การวิเคราะห์และประเมินประสิทธิภาพของแคชชิ่งในระบบดาตากริด

นายธีรยุทธ หิรัญทราภรณ์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2548
ISBN 974-53-2519-8

THE PERFORMANCE EVALUATION AND ANALYSIS OF CACHING IN DATA GRID SYSTEM

Mr. Teerayut Hiruntaraporn

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering Program in Computer Engineering
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2005
ISBN 974-53-2519-8

Thesis Title	The Performance Evaluation and Analysis of Caching in Data Grid System
By	Mr. Teerayut Hiruntaraporn
Field of Study	Computer Engineering
Thesis Advisor	Natawut Nupairoj, Ph.D.

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Master's Degree

[illegible] .... Dean of the Faculty of Engineering
(Professor Direk Lavansiri, Ph.D.)

THESIS COMMITTEE

[illegible] .... Chairman
(Yunyong Teng-amnuay, Ph.D.)

[illegible] .... Thesis Advisor
(Natawut Nupairoj, Ph.D.)

[illegible] .... Member
(Assistant Professor Putchong Uthayopas, Ph.D.)

[illegible] .... Member
(Veera Muangsin, Ph.D.)

ธีรยุทธ หิรัญทราภรณ์ : การวิเคราะห์และประเมินประสิทธิภาพของแคชชิ่งในระบบดาตากริด (THE PERFORMANCE EVALUATION AND ANALYSIS OF CACHING IN DATA GRID SYSTEM). อาจารย์ที่ปรึกษา : อ. ดร. ณัฐวุฒิ หนูไพโรจน์, 71 หน้า. ISBN 974-53-2519-8

ระบบ งาน ที่ เน้น ใช้ ข้อมูล ภายใน ระบบ ดา ตา กริด นั้น มักจะ ต้อง ทำ การ โอน ย้าย ข้อมูล ปริมาณมาก ซึ่งเป็นกระบวนการที่ใช้แบนด์วิทย์สูงและทำให้ประสิทธิภาพของระบบเครือข่ายลดลง แคชชิ่งเป็นเทคนิคอย่างหนึ่งที่ใช้ในการลดปริมาณการใช้งานแบนด์วิทย์โดยการจัดเก็บข้อมูลที่มีแนวโน้มในการใช้งานต่อในอนาคตมาเก็บในพื้นที่ของตัวเองซึ่งวิธีนี้สามารถใช้งานได้เป็นอย่างดีโดยเฉพาะอย่างยิ่งในระบบอินเตอร์เน็ต อย่างไรก็ตาม การที่จะนำเอาวิธีเดียวกันนี้มาใช้ในระบบกริดเลยนั้น ไม่สามารถทำได้ในทุกแง่มุม เนื่องจากความแตกต่างในสภาวะระหว่างระบบกริดและอินเตอร์เน็ต ตัวอย่างเช่นขนาดของข้อมูลและการใช้งานเครือข่าย

วิทยานิพนธ์ฉบับนี้นำเสนอวิธีการแคชแบบบล็อคซึ่งสามารถนำมาประยุกต์ใช้กับระบบดาตากริดได้โดยการนำเอาข้อมูลที่มีขนาดใหญ่มาแบ่งเป็นบล็อคของข้อมูลที่มีขนาดเล็ก ซึ่งวิธีการนี้จะทำให้มีการใช้พื้นที่ของแคชที่ดีขึ้นซึ่งส่งผลให้ประสิทธิภาพการทำงานโดยรวมดีขึ้น ในการทดสอบจะทำการทดสอบโดยการจำลองสถานการณ์โดยใช้โปรแกรมเอ็นเอส2 ซึ่งมีการเพิ่มเติมส่วนในการจำลองระบบกริดเข้าไป จากผลการทดสอบสามารถสรุปได้ว่าการใช้แคชในระบบดาตากริดโดยเฉพาะการแคชแบบบล็อคนั้น มีประโยชน์ในด้านการเพิ่มประสิทธิภาพการทำงานของระบบ โดยสามารถนำไปพัฒนาต่อเพื่อการทำไปใช้งานจริง และสามารถทำการปรับปรุงค่าต่างๆเพื่อให้สามารถใช้งานได้ดีตามสภาพแวดล้อมได้

ภาควิชา ......วิศวกรรมคอมพิวเตอร์......	ลายมือชื่อนิสิต ..[illegible]....
สาขาวิชา......วิศวกรรมคอมพิวเตอร์......	ลายมือชื่ออาจารย์ที่ปรึกษา [illegible]......
ปีการศึกษา ....2548............	

##4670337721 : MAJOR COMPUTER ENGINEERING
KEYWORDS : DATA GRID/ GRID DATAFARM / CACHING / PERFORMANCE EVALUATION

TEERAYUT HIRUNTARAPORN : THE PERFORMANCE EVALUATION AND ANALYSIS OF CACHING IN DATA GRID SYSTEM. THESIS ADVISOR : NATAWUT NUPAIROJ, Ph.D., 71 pp. ISBN 974-53-2519-8

Data-intensive applications in the data grid environment usually transfer large amount of data, which can cause high network consumption and degrade the overall performance of the network. Caching is one of the techniques that can reduce network bandwidth usage by storing data likely to be used in the near future in its local storage. This technique is very successful, especially in the Internet. However, the conventional cache used in WWW can not be applied completely into the grid environment because of the differences between Web and Data Grid's behaviors, for example, data size, network consumption.

This thesis proposes the block-based data caching model for data grid environment by dividing data file in many data blocks which can achieve better space utilization which leads to better overall performance. The experiment conduct by simulation using grid extended NS-2. The result indicates that data caching, especially block-based data caching, can provide benefits for the data grid environment which can be further developed into real implementation and can fine-tune for different environments.

Department ... Computer Engineering ...	Student's signature ..[illegible].
Field of study .. Computer Engineering ..	Advisor's signature ..[illegible].........
Academic year 2005	

Acknowledgements

To accomplish all of this work, there are many people who provide supports and reviews. First of all, I would like to thank my advisor, Dr. Natawut Nupairoj, for his grateful advices and technical knowledge. Regarding data grid and Grid DataFarm information and support, I would like to thank Assistant Professor Dr. Putchong Uthayopas and Dr. Osamu Tatebe for their grateful help.

The friendly environment inside this department is one of the key that bring me to the final phase. I would like to thank every people in department of computer engineering and Information System Engineering Laboratory.

Finally, I would like to thank to my parents for their kindly support and love which make me strength and reach to this moment.

Contents

List of Tables

List of Figures

Page

Page

CHAPTER I

INTRODUCTION

1.1 Introduction

The emergence of grid technologies provides efficient resource sharing and problem solving environments in worldwide collaborations. The power of grid computing can overcome the complex problems which were difficult to solve using conventional technologies. The main idea of grid is to allow resource owners to share their resources to other users with individual access right.

The data grid, the grid technology that enhances high performance data management features, initiated in order to deal with the grid-enabled data intensive computing such as High Energy Physics, Climate Simulations, and Bioinformatical Engineering. Typically, the data grid systems have to manipulate large-scale data ranging from gigabytes to petabytes. There are several projects that deploy data grid infrastructure to solve specific problem for example, GriPhyn[1] and PPDG[2] for physics experiment, Information Power Grid[3] for aerospace computation, and EuDataGrid[4] for High Energy Physics. In addition, open data grid middlewares are initiated in order to facilitate grid infrastructures and services to users such as Storage Resource Broker[5] and Globus Toolkits[6].

The high volume of data and users which are dispersed throughout the world makes data management in grid environments complex and nontrivial. Moreover, high latency network and large data sets produced by applications in the data grid environment can put heavy burden on network resources and cause applications to spend longer times to finish. The emerging contributions mostly involve data replication for the grid environment such as replica creation, placement or elimination. But in some certain situations, data replication may not be effective. One of such scenarios is the system having inadequate disk spaces to store the entire replica on each computing node which makes replication less feasible, especially huge data replication. Furthermore, replication for large file consumes disk spaces on the destination node, even if that such node uses only a few records of such data.

Data caching is one of the techniques that can improve the efficiency of data usages in the distributed environment by storing popular data on the places closed to the clients. It is quite obvious that caching can reduce the bandwidth usages and minimize access latencies[7]. Although there are many studying of caching in grid environment, the study focused mostly on replacement policy[8, 9, 10]. Thus, the behaviors and the effects of data caching in grid environment is still relatively unknown.

In order to utilize effective caching mechanisms that can be use in real data grid environment, the study of caching behavior in data grid have to be performed and validated. This thesis aims to study the behavior of caching in data grid, especially in block-based caching approach which can outperform the other approaches in theirs space utilization ability. The study bases on grid simulations which used Grid Datafarm's data accessing characteristics as a data grid model. The result from this study can be used as a baseline for implementing feasible data grid caching in the future.

1.2 Objective

The main objective this work is to evaluate data caching in the data grid including its behavior and performance characteristics. In addition, our study emphasizes on space utilization in grid caching mechanisms. This will provide us insight on the effectiveness of cache space usage, especially when we use the block-based caching scheme.

1.3 Scope of this Work

1. This work proposes the framework for block-based data cache model for data grid environment.
2. The basic assumptions describe in Chapter 4 will be appled throughout this work.
3. The Simulation will be performed using Network Simulator (NS-2).

1.4 Research Process

1. Study background theories in the field of grid computing, data grid, data replication, data caching, and conventional web caching.
2. Perform literature survey in the relevant fields.
3. Design and develop the simulation framework for the experiments.

4. Conduct experiments to evaluate the performance.
5. Validate and summarize the simulation results.
6. Write Thesis.

1.5 Benefits

1. Provide framework to implement block-based data caching.
2. Evaluate the performance of block-based data caching.

1.6 Organization of the Thesis

In the next chapter, the background theory and related work included in this thesis are described. The concepts regarding conventional and block-based data cache are represented in Chapter 3. Chapter 4 demonstrates the detailed implementation of the block-based data cache and simulation framework for performance evaluation. The experiments are explained in Chapter 5. Finally, we provide the conclusion and the future works in Chapter 6.

1.7 Paper Publication

Our works related to this thesis have been published twice as followed:

1. *The Performance Evaluation of Block-Based Data Caching in Data Grid Environment*, The 9th Annual National Symposium on Computational Science and Engineering, Bangkok, Thailand, 23-25 March 2005.
2. *Block-based Grid Caching in Grid Datafarm*, The 8th Internation Conference on Advanced Communication Technology, Phoenix Park, Korea, 20-22 February, 2006.

CHAPTER II

BACKGROUND THEORY AND RELATED WORK

2.1 Background Theory

2.1.1 Grid Computing

Grid computing[11] is one of the emerging technologies in this decade. It is a system which for a large-scale resource sharing, and high-performance application. It was originally proposed for the advanced science and engineering efforts.

At present, the Grid concept is applied for the challenge concerning *coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations*. There are two interesting words in these emphasized phases. First, the term 'resource sharing', which includes only the data exchange in most conventional distributed system, covers diverse resources involving direct access to computer, software, sensor, and many more. Next, the term 'virtual organizations' is a set of individual or organizational units that define contracts of sharing for one another. Such rules are strictly controlled and have clearly definitions about permission to producers and consumers; for instance, the rule shows that who can share resources and the limitation of sharing in those resources.

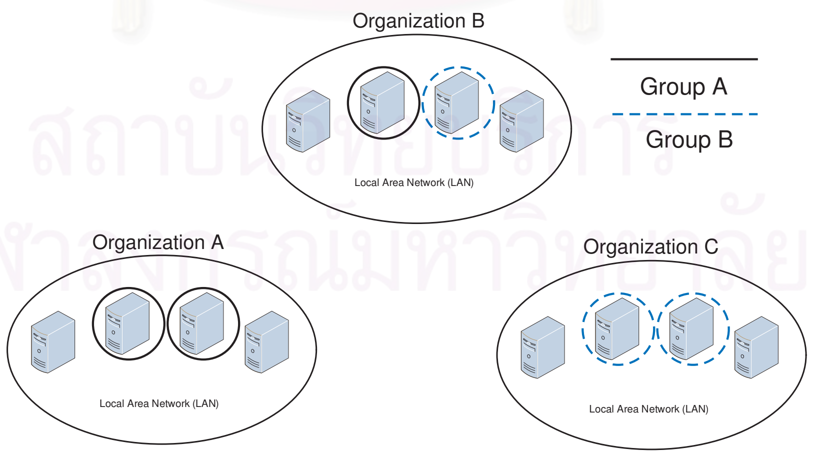


Figure 2.1: Example of VO scenario

The example of two virtual organizations is shown in figure 2.1. There are 3 actual organizations which are located in different places in the world and there are 2 projects forming 2 VOs, the astronomy research (as black circles) and the climate simulation (as dotted circles) atop of the resources in these organizations. From the figure, organizations can participate in each VO by sharing some resources. Furthermore, they can join more than one VO like Organization B.

According to the Grid problem and VOs' concept, the Grid architecture is established in order to fulfill aforementioned requirement. It is architecture providing basic mechanisms for users to service sharing relationships. Its structure can be divided into 5 main layers.

1. *Fabric*: This layer provides the resources to which shared access is mediated by Grid: for example, computational resources, storage system, network resources, etc. It can also be cluster computer, network file system too.
2. *Connectivity*: This layer defines core communication and authentication protocols for grid transaction. It should be contained basic requirement such as transport, routing and naming. This requirement will be integrated with security standards for VOs, for instance, single sign on, delegation and user-based trust relationships
3. *Resource*: This layer defines protocols for secure transaction on individual resources such as negotiation and monitoring. The protocols in this layer have to be limited in a small and focused set.
4. *Collective*: This layer makes interactions across collections of resources.
5. *Application*: This layer consists of user applications in grid environment.

There are many contributions to the grid protocol around the world. Globus Toolkit is one of the most dominant contributions which is a fundamental enabling technology for the 'Grid', letting people share computing power, databases, and other tools securely online across corporate, institutional, and geographic boundaries without sacrificing local autonomy[6]. It is consisted of software making grid easy to manipulate such as resource management, security and many more.

Due to attractive concepts and requirements for next-generation computation, several grid projects are initiated worldwide; for example, Information Power Grid which focused about aerospace computation, Virtual Laboratory applied grid for drug design, Fusion Grid initiated magnetic fusion experiment and so on[12]. In Thailand, Thailand E-Science Project is established in order to make computing environment for interdependent researches across organizations[13]. Every project will finally make worldwide collaborative researches and progress into next-generation large-scale computation.

2.1.2 The Data Grid

Due to an increasing number of large data sets and more complex research collaborations involving many users through the world, the conventional data management mechanisms become inadequate. *Data grid*[14], the grid infrastructure extended from the proclaimed grid concept with additional data management framework, is initiated in order to address above issues. The data grid architecture is driven from four principles derived from the requirement which is the operation in wide area, multi-institutional, heterogeneous environments:

1. *Mechanism neutrality.* The architecture should encapsulate peculiarities of low-level resources.
2. *Policy neutrality.* The design decisions are exposed to the user.
3. *Compatibility with grid infrastructure.* The data grid is compatible to lower-level grid mechanism.
4. *Uniformity of information infrastructure.* The same data model and interface are used in the underlying grid information infrastructure.

There are two basic services which are essential to data grid architecture. First, the data access service provides data access interface and abstraction of storage system. This allows application to have a uniform view of data and process them with the same mechanism. The latter is the metadata access which is concerned with the management of data information including publishing and accessing the metadata. From these underlying services, a number of high-level services are deployed in the upper layer, such as replica management, replica selection, and so on.

A data grid provides data management environment for distributed organizations. This imposes four basic requirements including global name space, latency management techniques, and persistency and uniform access mechanisms for data discovery, access,

Category	Project
Biological Data Grids	Federating Brain Data
	Bioinformatics
	Visible Embryo Project
	Virtual Drug Design
Physics & Astronomy Data Grids	GriPhyN
	PPDG
	Nile
	China Clipper
	Grid Datafarm
	Digital Sky Survey
	GIOD
	ALDAP
Earth Science Data Grids	Earth System Grid
	AVHRR DL
	ESIPS
Miscellaneous Data Grids	European Data Grid
	NARA
	ADEPT
	DVC
Software Systems	SRB
	Globus Data Grid
	ADR
	DataCutter
	Mocha
	Internet2 Distributed Storage
	Niagara

Table 2.1: Current active projects concerning a data grid.

and movement. At present, there are a lot of activities and researches involving data grids. Table 2.1 represent examples of the existing data grid project which can be reviewed at [15].

2.1.3 Grid Datafarm

Grid Datafarm (Gfarm)[16] is one of several middlewares that supports petascale data intensive applications in a grid across administrative domains. Gfarm facilitates virtual file system concept in which support for conventional POSIX API and Gfarm native API in order to drive application into a grid environment. In addition, Gfarm introduces two main concepts: *Gfarm file* or *Superfile* and *file-affinity scheduling*. Gfarm file is a group of physical files distributed in several nodes worldwide which can be viewed and managed as if it is a single logical file. File-affinity scheduling is a scheduling method that is adapted from *Owner Computes* concepts in which computations are happened in nodes that own data and processed in parallel style.

2.1.3.1 Grid Datafarm Filesystem

Grid Datafarm (Gfarm)[16] is one of several middlewares that supports petascale data intensive applications in a grid across administrative domains. Gfarm facilitates virtual file system concept in which support for conventional POSIX API and Gfarm native API in order to drive application into a grid environment. In addition, Gfarm introduces two main concepts: *Gfarm file* or *Superfile* and *file-affinity scheduling*. Gfarm file is a group of physical files distributed in several nodes worldwide which can be viewed and managed as if it is a single logical file. File-affinity scheduling is a scheduling method that is adapted from *Owner Computes* concept in which computations are operated in the nodes that own data and processed in parallel style.

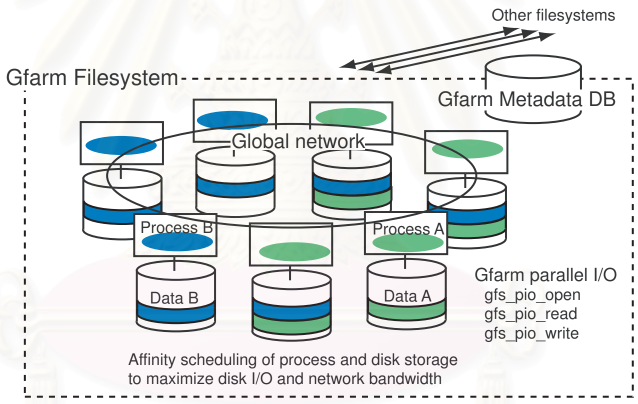


Figure 2.2: Grid Datafarm Architecture

The Gfarm filesystem node provides several utilities for filesystem operations. Most of them are similar to emerging parallel filesystem's. Additionally, it can support for *access locality* and *local file view* which are mechanisms for file-affinity scheduling. The Gfarm filesystem is invoked by Gfarm filesystem daemon (gfsd) which runs on each filesystem node and facilitates remote file operations, user authentication, file replication, node status monitoring and control.

The Gfarm metadata server maintains Gfarm metadata in metadata database, manipulated by Lightweight Directory Access Protocol (LDAP)[17]. The metadata consists of a mapping from a logical Gfarm filename to physically dispersed filename, a replica catalog, platform information, file status and file checksum. Metadata is updated consistently from Gfarm API according to file operation. However, Gfarm have metadata validation service that can detect and remove invalid information if there are invalid metadata which may be happened from unpredictable fault. An example of Gfarm metadata is depicted in figure 2.3

```
\# hkondo/hostname, pragma, grid
dn: pathname=hkondo/hostname, dc=pragma, dc=grid
objectClass: GFarmPath
pathname: hkondo/hostname
mode: 0100755
user: hkondo
group: *
atimesec: 1052719539
atimensec: 308766000
mtimesec: 1052719539
mtimensec: 308766000
ctimesec: 1052719539
ctimensec: 308766000
nsections: 0
```

Figure 2.3: Gfarm Metadata

2.1.3.2 Gfarm File

Gfarm File is a logical file that can be divided into several fragments which are spread throughout Gfarm filesystem. It is inherited from striping parallel filesystem with additional feature that each fragment can have variable file size and can store in any node in the filesystem. Figure 2.4 depicts Gfarm file conceptual view.

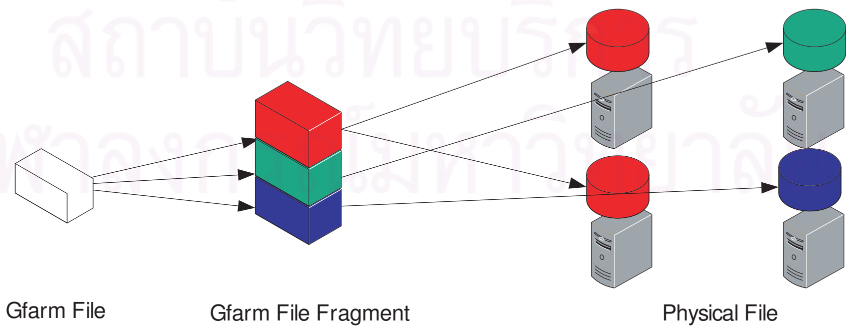


Figure 2.4: Gfarm File Concept

In Figure 2.4, Gfarm file can be replicated to make it fault-tolerance, enable low latency access, and perform load balancing over the grid. Gfarm file replication can be operated automatically or manually by using Gfarm commands and API.

The Gfarm API provides two file accessing views, global file view and local file view. The global file view is a generic file accessing behavior used by every conventional distributed filesystem. It manipulates logical files as if they are single files. On the other hand, local file view handles each fragment of logical files independently. Figure 2.5 depicts 2 file view models.

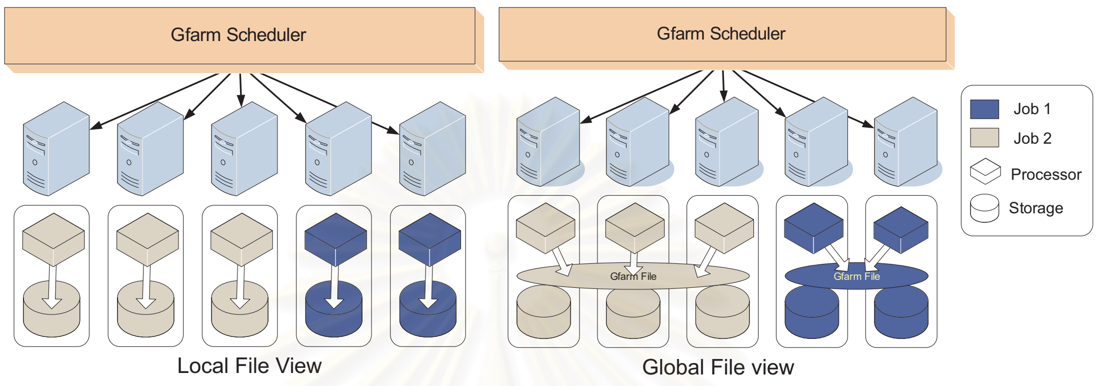


Figure 2.5: Local file view and affinity scheduling

2.1.3.3 Process Scheduler

The Gfarm process scheduler is a distributed scheduler designed mainly for file-affinity scheduling. This paradigm schedules Gfarm nodes that have specified Gfarm file in which Gfarm nodes have the same number as the corresponding Gfarm fragments. Then, the processors on each node use local file view to operate theirs own file fragments in parallel. However, if the node that owns file fragment has heavy workload, the scheduler may schedule to another node and access fragment via replication or remote access scheme.

2.1.3.4 Authentication

Gfarm provide two authentications in order to execute application or access Gfarm file on the Grid. Grid Security Infrastructure (GSI)[18] is basically used for mutual authentication and single sign-on. However, Gfarm requires a lot of authentication from processes, metadata server, and filesystem nodes that will cause execution overhead. Therefore, lightweight authentication like share-secret are preferred when there is no need to use full grid authentication method.

2.1.4 Web Caching

Caching is a basic approach for performance optimization in several fields of Computer Science, ranged from L1/L2 cache in cpus to DNS cache and proxy servers. The concept of caching is to store some data portions in the closer place to clients. Therefore, clients can retrieve the data faster than that of the origins. In the following part, we will focus on Caching in the WWW because it is quite similar with our research topic comparing to the other issues.

In the WWW, the major problem every user has to confront with is the network congestion and server overloading. A lot of people using the internet which is dramatically increased every year perform an amount of transactions causing the network congestion. Server overloading is caused by several requests from many clients that overcome the maximum capacity of servers. So servers have long responses, no response at worst. The improvement of system hardware such as internet backbone capacity or server performance seems not to be the appropriate solution because it can not fulfills the demand eventually. Web Caching, introduced in early 90's, is proposed to solve that issues by using special servers, especially 'the proxy server', to process clients' requests by looking up required data in themselves or forwarding them to server if the desired documents are not available.

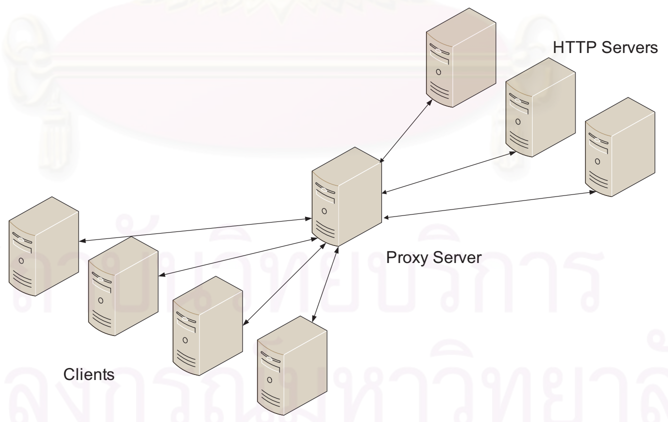


Figure 2.6: Simple Web Caching Architecture

2.1.4.1 Caching Benefits and Drawbacks

Using web caching provides several benefits for users and the internet[7]

1. Reducing bandwidth consumption, decreasing network traffic and then lessening network congestion.

2. Reducing network latency because data can be retrieved from the closer proxies instead of the servers. Moreover, since it can reduce network traffic, data can be fetched faster even though data is not in the cache.

3. Decreasing the workload of the web server because client can get the data in the nearby proxies.

4. Increasing availability and robustness of the web server, especially in the case that the web server is not available.

5. Using the data in the cache for analyzing the behavior of organizations.

However, there are several disadvantages of using web caching as well[7].

1. Client may fetch out-of-date data, if the proxies insufficiently perform data updating.

2. In the case of a cache miss, the access latency may increase.

3. A single proxy cache is a bottleneck and a single point of failure.

4. A proxy cache can degrade the hits on the original server which make fault for statistic analysis for the providers.

2.1.5 Cache Replacement Policy

Due to limited space, we can not cache everything into proxies without removing some of them from the storages. But there some issues that influence the caching's performance characteristics: will the evicted data be used in the future or do they not be used anymore. The art of decision on the data to be replaced from the cache storage is *Cache Replacement Policy*.

The performance of caching will be varied, which depended on what policies that we applied to. Different policies can lead to the various performance outcomes on different scenarios, for examples, some policies can outperform the others in minimizing latency, minimizing bandwidth, increasing space utilization, increasing hit rate. Therefore, we have to select the suitable policy in order to maximize the overall performance of the system.

There are several researches concerning cache replacement policy. We can classify them into 2 classes, deterministic policies and randomized policies. The deterministic policies perform cache replacement in predictable styles, for example, the Least Recently Used (LRU) use a decision primarily based on data's timestamp; the least used object will be evicted from the storage. On the other hands, the randomized policy uses the random algorithms which reduce the overhead used for the decision process, for example, to select evicted data randomly on the storage. The more detail about cache replacement algorithm can be found in [19, 20]

2.2 Related Works

2.2.1 File system Cache

Because we applied Grid Datafarm's access behavior throughout the research, we have to survey about caching in conventional distributed and parallel filesystem. The Sprite Network File System[21] use main memory in both servers and clients for caching file information. Figure 2.7 depicts the Sprite System which data cache can reduce the traffic in several manners.

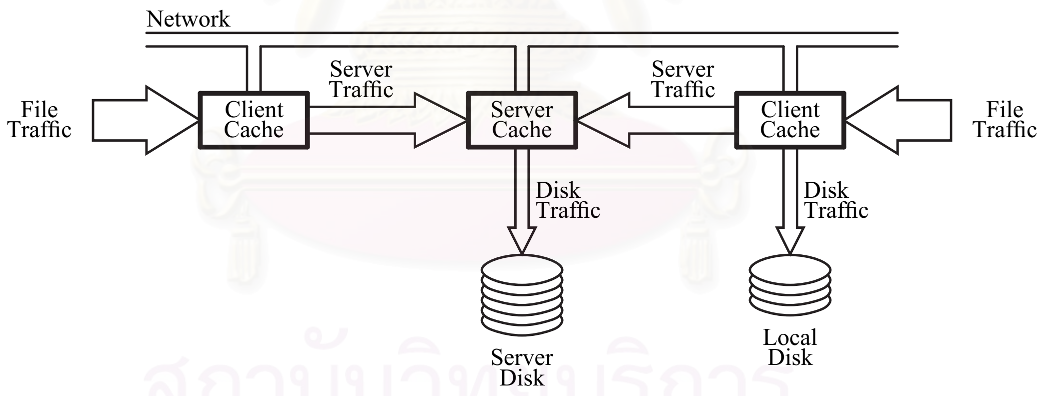


Figure 2.7: Sprite Network File System

2.2.2 Storage Resource Manager

The other grid-enabled component that have a data caching feature is the Storage Resource Managers (SRMs)[22, 23]. The SRMs are middleware components that do support dynamic space allocation and file management. There are 3 main types of SRMs, a Disk Resource Manager (DRM) which is for shared disks management, a Tape Resource Manager (TRM) which manipulates robotic tapes, and Hierarchical Resource

Manager (HRM) that is a combination of both. SRMs provide much functionality for multi-file request, data caching, and file pinning. The file pinning concept is analogous to that of file locking in many conventional distributed filesystems. But, instead of locking active file contents, it locks the active spaces in order to guarantee that data are absolutely available while file transfers are activating. E. Otoo et al. claimed in [8] that SRMs had some characteristics similar to either proxy servers' or reverse proxy servers' in the WWW and they compared their differences which is shown in table 2.2. Many contributions concerning SRMs' caching mechanisms are mostly focused on cache replacement policies including [8, 9, 10]. Figure 2.8 depicts SRM usage scenarios[8].

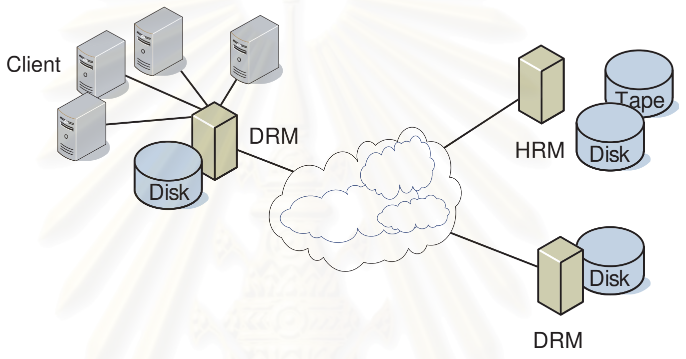


Figure 2.8: SRM Architecture

Characteristic Property	**Web Caching**	**Disk Caching in SRMs**
File/Object Size	Variable size objects of the order of megabytes	Variable size objects of the order of gigabytes
Cache Size	In the order of tens to hundreds of gigabyte	In the order of hundreds of gigabyte to ten of terabytes
Source Latency	A few milliseconds to seconds	In milliseconds to minutes
Object Transfer Time	Almost Instantaneous	In seconds up to a few minutes
Caching Requirement	Optional	Mandatory
Batched Requests	Typically one request references one object but may have a additional references to linked objects	May involve thousands of files in one request
Bundle Constraint	Only one object is referenced per request	May require that multiple files be accessed simultaneously
Cache Consistency	Cognizant of modified documents	Predominantly Read-Only and ignores consideration of cache coherence
Network Bandwidth Requirement	Standard Internet	High Speed Gigabit Networks

Table 2.2: Comparison between SRM and Web-Caching

2.2.3 Simulation Tools

Because data grids are in implemented state, The Performance analysis is mostly manipulated by simulations. However the previous simulation frameworks are not adequate to our requirement which is mainly for caching evaluation. GridSim[24] provides facilities to create simulation framework for design and evaluation of scheduling algorithm. However, the data transfer and caching behavior have not supported effectively yet. OptorSim[25] supports simulation over data grid environment, including Data Transfer and Replication, but not cover caching. Network Simulator (NS-2) is a discrete event simulator targeted at networking research. NS-2 provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks[26]. However, it has not supported for data grid simulations yet. Gridnet is a data grid simulator built on top on NS-2[27]. It provides basic grid network specifications and application level service including server node, cache node, client node, grid package. There is some limitations within the simulator, for example, it provide strict treed-like topology and can perform only simple file transfer mode. However, it provides basic structures that we can use to extend many features for perform simulations according to our data grid environments.

CHAPTER III

DATA CACHING MODEL IN GRID DATAFARM

We use Grid Datafarm model[16] as data grid reference model throughout this work. As Grid Datafarm does not defined cache management model in its architecture, we define a new caching model that is applicable to Grid Datafarm. This section describes the design of caching mechanism in Grid Datafarm used as the basic framework for all experiments throughout this thesis.

3.1 The Data Grid Scenario

We use the Tier-Model issued by MONARC Project[28] as a basis for the overall scenarios applied in this work. The tier model is illustrated in Figure 3.1.

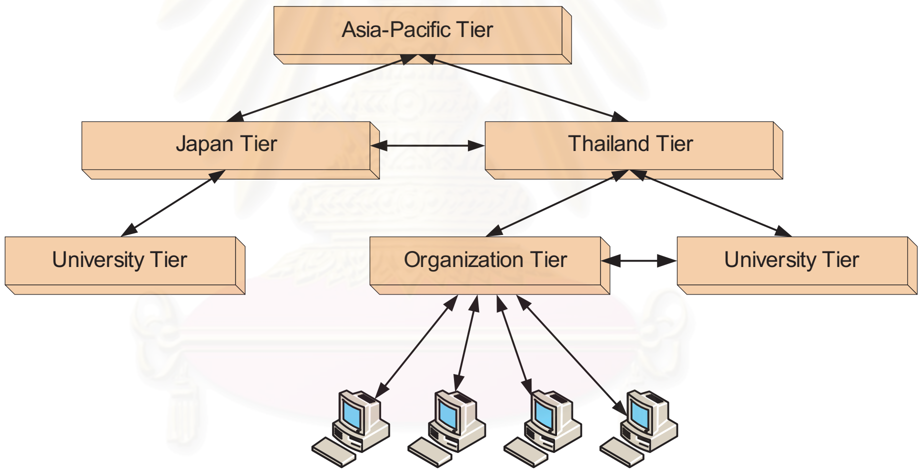


Figure 3.1: Data Grid Scenarios

The tier-model consists of several nodes in the virtual organization connecting in hierarchy topology. The topmost tier acts as a producer, which creates large-scale data derived from scientific experiment, for example, data collected from the astronomical Telescope. These data were conveyed to the lower tiers in order to perform data analysis and measurement. The transportation of these data can be classified in 2 approaches as follow:

1. Bulk data transfers — replicate all data entity to target nodes. Data processing takes place after whole data are transferred completely.

2. Remote access transfers — read data from the provider nodes in streaming-style. The providers then send block of data to clients. This method performs well when clients require just some portions of data and provides flexible data access mechanisms.

With this topology, the network requirements are very similar to the client-server model in its network-bandwidth behavior. The higher tiers have to deal with a lot of bandwidth consumptions from lower tiers and the large-scale loads from the lower tiers. Moreover, we assume that the network is not dedicated. Thus, the network bandwidths have to be shared with the other conventional network application, such as Internet, P2P, etc.

Based on our assumptions, the tremendous network usage degrades the overall performance including data transfer for the data grid environment. Caching data into intermediate node can be one of several promising methodologies that can reduce data accessing latency and network usages forwarding to the higher tiers.

3.2 Gfarm Data Access Behavior

Throughout this thesis, we use only Gfarm global file view described in Section 2.1.3. Using global file view, Gfarm I/O considers the fragment of physical related files as one logical file. The data accessing behavior is quite simple similar to a conventional filesystem with additional features of addressing the physical file. Figure 3.2 depicts the mechanism when Gfarm client accesses data inside VO which can be described in the following Steps:

1. The client queries the metadata server for the physical address of data.
2. The metadata server returns the FQDN location of data back to the client.
3. The client can connect to the server, with respect to physical address obtaining from the metadata server. However, if there are many physical file fragments, the offset and fragment numbers have to be calculated. Algorithm 1 shows the details of accessing for file *X* offset *Y*.
4. The server sends requested data back to client for processing.
5. If client wants to request additional data from the same data file but in different

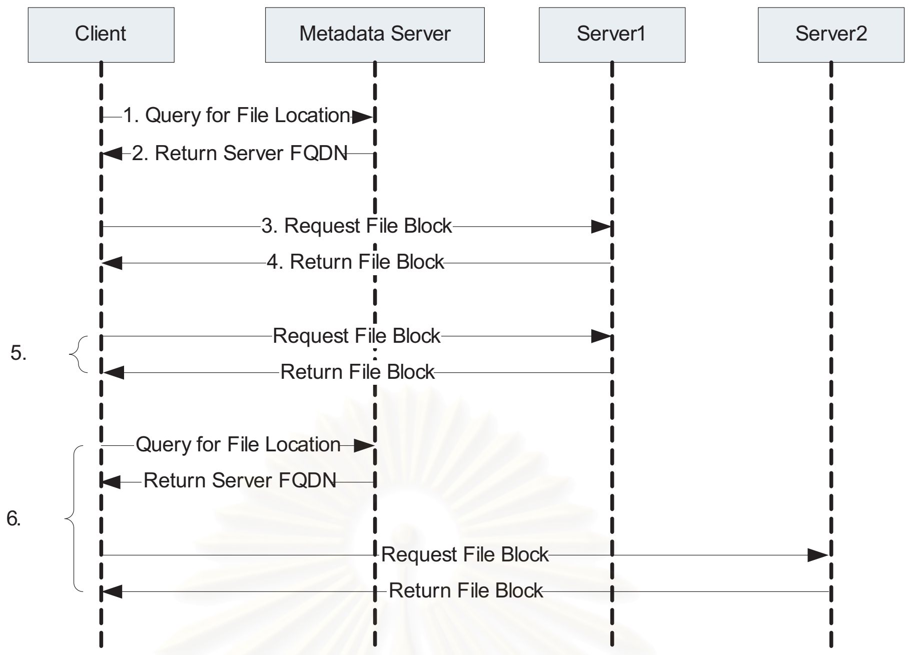


Figure 3.2: Gfarm Data Access Mechanisms

blocks, the client has to consider whether those data are resided in the same fragment as the prior data. If there are, the client can connect to the same server again in order to fetch additional data.

6. If additional data is not in current file fragment, the client has to locate new physical address for new fragment and, then, connect to the new server to get data.

Algorithm 1 *Determine Gfarm data access on file X offset Y*

```
1 begin
2     Store fragment number that contain offsets $Y$ in $section$.
3     set $offset = \sum_{i=0}^{section-1} fragmentsize(i)$.
4     Open fragment number $section$ and move file pointer to $pointer$.
5     Read data into buffer until buffers are filled or end-of-fragment detected.
6 end
```

3.3 GFarm Caching Model

In order to evaluate the caching behavior in data grid, we assume basic caching model as described in figure 3.3. In our model, the lower tiers share the intermediate

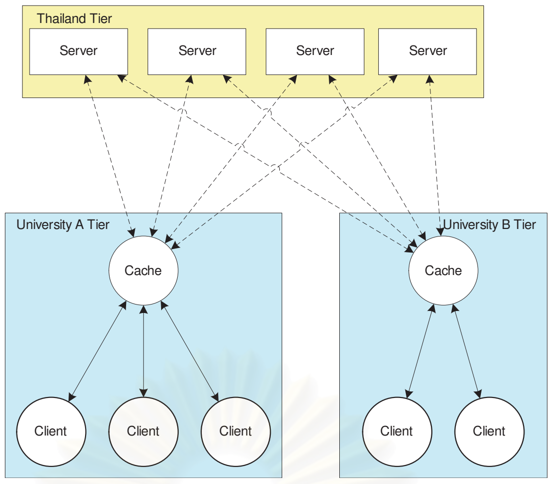


Figure 3.3: Gfarm Data Caching Model

cache server in order to obtain data from data server. Cache Server can connect to any data servers that share grid resources. The mechanisms of data caching can be described in Figure 3.4.

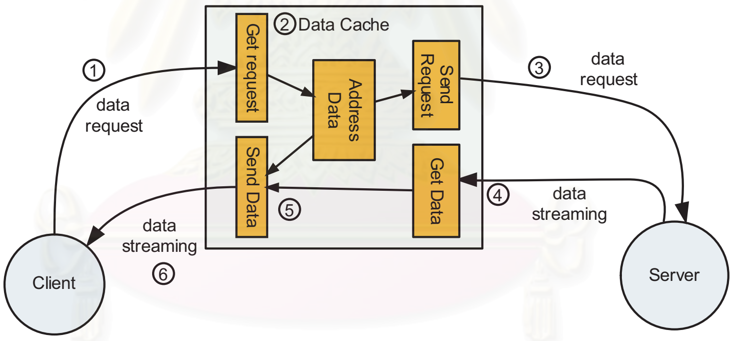


Figure 3.4: Gfarm Data Caching Mechanism

1. A client sends a request to a cache server. The request can be either for bulk data or partial data.
2. A cache server gets request from a client.
3. A cache server tries to locate the requested data from its storage spaces. If the data exist, the server will send them back to a client. Otherwise, the data request will be sent to the server own that data.
4. The server return data back to a cache server.

5. A cache server stores data with respect to its local caching policies. If there is no enough space, the cache replacement policy will be invoked in order to clear stale storage spaces.

6. Finally, a cache server returns data to the client.

3.4 Object-based Caching Model

Researches in grid caching always base on a web-based caching model. In this model, when intermediate cache server receives data request from its client, it will load the whole data as one big file from the server, store inside its storage spaces, and finally return the entire data to the client. Thus, cache treats each data file as one object no matter how large it is. We call this model "Object-Based Caching", as shown in Figure 3.5.

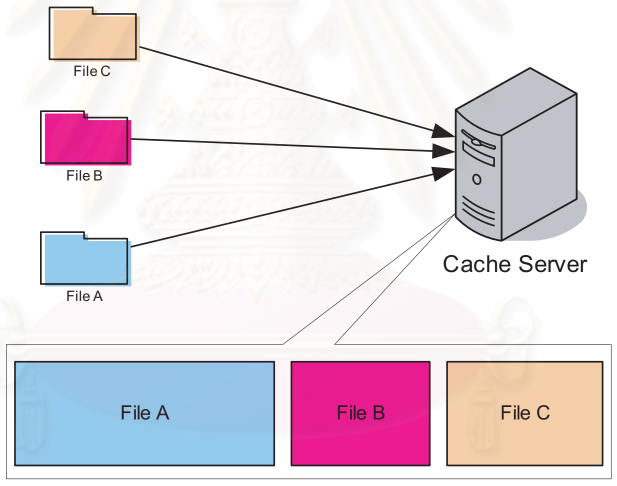


Figure 3.5: Object-Based Caching

Object-based caching has been widely used in distributed system, including the internet. As the data file inside the internet are mostly small, fetching whole data between the client and server is very trivial and flexible, for example, web browsers can download multiple objects from the same website concurrently. Moreover, the data objects are relatively so small, compared to the cache space that there are no need to perform space pre-allocation.

On the contrary, the data files in grid environment are very large, ranging from gigabytes to petabytes. Grid caching requires an additional process for allocating spaces to guarantee that data retrieved from server can be stored in the cache storage, which is called *file pinning*[23]. Therefore, when a cache miss occurs, the performance of the cache does suffer not only from long data latency for obtaining data, but also limited available spaces which finally cause cache server in grid environment less effective than web cache in WWW environment.

In addition, caching the whole data files does not flexible when some portions of data are being used. This reduces storage utilization; wastes network bandwidth, and degrades efficiency of the data cache.

3.5 Block-Based Cache Model

To improve the performance of data cache in grid environment, we propose a new caching model called "Block-Based Caching Model". Our proposed model handles each data file in fixed-size blocks. The object-based cache views its data as an individual object, no matter how large it is. On the contrary, the block-based cache manages each data file as a set of fixed-size blocks. If the data is too large to fit in one block, it will be divided into multiple blocks.

Figure 3.6 depicts the concept of our proposed block-based data caching model. When the cache server receives a request from a client, which can be a request for either the entire data file or just partial data, it will request data from servers and then sends data back to the client similar to typical object-based caching server. However, block-based cache stores data in blocks, instead of the whole file as one unit.

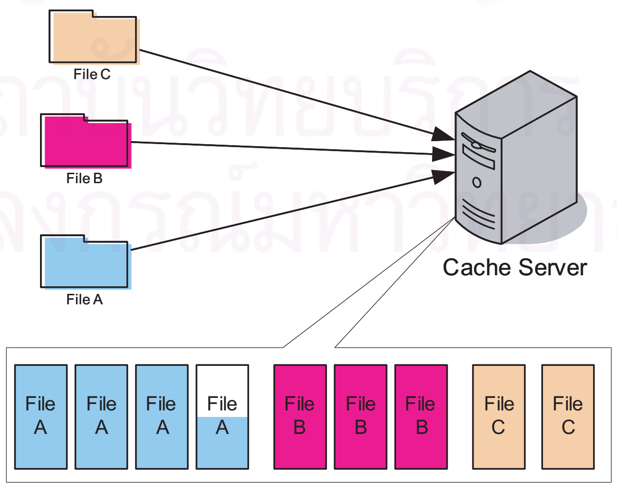


Figure 3.6: Block-Based Caching

The algorithms of object-based caching and block-based caching are shown in

Algorithm 2 and 3, respectively.

Algorithm 2 *Conventional Cache Manager Mechanisms:*

```
begin
  if The requested data is in cache.
    then
          Return data back to the clients.
     else
          Forward requests to the appropriate servers and wait for data.
          if Data can not be filled in cache spaces.
            then
                  Perform cache replacement policy.
          fi
          Store Data in the cache.
          Return data back to the clients.
  fi
end
```

Algorithm 3 *Block-based Cache Manager Mechanisms:*

```
begin
  if The requested data is in cache.
    then
          Return data back to the clients.
     else
          Forward requests to the appropriate servers and wait for data.
          if No block for storing data
            then
                  Perform cache replacement policy.
          fi
          Store Data in blocks.
          Return data back to the clients.
  fi
end
```

We evaluate the performance of both models using simulation. The results of our experiments are discussed in Chapter 5.

CHAPTER IV

FRAMEWORK IMPLEMENTATION AND SIMULATION SCENARIOS

This Chapter describes the implementation of simulation framework used in this thesis. We first discuss our basic assumptions our caching architecture, and our simulation framework.

4.1 Basic Assumptions

Throughout this thesis, we assume the following basic assumptions:

1. Data used in data grid environments are in read-only state.
2. Clients have to connect to only one data proxy in the data grid environment.
3. The basic topology consists of clients, proxies, and servers.
4. Each proxy is stand-alone. If there are multiple data proxies in the environments, they can not connect to other proxies.
5. Although GFarm servers can be used as computing nodes, this research assumes that computations always take place at client nodes only.

The first assumption is very important since grid files are often large and most of them are raw data containing the results of scientific productions or data sensors. Thus, these files are static. Moreover, this assumption leaves cache coherence as the future work and simplifies the ongoing model. The second assumption reduces the complexities of proxy connections. The third and last assumptions describe the topology that we will use throughout the research, in which there are three types of node and only one proxy can be in the paths from each client to servers. Our topology becomes more general than the topology described in [27] which is in tree-structured topology only.

4.2 Block-Based Cache Architecture

In the previous Chapter, the block-based cache paradigm was introduced with the empirical process flow model. This section will look into the detailed implementation of block-based data caching architecture.

4.2.1 Data Access Process of Client Node

For client, Gfarm provides 2 mechanisms to facilitate remote data manipulation.

1. Bulk Data Access — clients load all data file into the compute node. This approach is presented in Figure 4.1.
2. Streaming Data Access — clients load blocks of data incrementally to the compute node. Figure 4.2 summarizes this approach.

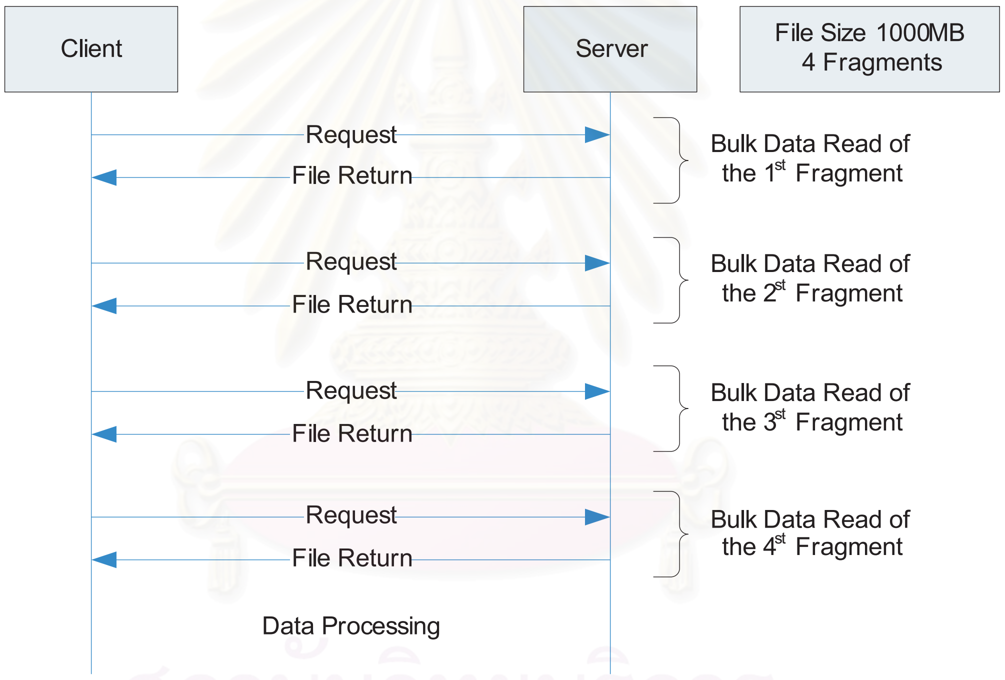

Client
Server
File Size 1000MB 4 Fragments
Request
File Return
Bulk Data Read of the 1st Fragment
Request
File Return
Bulk Data Read of the 2st Fragment
Request
File Return
Bulk Data Read of the 3st Fragment
Request
File Return
Bulk Data Read of the 4st Fragment
Data Processing

Figure 4.1: Bulk Data Access Mechanisms

The bulk data access loads all data into the compute node immediately, which incurs low transfer overhead against the streaming data access. However, this approach is not suitable if the data processing requires some portions of data files or random access behaviors are required.

The streaming data access loads parts of data to perform incremental data computation. This approach produces data output incrementally; however, the overheads from network connection and disk access will be increased, which depend on sizes of

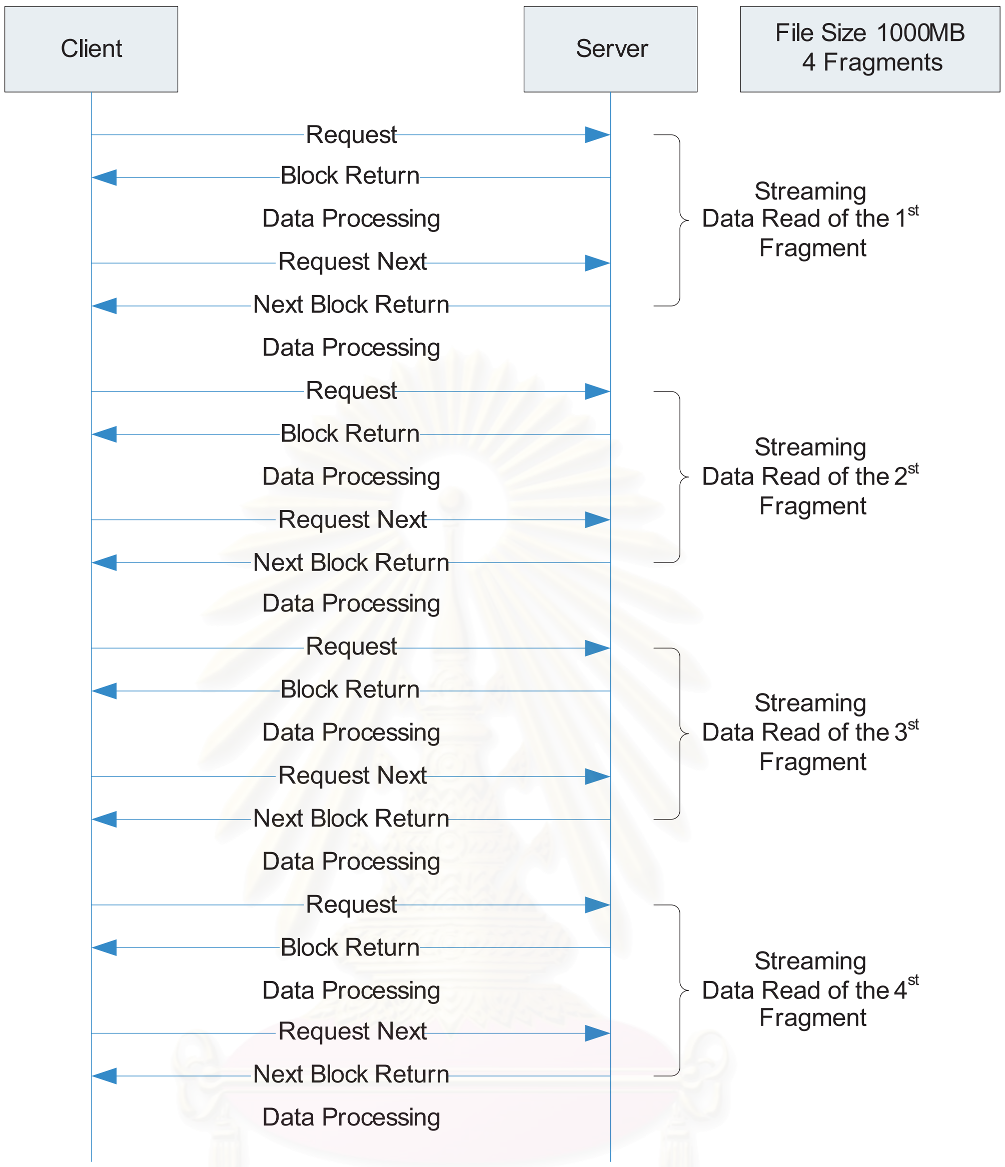


Figure 4.2: Streaming Data Access Mechanisms

the transfer data blocks. One of many situations that make this approach very effective is the scenarios that use some portions of data.

In GFarm Application, client nodes get data from server via GFarm API. For example:

```
GFS_FILE* gf = 0;
char buffer [4096];
int size = 4096;
int np = 0;
```

```
gfs_pio_open("gfarm://data.dat", GFARM_FILE_RDONLY, gf);
gfs_pio_read(gf, buffer, size, &np);
gfs_pio_close(gf);
```

In this example, GFarm will locate physical location of the data files from metadata with respect to the file descriptors and file record numbers. The metadata includes the detailed information of data files, e.g. physical address of the file fragments and file fragment numbers. The clients, then, measures the file record numbers related to the file fragments and forward a request to the GFarm server nodes. The server returns data corresponded to the requests from client nodes.

4.2.2 Architecture of Proxy Server

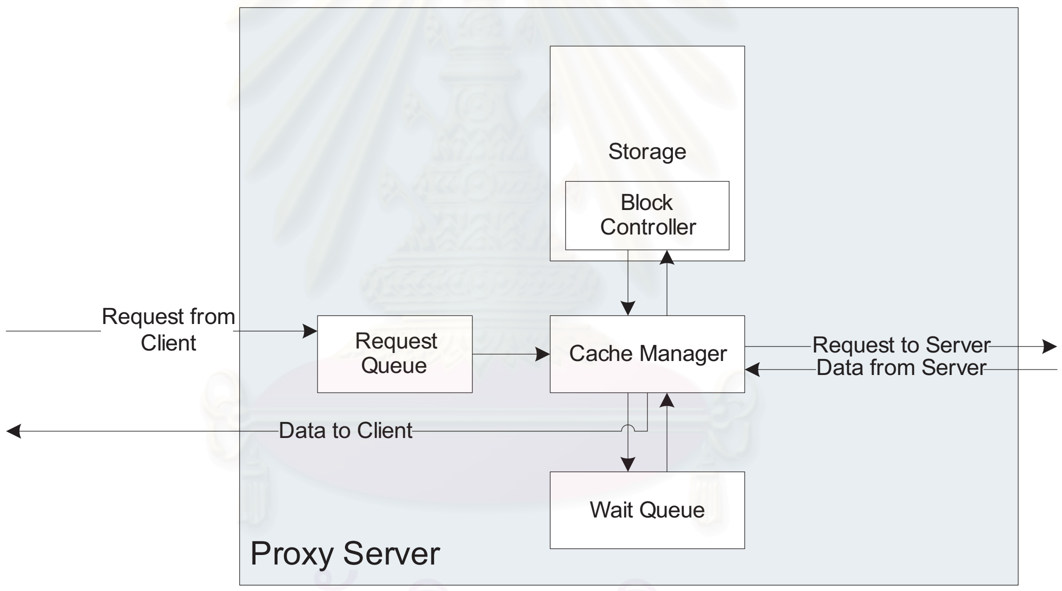


Figure 4.3: Proxy Server Architecture

Figure 4.3 represents the architecture of Grid Proxy Server. *Request Queue* receives data requests from clients and schedules them to the cache manager. It controls the number of operating task in the proxy server by delaying requests in the queue to assure that the concurrent processes are not exceeding maximum process limit. *The Cache Manager* takes care of clients' request processes; It addresses data location in Block Controller, communicates with GFarm servers, and sends request suspension to Wait Queue. *Wait Queue* receives requests suspension which target on the same data block as the ongoing task. We have to suspend such request because data files in the grid environment are heavily large. Repeating the same process again and again will cause a lot of burden to the network. Therefore, these requests must be waiting for the ongoing processes that operate on the same data blocks to be completed. *The Block Controller* deal with the manager of each individual data blocks. They have the metadata of their contents, eg. file name, file offset, file fragment number, and other value benefited for cache management policy.

4.2.3 The Process of Proxy Server

The Cache Process inside the proxy server can be described as follow.

1. *Data Request Establish* When clients send data requests to the proxy server, the data requests will be stored in the request queue. The request queue activates the requests to the cache manager if the amounts of ongoing process do not exceed maximum process configured by server. The requests that sent from the request queue are notified by the cache manager to begin the caching process.

2. *Data Query* The cache manager tries to locate the data defined in client's requests: It looks up the block description via the block controller.

3. *Space Reservation* If the data are in the cache storage, the cache manager returns that data to clients. However if the desired data are not available, the cache server allocates spaces for those data.

4. *Data Request to Servers* The cache manager sends data request to the servers hosted data files via GFarm API. The servers get request from the proxy server as if they get from GFarm Client and return data to the proxy server.

5. *Data Store and Return* These data are stored in the spaces pre-reserved by cache manager. and, finally, are sent back to the clients.

4.2.4 The Data Block Pre-Allocation and Pinning

The main issue that makes grid caching to have distinct characteristics against generic web caching is the size of data file. In web caching, the size of data file is quite small comparing to the overall workloads which is insignificant in performance metric, for example, the storage's read/write time is so short that it does not affect the cache efficiency.

However, when data are large, they have to take long times to complete the operations, e.g. data store, transfer, or eviction. These cause the race condition in the storage spaces which cause some data unable to be stored completely.

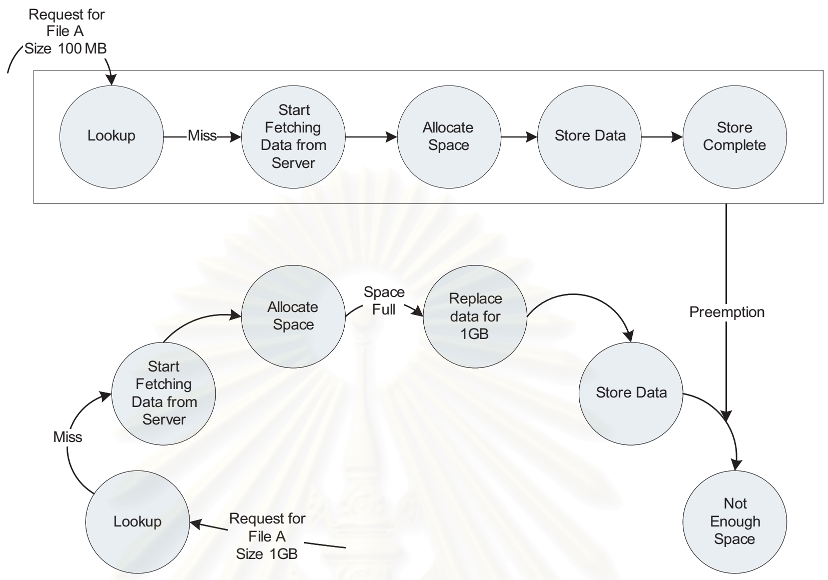


Figure 4.4: The Storage Preemption

Figure 4.4 demonstrates the storage preemption scenarios. A client requests for a 1-GB data file. Suppose that a cache server forwards this request to the data source server due to cache miss. When the connection between the cache server and the data source server is established, the cache server has to allocate 1 GB spaces which may need to perform cache replacement policy. Suppose that while data transfer for 1-GB data request is in progress, another client requests for 100-MB data file. The second request for 100MB data can fetch data faster than the first one. Thus, the spaces for 1 GB are preempted by 100MB data. This can cause a storage problem.

To solve this problem, 2 additional processes are needed. First, the space pre-allocation process is needed before sending a request to the data source server which can assure that there are enough spaces to store the requested data file. The other process is the *Block Pinning* which have been introduced in [23]. This process marks the cache spaces as busy and unavailable to be replaced in cache replacement process.

4.3 The Simulation Framework

In this work, we develop a simulation framework to evaluate the effectiveness of our grid caching. However, the existing popular grid simulators, including GridSim[29], OptorSim[30], ChickSim[31], and GridNet[27], can not fulfill our requirements which must model many parameters, for example, file access behaviors, topologies, various, cache manager, and so on. Table 4.1 describes the function of each simulator.

Simulator	Grid	Cache	Topology
EDG Optorsim	O	X	X
GridSim	O	X	X
NS-2	X	O	O
GridNet/NS2	O	O	X

Table 4.1: Preexisting Simulator's Function

Our data grid caching simulator extended from GridNet Simulator and implemented in C++ as a module for Network Simulator (NS-2). The architecture of the simulator is illustrated in figure 4.5. In figure 4.5, *gfarm client*, *gfarm cache* and

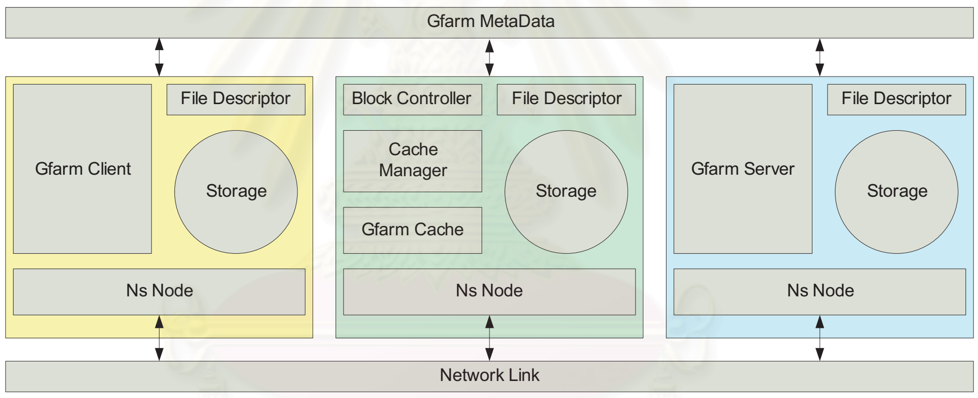


Figure 4.5: The Data Grid Cache Simulator

gfarm server are application agents that provide interfaces for user to manipulate the grid operation including requesting files and probing servers by using Tcl language. All three agents have *file descriptors* in order to control file access status for every transaction between entities. Gfarm client and server seem identical in logical architecture. However, the semantics and the functions are absolutely different. The client performs a role as a compute node which gets data from cache or servers and performs computations, whereas the server performs similar to a conventional file server like NFS. Gfarm cache node include *cache manager* and *block controller* for caching operation. Cache Manager deals with cache spaces, while block controller manages block information, such as filename and file position, in each block number. Several cache policies are feasible to implement by inheriting cache manager class and implement basic functions.

GFarm Metadata consists of the information about the entities in the grid environment and serves the information requests from GFarm nodes. The process of metadata request in metadata server takes short time comparing to data transfer between GFarm nodes which we can summarize that the metadata access time have no significant impact on the overall data processing time. Therefore, we do not include the metadata server in topology and assume that the metadata can be retrieved immediately after making metadata request.

CHAPTER V

THE EXPERIMENT AND EVALUATION

5.1 Simulation Scenarios

5.1.1 Network Topology and Parameters

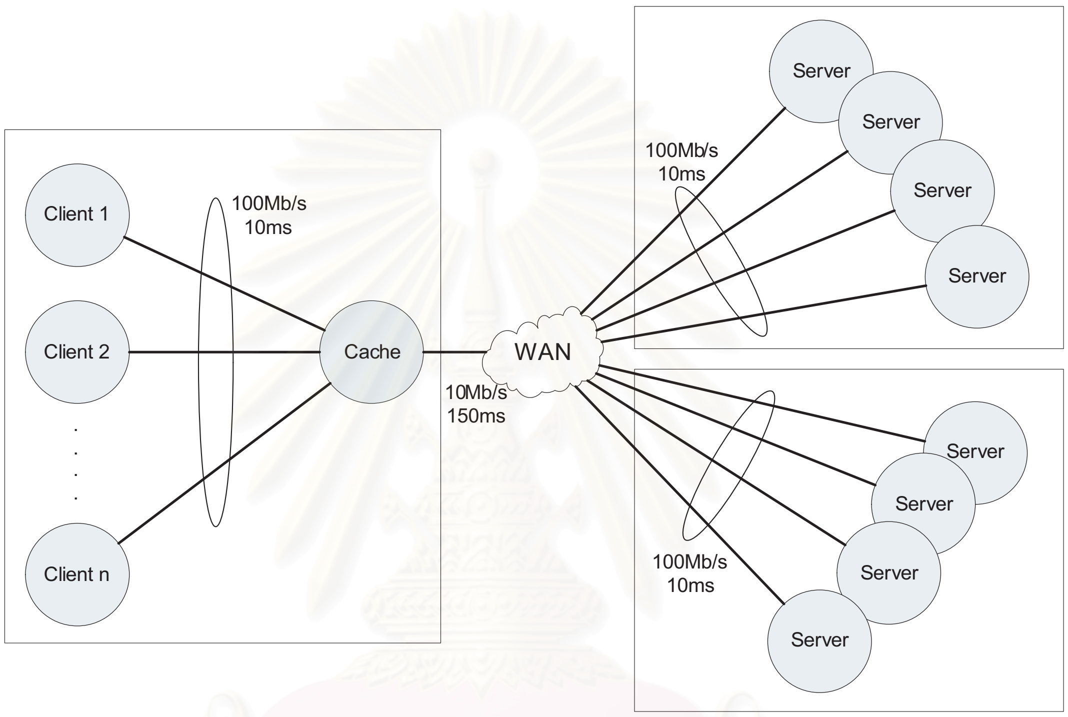


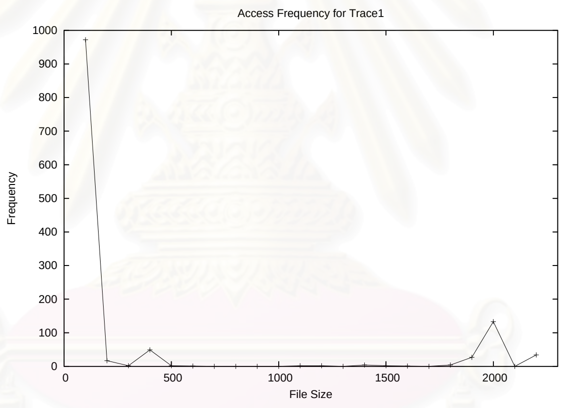
Figure 5.1: Simulation Scenarios

Figure 5.1 represents the topology used in our experiments. It consists of 3 domains; one domain hosts all clients and the other two domains host servers. Grid cache server is deployed in a client domain, which receives requests from clients and forwards requests to servers in other domains. The servers maintain data files used in the scenarios. Trace data is used to generate client requests. Note that clients and servers have unlimited storage spaces.

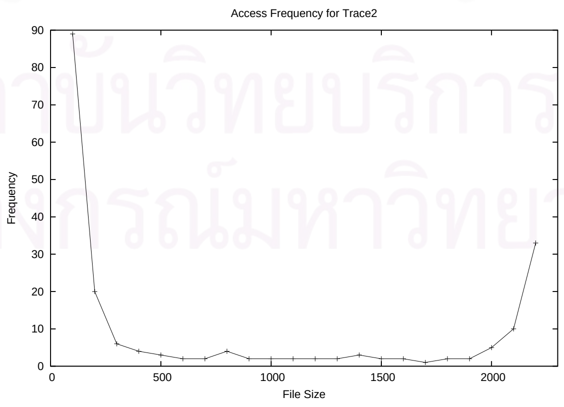
In our topology, we assume that the access time between client domain and the server domain is 150ms, which is derived from the propagation latency measured between Department of Computer Engineering, Chulalongkorn University and the site in Japan using BNR SpeedTest[32]. The propagation latency of each domain is set to 10 milliseconds, which is derived from the average latency of LAN inside Chulalongkorn University. The bandwidth between each node can be classified into 2 types. The intra-domain bandwidth, which is the bandwidth of LAN between nodes in the same domain, is set to 100Mb/s by default. The inter-domain bandwidth, which is the bandwidth of WAN between nodes in different domain, is set to 10 Mb/s by default.

5.1.2 Data Access Traces

To evaluate the model realistically, we use the real data access trace in our simulation. The trace data we use is the log of file access activities derived from Jefferson Lab Asynchronous Storage Manager(JASMine) at Jefferson Lab(JLab). These workloads involve Neuclear Physics experiments that conduct in JLab[33] as well as grid experiment in PPDG[2]. We extract the trace data into two trace logs. The first log consists of 1153 requests with 264GB overall data usages (17MB average file size). The later consists of 200 requests with 144GB data usages (500MB average file size). The characteristics of each data trace represent in Figure 5.2.



(a) Trace 1



(b) Trace 2

Figure 5.2: Access Pattern Characteristics

5.2 Simulation Results

We perform the experiments in 2 main scenarios. Firstly, we look into the impact of the inter-networking bandwidth to the performance of the cache model. Next, we determine the affect of block size to the performance of the cache model.

5.2.1 The Affect of The Inter-networking Bandwidth

In the first scenario, we evaluate the model based on bandwidth between computing nodes using block-based caching model with 100MB block size. The bandwidth are configured into 5 different situations shown in table 5.1.

Situation	Intra-Bandwidth	Inter-Bandwidth
1	10Mb/s	1Mb/s
2	50Mb/s	5Mb/s
3	100Mb/s	10Mb/s
4	500Mb/s	50Mb/s
5	1000Mb/s	100Mb/s

Table 5.1: Bandwidth Defined in Each Situation

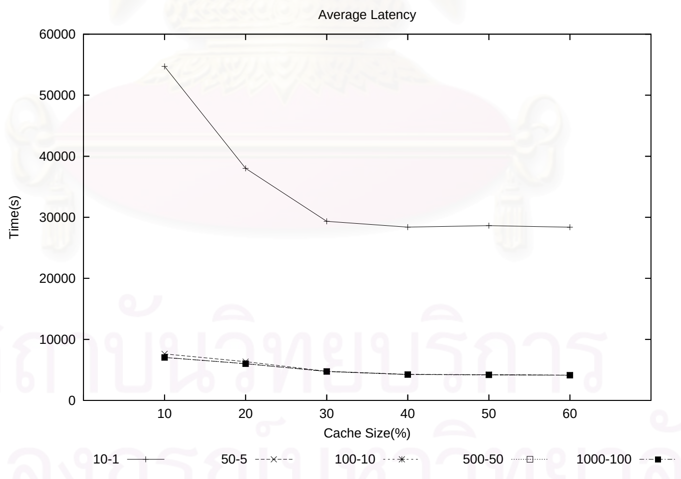


Figure 5.3: Average Latency on Different Bandwidth

Figure 5.3 represents the average latency fro the simulation result, which indicate the affect of the bandwidth to the data grid performance. This graph indicates that the smaller cache size, the longer latency to get data. However, when cache size become larger, the download speed improves gradually until the cache size is larger that saturated value, 30% in this scenario. This is because when cache size is small, the amounts of cache blocks to serve every data file become insufficient that leads to high probability of being replaced with new objects. Therefore, the overheads for remote data downloading and block replacement become larger and increase more loads. On the contrary, the large cache size provides more cache blocks which make high data availability to be downloaded from cache immediately. But if all data file can be stored in cache storage completely, the latency cannot be improved even through the cache size increased, which is called “saturated value”.

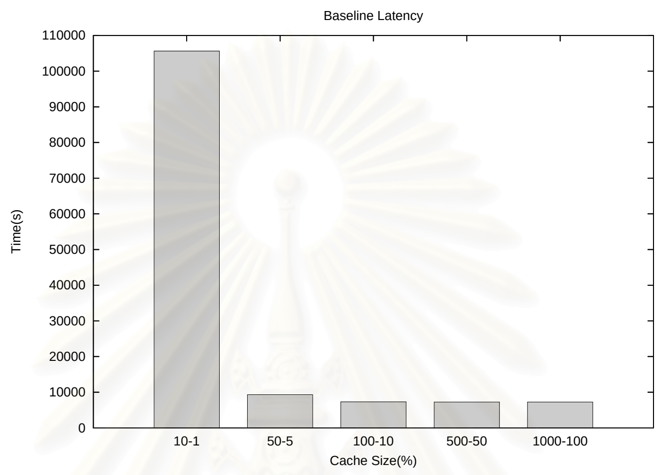


Figure 5.4: Baseline Latency on different Bandwidth

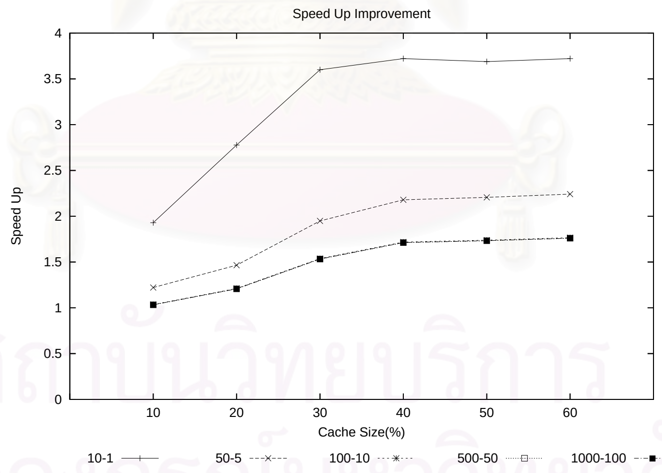


Figure 5.5: Latency Improvement on Different Bandwidth

If we normalize the average latencies in Figure5.3 with baseline latencies or average latencies when no caching in Figure 5.4, we have the results illustrated in Figure 5.5. The normalized results indicate that the low bandwidth network get more benefits from caching than high bandwidth network. For example, the 50% cache size in 10-1 bandwidth can get 3.5 times faster than the baseline, whereas the speed up ratio of 100-10 is only 1.5 to the baseline.

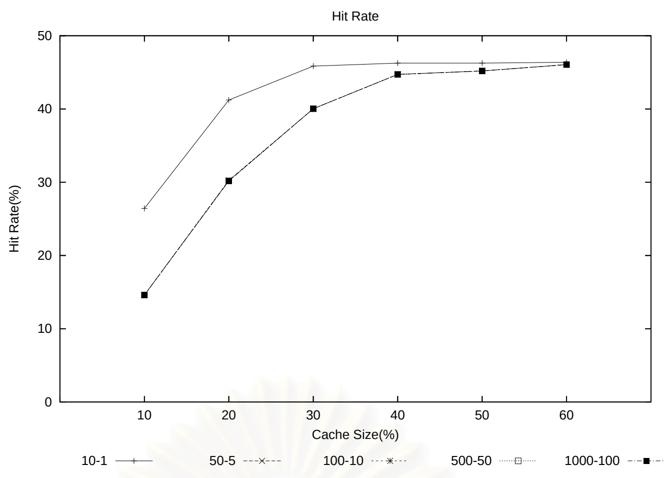


Figure 5.6: Hit Rate on Different Bandwidth

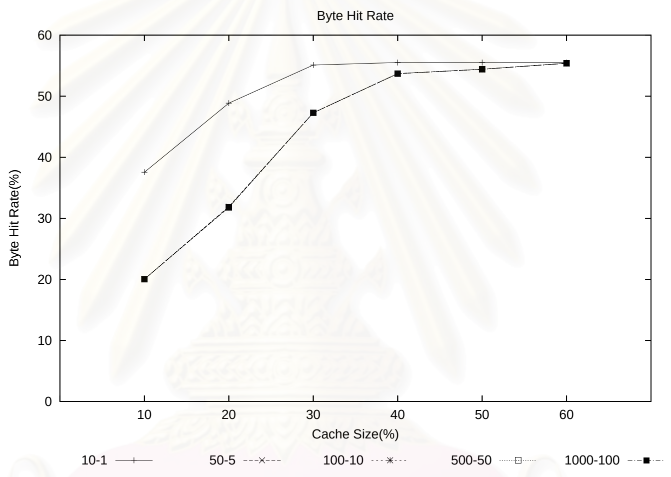


Figure 5.7: Byte Hit Rate on Different Bandwidth

Figure 5.6 and 5.7 represent hit rates and byte hit rates in this experiment. The results show that the smaller cache size, the lower hit rate and lower byte hit rate. The low hit rate causes cache server to download data from remote site frequently which incurs more latency overheads inside the data cache. The results from hit rate and byte hit rate are corresponding to the average latency.

5.2.2 The Effects of Block Size

In this scenario, we use block size as the performance factor. As the intra-domain bandwidth is commonly 100Mb/s, the inter-domain and intra-domain bandwidths are set to 10 and 100 Mb/s respectively. We use different cache block size as factor for scenarios as depicted in Table 5.2.2. In Table 5.2.2, “Baseline” is the situation when there are no cache server. Therefore, no caching policy applies to this scenario. “Obj”

Block Type	Description
Baseline	No Cache
Full	Object-Based Cache
Buff25	Block-Based Cache 25MB
Buff50	Block-Based Cache 50MB
Buff100	Block-Based Cache 100MB
Buff200	Block-Based Cache 200MB
Buff400	Block-Based Cache 400MB

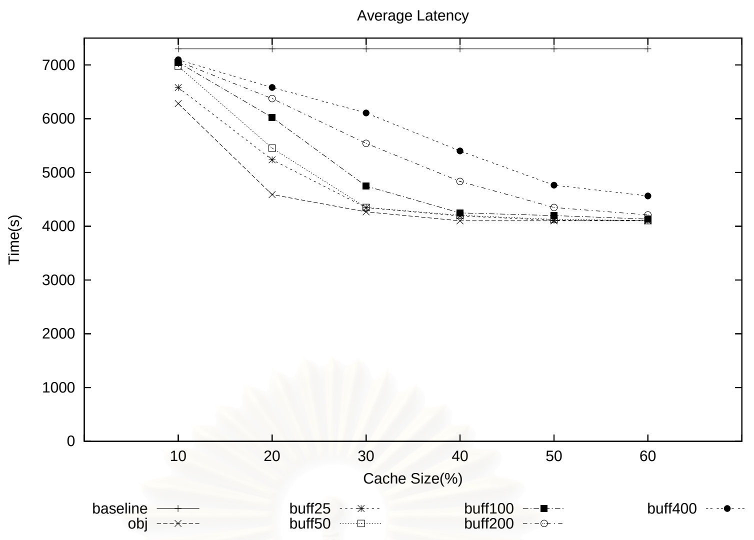
Table 5.2: The Block Type in The Scenarios

is the situation using object-based caching model.

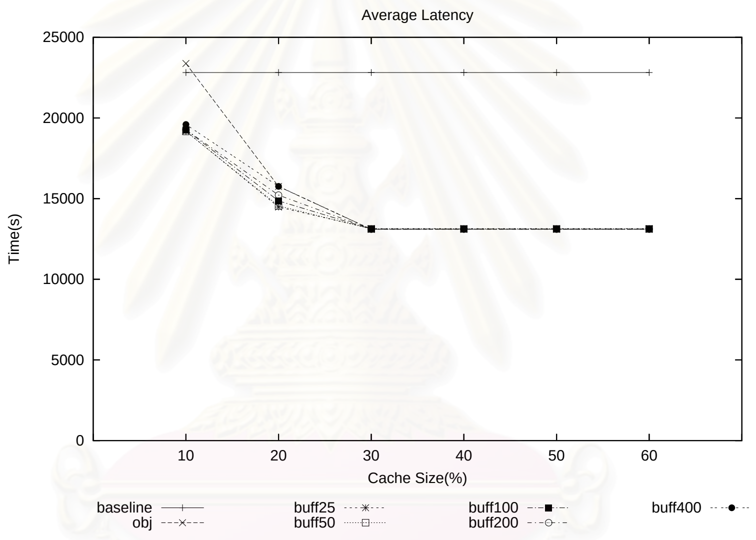
The result can be classified into several views. Figure 5.8 presents the average data accessing latency when we use different cache block sizes from trace no. 1 and 2 respectively. In addition, we compare the results with object-based data cache and the result when no data cache applied. Among the block-based data cache, the smaller block size can achieve better access latency against the larger ones. This is because given the same cache size, the bigger block-size, the smaller number of files the cache can keep. The smaller number of files a cache can keep, the lower the hit rates. Lower hit rate means higher cache misses occur. Thus, the average data accessing latency becomes worse.

When we compare the block-based caching with the object-based caching, however, there are very different characteristics between these two traces. In trace no.1, object-based data cache can outperform the block-based cache no matter what size they are. On the other hand, block-based data cache can overcome the object-based in trace no.2. This is because of the average data size of the data file using in each scenario. The first trace consists of many small data files (17MB average), whereas the other composes of large data files (500MB average). Small data files have 2 main advantages to object-based data cache.

1. It consumes less storage spaces which make data cache have more space to keep data file which lead to high hit ratio.
2. Object-based data cache spends lower connection overhead against the block-based. Figure 5.9 depicts the detailed latency of each data cache policy given the same byte hit ratio, namely optimal byte hit ratio. This graph shows that small block size causes more latency than large block size and object-based cache.



(a) Trace 1



(b) Trace 2

Figure 5.8: Average Latency on Different Block Size

The large data files do not enjoy the first advantage. This allows block-based data cache to overcome the object-based data cache as the bandwidth saving becomes more significant than the connection overhead.

Figure 5.10 and 5.11 depict the hit rates in the same scenario using trace no. 1 and 2. It can be stated that the cache hit rate becomes lower when the block size increases. This impacts the average latency. When hit rate decreases, the cache spends a lot of time to reload data from the servers. That is one of the main reasons that degrade the performance of the data cache. In trace no. 1, the object-based data cache get higher hit rate than the block-based because most data files in trace no.1 have

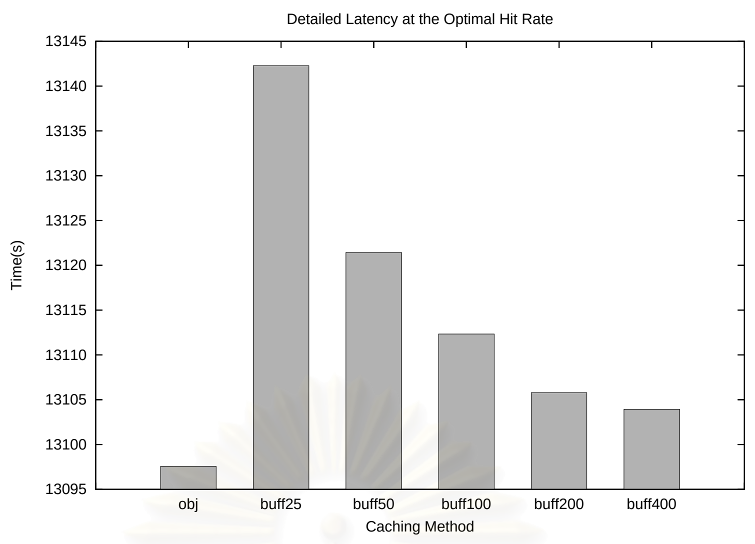
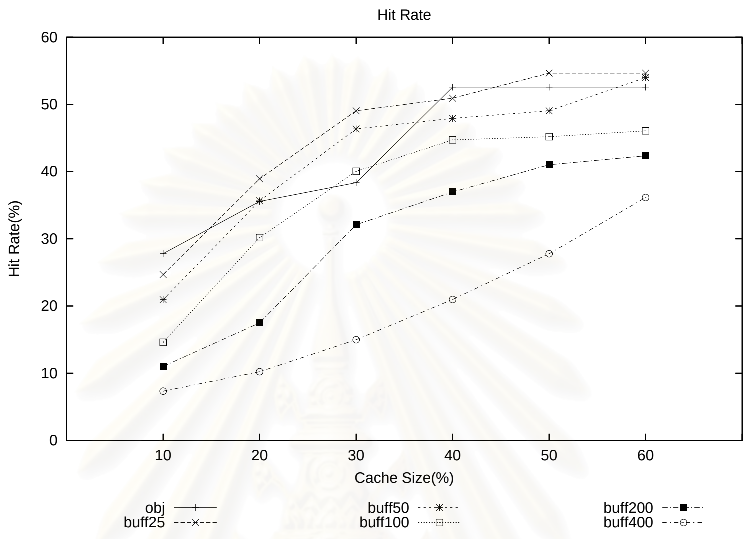


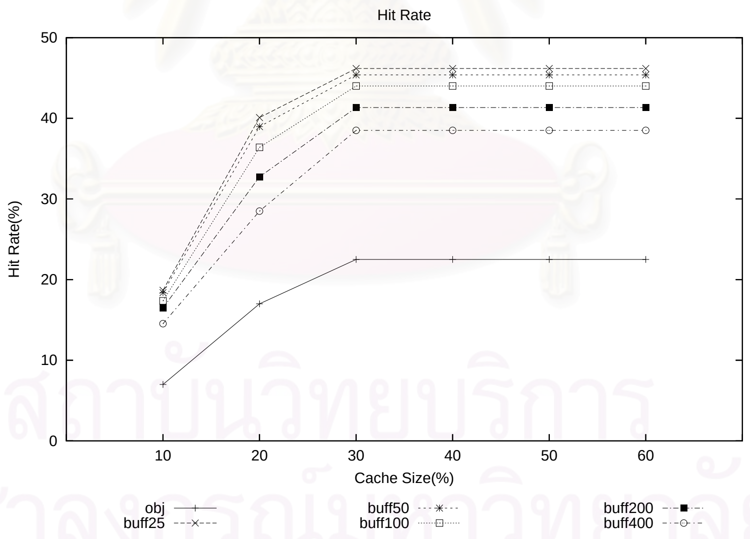
Figure 5.9: Detail Latency at the Optimal Hit Rate

small sizes. The smaller objects have more flexibility and “cache friendly” than large objects. Therefore, the average latency of object-based data cache can outperform the block-based data cache. In trace no. 2, however, the average sizes of the object are quite large. These make block-based data cache outperform object-based cache.

Figure 5.12 represents the first-byte latency which is the delay from a client issues a request until a client receives the first data from the cache server. The lower the first-byte latency, the sooner the clients can use data. This graph indicates that the cache with smaller block size have a better latency than biger block size. Having lower first-byute latency allows grid user to gain the benefits. Moreover, this can be extended to support additional data download policies. For example, we can select whether to use entire data download or block-based data download by calculating data usage threshold. If the data usage is below this value, the block-based policy, otherwise the entire download policy is used.

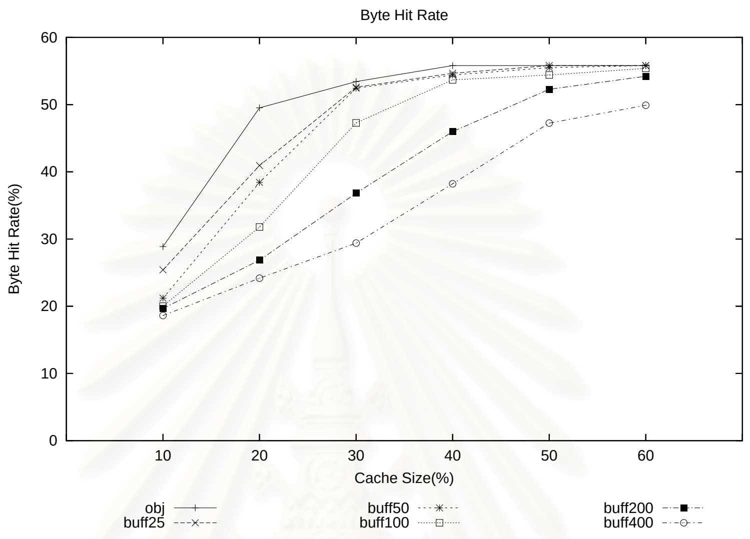


(a) Trace 1

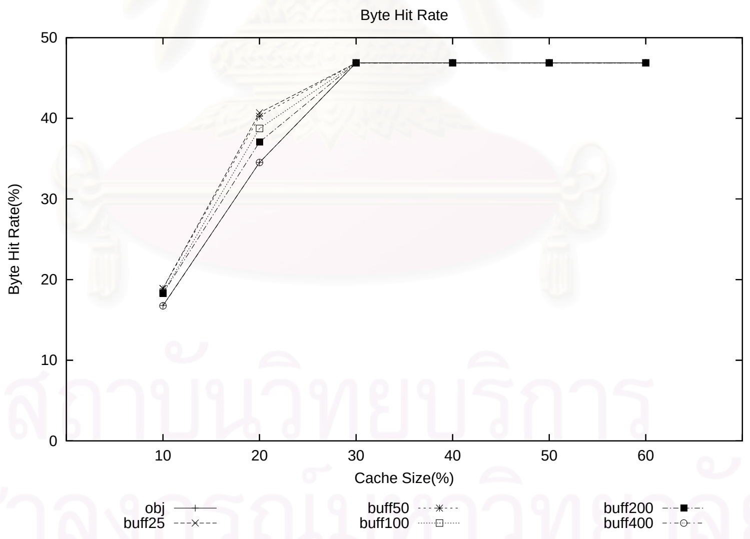


(b) Trace 2

Figure 5.10: Hit Ratio on Different Block Size

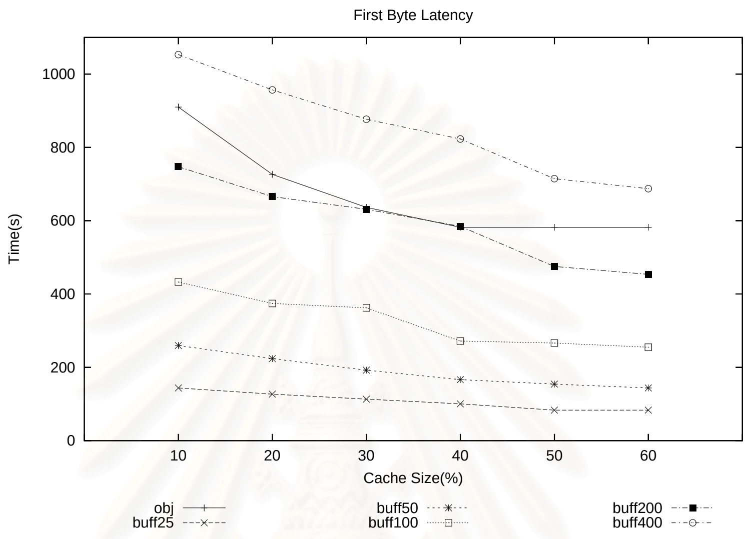


(a) Trace 1

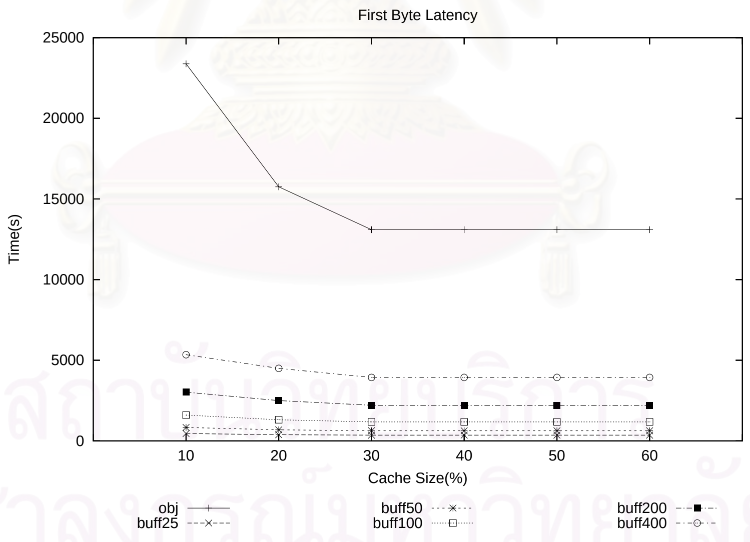


(b) Trace 2

Figure 5.11: Byte Hit Ratio on Different Block Size



(a) Trace 1



(b) Trace 2

Figure 5.12: First Byte Latency on Different Block Size

CHAPTER VI

CONCLUSION

6.1 Summary

This work proposes the block-based data caching model for the data grid environment. We argue that data cache is one of the essential components for data grid. We also show that the conventional object-based data cache model do not perform well in many cases, especially in the environment with large data transaction, or streaming data access behaviors. Therefore, we propose a new model to fit the characteristics of data grid environment.

Our proposed block-based model is derived from the conventional block-based data cache widely used in the cpu cache and file system. Our model segregates large data and stores them into multiple fixed-size data blocks. However, the block size must be large enough to avoid a lot of connection overhead and small enough to gain good storage space utilization.

We evaluate the performance of our proposed model by conducting simulation experiments. Our block-based data cache simulation framework is extended from the existing GridNet[27] Simulator. To make the evaluation become more realistic, the real data access traces JasMINE, at JLab[8] are used throughout the simulation. We conduct the experiments on the scenarios with various inter-domain bandwidths and the results clearly show that the block-based data cache enjoys good performance, especially in the small bandwidth environment. We also evaluate the performance of the data cache with different block sizes. The results indicate that different block sizes can affect the performance of the data grid. In small cache size, small data blocks can gain benefits from better utilization of the data cache storage. However, having too small block-size can suffer from connection overhead of either the disk connection, or network connection.

6.2 The Future Works

As our works have been focused on the basic mechanisms of caching, future works may introduce additional fine-tuning techniques adapting from conventional cache. For example, the studies of proper page replacement policies are needed. Data prefetching technique can also be applied to our caching model.

In addition, a variation of data block downloading can be performed. For example, if we can detect that the data requests want all data files, we can download all of them and separate them into the blocks later, instead of downloading one data block at a time.

References

1. National Science Foundation. GriPhyN Project. Available from: http://www.griphyn.org.

2. PPDG Collaboratory Pilot. PPDG. Available from: http://www.ppdg.net.

3. NASA. NASA Information Power Grid (IPG). Available from: http://www.ipg.nasa.gov.

4. European Union. Eu-DataGrid. Available from: http://eu-datagrid.web.cern.ch/eu-datagrid.

5. San Diego Supercomputer Center. Storage Resource Broker. Available from: http://www.npaci.edu/DICE/SRB.

6. The Globus Alliance. The Globus Toolkits. Available from: http://www.globus.org.

7. J. Wang. A survey of web caching schemes for the internet. ACM Computer Communication Review 29, 5(1999): 36-46.

8. E. Otoo, F. Olken, and A. Shoshani. Disk cache replacement algorithm for storage resource managers in data grids. In Proceedings of the 2002 ACM/IEEE conference on Supercomputing, pp. 1–15, Baltimore, Maryland. IEEE Computer Society Press, 2002.

9. S. Jiang and X. Zhang. Efficient distributed disk caching in data grid management. In IEEE International Conference on Cluster Computing(Cluster2003), pp. 446–451, 2003.

10. J. H. Abawajy. File replacement algorithm for storage resource managers in data grids. In International Conference on Computational Science 2004, pp. 339–346, 2004.

11. I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. International Journal Supercomputer Applications 15, 3(2001).

12. M. Baker, R. Buyya, and D. Laforenza. Grids and grid technologies for wide-area distributed computing. SOFTWARE -PRACTICE AND EXPERIENCE 32, 15(2002): 1437–1466.

13. Thailand E-Science. Thailand E-Science Project. Available from: http://www.thai-escience.net.

14. 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(2001): 187-200.

15. R. Oldfield. Summary of existing and developing data grids. March 2001. Available from: http://www.sdsc.edu/GridForum/RemoteData/Papers/paper.pdf.

16. 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, pp. 92–100. IEEE Computer Society, 2002.

17. OpenLDAP Foundation. OpenLDAP. Available from: http://www.openldap.org.

18. The Globus Aliance. GSI Documentation. Available from: http://www-unix.globus.org/toolkit/docs/3.2/gsi/index.html.

19. A. Balamash and M. Krunz. An overview of web caching replacement algorithms. IEEE Communications Surveys & Tutorials 6, 2(2004): 44-56.

20. P. Stefan and B. Laszlo. A survey of web cache replacement strategies. ACM Computing Surveys 35, 4(2003): 374-398.

21. N. N. Michael, B. W. Brent, and K. O. John. Caching in the sprite network file system. ACM Transactions on Computer Systems 6, 1(1988): 134-154.

22. A. Shoshani, A. Sim, and J. Gu. Storage resource managers: Middleware components for grid storage. In 19th IEEE Symposium on Mass Storage Systems (MSS'02), 2002.

23. A. Shoshani, A. Sim, and J. Gu. Storage resource managers:essential components for the grid. In J. Nabrzyski, J. Schopf, and J. Weglarz, editors, Grid resource management: State of the art and future trends. Kluwer Academic Publishers, 2003.

24. R. Buyya and M. Murshed. Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. The Journal of Concurrency and Computation: Practice and Experience (CCPE) 14, 13-15(2002): 1175-1220.

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

26. Information System Institute, University of Southern California. Network Simulator. Available from: http://nsnam.isi.edu/nsnam/index.php.

27. H. Lamehamedi, Z. Shentu, B. K. Szymanski, and E. Deelman. Simulation of dynamic data replication strategies in data grids. In 17th International Parallel and Distributed Processing Symposium (IPDPS 2003), pp. 100, 2003.

28. MONARC Members. Models of networked analysis at regional centres for lhc experiments: Phase 2 report. Available from: http://monarc.web.cern.ch/MONARC/docs/phase2report/Phase2Report.pdf, 2000.

29. R. Buyya. GridSim Simulator. Available from: http://www.buyya.com/gridsim.

30. European DataGrid Work Package 2. OptorSim Simulator. Available from: http://edg-wp2.web.cern.ch/edg-wp2/optimization/optorsim.html.

31. K. Ranganathan and I. T. Foster. Identifying dynamic replication strategies for a high-performance data grid. In GRID '01: Proceedings of the Second International Workshop on Grid Computing, pp. 75–86. Springer-Verlag, 2001.

32. Broadband Network Report. Broadband Network Report SpeedTest. Available from: http://www.musen-lan.com/speed/.

33. I. Bird, B. Hess, and A. Kowalski. Building the mass storage system at jefferson lab. In The 18th IEEE Symposium on Mass Storage Systems, San Diego, California. 2001.

APPENDICES

APPENDIX A

THE IMPLEMENTATION OF SIMULATION FRAMEWORK

The data grid component for NS-2 are extended from GridNet Simulator as described in Chapter 4. In this chapter, the detailed implementations of the data grid component for NS-2 are represented.

A.1 Data Grid Component Overview

Figure A.1 depicts the Component implemented for data grid environment for NS-2. The brief description of each component can be described as follow:

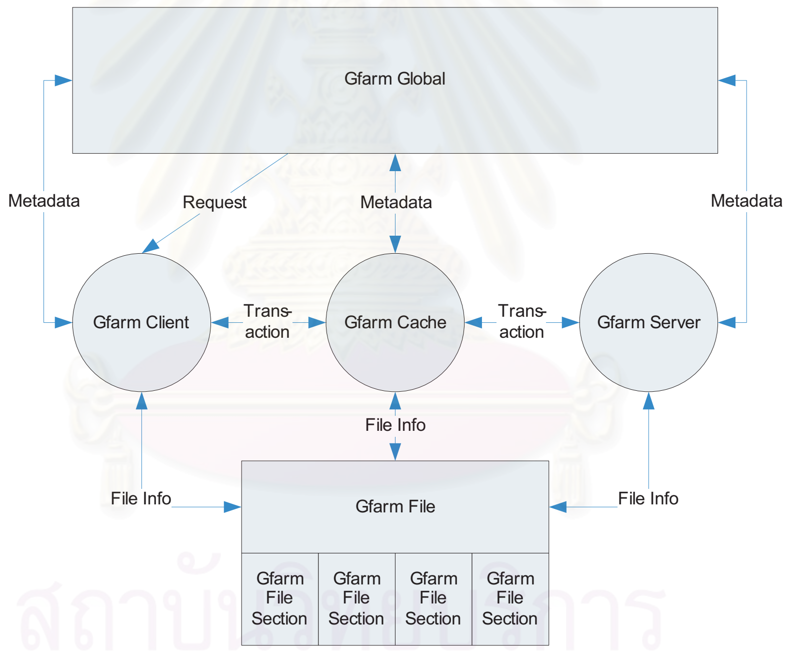


Figure A.1: Simulator Component Overview

1. *GFarm Global Component* manipulates the system in the initial phrase and behaves like users who send request information to data grid client. It consists of many utility functions to store the environment property including file mapping, agent list, and node connection.

2. *GFarm File* represents the file object in Grid DataFarm. Extended from original GridNet File, It consists of filename and many file section. Therefore, the user

can resolve file using filename instead of file id as in GridNet. Furthermore, the data file can be divided into many fragments and store in different GFarm Server.

3. *GFarm Server* behaves as if it is a file storage. Because the evaluations are deployed for caching evaluation, we set the infinite space for server storage.

4. *GFarm Client* is triggered by GFarm Global to perform data downloading.

5. *GFarm Cache* deals with data request from GFarm Client and data connection from GFarm Server if needs. It is a component which can be flexibly extended because of the use of OO model.

A.2 GridNet Modification

To accomplish the requirement, new components are implemented. The modification of the original framework, GridNet, has to be realized for the good simulation. There are many reasons to patch GridNet. First, GridNet does not support for large-scale data handling. The data structure to handle data support only about 2 GB, whereas, the overall data usages in the scenarios are larger, up to Terabyte. Second, the protocol inside grid packet object has to be extended for remote data access behavior. The information such as file pointer, block to read, have to be passed to cache or server nodes to make data connection. Finally, the topology has to be flexible enough to establish connection applied to the scenarios.

A.2.1 Grid Packet Protocol

GridNet provides grid protocol header to communicate between nodes. The grid protocol in the simulator consists of grid protocol command and the data parameter. The grid protocol command composes of 3 main actions: GRID_PROTOCOL_WRITE, GRID_PROTOCOL_READ, and GRID_PROTOCOL_PROBE. However, this thesis focused on read process only. The GRID_PROTOCOL_READ have 2 states:

1. GRID_READ_REQUEST sends data request with request information.

2. GRID_READ_RESPONSE returns response information including data.

During the node connection, the transfer parameters are enclosed together with grid protocol command. This information indicates what data to be picked up, how many block to be transfer and who request them. The transfer parameters are described as follow:

1. *Filename* The filename to be requested by clients.
2. *File Section No.* This specify which file fragment to be requested.
3. *File Block No.* Address in the file to be requested.
4. *Sender* The node that make data request.
5. *Buffer Read* The Amount of Buffer to be read from server.
6. *Task ID* This is an unique number which maintain the process consistency betweeen nodes.

In data request process, all of parameters are set to send data information to server. However, when servers perform response process, the data requested from client are attached with the grid protocol. Figure A.2 represent the connection model between nodes.

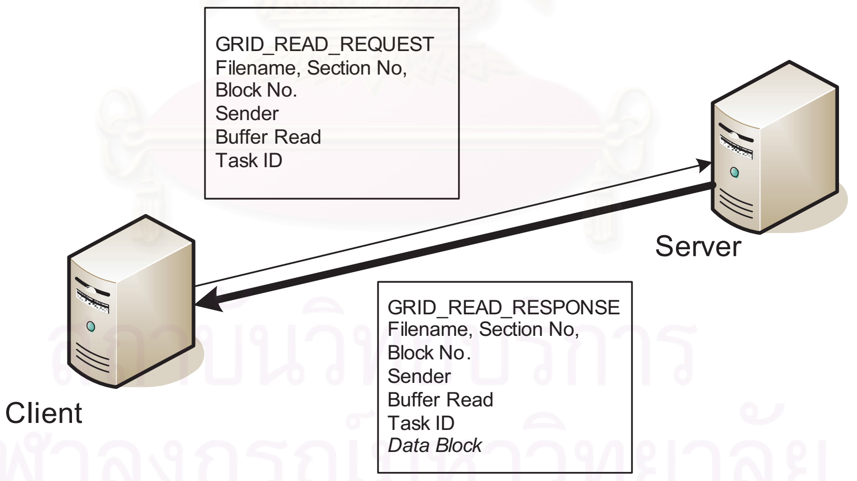


Figure A.2: Grid Connection Protocol

A.2.2 Data Access Mechanism

Data access inside each node, contrary to the network access, does not have pre-built latency simulation. Therefore, we have to model them to perceive the better simulation. Inside this simulation framework, we implemented the trivial process to

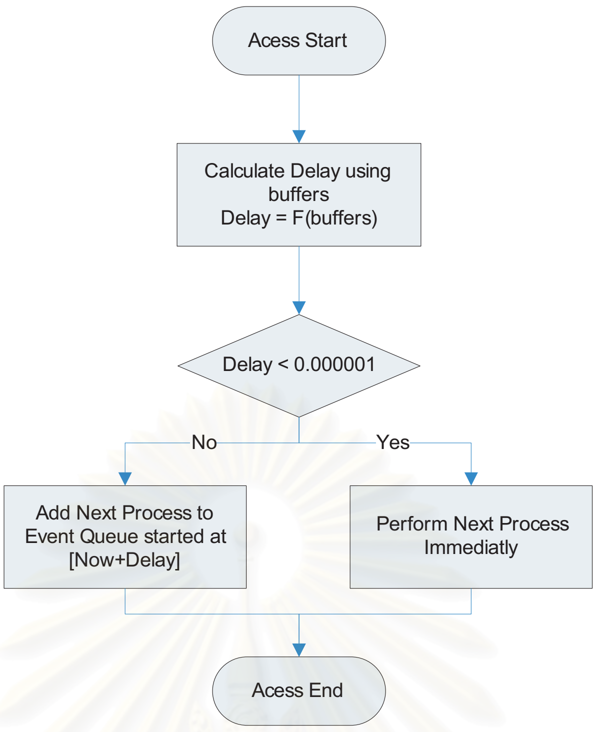

Acess Start
Calculate Delay using buffers
Delay = F(buffers)
Delay < 0.000001
No
Yes
Add Next Process to Event Queue started at [Now+Delay]
Perform Next Process Immediatly
Acess End

Figure A.3: Access Latency Simulation Process

retrieve disk access latency. Figure A.3 presents the data access delay process, which is quite straightforward. When data access occurs, the file pointers will be moved by sizes of buffer. Next, the delay time which is the function of buffer size is calculated. Because of the limitation on the precision of time event, the delay interval which is less than 0.000001 have no significant. Therefore, the next process will be done immediately. However, the next process will be sent to event queue if the delay intervals are more than 0.000001.

A.3 GFarm Component Implementation

A.3.1 GFarm Client

GFarm Client Component provides functions to make data request and emulate the data read/write behavior. Most of client event take place from Tcl functions listed below:

- *gfarm-request* begins data request.
- *set-proxy* sets client node whether to connect to cache server.
- *sendData* establishes data connection.

To manage sessions for each connection, the descriptor class is implemented to serve every request states. It stores file access information including filename, current access position, and fragment number. GFarm client map the request number to the right descriptor for session consistency.

The process of GFarm client is started when Tcl command "gfarm-request" invoked. GFarm client establishes request connection to cache server or data server, which depends on the client setting. When the connection is established, the GFarm client waits for request response from servers. After getting response, client updates file descriptor according to the request number and perform latency emulation. If the data are not fetched completely, the client resend the request to get the remaining data, or else the request job is finished.

A.3.2 GFarm Server

In the simulation, GFarm Server Component provides only the data response to client or cache node. Therefore, it has simple mechanisms to response to the client requests.

When client or cache nodes send grid read protocol with read request flag, The GFarm server read the data inside grid packet and opens corresponding file descriptor to begin data read. After file reading, all of the information including metadata and raw data are encapsulated into grid packet and return to clients.

A.3.3 GFarm Cache

GFarm Cache is the main component for data grid performance evaluation. Inside GFarm cache object, there are two components to facilitate cache storage management. The BlockData hold the data block information; it compose of file information stored in that blocks and the cost value which is a heuristic value to determine whether that blocks should be preserved for future use. The BlockData have to be extended in order to apply for each cache replacement scenario. The Cache Manager deals with the operation for cache policies. It allows researchers to implement new cache policy by define new block look up, store data block, and evict block data. The example usages of these two classes are the LRU Cache Manager and DataBlock which implements the LRU Cache Policy. Many GFarm cache operation can be invoked and initialized with Tcl Function. The Tcl methods are listed in Table A.1.

Function Name	Description
set-buffersize	set data size for each data block
set-cachesize	set storage size for cache server
get-cachesize	get storage size
get-buffersize	get data block size
sendData	establish connection and return data back to clients
sendRequest	establish connection and forward request to servers
response	start data cache process
set-concurrency	set maximum concurrency process
set-waittime	set polling time if the process have to be waited by scheduler

Table A.1: Tcl Method inside GFarm Cache

The process of data cache starts when the packet from client arrived. There are two types of incoming packet: the startup requests which are the requests that reach the cache server for the first time, which have to be determine availability status, and the established requests which are the requests that have been determined the status already and performed data downloading. If the packet is established request, the cache server brings it to its current download status without any decisions. However, if the packet is a startup request, the availability status has to be determined. The cache manager looks up the data blocks to find desired data. If cache hit occurs, the cost value for that data block has been updated and the data become ready to return to a client. However, if cache miss occurs, a cache server forwards request to servers and wait for data return. When data are return completely, the data block updates its cost value and become ready. The GFarm cache, then, returns data to clients. When clients receive data, it will send established requests to cache server if the data are not complete which cause the cache server return the remaining back until the transfers are completed.

APPENDIX B

EXPERIMENTAL RESULT IN DETAIL

B.1 Result on Bandwidth

	CacheSize					
Bandwidth	10	20	30	40	50	60
10-1	54724.11	38022.41	29335.45	28385.40	28633.25	28382.25
50-5	7626.13	6356.52	4777.48	4271.38	4220.18	4153.40
100-10	7046.83	6024.39	4745.93	4248.84	4198.95	4132.26
500-50	7035.72	6014.89	4738.91	4242.67	4192.82	4126.24
1000-100	7034.68	6014.00	4738.16	4241.99	4192.15	4125.57

Table B.1: Average Latency for Bandwidth

	CacheSize					
Bandwidth	10	20	30	40	50	60
10-1	26.43	41.23	45.85	46.26	46.26	46.38
50-5	14.61	30.26	40.04	44.72	45.19	46.06
100-10	14.61	30.18	40.04	44.72	45.19	46.06
500-50	14.61	30.18	40.04	44.72	∗45.19	46.06
1000-100	14.61	30.18	40.04	44.72	45.19	46.06

Table B.2: Hit Rate for Bandwidth

	CacheSize					
Bandwidth	10	20	30	40	50	60
10-1	37.54	48.85	55.10	55.51	55.51	55.51
50-5	20.02	31.92	47.28	53.69	54.41	55.39
100-10	20.02	31.79	47.28	53.69	54.41	55.39
500-50	20.02	31.79	47.28	53.69	54.41	55.39
1000-100	20.02	31.79	47.28	53.69	54.41	55.39

Table B.3: Byte Hit Rate for Bandwidth

Bandwidth	CacheSize					
	10	20	30	40	50	60
10-1	51.81	36.00	27.77	26.87	27.11	26.87
50-5	81.90	68.26	51.31	45.87	45.32	44.60
100-10	96.54	82.53	65.01	58.21	57.52	56.61
500-50	96.87	82.81	65.24	58.41	57.73	56.81
1000-100	96.87	82.81	65.25	58.41	57.73	56.81

Table B.4: Percent of Latency Improvement for Bandwidth

B.2 Result on Buffer Size

B.2.1 Trace Log 1

Buffer Size	CacheSize					
	10	20	30	40	50	60
full	910.11	726.24	636.57	582.04	582.04	582.04
25	143.62	126.84	113.30	100.36	83.19	83.19
50	259.49	223.84	192.23	166.48	154.36	143.71
100	432.87	374.09	362.23	271.76	266.28	254.85
200	747.21	665.94	631.29	584.53	475.20	453.72
400	1053.21	956.88	876.88	822.97	714.81	687.28

Table B.5: First Byte Latency for Buffer Size

Buffer Size	CacheSize					
	10	20	30	40	50	60
full	27.80	35.55	38.35	52.57	52.57	52.57
25	24.69	38.92	49.04	50.92	54.65	54.65
50	20.96	35.65	46.35	47.92	49.04	53.98
100	14.61	30.18	40.04	44.72	45.19	46.06
200	11.01	17.49	32.10	36.97	41.01	42.35
400	7.34	10.22	14.99	20.97	27.77	36.15

Table B.6: Hit Rate for Buffer Size

Buffer Size	CacheSize					
	10	20	30	40	50	60
full	28.87	49.52	53.43	55.81	55.81	55.81
25	25.42	40.94	52.58	54.67	55.81	55.81
50	21.20	38.43	52.44	54.41	55.51	55.81
100	20.02	31.79	47.28	53.69	54.41	55.39
200	19.67	26.85	36.83	45.99	52.28	54.23
400	18.62	24.18	29.39	38.22	47.25	49.91

Table B.7: Byte Hit Rate for Buffer Size

Buffer Size	CacheSize					
	10	20	30	40	50	60
baseline	7299.74	7299.74	7299.74	7299.74	7299.74	7299.74
full	6282.53	4588.41	4268.72	4100.95	4100.95	4100.95
25	6578.97	5236.29	4345.30	4190.35	4111.96	4111.94
50	6978.26	5452.03	4349.74	4202.09	4126.96	4105.85
100	7046.83	6024.39	4745.93	4248.84	4198.95	4132.26
200	7065.87	6375.60	5541.71	4832.77	4348.58	4210.55
400	7098.08	6581.84	6107.19	5401.74	4764.78	4563.62

Table B.8: Average Latency for Buffer Size

B.2.2 Trace Log 2

Buffer Size	CacheSize					
	10	20	30	40	50	60
full	23378.80	15754.24	13098.18	13097.92	13097.64	13097.56
25	456.64	385.44	355.46	355.45	355.45	355.44
50	831.11	688.87	628.97	628.96	628.95	628.95
100	1599.60	1309.27	1174.58	1174.55	1174.54	1174.53
200	3022.29	2503.30	2209.22	2209.18	2209.13	2209.12
400	5347.15	4499.08	3932.82	3932.78	3932.65	3932.61

Table B.9: First Byte Latency for Buffer Size

	CacheSize					
Buffer Size	10	20	30	40	50	60
full	7.00	17.00	22.50	22.50	22.50	22.50
25	18.68	40.09	46.18	46.18	46.18	46.18
50	18.38	38.97	45.39	45.39	45.39	45.39
100	17.37	36.40	44.01	44.01	44.01	44.01
200	16.49	32.74	41.34	41.34	41.34	41.34
400	14.54	28.49	38.51	38.51	38.51	38.51

Table B.10: Hit Rate for Buffer Size

	CacheSize					
Buffer Size	10	20	30	40	50	60
full	16.79	34.52	46.88	46.88	46.88	46.88
25	18.91	40.69	46.88	46.88	46.88	46.88
50	18.87	40.25	46.88	46.88	46.88	46.88
100	18.35	38.75	46.88	46.88	46.88	46.88
200	18.28	37.05	46.88	46.88	46.88	46.88
400	16.75	34.52	46.88	46.88	46.88	46.88

Table B.11: Byte Hit Rate for Buffer Size

	CacheSize					
Buffer Size	10	20	30	40	50	60
baseline	22814.21	22814.21	22814.21	22814.21	22814.21	22814.21
full	23378.80	15754.24	13098.18	13097.92	13097.64	13097.56
25	19189.32	14478.30	13143.07	13142.70	13142.38	13142.27
50	19166.45	14548.74	13122.01	13121.75	13121.54	13121.43
100	19263.98	14859.81	13112.85	13112.67	13112.47	13112.33
200	19270.79	15219.09	13106.71	13106.37	13106.02	13105.79
400	19596.62	15760.96	13104.63	13104.37	13104.13	13103.93

Table B.12: Average Latency for Buffer Size

Biography

Teerayut Hiruntaraporn was born in Bangkok, Thailand, on October, 1981. He received Bachelor Degree of Engineering from Chulalongkorn University since April, 2003. His field of study is Computer Engineering. His main interest is in the area of networking and algorithm. His current research is the performance evaluation for data cache in data grid environment.