum of access costs by taking into account the placed replicas. Jamin *et al.* [59] put forward l -greedy algorithm. l -greedy is similar to the simple greedy algorithm in [88] but it allows l step(s) trace back. When $l = 0$, the l -greedy is the same as the simple greedy algorithm. The computing complexity of l -greedy will increase intensively if l is large and the method becomes impractical to implement. So only 0-greedy, 1-greedy and 2-greedy are experimented in [59]. The experimental results show that 1-greedy and 2-greedy do not have significant improvement on 0-greedy (the simple greedy algorithm).

Li *et al.* [73] proposed a tree-based algorithm for Web proxy placement, which assumes that the network topologies are trees. The computational time of the tree-based algorithm is $O(N^3M^2)$, where M is the number of desired replicas and N is the number of potential sites. This algorithm is impractical for the network with thousands of vertices, and it also does not perform as well as the algorithms for general graph topologies [88]. So in this research, the tree-based algorithm is not evaluated.

Radoslavov *et al.* [92] investigated various factors that impact the performances of replica placement algorithms. It is found that the vertex degree based placement method performs well in some conditions. The vertex degree based algorithm is evaluated in this research.

3.3 Replica Placement Algorithms

3.3.1 Model of Replica Placement Problem

In this section, the basic graph theoretical problem, minimum k -median, is presented. Then, the formal model of replica placement problem is defined based on the minimum k -median problem. An evaluation model of load balance among replicas is proposed.

3.3.1.1 Minimum k -Median Problem

The *minimum k -median* problem is defined as follows. Given a graph $G = (V, E)$, where vertex set $V = \{v_1, \dots, v_n\}$, the number of vertices $|V| = n$, and edge set $E \subseteq V \times V$. The distance between vertices of v_i and v_j is $d(v_i, v_j)$, where $v_i, v_j \in V$. The goal is to choose a k -median set, i.e., a subset $M \subseteq V$ with vertex number $|M| = k$, so that the sum of the distances from each vertex to its nearest median, i.e., $\sum_{v \in V} \min_{m \in M} d(v, m)$, is minimal.

3.3.1.2 The Model of Replica Placement Problem

The model of replica placement problem is based on the minimum k -median graph theory. The definition of the replica placement problem is as follows.

Given a network represented by an undirected graph $G = (V, E)$, where vertex set $V = \{v_1, \dots, v_n\}$, number of vertices $|V| = n$, and edge set $E \subseteq V \times V$. The replica candidate set is $A = \{a_1, \dots, a_f\}$, where $|A| = f$ and $A \subseteq V$. The client set is $B = \{b_1, \dots, b_g\}$, where $|B| = g$ and $B \subseteq V$. The distance between vertices v_i and v_j is $d(v_i, v_j)$, where $v_i, v_j \in V$. The distance can be measured using the metrics such as geographical distances, hops, and round-trip time (RTT). Network bandwidth can also be used to define the distance, that is, higher bandwidth means shorter distance. In this chapter the terms *distance* and *cost* are used interchangeably to represent the access cost from client. The

3.3 Replica Placement Algorithms

goal of replica placement problem is to choose a replica set $S = \{s_1, \dots, s_k\}$, where $|S| = k$ and $S \subseteq A$, so that the total access cost for every client to its nearest replica, i.e., $\sum_{b \in B} \min_{s \in S} d(b, s)$, is minimum.

3.3.1.3 Replica Load Balance Evaluation Model

The minimum k -median problem does not constrain the number of requests served by each replica. If the capabilities of the replicas are restricted, balancing the load among all replicas will be important. We study the load balance for the replica placement algorithms under the assumptions that all replica servers have the same processing capability, the clients only access the closest replicas, and all clients have the same request rate.

The load of the replica is measured by the sum of received requests from the clients. If there are x replicas that have the same distance and all are the closest replicas to client b , we assume that the requests from client b are uniformly dispersed among the x replicas. The load (service request) coming from client b is valued as $1/x$ for each of the x replicas. Figure 3.1 presents the algorithm to calculate the load of every replica.

Input: $S[1..k]$: array of replicas
 $B[1..g]$: array of clients
Output: $load[1..k]$: array of load for every replica in S

CALCULATE-LOAD()

- 1: Initialize array $load$
- 2: **for all** client $b \in B$ **do**
- 3: Search replica set $T(b)$, where $T(b) \subseteq S$, $T(b) = \{t_1, \dots, t_n\}$, $|T(b)| = x$,
satisfying $d(b, t_1) = \dots = d(b, t_n) = \min_{s \in S} d(b, s)$.
- 4: **for all** replica $t \in T(b)$ **do**
- 5: $load[t] \leftarrow load[t] + \frac{1}{x}$
- 6: **end for**
- 7: **end for**
- 8: return array $load$

Figure 3.1: The calculation algorithm for the load of every replica.

Chapter 3. Replica Placement

As all replicas have the same capabilities, the load balance among replicas can be calculated based on the concepts of *standard deviation* and *coefficient of variation* used in probability theory. Let w_i be the load of replica i . The mean value of load for all replicas is $\mu = \frac{1}{k} \sum_{i=1}^k w_i$. The variance of load is $\sigma^2 = \frac{1}{k} \sum_{i=1}^k (w_i - \mu)^2$. The performance metric of *load balance* among the replicas is defined as:

$$\mathcal{C} = \sigma / \mu \quad .$$

Note that $\mathcal{C}$ is the coefficient of variation of the replica load. Lower value of $\mathcal{C}$ stands for better load balance among the replicas.

3.3.2 Replica Placement Algorithms

In this section, some algorithms are presented to solve the replica placement problem as defined in the previous section. All the algorithms are based on the assumption that the clients only access the nearest replicas. Hop count is used as the metric of data access cost. After presenting the greedy placement and the vertex degree based placement algorithms, we put forward our set-cover replica placement algorithm.

3.3.2.1 Greedy Algorithm

The greedy algorithm used in this research was proposed by Qiu *et al.* [88], and it was also studied in [59] and [92]. This algorithm iteratively adds new replicas to the network. First it checks every vertex in the replica candidate set A and chooses the one that yields the lowest total access cost. Then, the algorithm looks for the second replica location that, in conjunction with the chosen replica, yields the lowest total access cost. Iterate this procedure until all k replicas are placed.

3.3 Replica Placement Algorithms

3.3.2.2 Vertex Degree Based Algorithm

The idea of the vertex degree based algorithm is the same as the Transit Node method [59] and the fanout-based method [92]. This algorithm assumes that the vertex with the highest degree is usually the closest to all clients. It chooses k replicas from set A in descending order of vertex degree.

3.3.2.3 Set-Cover Algorithm

The set-cover algorithm works in a greedy fashion, and it attempts to place the replicas in the centers of partial clients, or called clustered clients. The algorithm works as follows. Given a radius r , $1 \leq r \leq \lceil \text{graph_diameter}/2 \rceil$, where the graph diameter is the maximum distance of any two vertices in the graph. First, the algorithm checks each replica candidates a_i in A and draw a logical circle that is centered from a_i with the radius r . For simplification, we say that a_i covers vertex v if $d(v, a_i) \leq r$. The set-cover algorithm selects the replica candidate a_s from A that covers the most clients, and let it be the first location for the replica. Mark the clients covered by a_s . In each subsequent iteration, the algorithm chooses the replica candidate that covers the most unmarked clients. If all clients have been marked and we still want to place more replicas, the algorithm will unmark all clients and then repeat the above step. Iterate the above steps until all k replicas are placed.

Figure 3.2 shows an example application of the set-cover algorithm. In this example, the selection sequence of the replica locations is: $a_1 \rightarrow a_4 \rightarrow a_3 \rightarrow a_2$.

An important parameter that affects the performance of the set-cover algorithm is the radius r . We use a simple method to get the approximate best radius. Initially the method sets the radius to one and uses the set-cover algorithm to place replicas and get the total access cost. It then increases the radius in each step and checks whether the

Figure 3.2: An example application of the set-cover algorithm.

total access cost decreases. If the total access cost decreases, the method will store this new radius as the best one temporarily. If it is found that the total access cost increases with two continuous larger radii, the method will assume that all other larger radii have higher total access costs than current radius, i.e., current radius is already the best one. Thus, it returns the radius and the corresponding replica placement result.

3.3.2.4 Random Placement

Besides the above three replica placement algorithms, the random replica placement method, which randomly chooses k replicas from set A , is studied in this chapter for performance comparison.

3.3.2.5 Computational Complexities

The computational complexities of the replica placement algorithms are shown in Table 3.1, where N is the number of replica candidates, C is the number of clients, and M is the number of desired replicas.

3.4 Simulation Environment

Table 3.1: Complexities of the replica placement algorithms.

Greedy	Vertex degree based	Set-cover (with assigned radius)	Random
$O(NCM)$	$O(N + M \lg N)$	$O(NCM)$	$O(M)$

3.4 Simulation Environment

The network topology and client distribution are the two factors that may affect the performances of replica placement algorithms. Combinations of various network topologies and different client placements are used in the simulations. In this section, we outline four network topologies used in the simulations: Waxman random, Internet AS-level, degree power-law, and Transit-Stub. We then describe three client distribution methods. After the network topology is generated and clients are placed on the network, the replica placement algorithms are executed. The performances of the algorithms are evaluated in terms of total access cost and load balance.

3.4.1 Network Topologies

Various network topologies are used in our research to evaluate the performances of the replica placement algorithms.

Waxman Random. The Waxman random model [113] assigns the probability of an edge between u and v as $P(u, v) = \alpha e^{-d(u, v)/\beta L}$, where α and β are the only two parameters and their values are in the range of $(0, 1]$. The graph edges will increase with a bigger α , and the long edge proportion will increase with a bigger β . $d(u, v)$ is the Euclidean distance from vertex u to v , and L is the longest distance between any two vertices on the network. We set $\alpha = 0.01$ and $\beta = 0.6$, and use the tool of GT-ITM, created by Zegura *et al.* [120], to generate a Waxman random graph,

Chapter 3. Replica Placement

which has 1000 vertices and the average degree of the vertices is 5.4.

Internet AS-level. NLNR [82] provides the real Internet Autonomous Systems (AS) level topologies collected each day from Nov. 1997 to Jan. 2000. In our study, we use the Internet AS-level topologies from NLNR with the target dates of Nov. 8, 1997 and Jan. 2, 2000. There are 3015 AS vertices on Nov. 8, 1997 and 6474 vertices on Jan. 2, 2000, where each vertex stand for an AS node of Internet.

Degree Power-Law. Faloutsos *et al.* [37] found several power laws of the AS-level Internet topology. Degree power-law is one of these laws, that is, if let $f(d)$ be the frequency of vertices with degree d , then $f(d) \propto d^\eta$, where η is a constant. Bu and Towsley [16] showed that $\eta \approx -2.18$ by performing a linear regression on the AS-level topology of Jan. 2002. Inet-3.0 [115] is an AS-level Internet topology generator based on the degree power-law. We use Inet-3.0 to create a network topology with 5000 vertices.

Transit-Stub. The Transit-Stub model generates topologies composed of interconnected transit and stub domains [19]. Transit domains interconnect stub domains efficiently. Stub domains resemble the campus networks, while connected transit domains correspond to the network backbone of wide area. In our study, GT-ITM [120] is used to create a Transit-Stub graph, which has five transit domains, 225 stub domains, and 5250 vertices totally.

The major parameters of the network topologies used in simulations are shown in Table 3.2.

3.4.2 Client Distributions

The client distribution on the network may affect the performances of replica placement algorithms. We use random, affinity, and disaffinity client distribution methods in our

3.4 Simulation Environment

Table 3.2: Parameters of the network topologies used for simulation.

Topology Model	Vertices	Edges	Average Degree	Graph Diameter
Waxman random	1000	2744	5.4	8
Internet AS-level Nov. 8, 1997	3015	5539	3.7	9
Internet AS-level Jan. 2, 2000	6474	13895	4.3	9
Degree power-law	5000	8901	3.6	9
Transit-Stub	5250	10554	4.0	21

simulations.

Random client distribution. This method randomly selects client locations with uniform probability from the network graph.

Affinity/disaffinity client distribution. This method was proposed by Wong *et al.* [116].

The affinity distribution is to simulate clients clustering together, whereas the disaffinity distribution is to simulate clients spreading out. This distribution method is implemented as follows. The first client is selected randomly from network. To pick the h th client, assign a probability $p_i = \alpha/w_i^\beta$ to each vertex v_i that is not selected as client, where w_i is the shortest distance between v_i and any vertex that is already selected as a client. α is calculated satisfying $\sum_{v_i} p_i = 1$, and β is the parameter determining the grade of clients' affinity or disaffinity. Same as [116] and [92], we use $\beta = 15$ for extreme affinity client distribution and $\beta = -15$ for extreme disaffinity client distribution in the simulations.

3.5 Performance Evaluation

In this section, the performances of the replica placement algorithms are evaluated. The Waxman random, Internet AS-level on Nov. 8, 1997, degree power-law, and Transit-Stub topologies are used in the simulations, and their properties are shown in Table 3.2.

For each topology, 15% of vertices are selected as clients using different methods described in Section 3.4.2, and then the replica placement algorithms are used to place replicas ranging from one to around 1% of the number of vertices. The final performance results shown in each plot are the mean values of five runs that are on the same network topology but with different client locations.

To compare the data access performances of the algorithms, we use the total access cost produced by the random algorithm with only one replica on the network as the *baseline*. We then calculate the *relative total access cost* for each algorithm, which is the ratio of the total access cost obtained by the algorithm to the baseline cost. A smaller value of relative total access cost means better performance for the algorithm.

The load balance among replicas is computed with the model described in Section 3.3.1.3. A smaller value of load balance implies better balance performance. If every replica has the same load, the value of load balance performance is zero, which is the optimal case. The simulation results are shown in the following sections in accordance with the network topology.

3.5.1 Waxman Random

We use a Waxman topology with 1000 vertices for our simulations. The simulation results with random, affinity and disaffinity client distributions are shown in Figure 3.3, Figure 3.4 and Figure 3.5 respectively.

3.5 Performance Evaluation

(a) Relative total access cost performance.

(b) Load balance performance.

Figure 3.3: Waxman random topology. Random client distribution.

(a) Relative total access cost performance.

(b) Load balance performance.

Figure 3.4: Waxman random topology. Affinity client distribution.

Apparently, the relative total access cost of the greedy placement is always the lowest among all algorithms. The set-cover algorithm also works well and its performance is very close to greedy, especially when clients cluster together (see Figure 3.4(a)). The performance of the vertex degree based algorithm is fine when the number of replicas is few. For the greedy and set-cover algorithms, with 10 replicas, the total client access costs are less than 60% of the baseline, which is the total access cost when only one

Chapter 3. Replica Placement

Figure 3.5: Waxman random topology. Disaffinity client distribution.

replica is randomly placed on the network.

The load balance performances of the greedy and set-cover algorithms are usually less than 0.22, while those of the random algorithm reach 0.4 when placing 10 replicas.

Figures 3.3, 3.4 and 3.5 show that the client distributions have trivial impact on the relative performances of the algorithms. So in the following sections, we only show the simulation results with the random client distribution.

3.5.2 Internet AS-level

The Internet AS-level topology on Nov. 8, 1997 has 3015 vertices. We randomly select 452 vertices as clients (15% of all vertices).

From Figure 3.6(a), it can be noticed that the vertex degree based algorithm performs very well and its performance is close to the set-cover and greedy algorithms. The difference of relative total access cost between the vertex degree based algorithm and the greedy algorithm is less than 0.02. The gap of the random algorithm's relative total access cost to any other algorithm is normally larger than 0.2.

3.5 Performance Evaluation

Figure 3.6: AS-level topology on Nov. 8, 1997. Random client distribution.

The load balance performances of the greedy, set-cover, and vertex degree based algorithms are similar, which is around 1.3 when 30 replicas are placed (see Figure 3.6(b)).

The simulation results on the topology of Internet AS-level on Jan. 2, 2000 are similar to AS-level on Nov. 8, 1997, so they are not shown here.

3.5.3 Degree Power-Law

The degree power-law topology generated by Inet-3.0 has 5000 vertices, and the simulation results are shown in Figure 3.7.

Comparing plots in Figure 3.7 of the degree power-law topology with those in Figure 3.6 of the AS-level topology, it can be found that both topologies have similar impacts on the performances of the replica placement algorithms, which also proves that the degree power-law topology qualitatively resembles the AS-level topology of Internet.

Again we see that the vertex degree based algorithm's performance is close to the greedy and set-cover algorithms (see Figure 3.7(a)). In Figure 3.7(b), the load balance performance of the random algorithm is apparently better than other algorithms when

Chapter 3. Replica Placement

Figure 3.7: Degree power-law topology. Random client distribution.

the number of replicas is more than 15, which differs from the results obtained from the Waxman topology.

3.5.4 Transit-Stub

We use a Transit-Stub topology with 5250 vertices for simulations. Slightly different from the previous simulations, on the Transit-Stub topology, we assume that clients can only distribute on stub domains, while replicas can be in both transit and stub domains. It is reasonable for real network services because most clients will not reside at the top-level network. The results are shown in Figure 3.8.

For both performances in relative total access cost and load balance, the vertex degree based algorithm performs poorly on the Transit-Stub topology in comparison with other topologies. The reason is that Transit-Stub is a structured topology, and the vertices in the transit domains construct the network backbone. Placing replicas on the network backbone would yield good performance in total access cost. However, the vertices in the backbone do not evidently have higher degrees than the others, which induces the

3.5 Performance Evaluation

Figure 3.8: Transit-Stub topology. Random client distribution in stub domains.

improper decisions by the vertex degree based algorithm.

The set-cover algorithm's total access cost is close to the greedy, and its load balance performance is the best among all algorithms.

3.5.5 Discussion

From the above simulations we can see that the greedy algorithm's performance in total access cost is the best among all evaluated algorithms regardless of network topologies and client distributions, and the set-cover algorithm is a close second. With lower complexity, the vertex degree based algorithm can perform well on the topologies of AS-level and degree power-law, but poorly on the Transit-Stub topology.

The load balance performances of the algorithms depend a lot on the network topologies. On the Waxman random and Transit-Stub topologies, the greedy, set-cover, and vertex degree based algorithms are apparently better than the random algorithm. In particular, on the Transit-Stub topology, the set-cover algorithm's load balance performance is the best among all algorithms. But on the Internet AS-level and degree power-law

Chapter 3. Replica Placement

topologies, the load balance performance of the random algorithm is better than other algorithms when the number of replicas is large. The reason is that both AS-level and degree power-law topologies follow the degree power-law (see Section 3.4.1), in which the vertex with higher degree is usually closer to most of the clients. For all algorithms except the random method, the first placed few replicas will take over most of the requests, while the later placed replicas will only serve a small portion of the requests. Thus, the load is not well distributed among the replicas in comparison with the random replica placement. Of course, if data request redirection mechanisms are applied with respect to load balance among replicas, the global performance will be better for all algorithms.

The evaluation of the algorithm performances can be the guide to choose and apply the algorithms effectively in different environments.

3.6 Summary

In this chapter, the replica placement is discussed and the existing approaches are reviewed. We propose a replica load balance evaluation model. The set-cover replica placement algorithm, which attempts to place replicas in the centers of clustered clients, is proposed. The set-cover algorithm and the well-known greedy, vertex degree based and random replica placement algorithms are evaluated with simulations on various network topologies and client distributions. From the simulation results, the greedy algorithm shows the best performance in terms of total access cost among all algorithms, and the set-cover algorithm is very close to the greedy placement. On the Transit-Stub topology, the set-cover algorithm gives the best load balance. The vertex degree based algorithm performs well on the Internet AS-level and degree power-law topologies, but poorly on the Transit-Stub topology.

Chapter 4

Logarithmic Characteristic of Replica Placement Algorithms

4.1 Introduction

In this chapter, we report on the logarithmic characteristic of the replica placement algorithms that are described in Section 3.3. As the set-cover algorithm's performance in total access cost is very close to the greedy algorithm, we only choose the greedy, vertex degree based and random replica placement algorithms for analysis. The simulation methods are the same as Section 3.4, and all topologies in Table 3.2 are used. For each network topology, 15% of the vertices are chosen as clients. The replica placement algorithms are then used to select the replica locations on the network topologies.

From our simulations with diverse network topologies and client distributions, it is found that the number of replicas and the total access cost fit the logarithmic relationship tightly. The logarithmic relationship can be used to predict the data access performance with the specified number of replicas. The observed logarithmic relationship between the number of replicas and the total access cost is applied to determine the optimal number of replicas for the *minimum facility location* problem.

The rest of this chapter is organized as follows. The logarithmic characteristic of the

Chapter 4. Logarithmic Characteristic of Replica Placement Algorithms

replica placement algorithms is analyzed in Section 4.2. Section 4.3 discusses the application of the observed logarithmic characteristic. Section 4.4 summarizes the chapter.

4.2 Analysis

In this section, we study the simulation results and analyze the logarithmic relationship between the number of replicas and the total access cost for the replica placement algorithms.

We plot the number of replicas and the relative total access cost in log-linear scale, and use linear regression to fit a line in the two-dimensional points. The method of linear regression is based on the least squares, which minimizes the sum of the squares of the deviations of the data from the model. The correlation coefficient is used to verify the validity of the approximation of regression.

4.2.1 Impact of Client Distribution

First, we use the AS-level topology on Nov. 8, 1997 (refer Table 3.2) for our simulation. As client distribution may affect the performance of replica placement algorithm, we use the random, affinity, and disaffinity client distribution methods (see Section 3.4.2), and the results are shown in Figure 4.1, Figure 4.2, and Figure 4.3 respectively. The number of replicas is in $[1, K]$. The maximum number of replicas K in the plot is determined by the greedy algorithm. That is, when K replicas are already placed on the network using the greedy algorithm, the total access cost is minimal and does not decrease further even if putting more replicas. On the y -axis, we plot the relative total access cost. The lines in the graphs represent the results of linear regression.

A remarkable observation is that the plots fit well the linear regression. With the three different client distribution methods, the greedy algorithm's correlation coefficients

4.2 Analysis

Figure 4.1: 1997 AS-level. Random client distribution.

Figure 4.2: 1997 AS-level. Affinity client distribution.

Figure 4.3: 1997 AS-level. Disaffinity client distribution.

of linear regression are higher than 0.992, the coefficients of the vertex degree algorithm are higher than 0.978, and even the coefficients of the random algorithm are higher than 0.909. The summary of correlation coefficients is grouped in accordance with the client distribution methods and shown in Table 4.1.

It can be easily deduced that the number of replicas and the total access cost also fit the same linear regressions as those in the plots. This leads us to define the approximate

Chapter 4. Logarithmic Characteristic of Replica Placement Algorithms

Table 4.1: Correlation coefficients.

Client distribution	Algorithms		
	Random	Vertex degree based	Greedy
Random	0.9094	0.9941	0.9964
Affinity	0.9792	0.9779	0.9917
Disaffinity	0.9530	0.9954	0.9985

function about number of replicas and total access cost as follows:

$$C(k) = -p \ln k + q \quad , \quad (4.2.1)$$

where k is the number of replicas, $C(k)$ is the total access cost. The values of p and q are different for different algorithms, and $p, q > 0$. The network topology and client distribution may also impact the values of p and q .

From the simulation results and analysis, we can see that although the clients are placed with different methods, each algorithm still fits the logarithmic relationship well. Thus, we can only use the random client distribution when evaluating the impact of network topology on total access cost in the following section.

4.2.2 Impact of Network Topology

Further simulations are carried out on the AS-level topology on Jan. 2, 2000, degree power-law topology, Transit-Stub topology, and Waxman random topology. The metrics of these topologies are shown in Table 3.2. The results are displayed in Figures 4.4, 4.5, 4.6 and 4.7. The values of the correlation coefficient of linear regression are grouped in different topologies and shown in Table 4.2.

From the plots we observe that the number of replicas and the total access cost fit the logarithmic relationship very well, and the correlation coefficients of both the greedy algorithm and the vertex degree based algorithm are higher than 0.982 on all topologies

4.2 Analysis

Figure 4.4: AS-level on Jan. 2, 2000.

Figure 4.5: Degree power-law.

Figure 4.6: Transit-Stub.

Figure 4.7: Waxman random.

that are tested. The correlation coefficients of the random algorithm are higher than 0.909.

Comparing the plots of the two AS-level topologies (Figure 4.1 and Figure 4.4) and the degree power-law topology (Figure 4.5), it can be observed that these topologies have similar impacts on the performances of the replica placement algorithms. Based on the above three topologies, the coefficients of the greedy algorithm are higher than 0.996, the coefficients of the vertex degree based algorithm are higher than 0.986, and those of the random algorithm are higher than 0.909.

Chapter 4. Logarithmic Characteristic of Replica Placement Algorithms

Table 4.2: Summary of correlation coefficients.

Network topology	Algorithms		
	Random	Vertex degree based	Greedy
AS-level 1997	0.9094	0.9941	0.9964
AS-level 2000	0.9540	0.9940	0.9983
Degree power-law	0.9595	0.9865	0.9974
Transit-Stub	0.9890	0.9818	0.9876
Waxman	0.9983	0.9976	0.9996

On the Transit-Stub topology, the correlation coefficient of the random algorithm is 0.989 and even higher than the coefficients of the vertex degree based and greedy algorithms. On the Waxman random topology, it is notable that the correlation coefficients of all algorithms are higher than 0.998.

4.3 Discussion

The logarithmic relationship reflects a principle of the replica placement problem on the network. When adding the first few replicas to the network, the total access cost will decrease sharply. However, if the number of replicas reaches the threshold, adding new replicas will only have marginal performance improvement.

Our proposed approximate function describes the relationship between the number of replicas and the total access cost. This function can be used to predict the final result of total access cost with the given number of replicas effectively. Or vice versa, the function can estimate how many replicas are needed to achieve the desired total access cost.

The approximate function can be used to trade-off the total access cost and the cost of adding new replicas. Let us consider the typical minimum facility location problem. The objective of *minimum facility location* problem is to minimize the total facility cost plus the total access cost. To model the minimum facility location problem, we use the

4.4 Summary

same notations as defined in Section 3.3.1.2. The cost of opening a replica at site s is represented as $f(s)$, and $\sum_{b \in B} \min_{s \in S} d(b, s)$ is the total access cost (refer Section 3.3.1.2). Then we want to minimize the value of

$$F = \sum_{s \in S} f(s) + \beta \sum_{b \in B} \min_{s \in S} d(b, s) \quad .$$

We use a constant β to adjust the measurement metric of the access cost to be the same as the metric of the cost of opening facility, and its value can be determined in practice.

For simplification, we assume that the cost of opening a replica at every site is w , and use the approximate function Equation (4.2.1) to substitute the total access cost, we can get:

$$F(k) = wk + \beta(-p \ln k + q) \quad .$$

To get the minimum facility location cost, let the differential function $F'(k) = w - \beta \cdot p/k = 0$. So when $k = \beta \cdot p/w$, we get:

$$F_{min} = \beta[p + q + \ln(\beta \cdot p/w)] \quad .$$

With the logarithmic approximate function, we can calculate how many replicas should be created to get the minimum facility location cost.

4.4 Summary

This chapter is an extension of the research described in Chapter 3 about replica placement. From the simulations of the replica placement algorithms with diverse network topologies and client distributions, we find that the number of replicas and the total access cost fit the logarithmic relationship tightly. The logarithmic relationship can be used to predict the data access performance with the specified number of replicas effectively. As an example application, the proposed logarithmic function is used to solve the minimum facility location problem.

Chapter 5

Data Management for the IBP-based Data Grid

5.1 Introduction

In the Data Grid, data management coordinates the underlying network storage systems and provides a uniform operation interface to applications. With the data management service, users can easily publish, locate and access shared data files in the distributed environment. The underlying services supporting data management include file transfer protocol, storage management, replica catalog, and security mechanism.

The *logistical networking* is a technology that allows easy scaling for shared network storage in terms of the size of user community, the aggregate quantity of storage that can be allocated, and the distribution scope of service nodes on the network [11, 12]. As the base component of the logistical networking, the Internet Backplane Protocol (IBP) acts as a middleware to support sharing of distributed storage resources over the network. The functions of storage management, data transfer and depot management are provided by IBP. The other components of the logistical networking include the Logistical Backbone (L-Bone), exNode and Logistical Runtime System (LoRS), which facilitate manipulations of the storage resources and data files in IBP.

Chapter 5. Data Management for the IBP-based Data Grid

The IBP depots listed in the L-Bone now serve tens of Terabyte of publicly sharable storage around the world [69]. IBP provides a basic scalable infrastructure for the global Data Grid. The typical Data Grid services, such as data management, data access control and replica selection, can be developed on top of IBP.

As the logistical networking lacks the data management service, the data stored in IBP is not sharable to public. In this chapter, we describe the implementation of data management system for the IBP-based Data Grid. The Globus Replica Catalog (GRC) is used in the IBP data management system to provide the location service for the data stored in IBP. GRC is a general replica location service and it is already adopted by some Data Grid projects. Different from the normal data access protocols such as ftp, gsiftp and http, a simple URL is not enough for IBP to access the data. To manage a data file in IBP, the application must have the exNode file associated with the data file. So the ExNode File Warehouse (EFW) is developed to store and manage the exNode files. The IBP Data Management (IDM) provides a set of APIs to applications and it interacts with IBP, GRC and EFW. With IDM APIs, applications can easily manage the data stored in IBP.

The rest of this chapter is organized as follows. The logistical networking and Globus Replica Catalog are introduced in Section 5.2 and Section 5.3 respectively. Then the system architecture of the data management for IBP is described in Section 5.4. The implementation of EFW and IDM are described in details, and some examples of the operation protocols are shown for better understanding of the whole system. Section 5.5 discusses the support of the IBP data management system for the Grid computing. Section 5.6 summarizes the chapter.

5.2 Logistical Networking

5.2 Logistical Networking

The Logistical Networking project [75] is developed by the computer science department of the University of Tennessee. The logistical networking is a technology that allows easy scaling for shared network storage in terms of the size of the user community, the aggregate quantity of storage that can be allocated, and the distribution scope of service nodes on the network [11]. The network storage stack [6] of logistical networking is shown in Figure 5.1, and the layers of Internet Backplane Protocol (IBP), exNode, Logistical Backbone (L-Bone) and Logistical Runtime System (LoRS) are the main efforts of the logistical networking.

Applications	
LoRS	
exNode	L-Bone
IBP	
Local access layer	
Physical layer	

Figure 5.1: The network storage stack of the logistical networking [6].

5.2.1 Internet Backplane Protocol

The Internet Backplane Protocol (IBP) is a mechanism to share storage resources across networks ranging from LAN to WAN [11]. It supports the logistical networking and allows applications to control the storage, data and data transmission for IBP depots [86].

IBP storage resources are managed by *depots*, which are computers running IBP server software. Clients can perform remote storage operations on IBP depots. In the view of clients, a depot's storage resources are a collection of append-only byte arrays. There are no directory structures or client-assigned file names in the depot. The *capabilities*

Chapter 5. Data Management for the IBP-based Data Grid

are assigned by depots and they can be viewed as the names of the byte arrays. Each capability is a text string encoded with the IP address of the IBP depot, plus other information to be interpreted only by the depot.

The IBP client calls fall into three different groups:

- Storage management: `IBP_allocate` and `IBP_manage`.
- Data transfer: `IBP_store`, `IBP_load`, `IBP_copy`, and `IBP_mcopy`.
- Depot management: `IBP_status`.

`IBP_allocate` is used to allocate a byte array at an IBP depot with specifications of size, duration and other attributes. A chief design feature of IBP is the use of capabilities (cryptographically secure passwords). A successful `IBP_allocate` returns three capabilities for reading, writing and managing the allocated byte array. `IBP_manage` allows the client to manipulate the allocated IBP byte arrays. For example, the user can call `IBP_manage` to modify the time limit of the allocations and the maximum size of a byte array.

The `IBP_store` and `IBP_load` functions are two blocking calls that store and load the data on a particular capability, which is returned by `IBP_allocate`. `IBP_copy` allows the user to perform a third-party transfer that is to send data from one allocation to another allocation. `IBP_mcopy` allows the user to perform third-party transfer in point-to-point or point-to-multipoint mode.

`IBP_status` is used to query a depot about its status and to change some storage properties.

Because of IBP's limitations on the size and duration of allocation and its weak allocation semantics, IBP does not directly implement reliable storage abstractions provided

5.2 Logistical Networking

in conventional file management system. Instead, these must be built on top of IBP using techniques such as redundant storage.

5.2.2 exNode

Following the example of the Unix *inode*, which aggregates disk blocks to implement a file, the *exNode* is a single generalized data structure managing aggregate allocations that can be used in implementing network storage.

The exNode has two major components: arbitrary metadata and mappings. Mappings may have metadata as well. Metadata consists of $\langle name, type, value \rangle$ triplets where the type can be 64-bit integers, 64-bit floating point numbers, character strings, and metadata lists. The metadata lists allow nesting of metadata. An individual mapping typically includes the IBP capabilities for reading, writing, and managing the allocated storage. The mapping will also have specialized metadata, such as logical offset, length and physical offset as well as user defined arbitrary metadata.

The exNode library automates the handling of IBP capabilities and allows the user to associate metadata with the capabilities [6]. The exNode library has a set of functions that allow an application to create and destroy an exNode, to add or delete a mapping from it, to query it with a set of criteria, and to produce an XML serialization. When a user wants to store the exNode to disk or to pass it to another user, he can use the exNode library to serialize it to an XML file. We call this XML file as *exNode file*. With the exNode file, users can manage the corresponding allocated storage in IBP. A sample of exNode file is shown in Figure 5.2.

Chapter 5. Data Management for the IBP-based Data Grid

```
<?xml version="1.0"?>
<exnode xmlns:exnode="http://loci.cs.utk.edu/exnode">
  <exnode:metadata name="Version" type="string">3.0</exnode:metadata>
  <exnode:metadata name="filename" type="string">irm.o</exnode:metadata>
  <exnode:metadata name="lorsversion" type="double">0.700000</exnode:metadata>
  <exnode:mapping>
    <exnode:metadata name="alloc_length" type="integer">119068</exnode:metadata>
    <exnode:metadata name="alloc_offset" type="integer">0</exnode:metadata>
    <exnode:metadata name="e2e_blocksize" type="integer">0</exnode:metadata>
    <exnode:metadata name="exnode_offset" type="integer">0</exnode:metadata>
    <exnode:metadata name="logical_length" type="integer">119068</exnode:metadata>
    <exnode:read>ibp://pprg21.sas.ntu.edu.sg:6714/IBP-5499917932824474781950166730
194055740815651748929921167151885917339178424494218188667384755492112131920560
135968098302553354544009638203426407770739918/158588915/READ</exnode:read>
    <exnode:write>ibp://pprg21.sas.ntu.edu.sg:6714/IBP-5499917932824474781950166730
1940557408156517489299211671518859173391784244942181886673847554921121319205601
35968098302553354544009638203426407770739918/282147953/WRITE</exnode:write>
    <exnode:manage>ibp://pprg21.sas.ntu.edu.sg:6714/IBP-5499917932824474781950166730
19405574081565174892992116715188591733917842449421818866738475549211213192056013
5968098302553354544009638203426407770739918/1784051961/MANAGE</exnode:manage>
  </exnode:mapping>
</exnode>
```

Figure 5.2: A sample of exNode file.

5.2.3 Logistical Backbone

The Logistical Backbone (L-Bone) is a service allowing client to access a collection of IBP depots deployed in the Internet. The basic L-Bone service is IBP depot discovery, where clients can query the L-Bone for depots that meet certain requirements (e.g., available storage, time limits, location preference, and so on), and the L-Bone returns the list of candidate depots (see Figure 5.3). The L-Bone uses information such as IP addresses, zip codes, and country codes to determine the location proximity for the depots [86].

The L-Bone server maintains a list of public depots and metadata about those depots [6]. The metadata includes IBP information such as hostname, port, allocation duration policy, and recent available space values. The L-Bone periodically polls the depots to update the values of available space. The L-Bone can also store geographic location information as well as machine characteristics such as data backup policy, power

5.2 Logistical Networking

backup availability, and others. The L-Bone client library provides the ability to find depots matching specific criteria (available space, duration policy, IBP allocation type, etc.).

Figure 5.3: L-Bone servers and IBP depots.

5.2.4 Logistical Runtime System

Although the L-Bone makes it easier for the user to find depots and the exNode handles IBP capabilities for the user, the user still has to manually request allocations, store the data, create the exNode, attach mappings to the exNode, and insert the IBP allocations and metadata into the mappings. The Logistical Runtime System (LoRS) layer consists of a C API and a command line interface tool set, which can automatically find IBP depots via the L-Bone, operate IBP capabilities, and create exNodes. LoRS facilitates the operations on network files in IBP [6], and it provides the following basic functions:

- Upload: Store data to a network file.
- Download: Retrieve data from a network file.
- Argument: Add replicas to a network file.
- Trim: Remove replicas from a network file.

Chapter 5. Data Management for the IBP-based Data Grid

- Refresh: modify the expiration time of a network file.
- List: view the network file's metadata.

5.3 Globus Replica Catalog

The replica catalog provides location information of replicas in distributed system. With the location information, users can access the desired replicas.

Globus Replica Catalog (GRC) is built based on LDAP. A set of associated files is called a *collection* in GRC. To improve the ability to locate sets of files, GRC supports operations on collections of logical files. The physical location information is associated with each collection to avoid describing each file individual. So a single lookup is sufficient to locate the storage system and path to the collection, and the logical file name can be mapped into physical file name easily [27].

GRC allows users to register files as logical collections and provides mappings between the logical names for files and collections and the physical storage system locations of one or more replicas of these objects [5]. The catalog registers three types of entries: logical collections, locations and logical files.

- A logical collection is a user defined group of files. Users can register and manipulate a set of files as a collection, rather than deal with every file individually.
- Location entries in the replica catalog contain all the information required for mapping a logical collection to a particular physical instance of that collection. The location entry may register information about the physical storage system, such as the hostname, port and protocol. In addition, it contains all information needed to construct a URL that can be used to access particular files in the collection on the corresponding storage system.

5.3 Globus Replica Catalog

Each logical collection may have an arbitrary number of associated location entries, each of which contains a (possibly overlapping) subset of the files in the collection. Using multiple location entries, users can register logical collections that are located in multiple physical storage systems.

- Despite the benefits of registering and manipulating collections of files using logical collection and location objects, users and applications may also want to characterize individual files. For this purpose, the replica catalog includes optional entries that describe individual logical files. Logical files are entities with globally unique names that may have one or more physical instances. The catalog may optionally contain one logical file entry in the replica catalog for each logical file in a collection.

An example of replica catalog using GRC is shown in Figure 5.4 [4]. Collection-1 is a logical collection in the catalog, and three files are listed in its attributes, i.e., fa.dat, fb.dat, and fc.dat. A partial collection of Collection-1 is in Location-A and another partial is in Location-B. The location entries contain attributes of URL constructor of the physical storage and the list of all files that are stored in that location. The attribute of URL constructor in location entry provide all required information (protocol, host, port, path) to map logical file names to physical file names. The logical file entry is optional in GRC. In this example, two logical file entries are created for the files of fa.dat and fc.dat, and they provide size information for individual files.

Globus Replica Catalog provides the following services:

1. Registering a set of files as a logical collection.
2. Registering the physical location for a complete or partial replicas of a logical collection.
3. Registering information about a particular logical file in a logical collection.

Chapter 5. Data Management for the IBP-based Data Grid

Figure 5.4: An example catalog using Globus Replica Catalog [4].

4. Modifying the contents of registered entries in the catalog.
5. Responding to queries of the catalog, such as to find all physical locations for a set of logical files in a logical collection, or to list all the attributes associated with a registered logical collection, location or logical file.

5.4 Data Management

In the logistical networking, exNode provides basic management methods for files in IBP. The operations of uploading, downloading, removing, and extending expired date for files in IBP can be completed with exNode. When a file is uploaded to IBP successfully with LoRS tools, an exNode file is generated and it contains the information to read, write and manage the uploaded file in IBP. The exNode file is required if users want to access the corresponding file stored in IBP.

The logical networking project does not provide data location service for IBP. Al-

5.4 Data Management

though the network storage resources are shared, the files stored in IBP are not shared. After a user uploads a file to IBP using the LoRS tools, he is the only one who has the corresponding exNode file. Other users do not have the exNode file, so they cannot access the file in IBP.

The data management for IBP allows users to easily publish and access shared data resources in IBP, and the underlying details of the logistical networking are transparent to the applications.

5.4.1 System Architecture

The system architecture of data management for the IBP-based Data Grid is shown in Figure 5.5. The whole system includes the logistical networking, Globus Replica Catalog (GRC), exNode File Warehouse (EFW), and IBP Data Management (IDM).

Figure 5.5: System architecture of data management for IBP.

In the logistical networking, IBP depots provide shared network storage resources, and L-Bone servers maintain the directory of IBP depots and metadata about these depots.

Globus Replica Catalog is used to provide the file location service for IBP in this

Chapter 5. Data Management for the IBP-based Data Grid

project. A general logical collection is created in GRC and it contains the exNode file information for general logical files. A *general logical file* is a single file that does not belong to any specific collection.

The services of global file name space, replica consistency, and replica selection are not included in the system and they could be built separately. Our works are mainly on exNode File Warehouse (EFW) and IBP Data Management (IDM).

5.4.2 ExNode File Warehouse

ExNode File Warehouse (EFW) stores and maintains the exNode files of the data in IBP. Other information of the data files, such as the expiration time and time stamp, is also stored in EFW. Normally each organization could set up an EFW. In a global system, there are one or more EFWs.

The architecture of EFW is shown in Figure 5.6. In EFW server, the information of exNode files can be stored in a database or a plain file. In order to speed up the operations, the index of all stored exNode files will be loaded into memory when EFW process starts up. The information loaded in memory will be periodically written back to the secondary storage by the main thread of EFW for update and backup.

A daemon process runs in the EFW server. The main thread of EFW listens to the client connections and creates a new thread for each incoming client connection. The new thread deals with the requests from the client. An expiration checking thread of the EFW daemon checks the exNode files periodically. If it is found that a data file in IBP is expired, the thread will remove the expired exNode file and send request to GRC to remove the corresponding information about the data file. IBP will remove the expired data file automatically.

In order to support multiple replicas for a unique data file in IBP, we must get a way

5.4 Data Management

Figure 5.6: Architecture of ExNode File Warehouse (EFW).

to identify each replica. In this system, an exclusive key is assigned to each replica. The time stamp when the replica is stored in IBP is used as the key for the replica. To add a new exNode file, the user needs to provide the parameters of file name (mandatory), collection name (optional), and expiration time (mandatory). EFW will create a unique key for the uploaded exNode file and return this key to the user. For deleting and refreshing operations, the user must give the specific information for the desired replica, i.e., file name, collection name, and key, all are mandatory. For querying operation, user just needs to provide file name (mandatory) and collection name (optional).

5.4.3 IBP Data Management

IBP Data Management (IDM) interacts with the logistical networking, GRC, and EFW, and it provides APIs for applications. Because LoRS APIs can automatically find IBP depots via the L-Bone, operate IBP capabilities, and create exNodes, IDM calls LoRS APIs directly to access IBP depots.

Chapter 5. Data Management for the IBP-based Data Grid

IDM provides the functions of uploading, downloading, refreshing, deleting, and querying a file or a set of files in IBP. Some of the operations on a data file are shown as follows.

5.4.3.1 Uploading

The protocol for uploading a file is shown in Figure 5.7. First, the application needs to call `initService()` to set some environment variables, such as the GRC location and the default EFW location. Then the application can perform uploading, downloading, refreshing, and deleting operations. The main steps of uploading a file are:

1. The application calls `putFile()` to upload a file to IBP. It specifies the name of the file to be uploaded, the location preference of depot, the duration of the file replicas to be kept in IBP, the number of copies of the file to be created in IBP, etc.
2. Then IDM calls `lorsUploadFile()` to upload the file to IBP. The `exNode` for the uploaded file is returned if the uploading operation is successful.
3. IDM stores the returned `exNode` file to the EFW by calling `putExnode()`.
4. The information of the data file need be registered in GRC. As mentioned before, a single file will be registered to the general collection in GRC. So IDM calls `regLoc2Coll()` to register the EFW in the general collection, calls `regFile2Coll()` to register the file name to the general collection, and calls `regFile2Loc()` to register the file name to the location of the EFW.
5. After above operations are done, IDM notifies the application.

5.4.3.2 Downloading

The protocol for downloading a file is shown in Figure 5.8. The main steps are:

5.4 Data Management

Figure 5.7: Uploading a file with IDM.

1. Application calls `getFile()` to download a file. The only parameter is the logical file name.
2. IDM calls `queryFileXndLoc()` to GRC. If the file exists in IBP, GRC returns the location of the EFW that stores the exNode for the required file.
3. After getting the EFW location, IDM calls `getExnode()` to ask for the exNode from the EFW. The EFW returns the exNode file to IDM.
4. With the exNode file, IDM calls `lorsDownloadFile()` to download the replica from IBP.
5. After the file is downloaded successfully, IDM returns the result to the application.

A data file may have multiple replicas in IBP for fault-redundancy. The sophisticated replica selection service will be included in future work to choose the replica that satisfies

Chapter 5. Data Management for the IBP-based Data Grid

the user specified criteria.

Figure 5.8: Downloading a file with IDM.

5.4.3.3 Refreshing

Refreshing is the operation to extend the expiration time of the file in IBP. The protocol for refreshing a file is shown in Figure 5.9. The main steps are:

1. Application calls **refreshFile()** to extend the expiration time of a file in IBP. The parameters include the logical file name, the key that was returned when the application uploaded the file, and the desired new expiration time.
2. IDM calls **queryFileXndLoc()** to GRC. If the file exists in IBP, GRC returns the location of the EFW that stores the exNode for the required file.
3. After the getting the EFW location, IDM calls **getExnode()** to ask for the exNode from the EFW. The EFW returns the exNode file to IDM.
4. With the exNode file, IDM calls **lrsRefreshFile()** to refresh the replica in IBP.

5.4 Data Management

5. IDM calls `refreshExnode()` to request the EFW to refresh the expiration time of the `exNode` file.
6. IDM returns the result to the application.

Figure 5.9: Refreshing a file with IDM.

5.4.3.4 Deleting

The protocol for deleting a file is shown in Figure 5.10. The main steps are:

1. Application calls `delFile()` to delete a file. The parameters are the logical file name and the key that was returned when the application uploaded the file.
2. IDM calls `queryFileXndLoc()` to GRC. If the file exists in IBP, GRC returns the location of the EFW that stores the `exNode` for the required file.

Chapter 5. Data Management for the IBP-based Data Grid

3. After getting the EFW location, IDM calls `getExnode()` to ask for the `exNode` from the EFW. The EFW returns the `exNode` file to IDM.
4. With the `exNode` file, IDM calls `lorsTrimFile()` to delete the replica in IBP.
5. IDM calls `delExnode()` to request the EFW to delete the `exNode` file.
6. IDM returns the result to the application.

Figure 5.10: Deleting a file with IDM.

5.5 Support for Grid Computing

With IBP data management system, the IBP depots provide a world-wide data cache for Grid computing. A Grid computing architecture supported by the IBP data management system is shown in Figure 5.11. Suppose a user plans to submit a job to the Grid, and the computing sites are distributed in Singapore, US and UK. The job needs to process

5.5 Support for Grid Computing

a large number of data files. The executions of the job are distributed and parallel on different sites. If a data file has been processed by a computing site, the other sites do not need to process it anymore. However, the user can not predict which data files will be used in each computing site.

Figure 5.11: Support for Grid computing.

If without a global data cache, the user needs to stage all data files to every computing site before submitting the job, which wastes the network and storage resources because some data files will not be used at all. Another choice is to transfer the specified data files to individual computing sites till they are requested by the running jobs. Although the network bandwidth and storage are saved, the job execution time will be increased.

With the IBP data management system, the user can upload all data files to IBP first, and the data location information will be registered in GRC automatically. When the user uploads the data files, he can optionally specify the depot locations that are near to the computing sites, so the computing sites can download the desired data quickly. The cost of creating data replicas, such as bandwidth and storage consumption, should be considered under the given conditions.

Then the user can submit the job to the Grid. The only information that the job

Chapter 5. Data Management for the IBP-based Data Grid

needs to carry is all logical names of the data files to be processed. So the job can quickly start up in each computing site.

The job processes running on computing sites will dynamically check which files they need and download the files from IBP using the IDM APIs. At the same time, if the data results are in huge size, the user can also request the job processes to store the results to IBP temporarily and just return the result file names to the user. When the computing job is done, the user can collect the results from IBP at any time.

5.6 Summary

IBP is a middleware for sharing storage resources on network and it is implemented as the base of the logistical networking. As IBP lacks the data management service, the data files stored in IBP are not shareable to public. In this chapter, we put forward a data management system that integrates the technologies of IBP and Globus Replica Catalog. The data management system enables the sharing of data resources among distributed communities. Besides, the system provides a global data cache for the Grid computing to improve the data access performance.

The security services relating to data management for IBP, such as user authentication and access control, are left for future work. In order to implement security in the data management system for IBP, the Grid Security Infrastructure (GSI) [17] should be used. The GSI system is based on a Public Key Infrastructure (PKI), and it has become the de-facto standard for Grid security. GSI provides a set of protocols, libraries and tools that allow users and applications to securely access the Grid resources.

This project was carried out in the middle of 2002. At that time, the Globus Replica Catalog (GRC) was the only available general replica location software for the Data Grid, and it had been applied by several projects, such as European DataGrid [36], GriPhyN [49]

5.6 Summary

and Partical Physics Data Grid (PPDG) [87]. Hence we chose GRC to store the data location information in our system. The Replica Location Service (RLS) [26] was officially released as a component of the Globus Toolkit Version 3.0 in late June 2003. Same as GRC, RLS maintains and provides access to mapping information from logical names to physical locations of replicated data. However, RLS provides higher performance, reliability and scalability compared to GRC. For future work, it will be desirable to replace GRC by RLS for the IBP data management system. The architecture and working mechanism of the system can be kept the same as the original, and the effort of migrating to RLS will be mainly confined to adapting the interfaces of IDM and EFW to RLS.

Chapter 6

Dynamic Replication Algorithms for the Multi-Tier Data Grid

6.1 Introduction

Data replication methods can be classified as static and dynamic. For static replication, after a replica is created, it will exist in the same place till it is deleted manually by users or its duration expires. The drawback of static replication is evident, when client access patterns change in the Data Grid, the benefits brought by replicas will decrease. On the contrary, dynamic replication takes into consideration the changes in the Grid environment and automatically creates new replicas for popular data files or moves the replicas to other sites when necessary to improve the performance.

In this chapter, two dynamic replication algorithms, Simple Bottom-Up (SBU) and Aggregate Bottom-Up (ABU), are proposed aiming at the multi-tier Data Grid, which is a typical architecture adopted in some scientific projects to support large-scale distributed computing. In order to evaluate the performances of the dynamic replication algorithms, the Data Grid simulator called DRepSim is developed. With DRepSim, a simulated Data Grid is constructed according to the estimate for the future Compact Muon Solenoid (CMS) experiment in 2007 [54]. Various simulations are carried out with

Chapter 6. Dynamic Replication Algorithms for the Multi-Tier Data Grid

different data access patterns and replica server capacities. Besides SBU and ABU, the static replication method and Fast Spread dynamic replication strategy [95] are also studied for performance comparison. The performance metrics used are average response time, average bandwidth cost and replication frequency.

The rest of this chapter is organized as follows. Section 6.2 introduces the background of the multi-tier Data Grid and the architecture supporting dynamic data replication. In Section 6.3 two dynamic replication algorithms are proposed. The simulation methods and results are described in Sections 6.4 and Section 6.5 respectively. Section 6.6 summarizes the chapter.

6.2 The Multi-Tier Data Grid

The multi-tier Data Grid architecture was first proposed by MONARC project [79], which aims to model the global distributed computing for the Large Hadron Collider (LHC) [72] experiments. The hierarchical topology provides an efficient and cost-effective method for sharing data, computational and network resources.

The European Organization for Nuclear Research (CERN) is working on the LHC computing program. The users of LHC program will be in hundreds or even thousands and distributed world-wide, and the physicists will access and analyze the experimental data generated at CERN. The multi-tier hierarchical model is adopted as the key feature of the LHC computing. The multi-tier Data Grid architecture for the LHC computing is shown in Figure 6.1 [22, 54]. In the multi-tier model, the raw data generated by the experiments are stored in tier-0. Meanwhile, the data analysis will be carried out by several regional centers in tier-1, many national centers in tier-2, institutional centers in tier-3, and end-user workstations in tier-4. The project of EU DataGrid [36] works on the operations of importing and exporting data among the tiers. To minimize the data access

6.2 The Multi-Tier Data Grid

time and the network load, replicas of the data should be created and spread from the root center to regional centers, or even to national centers. The discussion on influences and costs of distributed and replicated data stores can be found in [105].

Figure 6.1: The multi-tier Data Grid architecture for the LHC computing [22].

The multi-tier Data Grid has many advantages. Firstly, it allows hundreds or even thousands of scientists everywhere to access the resources in a common and efficient way. Secondly, the datasets can be distributed to appropriate resources and accessed at multiple sites. The network bandwidth will be used efficiently because most of the data transfers only use local or national network resources, hence alleviating the workload of international network links. Thirdly, with the support of the Grid middlewares, the resources located in different centers and even end-users can be utilized to support data-intensive computing. Furthermore, the multi-tier structure enables the flexible and scalable management for datasets and users.

In this chapter, the dynamic replication algorithms proposed are aimed at the multi-tier Data Grid.

6.2.1 The System Architecture Supporting Dynamic Replication

The system architecture that supports dynamic replication for the multi-tier Data Grid is shown in Figure 6.2. In the system, each middle-tier node should provide the necessary resources to serve as a replica server for the nodes in lower tiers. A *Local Replica Manager* (*LRM*) runs in each middle-tier node, and it manages the replicas stored in the local site.

Figure 6.2: System architecture supporting dynamic replication.

The *Dynamic Replication Scheduler (DRS)* is the entity that determines the replication for the multi-tier Data Grid. DRS keeps a data access history that contains the information of client access patterns. DRS will invoke the dynamic replication algorithms to process the history at intervals, find out the popular data files, and decide the replication. The historical access information will be cleaned after the processing and the new data access requests occurring in the next phase will be recorded from scratch.

The replicas in the Data Grid must be identified and managed to enable effective

6.2 The Multi-Tier Data Grid

data access. The *Replica Catalog (RC)* stores the mappings from logical file names to physical file names, where the logical file name identifies the individual data file, and the physical file name designates the access method for the corresponding replica. The well-known replica catalog systems include Globus Replica Catalog [27] and its replacer Replica Location Service [26].

Multiple replicas for a data file may exist in the Data Grid. Hence the service should be provided to choose from these available replicas and select one that is the closest to the client. The *Replica Selector (RS)* implements this service. When RS receives a request for a specific data file, it will query RC to get all the available replicas. Then RS will evaluate and predict the performance that each replica can provide in the near future with the assistance of MDS [77] and NWS [83], and return the location of the replica that offers the highest transmission speed. Given the properties of the multi-tier Data Grid, each client will only access the replicas from its ancestor nodes, thus ensuring that the data access management is confined to the regional centers.

When a client wants to access a data file, it will send a request with the data file name to RS to get the location of the closest replica. With the returned replica location, the client will contact the associated LRM and start the data transmission. For each successful data file access, the LRM will notify DRS for logging.

At intervals, DRS checks the data access history to look for the popular data files. According to the current situations of the replicas and the replica servers, the dynamic replication algorithms embedded in DRS make the decision of replication. If DRS has determined that a replica should be created in a middle-tier node, it will send a request to the LRM of the node and ask it to carry out the replication. When the new replica is created successfully, the LRM will notify DRS and let it update RC to ensure consistency of the directory information.

The replica consistency management can be implemented efficiently in the system.

Chapter 6. Dynamic Replication Algorithms for the Multi-Tier Data Grid

The locks of all data files in the Data Grid are stored in RC and managed by DRS. If a node wants to update a data file, it must obtain the file write-lock from DRS. After the write is finished, DRS will propagate the updates to all replicas of data file and release the lock. As the experimental data files in the Data Grid are rarely changed [56], the replica consistency issue will not be addressed in this research.

6.3 Dynamic Replication Algorithms

In this section, the problem of dynamic replication in the multi-tier Data Grid is defined first. Then the dynamic replication algorithms of Simple Bottom-Up and Aggregate Bottom-Up are presented. The Fast Spread replication strategy, which is proposed by other researchers in [95], is described in Section 6.3.4 and it is studied in this chapter for performance comparison.

6.3.1 Problem Definition

On the network topology of the multi-tier Data Grid, each node belongs to a specific *tier*, which is measured by the link hops to the root. Let $tier(v)$ denote the tier number for node v . For any two nodes v and p in the system, if $tier(p) = tier(v) - 1$ and p connects v directly, then p is the *parent* of v , and v is the *child* of p .

The *node ID* is a unique number identifying each node in the Data Grid. Let $nid(v)$ denote the node ID for node v . Following the breadth-first search order, the node ID starts from 0 and increases from upper to lower tier and from left to right node in each tier logically. The node ID can be an integer or a fractional number so long as it is comparable. Keeping the node IDs in order is a simple and efficient method to identify the topological relationships of the nodes.

In the Data Grid, the nodes may be added and removed dynamically. The order of

6.3 Dynamic Replication Algorithms

node IDs can be maintained easily by the following method. If new node v wants to join the system, it must specify its desired parent node p . Place v as the last child of p , and the topological position of v is thus determined. Let $pre(v)$ be the node before v and $post(v)$ be the node after v in the preorder tree walk. If $post(v) \neq nil$, choose any number x satisfying $nid(pre(v)) < x < nid(post(v))$; otherwise, just choose any number x satisfying $nid(pre(v)) < x$. Assign x to the node ID of v . Hence the order of node IDs is maintained without any change to the existing nodes. To remove a node from the system, nothing need to be done if the node is a leaf. If a middle-tier node is removed, all of its descendants must be reinserted into the system. The processes are the same as the aforementioned addition of new nodes.

The dynamic replication algorithm determines when to perform replication, which files should be replicated, and where to place the new replicas. The main purpose of the dynamic replication algorithm is to increase the data access performance from the perspective of the clients.

In the real world, the data access pattern changes from time to time, so the dynamic replication must keep track of the system situation to guide the proper replication. The popularity of a file represents its access rate from the clients. Looking for the potential popular files is the main task of the dynamic replication algorithm. Usually, the popular files in the past phase will continuously be accessed more frequently than the others in the next phase. In this research, the popular data files are identified by analyzing the data access history in DRS. Each record in the history is a tuple of $\langle nodeID, fileID, numOfAccesses \rangle$, which means that the client of $nodeID$ has accessed the data of $fileID$ for $numOfAccesses$ times.

The dynamic replication algorithms are invoked at intervals by DRS and they will process the historical information to determine the replications. After each call of the dynamic replication algorithm, the history of data access will be cleared and the new

Chapter 6. Dynamic Replication Algorithms for the Multi-Tier Data Grid

data access requests will be recorded. In other words, the dynamic replication algorithm breaks the time into *epochs* [65]. At the beginning of each epoch, the replication algorithm is invoked. The duration of the epoch is mainly determined by the arrival rate of data access, and the short epoch should be chosen for the high arrival rate. If the epoch were too long, the collected historical records would be out of date. Thus, the dynamic replication algorithm might attempt to replicate the data that are not popular any more. On the other hand, if the epoch were too short, the collected data access records would be too few to indicate the popular data files properly.

The replica servers will be filled with replicas in the long run, and some replicas must be removed to make room for new ones. In this research, the Least Recently Used (LRU) policy is applied for replica replacement with one more constraint added, that is, the replicas created in the current dynamic replication epoch will not be removed. This additional constraint is to avoid the deletion of newly created replicas by the dynamic replication algorithms.

6.3.2 Simple Bottom-Up

The basic idea of *Simple Bottom-Up (SBU)* is to create the replicas as close as possible to the clients that request the data files with rates exceeding the pre-defined threshold. The details of SBU is shown in Figure 6.3.

The inputs of the algorithm are the data file access history and the threshold that is used to determine the popular files. To identify the creation time of the replicas, the current time is recorded in t at the commencement of the algorithm. Function SORT-DEC is called in line 2 to scan the access history and pick out the records whose *numOfAccesses* values are greater than or equal to the threshold. The records are sorted in descending order of *numOfAccesses* and stored in the local variable array A . Therefore, array A contains the information of the popular files for individual clients, and these files are to

6.3 Dynamic Replication Algorithms

SIMPLE-BOTTOM-UP($H, threshold$)

```

1:  $t \leftarrow \text{GET-TIME}()$ 
2:  $A \leftarrow \text{SORT-DEC}(H, threshold)$ 
3: for all record  $h \in A$  do
4:    $f \leftarrow h.fileID$ 
5:    $p \leftarrow \text{PARENT}(h.nodeID)$ 
6:   while  $p \neq \text{Root}$  do
7:     if  $\text{EXIST-IN}(f, p)$  then
8:        $\text{UPDATE-CTIME}(f, p, t)$ 
9:       break
10:    end if
11:    if  $\text{AVAILABLE-SPACE}(p, t) \geq \text{SIZE}(f)$  then
12:       $\text{REPLICATE}(f, p, t)$ 
13:      break
14:    end if
15:     $p \leftarrow \text{PARENT}(p)$ 
16:  end while
17: end for

```

Figure 6.3: The Simple Bottom-Up replication algorithm.

be replicated on the suitable servers. For each record h in A , SBU gets its associated file ID f and the parent node ID p of the client in lines 4–5.

The **while** loop of lines 6–16 determines where the replica should be created, and the decision path is from the parent node of the client to the root. If a replica of file f exists in node p , then there is no need to replicate this file. The creation time of the replica is updated to the current replication epoch time t by function UPDATE-CTIME (line 8), so that the replica of f in node p will be treated similarly as a newly created one. Break the **while** loop in line 9 and process the next record in array A . If the replica of f does not exist in p and the available space of node p is large enough for file f , REPLICATE is called to create a new replica of f in node p and the creation time of the new replica will be set to t (line 12). Thereafter the algorithm quits the **while** loop. Otherwise, p is modified to point to its parent node (line 15) and the **while** loop is repeated.

Each replica server has a storage limitation and the storage could be used up by

Chapter 6. Dynamic Replication Algorithms for the Multi-Tier Data Grid

replicas. If the free space of the replica server is less than the size of the new replica, then some removable replicas may be deleted to make room for the new replica. The *removable replicas* are defined as the replicas that are created before the current replication epoch and are not used/pinned by any client at present. Let D_p be the set of all replicas in server p , and the set of removable replicas $\tilde{D}_p$ is

$$\tilde{D}_p = \{d \mid d \in D_p, d \text{ is created before the current replication epoch, and } d \text{ is unpinned now}\}. \quad (6.3.1)$$

The first condition of the removable replica prevents the deletion of the newly created replicas, and the second condition avoids the interruption of the ongoing data transmission.

At a specific time, the *available space* of a replica server is the maximum space that the server can provide for the new replicas. Before creating new replicas on the server, its available space must be checked. The available space of the replica server includes its free space and the space occupied by the removable replicas. Denote the free space of server p by $FS(p)$, and denote the size of data d by $Size(d)$, then the available space $AS(p)$ of server p is

$$AS(p) = FS(p) + \sum_{d \in \tilde{D}_p} Size(d) \quad . \quad (6.3.2)$$

Using the above methods, function $AVAILABLE-SPACE(p, t)$ obtains the available space of node p , where the current replication epoch time t is passed in to determine the removable replicas. Only if the available space of the destination replica server is greater than or equal to the size of the data file, can function $REPLICATE$ be called. $REPLICATE(f, p, t)$ will reserve the storage space for file f in node p first, then invoke the transmission of file f to node p in the background. The source of the transmission is p 's closest ancestor node that has the replica of data file f . After the transmission is completed, the new replica's creation time is assigned to t .

6.3 Dynamic Replication Algorithms

6.3.3 Aggregate Bottom-Up

The SBU algorithm processes the records in the access history individually and does not study the relations among these records. Owing to the characteristics of the multi-tier Data Grid, i.e., every node only accesses the replicas that are in its ancestor nodes, the locations of the replica servers and the clients should be carefully considered when carrying out the dynamic replication to fully exploit the abilities of the replication resources.

An example of the data file access history and the network topology of the related nodes are shown in Figure 6.4. The history shows that node C_1 has accessed file X for 10 times, while C_2 and C_3 have accessed file Y for 9 and 8 times respectively. Nodes C_1 , C_2 and C_3 are siblings and their parent node is S_1 .

Figure 6.4: An example of the history and the node relations.

Assume that the SBU algorithm is adopted and the given threshold is 10, so the last two records in the history table will be skipped and only the first record will be processed. The result is that the file X will be created in node S_1 if it has enough space, and file Y will not be replicated. Consider the above example again and it can be found that the decision of SBU is not the optimal, because from the perspective of the whole system, file Y , which has an aggregate of 17 accesses by node C_2 and C_3 , is more popular than file X , which has only 10 accesses by node C_1 . So the better solution is to replicate file Y to S_1 first, then replicate file X to S_1 if it still has enough available space. The *Aggregate Bottom-Up (ABU)* algorithm works in the similar way as the second solution.

Chapter 6. Dynamic Replication Algorithms for the Multi-Tier Data Grid

The basic idea of ABU is to aggregate the history records to the upper tier step by step till it reaches the root. The aggregate method adds up the *numOfAccesses* for the records whose *nodeIDs* are siblings (having the same parent node) and *fileIDs* are same. The *nodeID* of the result record after aggregation will be the parent node ID of those siblings. Taking Figure 6.4 as an example, after aggregation, the results are two records: $\langle S_1, Y, 17 \rangle$ and $\langle S_1, X, 10 \rangle$.

The implementation of the ABU algorithm is shown in Figure 6.5. The inputs of the ABU algorithm include the data file access history and a set of thresholds that are used by each middle-tier. The functions of EXIST-IN, UPDATE-CTIME, AVAILABLE-SPACE and REPLICATE are similar to those in Section 6.3.2.

AGGREGATE-BOTTOM-UP($H, thresholds$)

```

1:  $t \leftarrow \text{GET-TIME}()$ 
2:  $A \leftarrow \text{PROC-HIST}(H)$ 
3: for  $tier \leftarrow ClientTier - 1$  downto  $RootTier + 1$  do
4:    $A \leftarrow \text{AGGREGATE}(A)$ 
5:   for all record  $h \in A$  do
6:     if  $h.numOfAccesses \geq thresholds[tier]$  then
7:       if  $\text{EXIST-IN}(h.fileID, h.nodeID)$  then
8:          $\text{UPDATE-CTIME}(h.fileID, h.nodeID, t)$ 
9:          $\text{REMOVE}(A, h)$ 
10:      else if  $\text{AVAILABLE-SPACE}(h.nodeID, t) \geq \text{SIZE}(h.fileID)$  then
11:         $\text{REPLICATE}(h.fileID, h.nodeID, t)$ 
12:         $\text{REMOVE}(A, h)$ 
13:      end if
14:    end if
15:  end for
16: end for
```

Figure 6.5: The Aggregate Bottom-Up replication algorithm.

The current replication epoch time is recorded in t (line 1). PROC-HIST sorts the records in history H in ascending order based on the *nodeID* field and the results are stored in array A . The **for** loop (lines 3–16) processes each middle-tier from the tier above the clients to the tier below the root. In line 4 the data access records in A are aggregated

6.3 Dynamic Replication Algorithms

to the current tier. The details of function AGGREGATE will be discussed later. After the aggregation, all records' *nodeIDs* are in the current processing tier. For each record *h* in *A*, if *h.numOfAccesses* exceeds the threshold for the current tier, it will be processed further (line 6). If the replica of file *h.fileID* exists in the node of *h.nodeID*, then its creation time is updated to the current replication epoch time and *h* is removed from *A* (lines 7–9). Otherwise, if node *h.nodeID* has enough space for file *h.fileID*, replicate the file to the node and remove record *h* from *A* (lines 10–12). After the inner **for** loop is done, the remaining records in *A* will be aggregated to the next higher tier in line 4, and the updated array *A* will be processed again as stated above.

```

AGGREGATE(A)
1: INIT(B)
2: count ← 0
3: for all record h ∈ A do
4:   merged ← FALSE
5:   for i ← 0 to count − 1 do
6:     if B[i].nodeID = PARENT(h.nodeID) and B[i].fileID = h.fileID then
7:       B[i].numOfAccesses ← B[i].numOfAccesses + h.numOfAccesses
8:       merged ← TRUE
9:       break
10:    else if B[i].nodeID > PARENT(h.nodeID) then
11:      break
12:    end if
13:  end for
14:  if not merged then
15:    B[count].nodeID ← PARENT(h.nodeID)
16:    B[count].fileID ← h.fileID
17:    B[count].numOfAccesses ← h.numOfAccesses
18:    count ← count + 1
19:  end if
20: end for
21: SORT-SEQ(B)
22: return B

```

Figure 6.6: The function of AGGREGATE.

The function of AGGREGATE is shown in Figure 6.6. The input parameter *A* is an

Chapter 6. Dynamic Replication Algorithms for the Multi-Tier Data Grid

array containing the data file access records whose *nodeIDs* are in the same tier and are sorted in ascending order. Array *B* is a buffer with the same structure as *A* and integer *count* indicates the number of records in *B*. The **for** loop in lines 3–20 processes all records in *A*. For each record *h*, the variable *merged* flags whether *h* has been merged with a record in *B*. Lines 5–13 try to merge *h* with a record in *B*. If there is a record *B*[*i*], whose *nodeID* is the parent of *h.nodeID* and whose *fileID* is the same as *h*'s (line 6), then record *h* can be merged with *B*[*i*]. The merging process is simply adding *h.numOfAccesses* to *B*[*i*].*numOfAccesses*, and the flag of *merged* is set to TRUE (lines 7–8). After the merging, the function quits the inner **for** loop and continues to process the next record in *A*. Records in *B* are always arranged in ascending order of *nodeID*, so if the judgement of line 10 is true, then all of *B*'s records whose indices are larger than *i* cannot be merged with *h*. Therefore, the function quits the inner **for** loop and jumps to line 14. If *h* cannot be merged, the function appends a new record to *B* and increases *count* in lines 15–18. The new record's *nodeID* is the parent of *h.nodeID*, and its *fileID* and *numOfAccesses* are the same as *h*'s. After all records in *A* are processed, SORT-SEQ is called in line 21 to sort *B* based on the sequence of {*nodeID*, *numOfAccesses*}, where *nodeID* is in ascending order, but *numOfAccesses* is in descending order. The AGGREGATE function returns *B* as its output.

6.3.4 Fast Spread

Several simple dynamic replication strategies, including the Fast Spread and Cascading replication methods, were put forward by Ranganathan and Foster [95] for a high-performance Data Grid. From their simulation results, Fast Spread has the best performance when the data access pattern is random, while Cascading works better when there is a small degree of geographical and temporal localities. In our simulations, the data access patterns are random, so only Fast Spread is compared with our dynamic

6.4 Simulation Method

replication algorithms.

The idea of Fast Spread is straightforward: for each data file request from the client, a replica of the data file will be created in every middle-tier node along the transmission path. There are two potential methods that can realize the Fast Spread strategy. The first method works in a store-and-forward manner: when a client requests a data file, the data will be transferred and replicated to the lower tier node that is on the transmission path. The step is repeated tier by tier until it reaches the client. The second method transfers the data from the source to the client directly, and each middle-tier node along the transmission path can intercept the data stream. When the data file is transferred to the client completely, the replicas exist at the same time in all middle-tier nodes along the transmission path. Apparently the second method is faster than the first one and there are no bandwidth costs for replication, hence it is adopted in this research.

6.4 Simulation Method

In order to evaluate the properties of the dynamic replication algorithms, a multi-tier Data Grid simulator named *DRepSim* is developed by referring to the architecture shown in Figure 6.2. The network bandwidth simulated by DRepSim is shared equally by all connections. The available bandwidth for a connection is determined by the lowest bandwidth along the transmission path, and the value may change from time to time. There is a replica server running in each middle-tier node, and the replica server supports concurrent data accesses.

With DRepSim, users can easily simulate a Data Grid with the desired parameters, including the topology, network bandwidth, storage size of each node, and data access pattern. Furthermore, various dynamic replication methods can be applied individually to the Data Grid for performance comparison.

Chapter 6. Dynamic Replication Algorithms for the Multi-Tier Data Grid

To demonstrate the advantages of the dynamic replication algorithms, the static replication method is also studied. As the data file popularity transforms with time, it is impossible to deduce an optimal static replication method without knowing the data access pattern in advance. Therefore, the *Random Static Replication (RSR)* policy is applied. Before the start of each simulation, the data files stored in the root node will be replicated to every middle-tier node randomly till all available storage is used up. The replication costs of this initial setup are not considered in our evaluation. We then evaluate the performances of RSR, SBU, ABU and Fast Spread under this environment. For RSR, all replicas created in the initialization step will not be changed and no new replicas will be created during the whole simulation session. On the contrary, the dynamic replication methods of SBU, ABU and Fast Spread will alter the replication status with different strategies after the initial random replication.

The values for interval checking and threshold for SBU and ABU are chosen based on the data access arrival rate, the data access distribution and the capacity of the replica servers. For each simulation case, two to five combinations of different checking intervals and threshold values are tested, and the average results are introduced.

The detailed simulation configurations, including the topology of the simulated Data Grid, data file access patterns and replica server capacities, are introduced as follows.

6.4.1 Topology of the Simulated Data Grid

The simulated Data Grid architecture and network bandwidth between the sites are adopted from the estimations for the CMS experiment to be in operation in 2007 [22, 54]. The infrastructure has four tiers, a root center at tier-0, five regional centers at tier-1, 25 national centers at tier-2, and 125 institute centers at tier-3. Each center in tier-0, 1 and 2 serves five centers in the lower tier. In the topology of the infrastructure, each center is represented as a node. All data requests are generated by the last tier, tier-3. There

6.4 Simulation Method

are 156 nodes in the Data Grid with 125 of them generating data requests in tier-3.

The network bandwidth between any node in tier-3 and its parent node is 622 Mbps, the bandwidth between any node in tier-2 and its parent node is 2.5 Gbps, and it is 2.5 Gbps between any node in tier-1 and the root node. The topology and network configurations of the modeled Data Grid are shown in Figure 6.7.

Figure 6.7: The topology of the modeled multi-tier Data Grid.

6.4.2 Data File Access Pattern

In this research, the main performance metrics are the average response time and the average bandwidth cost, which are defined in Section 6.5. The relative performances of the dynamic replication algorithms are not impacted by the data file size distributions. Furthermore, the data files can be chopped into uniform sized blocks to facilitate system deployment thereby eliminating the file size issue [46]. It is thus reasonable to assign the same size to all the files in the simulations.

There are 10,000 data files in the system of uniform size 2 GBytes. The data access requests from the clients follow Poisson arrival. On average, each client sends one request per 150 seconds. According to the properties of Poisson process, the merging of 125 Poisson streams results in a Poisson with about 0.8 requests per second for the whole system. For each simulation, there are 1,000,000 data file accesses requested by clients.

Chapter 6. Dynamic Replication Algorithms for the Multi-Tier Data Grid

The following distributions are used to simulate the file popularity in the system:

1. Geometric distribution. The probability of requests for the n -th most popular file is defined as $P(n) = p(1 - p)^{n-1}$, where $n = 1, 2, \dots$ and $0 < p < 1$. The geometric distribution is used to model the scenario that some data files are requested more frequently than others. A larger value of p means more requests for a smaller proportion of data files. In this research, the value of $p = 0.01$ is used to model the data request distribution, and this distribution is simply called Geo-.01 hereafter.
2. Zipf-like distribution. In Zipf-like distribution [121, 15], the number of requests for the n -th most popular file is proportional to $n^{-\alpha}$, where α is a constant. It is called *Zipf's law* when α is 1. Zipf-like distribution exists widely in the Internet world [15, 57], including Internet proxy, Web server and Gnutella [48], and the observed parameter values [57] are in the range of $0.63 < \alpha < 1.24$. In this research, parameters of $\alpha = 1.0$ and $\alpha = 0.8$ are used individually, and hereafter we refer to them as Zipf-1.0 and Zipf-0.8 distributions respectively.

In distributed systems, the data file popularity will change with time. With variation in clients' interests, some files that are currently popular may not be popular in the next phase, and vice versa. To generate the data access pattern with dynamically changed popularity, every simulation session is partitioned into 10 sub-sessions evenly according to the number of requests, resulting in 100,000 data accesses in each sub-session. The data file popularity is changed across sub-sessions. Without any loss of accuracy, we associate the file popularity with its ID. In the first sub-session, the most popular data file is the one with ID 0. In the second sub-session, the data file popularity is shifted and the file with ID 1000 becomes the most popular one. Likewise, for every subsequent sub-session, the most popular file ID is shifted by an increment of 1000. In each sub-session, the data file popularity follows the Geo-.01, Zipf-1.0 or Zipf-0.8 distribution.

6.4 Simulation Method

Taking Zipf-1.0 as an example, the distributions of data file request in the whole simulation session and in the fifth sub-session are shown in Figure 6.8 and Figure 6.9 respectively. From the figures it can be found that in each sub-session only a small portion of the data files are accessed with very high frequencies, while most of the other data files are rarely accessed. In the fifth sub-session, the most popular data files are clustered in the file with ID 4000. Overall, the data file popularity of Zipf-1.0 distribution holds in each sub-session, but the popularity of each file is varied in the different sub-sessions to better simulate the dynamically changing access patterns.

Figure 6.8: Distributions of data file requests in the whole simulation session (Zipf-1.0).

6.4.3 Replica Server Capacity

If the storage capacities of replica servers are increased, more replicas can be created. As a consequence, the performance of a particular replication method may be improved. In this research, the effectiveness of different replication methods are studied with a wide range of replica server capacities.

The *relative capacity* of the replica servers is defined as $r = S/D \times 100\%$, where S is the total storage size of the concerned replica servers and D is the total size of all data

Chapter 6. Dynamic Replication Algorithms for the Multi-Tier Data Grid

Figure 6.9: Distributions of data file requests in the fifth sub-session (Zipf-1.0).

files in the Data Grid. If $r = 100\%$, it implies that on average every data file may have a replica in the servers. As stated previously, there are 10,000 data files in the system and each one is 2 GB, so D is around 19.5 TB.

All original data files are stored in the root node, which has unlimited storage size. In all the simulations, every client storage is set as 2 GB, which is the size of one data file. The client storage is cache enabled, in other words, if the client has two successive requests to access a data file, the second request can get the file from the local storage directly.

All replica servers reside in the middle-tier (tier-1 and tier-2) nodes. The overall relative capacity of the replica servers in the Data Grid are varied from 76.8% down to 2.4% in six simulation cases. The detailed storage configurations for all simulation cases are shown in Table 6.1. Taking case 3 as an example, each node in tier-1 has 0.5 TB storage for replicas, so all 5 nodes of tier-1 contribute 2.5 TB storage, and the relative capacity is 12.8% in tier-1. Similarly, there are 25 nodes in tier-2 and each node has 0.05 TB storage, and $r = 6.4\%$ for tier-2. Therefore, the overall relative capacity of the replica

6.5 Performances

servers is 19.2% for case 3.

Table 6.1: Configurations of replica server capacities.

Case	Tier-1		Tier-2		Overall
	total storage size (TB)	relative cap. r	total storage size (TB)	relative cap. r	relative cap. r
1	$2 \times 5 = 10$	51.2%	$0.2 \times 25 = 5$	25.6%	76.8%
2	$1 \times 5 = 5$	25.6%	$0.1 \times 25 = 2.5$	12.8%	38.4%
3	$0.5 \times 5 = 2.5$	12.8%	$0.05 \times 25 = 1.25$	6.4%	19.2%
4	$0.25 \times 5 = 1.25$	6.4%	$0.025 \times 25 = 0.625$	3.2%	9.6%
5	$0.125 \times 5 = 0.625$	3.2%	$0.0125 \times 25 = 0.3125$	1.6%	4.8%
6	$0.0625 \times 5 = 0.3125$	1.6%	$0.00625 \times 25 = 0.15625$	0.8%	2.4%

6.5 Performances

In this section, the performance results of the dynamic replication algorithms are presented and discussed. The studied performance metrics include the average response time, average bandwidth cost and replication frequency.

Average response time. The *response time* for a data file access is the interval between the beginning of the data request sent by the client and the end of the data transmission. The *average response time* is the mean value of the response times for all data accesses. This performance metric is from the perspective of the clients.

Average bandwidth cost. Normally, the price of the international link is higher than that of the national link [1], and the prices for LAN and campus network are negligible. In this research, the unit price is set as 5 for the link between tier-0 and tier-1, as 2 for the link between tier-1 and tier-2, and as zero for the link between tier-2 and tier-3. For each data transmission, the *bandwidth cost* is the data size multiplied by the summation of unit prices along the data transmission path, and

Chapter 6. Dynamic Replication Algorithms for the Multi-Tier Data Grid

it can be defined as $cost = size \cdot \sum w_i$, where w_i is the unit price for link i in the path. The value of the *average bandwidth cost* is the total bandwidth cost divided by the number of client data requests, where the total bandwidth cost includes the bandwidth consumed by the client requests and the data replications.

Replication frequency. The metric of *replication frequency* is defined as the ratio of the frequency of replication to the frequency of data access from clients, namely it is the value of how many replications occur per data access. This metric indicates the efficiency of data replication.

6.5.1 Average Response Time

The simulation results for the methods of RSR, SBU and ABU in the first four replica server capacity cases (refer Table 6.1) are shown in Figure 6.10. As expected, the average response times of both dynamic replication algorithms are less than the random static replication method for the same data access pattern and storage capacity. In particular when the storage size of replica server is small, the benefits of dynamic replication are distinct. The average response time of RSR increases drastically when the replica server capacity (r) decreases.

To demonstrate the performances of the dynamic replication algorithms clearly, Figure 6.10 is zoomed in and shown in Figures 6.11, 6.12 and 6.13 according to the different data access distributions. In these figures, the results of simulation cases 5 and 6 in Table 6.1 are added, and the results of Fast Spread are plotted for comparison. Evidently, with increasing storage size, the performances of all methods are improved by different degrees.

For the data access distribution of Geo-.01 (refer Figure 6.11), the average response time of ABU is less than 70% of Fast Spread in any case, and SBU is higher than

6.5 Performances

Figure 6.10: Performance comparison of RSR, SBU and ABU.

Figure 6.11: Average response time vs. replica server capacity for Geo-.01.

Chapter 6. Dynamic Replication Algorithms for the Multi-Tier Data Grid

Figure 6.12: Average response time vs. replica server capacity for Zipf-1.0.

Figure 6.13: Average response time vs. replica server capacity for Zipf-0.8.

6.5 Performances

Fast Spread when the relative storage capacity is less than 38.4%. For Zipf-1.0 (refer Figure 6.12), the average response time of ABU is slightly higher than Fast Spread when the relative storage capacity is less than 19.2%, but the differences are less than 7% of Fast Spread. When the relative capacity is larger than 38.4%, ABU's average response time is slightly smaller than Fast Spread. The average response time of SBU is always the highest among the three dynamic replication algorithms. For Zipf-0.8 (Figure 6.13), ABU outperforms Fast Spread for any relative storage capacity, and it exhibits more advantages than Fast Spread when the storage capacity is decreased. SBU's average response time is higher than Fast Spread when the relative storage capacity is less than 19.2%.

In most conditions, ABU gives the best average response time among the studied dynamic replication methods. The only exception is when the data access pattern is Zipf-1.0 and the relative replica server capacity is less than 19.2% (refer Figure 6.12). This is because when the data access pattern follows Zipf-1.0, the clients focus on a smaller range of data files with higher frequencies compared to Zipf-0.8 and Geo-.01. Therefore, the popular data files can be easily identified even by the simple strategy of Fast Spread. Consequently the advantage of ABU over Fast Spread in exploring popular data files is diminished. In another aspect, ABU consumes network bandwidth when creating new replicas, but Fast Spread does not in the simulations (refer Section 6.3.4). The impact of the bandwidth costs of data replications on ABU is amplified when the replica server storage size is small.

6.5.2 Average Bandwidth Cost

The average bandwidth cost measures the network resource consumption from the view of the whole system, while the average response time is the performance metric from the perspective of end-users.

Chapter 6. Dynamic Replication Algorithms for the Multi-Tier Data Grid

To effectively demonstrate the cost results, we calculate the average bandwidth cost when there is no replica in the Data Grid and all data are transferred from the root node to the clients, and this cost is used as the *baseline*. Then the *relative average bandwidth cost* is calculated as the ratio of the average bandwidth cost to the baseline. A smaller relative average bandwidth cost means better performance. The results are shown in Figure 6.14, and the solid lines, dash-dot lines and dotted lines are for Geo-.01, Zipf-0.8 and Zipf-1.0 data access distributions respectively.

Figure 6.14: Average bandwidth cost vs. replica server capacity.

For RSR, the lines are clustered together for all access patterns, namely its network bandwidth costs do not change evidently for different data access patterns. When the storage size is increased, the bandwidth cost of RSR is reduced in a proportional degree.

Because the dynamic replication algorithms create replicas close to the clients for the popular data files, a majority of data access requests from the clients are served by the middle-tier nodes. As a result, the loads of the upper tier links are alleviated and the average bandwidth costs of the dynamic replication algorithms are comparatively low. It also shows the advantages of dynamic replication in reducing the network resource

6.5 Performances

consumption.

For any data access pattern and storage capacity, the average bandwidth cost of ABU is the lowest, Fast Spread is the medium, and SBU is the highest among the three dynamic replication methods. Concerning the impacts of the data access patterns, the dynamic replication algorithms' bandwidth costs for Zipf-1.0 data access pattern (dotted lines) are lower than those for Zipf-0.8 (dash-dot lines) as a whole. As mentioned before, that is because when the data access pattern follows Zipf-1.0, the clients focus on fewer data files with higher frequencies than Zipf-0.8. As a consequence, the popular data files can be easily identified by the dynamic replication algorithms and the performance results are improved. For both Zipf-like data access patterns, the differences in the relative average bandwidth costs between any two dynamic replication methods are less than 0.05 on average. However, for Geo-.01 data access pattern, the performance differences between any two dynamic replication methods are more distinct and at values higher than 0.12.

6.5.3 Replication Frequency

For each replication operation, not only is the network bandwidth resource consumed, but also the replica server load is increased because of the disk I/O and CPU utilizations. Therefore, the frequency of replication operation must be controlled to avoid heavy network and server load.

The results of the replication frequency are shown in Figure 6.15. In any case, the replication frequencies of SBU and ABU are less than 0.03, that is, at most three replicas are created per 100 data accesses from clients. However, the frequencies are higher than 1.3 for Fast Spread in all conditions, which means that at least 1.3 replicas have been created for each data access.

It can be concluded that the replication frequencies of both SBU and ABU are rea-

Chapter 6. Dynamic Replication Algorithms for the Multi-Tier Data Grid

Figure 6.15: Replication frequency of the dynamic replication methods.

sonably low even if the replica server capacity is very small. However, the replication frequency of Fast Spread is too high which renders it not feasible in the real world.

6.5.4 Discussion

The primary goal of dynamic data replication is to shorten the average response time that is experienced by the end-user. At the same time, from the view of the whole system, the performance metrics of bandwidth consumption and replication frequency must be considered to make sure that the dynamic replications do not cause heavy load on the system.

From the simulation results we know that the dynamic replication methods of SBU, ABU and Fast Spread can significantly reduce the average response time and average bandwidth cost when compared to the static replication. In most situations, ABU surpasses SBU and Fast Spread on the performance metrics of average response time and average bandwidth cost. The reason is that ABU can find the popular data files effectively based on past data access history, and furthermore, it considers the locations of

6.6 Summary

replica servers and clients when deciding the replication destinations, so that the replica server storage is properly utilized.

Although Fast Spread's performances in terms of average response time and average bandwidth cost are generally better than SBU, the Fast Spread strategy is not practical in the real world because it causes high replication frequency, which inflicts heavy load on replica servers. In the simulations, the overheads incurred by high-frequency replication are not considered for Fast Spread, so the simulated performance results of Fast Spread are ideally optimal.

The benefits of SBU and ABU are achieved by creating only a small number of replicas, which indicates that both algorithms are successful in deciding which data file should be replicated and where the replica should be created.

The capacity of replica server has a major impact on the performances of the dynamic replication algorithms. Increasing the replica server capacity leads to performance improvement in average response time and average bandwidth cost. ABU is less sensitive to replica server capacity than SBU, so even when the storage size of the replica server is very small, its average response time is still favorable.

6.6 Summary

In this chapter, two dynamic replication algorithms, SBU and ABU, are put forward for the multi-tier Data Grid. At intervals, the dynamic replication algorithms exploit the data access history for the popular data files and compute the replication destinations to improve data access performance.

The simulator DRepSim is created to study the performances of the dynamic replication algorithms. A simulated Data Grid is built with DRepSim and diverse simulations are carried out by varying the settings of data access patterns and replica server ca-

Chapter 6. Dynamic Replication Algorithms for the Multi-Tier Data Grid

capacities. The results demonstrate that the dynamic replication can shorten the average response time of data access greatly and reduce the bandwidth consumptions compared to the static replication method.

The performances of SBU and ABU are compared with the Fast Spread strategy. In all studied situations of diverse data access patterns and replica server capacities, ABU exhibits outstanding performances in terms of both average response time and average bandwidth cost, and its performances are better than SBU and Fast Spread in most cases. Although Fast Spread's performances in average response time and average bandwidth cost are better than SBU generally, considering its high replication frequency, Fast Spread is not a practical replication strategy because it induces heavy I/O and CPU load on the replica servers. On the contrary, the low replication frequencies of SBU and ABU indicate that both algorithms are cost-effective.

Chapter 7

Combining Data Replication and Job Scheduling in the Data Grid

7.1 Introduction

The Grid resources, including computing facility, data storage and network bandwidth, are consumed by the jobs. For each incoming job, the Grid scheduler decides where to run the job based on the job requirements and the system status. Job scheduling heuristics for the distributed computing systems have been extensively studied. Maheswaran *et al.* [76] introduced matching and scheduling heuristics for a class of independent jobs. The heuristics are classified into online and batch modes. Hamscher *et al.* [52] presented three scheduling structures, including centralized, hierarchical and decentralized, for the Grid. Schopf [100] proposed a general architecture for scheduling on the Grid. The stages of Grid scheduling, including resource discovery, system selection and job execution, were described. Shan and Olier [101] proposed a superscheduler (Grid scheduler) architecture and three distributed job migration algorithms. The performances of idealized, centralized and distributed scheduling schemes were compared using real workloads. However, all these scheduling research does not take into account the influence of data replicas.

For data-intensive applications, the location of the data required by the job is an

Chapter 7. Combining Data Replication and Job Scheduling in the Data Grid

important parameter for the Grid scheduling. Creating data replicas can reroute the data requests to certain replica servers and offer remarkably higher access speed compared to a single server. At the same time, the replicas provide broader decision space for the Grid scheduler to achieve better performance from the perspective of the job. On the other hand, Grid job scheduling can change the data access pattern and then affect the replication status. Therefore, Grid scheduling and data replication are two tightly correlated problems in the Data Grid, and it will be beneficial to study their interactions.

In this chapter, we propose a Data Grid architecture supporting efficient data replication and job scheduling. The computing sites are organized into individual domains according to the network connection, and a replica server is placed in each domain. Two centralized dynamic replication algorithms with different replica placement methods and a distributed dynamic replication algorithm are put forward. Data replication is not an isolated problem in the Data Grid. In particular, Grid job scheduling is tightly correlated with data replication. Hence, the representative Grid scheduling heuristics of Shortest Turnaround Time, Least Relative Load and Data Present are proposed. In order to evaluate the performance of the scheduling heuristics combined with different data replication methods, a Data Grid simulator called XDRepSim is developed. Various simulations are carried out with different system configurations and job workloads.

The rest of this chapter is organized as follows. Section 7.2 introduces the system model and performance metric for the Data Grid. Section 7.3 presents our dynamic data replication algorithms. The Grid scheduling heuristics are put forward in Section 7.4. The simulation methods and results are described in Section 7.5 and Section 7.6 respectively. Section 7.7 summarizes the chapter.

7.2 System Model and Performance Metric

7.2 System Model and Performance Metric

The Data Grid architecture supporting data replication and job scheduling is shown in Figure 7.1. The Data Grid consists of a set of domains, and each domain contains a replica server and many computing sites. The replica server provides storage for the replicas, and the computing site offers computational resources for the jobs. Computing site and replica server are generally called *nodes*. All nodes in a domain are served by a local-area network (LAN), typically the network bandwidth between each two nodes in a domain is in the range from 10Mbps to 100Mbps. The domains are connected by wide-area network (WAN), and the network bandwidth between any nodes in different domains is normally less than 10Mbps. Some isolated replica servers might exist in the Data Grid aiming to improve remote data access performance, and each of them constructs a domain by itself.

Figure 7.1: System architecture of the Data Grid.

If a computing site and a replica server are in the same domain, the replica server is defined as the computing site's *Primary Replica Server (PRS)* and all other replica

Chapter 7. Combining Data Replication and Job Scheduling in the Data Grid

servers residing in other domains are the computing site's *Secondary Replica Servers* (*SRS*).

There are one or more *Grid schedulers* in the Data Grid, which assign jobs to the computing sites based on particular strategies. Due to the large size of the data and the scale of the Data Grid, online access to data repositories is not efficient for large numbers of jobs [110]. Therefore, before a job runs in a computing site, its required input data should be loaded to the local storage of the computing site in advance. The data may be accessed from the computing site's data cache directly. If the required data is not in the computing site's data cache, the site will send the data access request to its PRS, which will locate all replicas of the data and select the one that provides the highest available bandwidth to the computing site. The selected replica will be transferred to the computing site.

To simulate the real world characteristic, if there is no replica server in a domain, the above architecture can still be applied by placing a dummy replica server in the domain. The storage size of the dummy replica server is zero so that no replicas will be created in the server. All data requests from the computing sites in the domain will be forwarded to suitable SRS. On the other hand, if a domain contains multiple replica servers physically, we can regard them as a single PRS, which aggregates the capabilities of these replica servers.

Let W be the set of domains in the Data Grid. For each domain $w \in W$, the set of computing sites located in w is denoted by $CS(w)$, and the replica server in w is denoted by $RS(w)$. The set of nodes in domain w is denoted by $NS(w)$, where $NS(w) = CS(w) \cup RS(w)$. For a computing site i , its computing capability is C_i .

The notion of the domain in our study is similar to the *I/O community* proposed by Thai *et al.* [110]. In their study, execution and storage sites are bound into I/O communities, which reflect the physical reality and participate in the wide-area system.

7.2 System Model and Performance Metric

Each I/O community hosts a storage appliance to serve both local and remote data access. A job requiring particular data may be moved to a community where the data has been staged, or data may be staged to the community in which the job has been placed.

7.2.1 Performance Metric

In distributed and parallel systems, widely used performance metrics for job scheduling include turnaround time, throughput, utilization, makespan and slowdown. However, the performance metrics are not applicable to all environments. *Turnaround time* measures how long a job takes from its submission to completion. Feitelson *et al.* [38] claim that turnaround time is the main metric for the open online system. As the system utilization and throughput are largely determined by the job arrival process and the job resource requirements rather than by the scheduler, they are only suitable for closed systems, in which every job is re-submitted to the system when it terminates. Makespan is used for batch mode scheduling [38, 76]. The makespan of a schedule is the latest completion time of any job in the collected set of jobs. Slowdown is defined as the job turnaround time divided by its execution time [38, 29]. In this chapter, we study the moldable job workload [39], i.e., jobs can run with multiple partition sizes. As the execution time of a moldable job depends on the partition size it uses, to let a moldable job run with small partition size can increase its execution time. Although the slowdown is improved as a result, the job turnaround time is increased. Hence slowdown is not an appropriate performance metric for moldable jobs [29].

Although the average turnaround time is a straightforward metric for the Grid scheduling performance, it is dominated by long jobs. Its value will increase drastically if there are a few very long jobs. The geometric mean of turnaround time was used by the previous research [29] as the main performance metric. The *geometric mean of turnaround*

Chapter 7. Combining Data Replication and Job Scheduling in the Data Grid

time is defined as:

$$GM\!T\!T = \left(\prod_{k \in K} TT_k \right)^{\frac{1}{|K|}},$$

where TT_k is the turnaround time of job k , K is the set of jobs in the concerned workload, and $|K|$ is the number of jobs. A lower GM $\!T\!T$ implies better performance. The geometric mean equally considers the performance improvement of any job and it does not favor long jobs [29], so it can evaluate the scheduling performance more objectively than the arithmetic mean. For example, for 100 jobs whose turnaround times are the same as one-hour, and one job whose turnaround time is one-week, the arithmetic mean of turnaround time is 2.65 hours, while the GM $\!T\!T$ is 1.05 hours. Hence GM $\!T\!T$ is chosen by this research as the performance metric.

7.3 Dynamic Data Replication

As stated in the previous chapter, the dynamic replication mechanism takes into consideration changes in the Grid environment, and automatically creates new replicas for popular data files or moves the replicas to other sites when necessary to improve the data access performance. It is assumed that popular data in the previous phase will remain popular in the near future. The dynamic data replication algorithms discussed in this chapter determine the popularity of data files by analyzing the data access history.

In this research, replication aims to shorten data access time perceived by the job. As the job could be submitted to any computing site for execution, which is determined by the Grid scheduler, there is no inherent data access pattern for individual computing site. Therefore, the locality information of which computing site had previously accessed the data file is not recorded in the data access history.

Let FID be the abbreviation for *File ID*, and NOA for *Number Of Accesses*. The history table of data access is in the format of $\langle FID, NOA \rangle$, which indicates that file

7.3 Dynamic Data Replication

FID has been accessed for NOA times. The FID field is the primary key for the history table. For each record h in history H , let $FID(h)$ denote its corresponding file ID, and $NOA(h)$ be the number of accesses.

The dynamic replication algorithms are invoked at intervals and they will process the historical information to determine the replications. After each call of the dynamic replication algorithm, the historical content will be cleared and the new data access requests will be recorded. The time interval or called *epoch* (refer Section 6.3.1) to invoke the dynamic replication algorithms is mainly determined by the arrival rate of data access, and a short interval should be chosen for high arrival rate. The replica servers will be filled with replicas in the long run and some replicas must be evicted to make room for new ones. In this research, the Least Recently Used (LRU) algorithm is applied for replica replacement with the constraint that only the replicas created in previous epochs can be evicted from the replica servers. Equation (6.3.1) is used to identify the replicas that can be removed from the replica server, while Equation (6.3.2) is used to measure the available space of the replica server.

According to the architecture of the replication system, the proposed dynamic replication algorithms are classified into centralized and distributed ones. In addition to looking for popular data files, the centralized replication algorithm needs to determine the replica placement.

7.3.1 Centralized Dynamic Replication (CDR)

In the centralized dynamic data replication architecture, there is a replication master running in the Data Grid. Every PRS collects the records of data accesses that are initiated by the computing sites in its domain. When it is time to replicate data, all PRSs send the collated historical information to the replication master. The master aggregates the histories and summarizes the values of NOA for the same FID . The

Chapter 7. Combining Data Replication and Job Scheduling in the Data Grid

aggregation results are stored in the history table H , which is maintained by the master. Each record h in H indicates that data $FID(h)$ has been accessed for $NOA(h)$ times for the Data Grid in the past epoch.

The centralized dynamic data replication algorithm is invoked by the replication master, which commands the replica servers to carry out the data replications. The algorithm is presented as follows:

1. Compute the average number of accesses $\overline{NOA} = \frac{1}{|H|} \sum_{h \in H} NOA(h)$, where $|H|$ is the number of records in H (also the number of data files that have been requested).
2. Remove the history records whose NOA values are less than $\overline{NOA}$. Sort the remaining history records based on NOA in descending order. Let l be the last record in H , and denote $MNOA = NOA(l)$, where $MNOA$ stands for the *minimal number of accesses* among the popular files to be replicated.
3. While H is not empty:
 - (a) Pop the first record h off H .
 - (b) Invoke the RTPlace (refer Section 7.3.1.1) or SMPlace (refer Section 7.3.1.2) method to determine the placement for the replica of $FID(h)$.
 - (c) Update record h and let $NOA(h) \leftarrow NOA(h) - MNOA$. If $NOA(h) \geq MNOA$, then re-insert record h into H according to the descending order of NOA field.

In the algorithm, the average number of accesses $\overline{NOA}$ is used as the threshold to distinguish popular data files, and only the files that have been accessed for more than $\overline{NOA}$ times will be replicated. Among the popular data files sorted out, the files with larger NOA may be replicated more times, and the files that have the fewest number of accesses ($MNOA$) will be replicated once. Every time a data file is replicated to a server, its NOA is decreased by $MNOA$. If the new NOA value of the data file is greater than

7.3 Dynamic Data Replication

or equal to *MONA*, the corresponding record will be re-inserted into the sorted list for more replications.

The replica placement methods of RTPlace and SMPlace are introduced in the following two sub-sections respectively.

7.3.1.1 Response-Time Oriented Replica Placement (RTPlace)

The computing sites with higher CPU speed can finish more jobs in a given duration. This results in the sites accessing data files more frequently. The computing capability of domain w is defined as $C_w = \sum_{i \in CS(w)} C_i$, and the computing capability of all domains is $C_W = \sum_{w \in W} C_w$. We assume that the data request rate from domain w is proportional to its computing capability. Let θ be the factor that measures the proportional relationship between the computing capability and the data request rate for any domain. Then the data request rate from domain w can be denoted by $Q(w) = \theta \cdot C_w$. Let $Prob_f$ be the request probability for data f , then the request rate from domain w for data f can be denoted by $Q(w, f) = Prob_f \cdot Q(w) = \theta \cdot Prob_f \cdot C_w$.

In the Data Grid, every computing site always accesses the replica that offers the highest available bandwidth. All computing sites in the same domain should have equivalent bandwidth to a specified replica server. Let $B_{w,j}$ be the bandwidth capacity from any node in domain w to replica server j . Let R_f be the set of replica servers that contain the replicas or the original copy of data f . The bandwidth capacity for any node in domain w accessing data f can be defined as $AB(w, f) = \max_{j \in R_f} B_{w,j}$. The size of data f is denoted by $Size_f$. The average response time of all requests for data f in the Data Grid can be defined as:

$$AvgRespTime(f) = \frac{\sum_{w \in W} \frac{Q(w, f) \cdot Size_f}{AB(w, f)}}{\sum_{w \in W} Q(w, f)} = \frac{Size_f}{C_W} \cdot \sum_{w \in W} \frac{C_w}{\max_{j \in R_f} B_{w,j}}.$$

As $\frac{Size_f}{C_W}$ is constant, we only need to consider $\sum_{w \in W} \frac{C_w}{\max_{j \in R_f} B_{w,j}}$ to get the minimal

Chapter 7. Combining Data Replication and Job Scheduling in the Data Grid

average response time for data f .

Let $\mathcal{R}_f$ be the set of servers that contain the replicas created in the current replication epoch or the original copy of data f . To create one more replica for data file f , the replica placement strategy will evaluate every candidate replica server x , which has enough available storage (refer Equation (6.3.2)) and $x \notin \mathcal{R}_f$. Attempt to let x be the chosen replica server and calculate the value of:

$$Y(f, x) = \sum_{w \in W} \frac{C_w}{\max_{j \in \mathcal{R}_f \cup x} B_{w,j}} \quad .$$

Select replica server $\hat{x}$ to minimize function Y and let it be the location of the new replica for data f . If data file f already exists in server $\hat{x}$, just stamp the replica's creation time as the current replication epoch, so that it will be treated as a newly created one. Otherwise, transfer data f to server $\hat{x}$ from a replica server that stores f and offers the highest transfer bandwidth. Let $\mathcal{R}_f \leftarrow \mathcal{R}_f \cup \hat{x}$.

7.3.1.2 Server Merit Oriented Replica Placement (SMPlace)

The replica placement can be carried out by studying the location characteristics of the replica servers relative to all domains in the Data Grid. Placing the replica in the server that is close to every domain may induce small average data access cost, where the higher bandwidth means the shorter distance. The domain with larger computing capability may access the data file more frequently than others. We define the *location merit* of replica server j as:

$$Merit(j) = \sum_{w \in W} \frac{C_w}{B_{w,j}} \quad .$$

A smaller value of $Merit(j)$ implies that replica server j is in a more crucial location. To place a replica in the server with a small merit value may result in a short average data transfer time.

After calculating the merits of all replica servers, the centralized replication master

7.3 Dynamic Data Replication

keeps a replica server list that is sorted in ascending order of server's merit. The ordered list is used in deciding replica placement.

For each request to create a replica for data f , the replica placement function will check the replica server list sequentially. Once it finds a server that does not contain replica f created in the current replication epoch and has enough available space (refer Equation (6.3.2)) for data f , the data will be replicated to that server.

Hereafter the centralized replication algorithms with RTPlace and SMPlace placement methods are referred to as CDR_RTPlace and CDR_SMPlace for short, respectively.

7.3.2 Distributed Dynamic Replication (DDR)

In the distributed dynamic data replication architecture, for every data access request from a computing site, the PRS records the request into its history table. The historical information will be exchanged among all replica servers. Every replica server aggregates NOA over all domains for the same data file and creates the overall data access history of the Data Grid. At intervals, each replica server will use the replication algorithm to analyze the history and determine data replications. The distributed dynamic data replication algorithm is shown as follows:

1. Compute the average number of data accesses $\overline{NOA}$ and let $threshold = \overline{NOA} + \delta$, where $\delta \geq 0$.
2. Remove the history records whose NOA values are less than $threshold$. Sort the rest of the history records based on the field of NOA in descending order.
3. For each record h in the history, try to create a replica of data file $FID(h)$ at the local replica server till the storage is full.

Chapter 7. Combining Data Replication and Job Scheduling in the Data Grid

The increment δ is used to avoid excessive data replications that will cause heavy network traffic, and it can be chosen depending on how much we are willing to compromise on the quality of replication. To avoid the network contention caused by the large number of data replications happening simultaneously, each replica server invokes the distributed replication algorithm with a different start time.

7.4 Grid Scheduling

As Grid resources are under the control of local schedulers, which provide access points to the resources, the Grid schedulers must be built on top of the existing local schedulers. In order to get the information about the Grid system especially network bandwidth and data replicas, the Grid schedulers need the support of MDS [77], NWS [83] and RLS [97]. According to the requests of the incoming jobs, the Grid schedulers analyze the system situations, communicate with the low-level local schedulers and decide which resources should be allocated to the jobs. On behalf of the end-users, the Grid schedulers submit the jobs to the local schedulers.

The experimental results in [101] indicate that the local scheduling algorithm has significant impact on the Grid performance in their experiment. The First-Come-First-Serve with backfilling (FCFS+BF), First-Fit (FF) and Shortest-Job-First (SJF) algorithms are evaluated with simulations, and the results show that FCFS+BF has the best performance in terms of average wait time and turnaround time. Thus, in this research the FCFS algorithm is adopted by the local schedulers. In addition, following the idea of the aggressive backfilling [81] to improve the scheduling performance, the local schedulers can move a job ahead of others in the queue and begin execution as long as it does not delay the first queued job. Therefore, if the first queued job cannot be executed since it is still waiting for the completion of transmission for the input data file, another later job whose input data has been loaded in the computing site can be executed immediately.

7.4 Grid Scheduling

This method can improve the system utilization.

The jobs are moldable [39], that is, they can run with multiple partition sizes. Each job is scheduled to a single computing site, and it runs on all processors in the site. Same as [96, 106, 78, 32], we assume that every job requires an input data file to process. Since the job's output is significantly smaller than the job's input in general [55], the output cost is ignored in this research.

For each job submission, the user needs to estimate the computational cost and specify the input data file. The job *computational cost* is the execution time when the job runs on a benchmark computer, and it is also called the *benchmark execution time*. Let BT_k be the user estimated computational cost and $f(k)$ be the input data file for job k .

Job scheduling heuristics are classified into batch mode and online mode [76]. In the batch mode, the jobs are gathered into a set during a specific duration, and the set of jobs is considered and scheduled at predetermined events. On the contrary, with the online mode scheduling, a job is scheduled to the processors as soon as it arrives at the scheduler. Although the batch mode scheduling has a wider decision scope over the collected jobs, the online mode scheduling is more appropriate for the Grid because of its simple and fast implementation and zero delay scheduling. Furthermore, for data intensive computing, the data transfer induces heavy overhead for the job turnaround time. After a job is scheduled to a computing site, the required data can be transferred to the site simultaneously when the job is in the queue. Thus, early determination of the computing site for the job to run can shorten the job turnaround time. The Grid scheduling heuristics studied in this chapter are of online mode.

We study the representative Grid scheduling heuristics that take into account different factors impacting the job turnaround time. In the following three proposed Grid scheduling heuristics, the Shortest Turnaround Time heuristic considers the data transfer time, queuing time, and execution time all together for the job; the Least Relative Load

Chapter 7. Combining Data Replication and Job Scheduling in the Data Grid

heuristic tries to minimize the job's queuing and execution time by balancing the load among computing sites; and the Data Present heuristic mainly attempts to minimize the data transfer time for the job.

7.4.1 Shortest Turnaround Time (STT)

For each incoming job, the *Shortest Turnaround Time* (*STT*) heuristic estimates the job turnaround time on every computing site and assigns the job to the site that provides the shortest turnaround time. The estimated turnaround time of job k running in computing site i is denoted by:

$$TT_{k,i} = \max\{QT(i), DT(f(k), i)\} + ET_{k,i} \quad (7.4.1)$$

where $QT(i)$ is the queuing time, $DT(f(k), i)$ is the data transfer time, and $ET_{k,i}$ is the job execution time.

As every job's actual computational cost is unknown to the scheduler, it is hard to accurately predict the queuing time for the new job in a computing site. Assume that every job in the queue executes immediately after the previous job completes, that is, data transmission does not delay the job start time. Thus, the new job queuing time in computing site i can be approximated as:

$$QT(i) = \frac{\sum_{job \in Queue(i)} BT_{job}}{C_i} \quad ,$$

where $Queue(i)$ is the set of jobs already in the queue of computing site i .

If job k 's input data file $f(k)$ is in computing site i 's data cache, then $DT(f(k), i) = 0$. Otherwise, let $R_{f(k)}$ be the set of replica servers that have data file $f(k)$, and $BW(i, j)$ be the available bandwidth between computing site i and replica server j at the moment. The data file downloading time can be estimated as:

$$DT(f(k), i) = \frac{Size_{f(k)}}{\max_{j \in R_{f(k)}} BW(i, j)} \quad .$$

7.4 Grid Scheduling

Let the benchmark computer's computing capability be normalized as one, then the execution time in Equation (7.4.1) for job k running in computing site i can be estimated as:

$$ET_{k,i} = BT_k / C_i \quad .$$

Hence, according to the estimated turnaround time with Equation (7.4.1), the job is scheduled to the computing site that yields the shortest estimated turnaround time.

7.4.2 Least Relative Load (LRL)

The *relative load* of computing site i is defined as:

$$RL_i = \frac{NumOfJobs(i) + 1}{C_i} \quad ,$$

where $NumOfJobs(i)$ is the number of jobs, including the running and queuing ones, in computing site i at the moment.

The *Least Relative Load (LRL)* heuristic assigns the new job to the computing site that has the least relative load. This scheduling heuristic attempts to balance the load among the computing sites in the Data Grid.

7.4.3 Data Present (DP)

The *Data Present (DP)* heuristic is an extension of JobDataPresent in [96], and it takes the data location as the major factor when assigning the job. According to different situations of the data file required by the job, DP works in the following manner:

1. **If** there are computing sites having the data file in their caches, then assign the job to the one with the least relative load among these sites.
2. **Else if** there are non-isolated replica servers having the data file, then assign the

Chapter 7. Combining Data Replication and Job Scheduling in the Data Grid

job to the computing site that is in the same domain as one of these replica servers and has the least relative load.

3. **Else**, all replicas of the data file must be in the isolated replica servers. Assign the job to the computing site with the least relative load in the Data Grid. In this case on selection of computing site, DP is similar to LRL.

7.5 Simulation Method

In order to evaluate the performances of the dynamic replication algorithms and the Grid scheduling heuristics, a Data Grid simulator named *XDRepSim* is developed in accordance with the architecture shown in Figure 7.1. With XDRepSim, users can easily create a simulated Data Grid with the desired parameters, including the system organization, computing site capabilities, replica server capacities, network bandwidth, and job workload. Under the same conditions of the Data Grid, various data replication methods and Grid scheduling heuristics can be chosen and combined for performance evaluation.

In this section, the simulated environment of the Data Grid and job workload are introduced.

7.5.1 Simulated Environment of the Data Grid

The simulated environment of the Data Grid refers to some existing Grid projects, such as EGEE [34], TeraGrid [109] and LCG [70]. There are 25 domains in the simulated Data Grid and each domain has a replica server. A total of 200 computing sites are randomly scattered across twenty domains. The remaining five domains only have replica servers. Let $U(x, y, \Delta)$ be a number sampled from the uniform distribution with a range from x to y , and the sampling granularity is Δ . The computing capability of each computing site is set to $U(1, 20, 1)$.

7.5 Simulation Method

Every computing site has a data cache, which is large enough to contain all data files required by the jobs running or queuing in the site at any time. If a job is scheduled to a computing site whose cache has the job input data, it can access the data from the cache directly. Otherwise, the job would retrieve the data from a replica server. The cached data files are evicted automatically if they are not referred to by any job in the site.

7.5.1.1 Data Files

There are 10,000 data files in the system, and the size of each file is set to $U(500\text{MB}, 5\text{GB}, 100\text{MB})$. Every data file has a primary copy, which is kept in one of the replica servers permanently during the whole simulation session. Before a simulation starts, the primary copies of all data files are randomly distributed among the replica servers.

Zipf-like distribution [121, 15] is used to simulate the file popularity, that is, the number of requests for the n th most popular file is proportional to $n^{-\alpha}$, where α is a constant (refer Section 6.4.2). In this research, $\alpha = 1.0$ is used.

In distributed systems, the data file popularity may change with time. With variation in users' interests, the files that are currently popular may not be popular in the next phase, and vice versa. To simulate the data access pattern with dynamically changing popularity, the method presented in Section 6.4.2 is used. Every simulation session is partitioned evenly into 10 sub-sessions according to the number of submitted jobs, and the data file popularity is changed across sub-sessions. Without any loss of accuracy, we associate the file popularity with its ID. Let the most popular file have ID 0 in the first sub-session. In the second sub-session, the data file popularity is shifted and the file with ID 1000 becomes the most popular one. Likewise, for every subsequent sub-session, the most popular file ID is shifted by an increment of 1000. In each sub-session, the data file popularity follows Zipf-like ($\alpha = 1.0$) distribution (refer Figure 6.8 and Figure 6.9).

Chapter 7. Combining Data Replication and Job Scheduling in the Data Grid

7.5.1.2 Network Model

To model the network bandwidth sharing behaviors, we use the following methods. For any two nodes u and v in the system, let the bandwidth capacity between them be $B_{u,v}$. Each replica server can serve multiple connections concurrently, and the outbound bandwidth limitation of server j is denoted by OB_j . Assume that every node can receive multiple data transfers from different replica servers simultaneously, but only one inbound data connection from the same replica server is allowed at any time. For any data transfer, the bandwidth bottleneck lies either along the link path or in the server side rather than in the client side. The server's outbound bandwidth is proportionally shared among the connections based on the requested bandwidth. Let $V(j)$ be the set of nodes that are being served by replica server j . For any node $u \in V(j)$, the available bandwidth between node u and replica server j is modeled as:

$$BW(u, j) = \min\{B_{u,j}, OB_j \times \frac{B_{u,j}}{\sum_{v \in V(j)} B_{v,j}}\} \quad (7.5.1)$$

The above model can characterize network bandwidth sharing for both LAN and WAN in the Grid. For example, there is a replica server j whose outbound bandwidth limitation is 100Mbps; computing sites a and b are in the same domain as j , where $B_{a,j} = B_{b,j} = 100$ Mbps; and two remote nodes c and d , where $B_{c,j} = 2$ Mbps and $B_{d,j} = 1$ Mbps. Using Equation (7.5.1), we can get the following scenario results:

- If a and b are being served by j , the available bandwidth for both connections is 50Mbps.
- If c and d are being served by j , the available bandwidth is 2 and 1Mbps for nodes c and d respectively.
- If local node a and remote nodes c and d are being served by j , the available bandwidth is 97.09, 1.94 and 0.97Mbps for nodes a , c and d respectively.

7.5 Simulation Method

- If all of the four nodes are being served by j , the available bandwidth is 49.26, 49.26, 0.99 and 0.49Mbps for nodes a , b , c and d respectively.

As each domain is connected by a LAN and different domains are interconnected by a WAN, to simplify the model of the Grid network, we assume that the bandwidth capacity between any computing site in a domain and the domain's replica server should be the same, and all nodes in a domain should have the same bandwidth capacity to a remote replica server. Therefore, we only need to specify the bandwidth capacity between every two domains instead of between every two nodes. Let B_{w_1, w_2} be the assigned bandwidth capacity for domains w_1 and w_2 , where $B_{w_1, w_2} = B_{w_2, w_1}$ symmetrically. If $w_1 = w_2$, the assigned value means the intra-domain bandwidth. For any two different nodes $u \in NS(w_1)$ and $v \in NS(w_2)$, the bandwidth capacity between them is $B_{u, v} = B_{w_1, w_2}$. In our simulated Data Grid, the bandwidth capacity between any two different domains is set to $U(1, 10, 1)$ Mbps, and the intra-domain bandwidth capacity is set to $U(10, 100, 10)$ Mbps. The outbound bandwidth limitation of every replica server is 200Mbps uniformly. The available bandwidth for every connection dynamically changes according to Equation (7.5.1).

7.5.1.3 Replica Server Capacity

The specified replica server capacity only refers to the space for the data replicas, and it does not include the space occupied by the primary copies. The *relative capacity* of all replica servers is defined as $r = S/D \times 100\%$, where S is the sum of storage sizes of all replica servers and D is the sum of sizes of all data files in the Data Grid (refer Section 6.4.3). If $r = 50\%$, it implies that on average half of the data files may have one replica in addition to their primary copies.

In each simulation, only the relative capacity of all replica servers r is given, so that the total storage size S of the servers is known with respect to the total data file size D .

Chapter 7. Combining Data Replication and Job Scheduling in the Data Grid

To specify the storage size for every replica server, simply let all isolated replica servers have the same space as the mean value of storage sizes for all replica servers, and let every non-isolated replica server have the storage size that is proportional to the computing capability of the domain in which the server locates. As there are 25 replica servers in the system and five of them are isolated, the storage size of each isolated replica server is $\frac{S}{25}$. For non-isolated replica server j in domain w , its storage size is $\frac{20S}{25} \times \frac{C_w}{C_W}$.

In our simulated Data Grid, the relative capacity of all replica servers r is set to 50%.

7.5.2 Workload

There are 100,000 jobs in the workload for every simulation. Job arrivals follow Poisson distribution. On average, a new job arrives per 10 second. The span of the workload is about 12 days. The workload format refers to the standard defined in [24].

The actual computational cost of each job is set to $U(30\text{sec}, 10\text{hour}, 1\text{sec})$. The estimated computational cost of the job may be different from the actual value [71]. To simulate the accuracy of the estimated job computational cost, we use the following method that is derived from the model in [76] with modifications to conform to the fact that users often overestimate their jobs' computational costs [71]. For a given job whose actual computational cost is t_a , its estimated computational cost t_e is a sampling from the Gaussian distribution with mean $\mu = t_a$ and standard deviation $\sigma = 3t_a$. If the sampling value is negative, discard the value and sample again. Let $\Delta t = |t_e - t_a|$ denote the *estimation error* of the job computational cost, and the *relative estimation error* is defined as:

$$e = \frac{\Delta t}{t_a} \quad (7.5.2)$$

The *average relative estimation error* ($\bar{e}$) for a workload is the mean value of relative estimation errors for all jobs. Using the above method, $\bar{e}$ of the generated workload is measured as 2.0.

7.6 Results

7.6 Results

In order to demonstrate the advantages of the Data Grid architecture in which every domain has a replica server, the method without any data replication is studied. We refer to this method as *NoR*.

To illustrate the performances of the dynamic replication algorithms, the static replication method is also studied. As the data file popularity changes with time (refer Section 7.5.1.1), it is impossible to deduce an optimal static replication method without knowing the data access pattern in advance. Therefore, the *Random Static Replication* (*RSR*) method is applied. Before the start of each simulation, the data files are replicated at every server randomly till all available storage is used up. This step can also be deemed as the warm-up process to prime the replica servers before collecting statistics. We then evaluate the performances of the replication methods of *RSR*, *CDR_RTPlace*, *CDR_SMPlace* and *DDR* under the same environment. For *RSR*, all replicas created in the initialization step will not be changed and no new replicas will be created during the whole simulation session. On the contrary, the dynamic replication algorithms of *CDR_RTPlace*, *CDR_SMPlace* and *DDR* will alter the replication status with different strategies after the initial random replication.

With five replication methods (*NoR*, *RSR*, *CDR_RTPlace*, *CDR_SMPlace* and *DDR*) and three scheduling heuristics (*STT*, *LRL* and *DP*), there is a total of fifteen combinations to evaluate. Each combination is referred to as a *policy*. With the simulation configuration stated in Section 7.5, Figure 7.2 shows the geometric mean of turnaround time (*GMTT*) of the different policies.

For the same scheduling heuristic, the performance of *NoR* is always the worst and its *GMTT* is evidently larger than any static and dynamic replication algorithm, which proves that data replication can shorten the job turnaround time effectively. The per-

Chapter 7. Combining Data Replication and Job Scheduling in the Data Grid

Figure 7.2: Policy performances under the base simulation configuration.

formance of RSR is better than NoR but worse than all dynamic replication algorithms. This is because RSR does not adapt to the data file access pattern. Dynamic replication algorithm CDR_RTPlace is the best among all replication algorithms, and CDR_SMPlace is slightly better than DDR.

Generally, DP scheduling heuristic works better than STT and LRL in the simulated environment. The performance differences between the static and dynamic replication algorithms are not distinct for DP scheduling heuristic.

As a whole, the STT+CDR_RTPlace and DP+CDR_RTPlace policies have the shortest GMTT.

7.6.1 Impact of Data Grid Environment

In this sub-section, we study the impacts of network bandwidth capacity and replica server capacity on the policy performances. The other parameters of the Data Grid environment and the job workload are kept the same as the base simulation configuration

7.6 Results

stated in Section 7.5.

7.6.1.1 Network Bandwidth Capacity

The network bandwidth capacity is increased to five times the base configuration, namely the LAN and WAN bandwidth capacities are set to $U(5, 50, 5)$ Mbps and $U(50, 500, 50)$ Mbps respectively, and the outbound limitation of each replica server is 1Gbps. The simulation results are shown in Figure 7.3.

Figure 7.3: Policy performances when the network capacity is scaled up.

When the network capacity is increased, the performances of all policies are improved significantly compared to Figure 7.2. STT scheduling heuristic performs the best, and DP is the worst. As the data transfer time is decreased for all policies, the job execution time becomes the major factor affecting the turnaround time. Consequently the scheduling heuristic that tries to minimize the job's queuing and execution time (LRL) works better than the heuristic that attempts to minimize data transfer time (DP).

All policies with STT scheduling heuristic produce similar results. Although the data replication methods improve the performance in terms of job turnaround time,

Chapter 7. Combining Data Replication and Job Scheduling in the Data Grid

the improvements are not significant. The reason is that when the network capacity is increased, the impact of data transfer cost on the job turnaround time is diminished, resulting in marginal performance improvement by replication. Besides, STT takes into consideration both the computational cost and the data transfer cost for the job, so it is adaptable to the inferior situations of data distribution.

We also have studied the situation when each data file size is scaled down to one-fifth of the original size, that is, the file size is set to $U(100\text{MB}, 1\text{GB}, 100\text{MB})$. The simulation results are consistent with Figure 7.3.

7.6.1.2 Replica Server Capacity

To study the impact of the replica server storage size on the job turnaround time, the relative capacity of all replica servers r is varied from 10% to 100% in ten simulation cases. The performance results of STT scheduling heuristic combined with each data replication method are shown in Figure 7.4.

Figure 7.4: Geometric mean of turnaround time vs. replica server capacity for STT.

As NoR does not create replicas, its performance remains constant irrespective of the

7.6 Results

storage sizes of the replica servers. The difference between the horizontal line of NoR and the replication algorithms represents the benefit gain of data replication in reducing the job turnaround time. For any storage capacity, the dynamic replication algorithms work better than RSR. CDR_RTPlace gives the best performance and CDR_SMPlace is better than DDR.

By increasing the replica server capacity, the performances of RSR and all dynamic replication algorithms are improved by different degrees. When the total storage size of all replica servers is the same as the total size of all data files ($r = 100\%$), the GMTT of RSR is only forty percent of NoR, and the GMTT of any dynamic replication algorithm is less than thirty percent of NoR.

Similar results are observed for LRL scheduling heuristic in Figure 7.5, where all replication methods can improve the performance. In particular, LRL performs best with the CDR_RTPlace replication algorithm.

Figure 7.5: Geometric mean of turnaround time vs. replica server capacity for LRL.

The results for DP scheduling heuristic are shown in Figure 7.6. Unlike STT and LRL,

Chapter 7. Combining Data Replication and Job Scheduling in the Data Grid

DP scheduling heuristic does not show any performance improvement when applying the dynamic data replication algorithms compared to using the static replication method. The similar property can also be observed in Figures 7.2 and 7.3. The reason is that before the dynamic replication algorithms are invoked, the popular data files have already been loaded into the caches of certain computing sites to which the jobs had been submitted. DP chooses the job's running site by considering whether the input data file is in the caches of the computing sites first. From the simulation logs, it is found that around half of the jobs in the workload access data files from the computing sites' caches directly for all policies with DP scheduling heuristic. Therefore, the data replicas created by the dynamic replication algorithms do not impact DP scheduling decisions evidently when the data caches for the computing sites are large as we set.

Figure 7.6: Geometric mean of turnaround time vs. replica server capacity for DP.

7.6.2 Impact of Job Workload

The impacts of the workload parameters, including the relative estimation error of job computational cost, job arrival rate and job computational cost, on the performances of

7.6 Results

the policies are studied in this sub-section.

7.6.2.1 Relative Estimation Error of Job Computational Cost

Among the three Grid scheduling heuristics, only STT takes the job computational cost into consideration. The inaccurate estimate of job computational cost will lead to improper scheduling decision for STT. Consequently the effectiveness of STT will be diminished. To comprehensively study the properties of the policies, we evaluate them with the job workload that is the same as the base configuration except that the estimates of job computational costs are accurate, namely $\bar{e} = 0$.

Figure 7.7 shows the performance results when the estimates of job computational costs are accurate. It can be noted that the performance of STT with any replication method shows distinct improvement compared to Figure 7.2. As the local scheduler can move a job ahead of others in the queue and start execution as long as the job does not delay the first queued job (refer Section 7.4), the GMTT values of LRL and DP are also reduced when the estimates of job computational costs are accurate. However, the improvements are marginal. Overall, the performance of STT scheduling heuristic is far better than LRL and DP, and particularly STT+CDR_RTPlace has the best performance.

By comparing Figure 7.7 with Figures 7.2, it can be noticed that the performance of STT scheduling heuristic is sensitive to the estimation error of job computational cost. As expected, the influence of the estimation error on the scheduling heuristics of LRL and DP is negligible. Thus, with the same workload, STT is evaluated with the average relative estimation error $\bar{e}$ ranging from 0 to 2.0, and the results are show in Figure 7.8.

When $0 < \bar{e} < 1.0$, the performances of all studied policies with STT scheduling heuristic are very close, and their GMTT values rise gently with increasing $\bar{e}$. When $\bar{e} > 1.0$, the GMTT of every policy increases rapidly, especially the policies without dynamic data replication. The plot also demonstrates that the policy of STT+CDR_RTPlace can

Chapter 7. Combining Data Replication and Job Scheduling in the Data Grid

Figure 7.7: Policy performances when $\bar{e} = 0$.

tolerate a large estimation error and still yield acceptable performance.

7.6.2.2 Job Arrival Rate

We decrease the job arrival rate and set the average arrival interval to 12 seconds. The simulation results are shown in Figure 7.9.

As the job arrival rate is decreased, the load of the system is diminished. Therefore, the turnaround time of any policy is shortened compared to Figure 7.2. In particular, the policies with STT scheduling heuristic outperform the other policies. Although the estimate of job computational cost is very inaccurate ($\bar{e} = 2.0$), the number of jobs in the local queues is not large due to the low job arrival rate, so the estimation error is not accumulated. Thus, STT can decide scheduling properly and achieve superior performance.

7.6 Results

Figure 7.8: Geometric mean of turnaround time vs. average relative estimation error for STT.

Figure 7.9: Policy performances when the job arrival rate is decreased.

7.6.2.3 Job Computational Cost

We set the job computational cost to $U(15\text{sec}, 5\text{hour}, 1\text{sec})$, which is half of the base configuration. The simulation results are shown in Figure 7.10.

Chapter 7. Combining Data Replication and Job Scheduling in the Data Grid

Figure 7.10: Policy performances when the job computational cost is halved.

By halving the job computational cost, the turnaround times of all policies are significantly reduced in comparison with Figure 7.2. Specifically, the turnaround time of any policy with STT scheduling heuristic is only about 10% of the original value shown in Figure 7.2, and the percentages are about 40% and 30% for LRL and DP respectively. For all policies, the performance improvements result from shortened job execution time and job queuing time. Moreover, as the job computational cost (t_a) is reduced and the relative estimation error (e) keeps the same, the estimation error ($\Delta t = t_a \cdot e$) of the job computational cost is decreased consequently (see Equation (7.5.2)). Hence STT can schedule more effectively and achieve better performance than LRL and DP.

7.6.3 Discussion

From the simulation results we know that for the same Grid scheduling heuristic, the data replication methods of RSR, CDR_RTPlace, CDR_SMPlace and DDR can reduce the job turnaround time significantly compared to the method without any data replication (NoR). Under the same scheduling heuristic of either STT or LRL, the dynamic

7.6 Results

replication algorithms perform better than the static replication algorithm because the dynamic replication algorithms can detect the changes in file popularity and update the data replication status dynamically. As the cached data in the computing sites dominate the decisions of DP scheduling heuristic, the advantages of the dynamic replication algorithms are not apparent for DP. Thus, we only focus on the policies that are combined with STT or LRL scheduling heuristic when we compare the performances of the data replication methods.

In general, the centralized data replication algorithms (CDR_RTPlace and CDR_SMPlace) outperform the distributed data replication algorithm (DDR) for the same scheduling heuristic and same system configuration. The reason is that the centralized replication can make replication decision based on the global view of the system, so the redundant replications are avoided and the storage resources are utilized efficiently. On the contrary, by applying DDR, every replica server tries to replicate the most popular data files to local storage, hence the contents of all replica servers are similar. Consequently, the popular data files are replicated at too many servers, while many other data files do not have chances to be replicated due to the storage limitation. DDR does not maximize the use of the replica server capacity efficiently. As CDR_RTPlace takes the existing replicas into consideration when placing new replicas, its performance is better than CDR_SMPlace.

The Data Grid environment and workload properties must be considered when choosing the scheduling heuristics and the data replication methods. STT is impacted by the accumulated estimation error of job computational cost, but DP and LRL are not impacted by it. The accumulated estimation error is determined by the factors of the relative estimation error, the job computational cost, and the job arrival rate all together. Under the extreme condition when the network bandwidth is low, the estimate of job execution time is very inaccurate, the job arrival rate is high, and the job computational cost is large, the policy of DP scheduling heuristic combined with any data replication

Chapter 7. Combining Data Replication and Job Scheduling in the Data Grid

method and policy STT+CDR_RTPlace outperform the others. However, in other situations where the accumulated estimation error of job computational cost is not large, the performance of STT scheduling heuristic combined with any data replication algorithm is far better than other policies.

Overall, the policy of STT+CDR_RTPlace is a sound choice as it works well under various situations of the system environment and job workload.

7.7 Summary

In this chapter, a Data Grid architecture is introduced to support efficient data access for the job. Two centralized dynamic replication algorithms, CDR_RTPlace and CDR_SMPlace, and a distributed algorithm, DDR, are put forward. At intervals, the dynamic replication algorithms exploit the data access history for the popular data files and determine the replications to improve data access performance from the perspective of the Grid job. The Grid scheduling heuristics of STT, LRL and DP, are proposed, and they are combined with different data replication methods.

The simulator XDRepSim is developed to study the performances of the dynamic replication algorithms and Grid scheduling heuristics. A simulated Data Grid is built with XDRepSim and diverse simulations are carried out by varying the settings of the Data Grid environment and job workload. The results demonstrate that the proposed dynamic replication algorithms can shorten the job turnaround time. In particular, the policy of STT+CDR_RTPlace exhibits remarkable performance relative to the other evaluated policies under various conditions of the system environment and job workload.

Chapter 8

Conclusion

8.1 Summary

Enabled by rapid improvements in the capability and capacity of sensors, storage systems, computers, and networks, large volumes of data are coming forth in many disciplines such as high energy physics, biology and meteorology [43]. The large-scale data need to be analyzed in pursuit of profound understanding for the problems. Multiple communities would be involved and collaborate on the data-intensive computing. The Data Grid is an infrastructure that coordinates the data and computational resources across widely distributed communities to support the data-intensive computing. As enormous data transfers are induced by the computing, optimization for remote data access is crucial for the Data Grid. Therefore, this thesis focuses on the techniques of data replication and management to improve data access performance for the Data Grid.

We have investigated the performances of the replica placement algorithms that are derived from the minimum k -median graph theory. We propose the set-cover replica placement algorithm that attempts to place the replicas in the centers of the clustered clients. The set-cover, greedy, vertex degree based, and random replica placement algorithms are evaluated with diverse simulation parameters, and their performances in terms

Chapter 8. Conclusion

of total access cost and load balance are analyzed. Among these algorithms, the greedy algorithm shows the best performance in total access cost, and the set-cover algorithm is a close second. However on the Transit-Stub topology, the set-cover algorithm has the best load balance among replicas. The vertex degree based algorithm performs well on the Internet AS-level and degree power-law topologies, but poorly on the Transit-Stub topology.

By analyzing the simulation results of the replica placement algorithms, we find that the number of replicas and the total access cost follow the logarithmic relationship with high correlation coefficients. This logarithmic relationship can be used to predict the access performance with the specified number of replicas effectively. As an example application, the logarithmic characteristic is applied to the *minimum facility location* problem and a function is deduced to get the optimal number of replicas.

In order to harness the tens of Terabytes of public storage resources driven by the Internet Backplane Protocol (IBP), we develop a data management system that facilitates data sharing for IBP. The system uses the Globus Replica Catalog (GRC) to keep the data location information. Different from the normal data access protocols such as ftp, gsiftp and http, a single URL is not enough to access the data in IBP. To manage a data file in IBP, the application must have the exNode file associated with the data. Thus, the ExNode File Warehouse (EFW) is developed to store and manage the exNodes of the data files in IBP. Interacting with IBP, GRC and EFW, the IBP Data Management (IDM) encapsulates the complex operations in the underlying infrastructure and provides a transparent and easy-to-use interface to applications for managing the data in IBP.

The multi-tier Data Grid architecture proposed by MONARC project [79] aims at supporting the global distributed computing for the high energy physics experiments. In the multi-tier model, the raw data generated by the experiments are stored at the top tier. The data analysis will be carried out by the regional centers, national centers, in-

8.1 Summary

stitutional centers, and end-user workstations, which are located at different lower tiers. According to the properties of the multi-tier Data Grid, we propose two dynamic replication algorithms, Simple Bottom-Up (SBU) and Aggregate Bottom-Up (ABU). Both algorithms work in the manner of finding the popular data files by analyzing the data access history, and then determining where to place the replicas for the popular data files. The simulation results show that the SBU and ABU replication algorithms can reduce the average response time and bandwidth cost of data access effectively. The performances of SBU and ABU are compared with the Fast Spread replication strategy. In all studied situations of diverse data access patterns and replica server storage capacities, ABU exhibits outstanding performances in terms of both average response time and average bandwidth cost, and its performances are better than SBU and Fast Spread in most cases. Although Fast Spread's performances in average response time and average bandwidth cost are better than SBU generally, considering its high replication frequency, Fast Spread is not a practical replication strategy because it induces heavy I/O and CPU load on the replica servers. On the contrary, the low replication frequencies of SBU and ABU indicate that both algorithms are feasible and cost-effective.

Data replication is not an isolated problem in the Data Grid. In particular, Grid job scheduling is close correlated with data replication. For above concerns, we present a Data Grid architecture that supports efficient data access for the Grid job. Two centralized dynamic replication algorithms (CDR_RTPlace and CDR_SMPlace) and a distributed algorithm (DDR) are put forward. The replication algorithms are combined with different Grid scheduling heuristics, including Shortest Turnaround Time (STT), Least Relative Load (LRL) and Data Present (DP). Diverse simulations are carried out by varying the settings of the Data Grid environment and job workload, and the results demonstrate that the proposed dynamic replication algorithms can shorten the job turnaround time. In particular, the policy of STT+CDR_RTPlace displays remarkable performance under

various conditions of system environment and workload.

8.2 Future Work

We have studied the replica placement algorithms for the purpose of enhancing the global data access performance. The load balance among replicas is also measured as the secondary criterion for the replica placement algorithms, and it can be improved by applying the request redirection mechanism that takes into consideration the load of the replicas. We will study the request redirection model that makes proper tradeoffs between selecting the closest replica to redirect a request and attempting to balance the replica load. Furthermore, the problem of replica placement can also be considered for the objectives of maximizing data availability and maximizing network bandwidth utilization. To this end, the replicas should be placed across network domains to ensure that the data remains available even if an entire domain is out of service, for example, due to a failure of a shared resource like a network gateway or power circuit. At the same time, the aggregate bandwidth of multiple domains can be exploited to improve the data transfer speed if parallel streams from different replicas are supported.

Although a prototype system of data management for the IBP-based Data Grid has been built, there is still a long way to go to implement the entire system. As the Replica Location Service (RLS) provides higher reliability and scalability than the Globus Replica Catalog (GRC), we will use RLS to replace GRC in the data management system. Replica selection service, global naming service, security, user management and data access control will be included.

For dynamic data replication, the adaptation triggering function is responsible for identifying changes in the system and for adapting the replication configurations to the system changes [102]. Aiming to perform global optimizations, the dynamic replication

8.2 Future Work

mechanisms studied in this thesis use the periodic adaptation triggers, which invoke the data replication algorithm to analyze the data access history and determine data replication at intervals. The periodic adaptation trigger is effective in optimizing the system performance in a long-term perspective, but it cannot respond to the burst event instantly. The event-based adaptation trigger monitors the system status continuously and adapts the replication configurations once emergency events, such as the burst of data requests and the congestion of network traffic, are detected. In order to adapt replication to emergency situations as well as to slow changes of the system, the event-based adaptation triggering function will be used in addition to the periodic adaptation triggering function.

As the Data Grid is still in the initial phase, the traces of data access are not available yet. Thus, the synthetic data access requests are used in the simulations. In future work we will investigate the real data access patterns and job workloads of the scientific and engineering applications, and they will be used to evaluate the performances of our dynamic data replication algorithms and job scheduling heuristics.

In the Supercomputing 2004 (SC2004) Bandwidth Challenge [99], the team of *High Speed TeraByte Transfers for Physics*, which was led by Caltech and included some other research partners, achieved a maximum bandwidth of 101Gbps between Pittsburgh and Los Angeles and won the challenge. This success demonstrates that multiple 10Gbps links can be incorporated and used as the foundation of the global Grid system to support Terabyte-scale data transfers. It also makes it possible to construct *hybrid* networks that can integrate traditional packet routing and switching with dynamically constructed optical paths to support extremely large data flows [18]. In accordance with the vision of a world-wide Data Grid that can support Terabyte-scale data transfers for data-intensive computing, we will study more sophisticated Grid scheduling strategies, which need to schedule both computing resources and data transfers by constructing optical

Chapter 8. Conclusion

paths dynamically. As data transfer paths of hybrid networks are controllable, new data replication algorithms will be developed for such environment.

So far, very little research on replica consistency management has been carried out for data files in the Data Grid since most data is read-only. Nevertheless, the writable data still exists and replica consistency management is unavoidable. The challenges of replica consistency management for the Data Grid come from the factors that a large number of machines are involved, data files are large in size and quantity, file systems are heterogeneous, and the whole system changes dynamically. As the consistency requirements for various data could be different, associating each writable data with the most suitable consistency scheme could be a solution. For example, if the data requires strict consistency, the eager scheme [50], which keeps all replicas synchronized by updating all the replicas as part of one atomic transaction, can be used. On the contrary, if the data allows weak consistency, the lazy scheme, which asynchronously propagates replica updates to other servers after the writing, can be associated with it. To support the consistency management, the replica metadata catalog should include the consistency information for the replica, such as the associated consistency scheme, read/write lock, version and time stamps. Furthermore, our replication algorithms will be improved to take into account the cost of consistency management for the replicas. For the data that requires strong consistency and is updated frequently, the number of its replicas should be controlled in a small range, and the replication algorithms would tend to migrate its existing replicas to other locations for better performance rather than to create new replicas.

Acronyms

ABU	Aggregate Bottom-Up
AS	Autonomous System
CDN	Content Distribution Network
CDR	Centralized Dynamic Replication
CERN	European Organization for Nuclear Research
CMS	Compact Muon Solenoid
DDR	Distributed Dynamic Replication
DP	Data Present
DRS	Dynamic Replication Scheduler
EFW	ExNode File Warehouse
FCFS	First-Come-First-Serve
FID	File ID
GMTT	Geometric Mean of Turnaround Time
GRC	Globus Replica Catalog
GSI	Grid Security Infrastructure
HEP	High Energy Physics
IBP	Internet Backplane Protocol
IDM	IBP Data Management
LAN	Local-Area Network
L-Bone	Logistical Backbone
LDAP	Lightweight Directory Access Protocol
LFN	Logical File Name
LFU	Least-Frequently Used
LHC	Large Hadron Collider
LoRS	Logistical Runtime System
LRC	Location Replica Catalog
LRL	Least Relative Load
LRM	Local Replica Manager
LRU	Least-Recently Used
MDS	Monitoring and Discovery Service (Globus)
MCAT	Metadata Catalog

ACRONYMS

NoR	No Replication
NOA	Number of Accesses
NWS	Network Weather Service
PFN	Physical File Name
PPDG	Partical Physics Data Grid
PRS	Primary Replica Server
RC	Replica Catalog
RLI	Replica Location Index
RLS	Replica Location Service
RS	Replica Selector
RSR	Random Static Replication
RTPlace	Response-Time Oriented Replica Placement
SBU	Simple Bottom-Up
SMPlace	Server Merit Oriented Replica Placement
SRB	Storage Resource Broker
SRS	Secondary Replica Server
STT	Shortest Turnaround Time
WAN	Wide-Area Network

Author's Publications

1. Bu-Sung Lee and Ming Tang, "The Performance of Replica Placement Algorithms on the Network," in *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'03)*, pp. 831–837, June 2003, Las Vegas, USA.
2. Ming Tang and Bu-Sung Lee, "Logarithmic Characteristic of Replica Placement Algorithms," *IEICE Transactions on Communications*, vol. E86-B, no. 7, pp. 2227–2231, July 2003.
3. Ming Tang, Bu-Sung Lee, Chai-Kiat Yeo, Micah Beck, James S. Plank, and Scott Atchley, "Data Management for the IBP-based Data Grid," in *Proceedings of International Conference on Scientific and Engineering Computation (IC-SEC)*, June 2004, Singapore.
4. Ming Tang, Bu-Sung Lee, Chai-Kiat Yeo, and Xueyan Tang, "Dynamic Replication Algorithms for the Multi-tier Data Grid," *Journal of Future Generation Computer Systems (FGCS)*, vol. 21, no. 5, pp. 775–790, 2005.
5. Ming Tang, Bu-Sung Lee, Xueyan Tang, and Chai-Kiat Yeo, "Combining Data Replication Algorithms and Job Scheduling Heuristics in the Data Grid," in *Proceedings of Euro-Par 2005*, pp. 381–390, August 2005, Portugal.

AUTHOR'S PUBLICATIONS

6. Ming Tang, Bu-Sung Lee, Xueyan Tang, and Chai-Kiat Yeo, "The Impact of Data Replication on Job Scheduling Performance in the Data Grid," *Journal of Future Generation Computer Systems (FGCS)*, vol. 22, no. 3, pp. 254–268, 2006.

Bibliography

- [1] The Historical Charges for AARNet Services.
<http://www.aarnet.edu.au/services/historicalcharges.html>.
- [2] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox. Caching proxies: Limitations and potentials. In *Fourth International World Wide Web Conference*, Boston, MA, Dec 1995.
- [3] C. Aggarwal, J. L. Wolf, and P. S. Yu. Caching on the world wide web. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):94–107, 1999.
- [4] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, and S. Tuecke. Data management and transfer in high performance computational Grid environments. *Parallel Computing Journal*, 28(5):749–771, May 2002.
- [5] B. Allcock, S. Tuecke, I. Foster, A. Chervenak, and C. Kesselman. Protocols and services for distributed data-intensive science. In *ACAT2000 Proceedings*, pages 161–163, 2000.
- [6] S. Atchley, M. Beck, J. Millar, T. Moore, J. S. Plank, and S. Soltesz. The logistical networking testbed. Technical Report UT-CS-02-496, University of Tennessee, Dec 2002.

BIBLIOGRAPHY

- [7] P. Avery. Data grids: a new computational infrastructure for data-intensive science. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical Physical and Engineering Sciences*, 360(1795):1191–1209, 2002.
- [8] P. Avery. Grid computing in high energy physics. In *Proceedings of the International Beauty 2003 Conference*, Pittsburgh, PA, Oct 2003.
- [9] A. Balamash and M. Krunz. An overview of web caching replacement algorithms. *IEEE Communications Surveys and Tutorials*, 6(2):44–56, 2004.
- [10] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Resource Broker. In *Proceedings of CASCON'98 Conference*, Toronto, Canada, Nov 1998.
- [11] A. Bassi, M. Beck, T. Moore, and J. S. Plank. The Logistical Backbone: Scalable infrastructure for global Data Grids. In *Asian Computing Science Conference 2002*, Hanoi, Vietnam, Dec 2002. December, 2002. Springer Verlag.
- [12] M. Beck, T. Moore, J. S. Plank, and M. Swamy. Logistical networking: Sharing more than the wires. In C. A. Lee S. Hariri and C. S. Raghavendra, editors, *Active Middleware Services*. Kluwer Academic, Norwell, MA, 2000.
- [13] W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar, K. Stockinger, and F. Zini. Optorsim - a Grid simulator for studying dynamic data replication strategies. *International Journal of High Performance Computing Applications*, 17(4):403–416, 2003.
- [14] W. H. Bell, D. G. Cameron, R. Carvajal-Schiaffino, A. P. Millar, K. Stockinger, and F. Zini. Evaluation of an economy-based file replication strategy for a Data Grid. In *International Workshop on Agent based Cluster and Grid Computing at CCGrid 2003*, Tokyo, Japan, May 2003.

BIBLIOGRAPHY

- [15] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: evidence and implications. In *Proceedings of IEEE INFOCOM*, pages 126–134, New York, Mar 1999.
- [16] T. Bu and D. Towsley. On distinguishing between Internet power law topology generators. In *Proceedings of IEEE INFOCOM*, 2002.
- [17] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch. A national-scale authentication infrastructure. *IEEE Computer*, 33(12):60–66, 2000.
- [18] Internet Speed Quadrupled by International Team During 2004 Bandwidth Challenge. http://pr.caltech.edu/media/Press_Releases/PR12620.html, Nov 2004.
- [19] K. L. Calvert, M. B. Doar, and E. W. Zegura. Modeling Internet topology. *IEEE Communications Magazine*, Jun 1997.
- [20] 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*, 2(1):57–69, 2004.
- [21] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. In *USENIX Symposium on Internet Technologies and Systems*, pages 193–206, Monterey, CA, 1997.
- [22] P. Capiluppi. CMS world wide computing. reported at the LHCC Comprehensive Review of CMS Software and Computing, Oct 2000. http://cmsdoc.cern.ch/cms/software/reviews/LHCC_review_oct_00/.
- [23] M. Carman, F. Zini, L. Serafini, and K. Stockinger. Towards an economy-based optimisation of file access and replication on a Data Grid. In *the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)*, Berlin, Germany, May 2002.

BIBLIOGRAPHY

- [24] 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.
- [25] L. Cherkasova and J. Lee. FastReplica: Efficient large file distribution within content delivery network. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.
- [26] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunst, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, and B. Tierney. Giggle: a framework for constructing scalable replica location services. In *Proceedings of the IEEE/ACM Supercomputing 2002 (SC2002)*, pages 1–17, Baltimore, Maryland, Nov 2002.
- [27] 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*, 23(3):187–200, 2000.
- [28] A. L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf. Performance and scalability of a replica location service. In *Proceedings of the International IEEE Symposium on High Performance Distributed Computing (HPDC-13)*, Jun 2004.
- [29] 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*, 14(2):181–192, 2003.
- [30] The Condor Project. <http://www.cs.wisc.edu/condor/>.

BIBLIOGRAPHY

- [31] The Directed Acyclic Graph Manager (DAGMan).
<http://www.cs.wisc.edu/condor/dagman/>.
- [32] F. Desprez and A. Vernois. Simultaneous scheduling of replication and date-intensive computation for applications on the grid. Technical Report RR-5460, INRIA, Jan 2005.
- [33] D. Duellmann, W. Hoschek, J. Jean-Martinez, A. Samar, B. Segal, H. Stockinger, and K. Stockinger. Models for replica synchronisation and consistency in a Data Grid. In *10th IEEE Symposium on High Performance and Distributed Computing (HPDC-10)*, San Francisco, CA, Aug 2001.
- [34] The Enabling Grids for E-science (EGEE) Project. <http://public.eu-egee.org>.
- [35] UK e-Science Project. <http://www.nesc.ac.uk>.
- [36] The European DataGrid Project. <http://eu-datagrid.web.cern.ch/eu-datagrid/>.
- [37] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the Internet topology. In *Proceedings of ACM SIGCOMM*, pages 251–262, 1999.
- [38] 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*, pages 1–24, 1998.
- [39] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong. Theory and practice in parallel job scheduling. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 1–34. Springer Verlag, 1997.
- [40] I. Foster. The Grid: A new infrastructure for 21st century science. *Physics Today*, 55(2):42–47, 2002.

BIBLIOGRAPHY

- [41] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.
- [42] I. Foster and C. Kesselman. GriPhyN/PPDG Data Grid architecture, toolkit, and roadmap. Technical Report GriPhyN-2001-15, 2001.
- [43] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, second edition, 2004.
- [44] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal High Performance Computing Applications*, 15(3):200–222, 2001.
- [45] The Grid Datafarm Project. <http://datafarm.apgrid.org/>.
- [46] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, Lake George, NY, Oct 2003.
- [47] D. K. Gifford. Weighted voting for replicated data. In *Proceedings of 7th ACM Symposium on Operating System Principles*, pages 150–162, 1979.
- [48] GNUTELLA.WEGO.COM. Gnutella: Distributed Information Sharing. <http://gnutella.wego.com/>, 2000.
- [49] Grid Physics Network (GriPhyN). <http://www.griphyn.org>.
- [50] J. Gray, P. Helland, P. O’Neil, and D. Shasha. The dangers of replication and a solution. In *Proceedings of SIGMOD*, pages 173–182, 1996.
- [51] J. Gray, D. T. Liu, M. A. Nieto-Santisteban, A. S. Szalay, G. Heber, and D. DeWitt. Scientific data management in the coming decade. Technical Report MSR-TR-2005-10, Microsoft Research, 2005.

BIBLIOGRAPHY

- [52] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of job-scheduling strategies for Grid computing. In *Proceedings of the First International Workshop on Grid Computing*, pages 191–202, 2000.
- [53] A. Helal, A. Heddaya, and B. Bhargava. *Replication Techniques in Distributed Systems*. Kluwer Academic Publishers, 1996.
- [54] K. Holtman. CMS Data Grid system overview and requirements. CMS Experiment Note 2001/037, CERN, Switzerland, 2001.
- [55] K. Holtman. HEPGRID2001: A model of a virtual Data Grid application. In *Proceedings of HPCN Europe*, volume 2110, pages 711–720, Amsterdam, 2001. Springer LNCS.
- [56] 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*, pages 77–90, 2000.
- [57] A. Iamnitchi, M. Ripeanu, and I. Foster. Locating data in (small-world?) peer-to-peer scientific collaborations. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 232–241, 2002.
- [58] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. On the placement of Internet instrumentation. In *Proceedings of IEEE INFOCOM*, pages 295–304, 2000.
- [59] S. Jamin, C. Jin, A. R. Kurc, D. Raz, and Y. Shavitt. Constrained mirror placement on the Internet. In *Proceedings of IEEE INFOCOM*, Mar 2001.
- [60] Y. Jiang, M.-Y. Wu, and W. Shu. Web prefetching: Costs, benefits and performance. In *The 7th International Workshop on Web Content Caching and Distribution (WCW2002)*, Boulder, Colorado, Aug 2002.

BIBLIOGRAPHY

- [61] W. E. Johnston. The computing and Data Grid approach: infrastructure for distributed science applications. *Computing and Informatics*, 21(4), 2002.
- [62] J. Joseph and C. Fellenstein. *Introduction to Grid Computing*. Prentice Hall PTR, 2004.
- [63] K. Kalpakis, K. Dasgupta, and O. Wolfson. Optimal placement of replicas in trees with read, write, and storage costs. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):628–637, Jun 2001.
- [64] J. Kangasharju, J. W. Roberts, and K. W. Ross. Object replication strategies in content distribution networks. *Computer Communications*, 25(4):367–383, March 2002.
- [65] M. R. Korupolu and M. Dahlin. Coordinated placement and replacement for large-scale distributed caches. *IEEE Transactions on Knowledge and Data Engineering*, 14(6):1317–1329, 2002.
- [66] T. Kosar and M. Livny. Stork: Making data placement a first class citizen in the grid. In *Proceedings of 24th IEEE Int. Conference on Distributed Computing Systems (ICDCS2004)*, Tokyo, Japan, 2004.
- [67] P. Krishnan, D. Raz, and Y. Shavitt. The cache location problem. *ACM/IEEE Transactions on Networking*, 8(5), Oct 2000.
- [68] H. Lamahamedi, Z. Shentu, and B. Szymanski. Simulation of dynamic data replication strategies in Data Grids. In *Proceedings of 12th Heterogeneous Computing Workshop (HCW'03)*, Apr 2001.
- [69] The Logistical Backbone (L-Bone). <http://loci.cs.utk.edu/lbone/>.
- [70] LHC Computing Grid (LCG). <http://lcg.web.cern.ch/lcg/>.

BIBLIOGRAPHY

- [71] C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely. Are user runtime estimates inherently inaccurate? In *Proceedings of 10th Job Scheduling Strategies for Parallel Processing*, Jun 2004.
- [72] LHC - The Large Hadron Collider. <http://lhc-new-homepage.web.cern.ch>.
- [73] B. Li, M. J. Golin, G. F. Ialio, and X. Deng. On the optimal placement of web proxies in the Internet. In *Proceedings of INFOCOM'99*, Mar 1999.
- [74] J.-H. Lin and J. S. Vitter. ε -approximations with minimum packing constraint violation. In *Proceedings 24th ACM Symp. on Theory of Computing*, pages 771–782, 1992.
- [75] Logistical computing and internetworking lab, computer science department of the university of tennessee. <http://loci.cs.utk.edu>.
- [76] 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*, 59(2):107–131, Nov 1999.
- [77] Globus Monitoring and Discovery Service (MDS). <http://www.globus.org/mds/>.
- [78] H. H. Mohamed and D. H. J. Epema. An evaluation of the close-to-files processor and data co-allocation policy in multiclusters. In *IEEE International Conference on Cluster Computing 2004*, San Diego, California, Sep 2004.
- [79] The MONARC project. <http://monarc.web.cern.ch/MONARC/>.
- [80] R. W. Moore, A. Rajasekar, and M. Wan. Data grids, digital libraries, and persistent archives: an integrated approach to sharing, publishing, and archiving data. *Proceedings of the IEEE*, 93(3):578–588, 2005.

BIBLIOGRAPHY

- [81] A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):529–543, Jun 2001.
- [82] NLANR Measurement and Network Analysis Group. <http://moat.nlanr.net/AS>.
- [83] Network Weather Service (NWS). <http://nws.cs.ucsb.edu/>.
- [84] G. On, J. Schmitt, and R. Steinmetz. Quality of availability: Replica placement for widely distributed systems. In *IWQoS 2003*, pages 325–342, Montrey, CA, USA, June 2003.
- [85] G. Pierre and M. van Steen. Globule: a platform for self-replicating Web documents. In *Proceedings of the 6th International Conference on Protocols for Multimedia Systems*, pages 1–11, October 2001.
- [86] J. S. Plank, A. Bassi, M. Beck, T. Moore, D. M. Swany, and R. Wolski. Managing data storage in the network. *IEEE Internet Computing*, 5(5):50–58, Sep/Oct 2001.
- [87] Particle Physics Data Grid (PPDG). <http://www.ppdg.net/>.
- [88] L. Qiu, V. N. Padmanabhan, and G. M. Voelker. On the placement of web server replicas. In *Proceedings of IEEE INFOCOM*, Mar 2001.
- [89] M. Rabinovich. Issues in web content replication. *IEEE Data Engineering Bulletin*, 21(4), Dec 1998.
- [90] M. Rabinovich and A. Aggarwal. Radar: a scalable architecture for a global web hosting service. In *Proceeding of the eighth international conference on World Wide Web*, pages 1545–1561, Toronto, Canada, 1999.
- [91] M. Rabinovich, I. Rabinovich, and R. Rajaraman. Dynamic replication on the Internet. Technical Report HA6177000-980305-01-TM, AT&T Labs, Mar 1998.

BIBLIOGRAPHY

- [92] P. Radoslavov, R. Govindan, and D. Estrin. Topology-informed Internet replica placement. *Computer Communications*, 25:384–392, 2002.
- [93] A. Rajasekar, M. Wan, R. Moore, G. Kremenek, and T. Guptill. Data Grids, collections and grid bricks. In *20th IEEE/ 11th NASA Goddard Conference on Mass Storage Systems & Technologies (MSST2003)*, San Diego, California, Apr 2003.
- [94] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, Chicago, IL, Jul 1998.
- [95] K. Ranganathan and I. Foster. Identifying dynamic replication strategies for high performance Data Grids. In *Proceedings of the Second International Workshop on Grid Computing*, pages 75–86, Nov 2001.
- [96] K. Ranganathan and I. Foster. Simulation studies of computation and data scheduling algorithms for Data Grids. *Journal of Grid Computing*, 1:53–62, 2003.
- [97] Globus Replica Location Service (RLS). <http://www.globus.org/rls/>.
- [98] P. Rodriguez and E. W. Biersack. Dynamic parallel access to replicated content in the Internet. *IEEE/ACM Transactions on Networking*, 10(4):455–465, 2002.
- [99] SC 2004 HPC Bandwidth Challenge. <http://www.sc-conference.org/sc2004/bandwidth.html>.
- [100] J. M. Schopf. A general architecture for scheduling on the Grid. Technical Report ANL/MCS-P1000-10002, Argonne National Laboratory, 2002.

BIBLIOGRAPHY

- [101] H. Shan and L. Oliker. Job superscheduler architecture and performance in computational Grid environments. In *Proceedings of the ACM/IEEE SC2003 Conference*, Nov 2003.
- [102] S. Sivasubramanian, M. Szymaniak, G. Pierre, and M. van Steen. Replication for web hosting systems. *ACM Computing Surveys*, 36(3):291–334, Sep 2004.
- [103] The SDSC Storage Resource Broker (SRB). <http://www.npaci.edu/DICE/SRB/>.
- [104] H. Stockinger, O. Rana, R. Moore, and A. Merzky. Data management for grid environments. In *European High Performance Computing and Networking (HPCN 2001)*, Amsterdam, Holland, Jun 2001.
- [105] H. Stockinger, K. Stockinger, E. Schikuta, and I. Willers. Towards a cost model for distributed and replicated data stores. In *Proceedings of 9th Euromicro Workshop on Parallel and Distributed Processing (PDP2001)*, pages 461–467, Mantova, Italy, Feb 2001.
- [106] 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. In *Proceedings of 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*, Jun 2003.
- [107] A. S. Tanenbaum. *Distributed Operating Systems*. Prentice-Hall, 1995.
- [108] O. Tatebe, Y. Morita, S. Matsuoka, N. Soda, and S. Sekiguchi. Grid datafarm architecture for petascale data intensive computing. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2002)*, 2002.
- [109] The TeraGrid Project. <http://www.teragrid.org>.

BIBLIOGRAPHY

- [110] D. Thain, J. Bent, A. Arpaci-Dusseau, R. Arpaci-Dusseau, and M. Livny. Gathering at the well: Creating communities for Grid I/O. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, Denver, Colorado, Nov 2001.
- [111] A. Venkataramani, P. Yalagandula, R. Kokku, S. Sharif, and M. Dahlin. Potential costs and benefits of long-term prefetching for content-distribution. *Elsevier Computer Communications*, 25(4):367–375, March 2002.
- [112] J. Wang. A survey of web caching schemes for the Internet. *ACM SIGCOMM Computer Communication Review*, 29(5):36–46, Oct 1999.
- [113] B. M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, Dec 1988.
- [114] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox. Removal policies in network caches for World-Wide Web documents. In *Proceedings of the ACM SIGCOMM '96 Conference*, Stanford University, CA, 1996.
- [115] J. Winick and S. Jamin. Inet-3.0: Internet topology generator. Technical Report CSE-TR-456-02, University of Michigan, 2002.
- [116] T. Wong and R. Katz. An analysis of multicast forwarding state scalability. In *Proceedings of the 8th IEEE International Conference on Network Protocols (ICNP 2000)*, Osaka, Japan, Nov 2000.
- [117] R. Wooster and M. Abrams. Proxy caching that estimates page load delays. In *the sixth international conference on World Wide Web*, Santa Clara, CA, 1997.
- [118] N. Yamamoto, O. Tatebe, and S. Sekiguchi. Parallel and distributed astronomical data analysis on grid datafarm. In *Proceedings of 5th IEEE/ACM International Workshop on Grid Computing (Grid 2004)*, 2004.

BIBLIOGRAPHY

- [119] N. E. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, 1994.
- [120] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proceedings of IEEE INFOCOM'96*, San Francisco, CA, 1996.
- [121] G. K. Zipf. *Human Behavior and the Principles of Least Effort*. Addison-Wesley, 1949.