私立東海大學

資訊工程與科學研究所

碩士論文

指導教授：楊朝棟　博士

強化動態預測調整機制於資料網格中之平行檔案傳輸

The Enhancement of Anticipative Recursively Adjusting Mechanism for Redundant Parallel File Transfer in Data Grids

研　究　生：楊明峰

中華民國九十八年六月

摘要

在資料網格環境中，資料集被重複製為複本且分送到多重的站台。由於資料集的檔案通常很大，如何有效率的存取及傳輸成為重大的課題。因此先前有學者發展出協同配置的架構，使得同時從多重站台平行下載資料變成可能，目前發展出數種協同配置法用來解決傳輸時本地端與伺服端網路傳輸率不斷變動的問題。例如將欲傳輸的檔案切割成數個均等的檔案大小，或是將檔案動態的切割置於工作佇列，透過連線品質較佳者傳送佇列中未完成傳輸的檔案區塊，來解決網路變動的問題。無論各個下載連線的效率為何，當伺服端傳送最後一個檔案區塊時，發生速度快的伺服器閒置的等待最慢的伺服器完成最後一個檔案區塊的傳送，或是因為不同伺服器傳送相同的檔案區塊，造成網路頻寬資源的浪費，因此，若能在一群候選伺服器中找到最大的頻寬資源，有效分配工作減少各伺服器間完成傳輸時間的差異，將成為最重要的工作。近年，在全球各地的學者先後提出頗具新意的資料網格平行檔案傳輸策略；本研究中，匯集 8 種各具代表性的平行檔案傳輸方法，融合各方法的優點改善其缺點，採用 TCP 頻寬估計模型與突發模式等新策略，藉此強化預測性遞迴調整的協同配置法，進一步提高大量資料集於資料網格中的傳輸效能。我們的方法能有效地找出一群快速伺服器並分配較多的工作量提高其資源利用率，動態計算出檔案切割量，有效減少各伺服器間的相互等待時間。藉由各項實驗證明其傳輸的高效能，並具有網路自適應性與高度容錯性，有效因應不同的網格環境。

關鍵字： 資料網格、協同配置、動態協同配置、平行傳輸

Abstract

Data grid enable the sharing, selection, and connection of a wide variety of geographically distributed computational and storage resources for content that the large-scale data-intensive application needs, such as high-energy physics, bioinformatics, and virtual astrophysical observatories. Data grid consists of scattered computing and storage resources located in different regions yet accessible to users. Co-allocation architectures can be used to enable parallel transfers of data file from multiple replicas in data grids which are stored at different grid sites. Schemes based on co-allocation models have been proposed and used to exploit the different transfer rates among various client-server network links and to adapt to dynamic rate fluctuations by dividing data into fragments. These schemes show that the more fragments used the more performance. In fact, some schemes can be applied to specific situations; however, most situations are not common actually. For example, how many blocks in a data set should be cut? For this issue, we proposed the anticipative recursively adjusting mechanism (ARAM) in a previous research work. Its best feature is performance tuning through alpha value adjustment. It relies on special features to adapt to various network situations in data grid environments. In this thesis, the TCP Bandwidth Estimation Model (TCPBEM) is used to evaluate dynamic link states by detecting TCP throughputs and packet lost rates between grid nodes. We integrated the model into ARAM, calling the result the anticipative recursively adjusting mechanism plus (ARAM+); it can be more reliable and reasonable than its predecessor. We also designed a Burst Mode (BM) that increases ARAM+ transfer rates. This approach not only adapts to the worst network links, but also speeds up overall performance.

Keywords: Data Grid, Co-allocation, Dynamic Co-allocation, Parallel file transfer

Acknowledgements

研究所 2 年時光匆匆，回想當初決定報考東海大學到選擇加入高效能計算實驗室，我首先要感謝指導教授—楊朝棟 博士，在學期間給予我最多耐心且最大支持與最真誠的鼓勵，無論是參與各項比賽與國際論文發表，研究的過程中充滿挑戰且富有成就感，有幸能成為楊老師實驗室的一員，回想起過程就好比男人當兵 2 個寒暑，從男孩蛻變男人，在東海大學高效能計算實驗室培養出的韌性足以應付未來更多的挑戰。

接著要感謝我的太太雯如與大女兒佳恩，沒有他們的體諒與支持，我將無法順利完成論文，更遑論畢業取得學位。

最後，特別感激修平技術學院的長官 姜文忠博士與東海大學研究所的同學們，志豪、秋雄與共同合作的學長學弟們，沒有你們的指導與無私付出，我將不可能完成我的研究工作，謝謝你們大家，並且再一次感謝 楊朝棟老師的指導。

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

An increasing number of scientific applications. e.g., arising from Genomics, Proteomics, and Bioinformatics require exchanges of large volumes of data to support computation. Downloading large data sets from replica locations may result in different performance rates because replica sites may have different architectures, system loading, and network connectivity. Bandwidth quality is the most important factor affecting internet transfers between clients and servers, with download speeds being bounded by traffic congestion due to bandwidth limitations.

One method for improving download speeds uses replica selection techniques to determine the best replica locations [28]. However, downloading data sets from single best servers often results in ordinary transfer rates because bandwidth quality varies unpredictably due to the shared nature of the Internet.

Another method uses co-allocation [27] technology to download data. Co-allocation architectures were developed to enable clients to download data from multiple locations by establishing multiple connections in parallel, thus improving performance over single-server transfers and helping to alleviate the internet congestion problem [27]. Parallel downloading [23, 24, 25, 26] is a technique used to fetch and download files from multiple sources including Web servers, file servers, P2P nodes, etc. Parallel downloading has been integrated into many Internet applications and has become the core of many P2P systems. It speeds up download times and eliminates the server selection problem [21, 22, 23]. Several co-allocation strategies were addressed in previous works [15, 27], but drawbacks remain, such as faster servers having to wait for the slowest one to deliver its final block. As shown in

[15, 16], this may degrade network performance by repeatedly transferring the same block. Hence, it is important to minimize differences in finish times among servers, and to prevent the same blocks from being transferred over different links between servers and clients.

In our previous research work, we presented a method for regulating next-section workloads by continuously adjusting the workloads on selected replica servers. The anticipative recursively adjusting mechanism (ARAM) scheme measures the actual bandwidth performance during data file transfers, and, according to previous transfer finish rates, anticipates bandwidth statuses at the next transfer section. The basic idea is to assign less data to selected replica servers with greater network link performance variations since links with more bandwidth variations will have smaller effective bandwidths, as well as smaller transfer finish rates. The goal is to make the expected finish times of all servers be the same.

1.2 Contribution

In this thesis, we first present our new approach based on the ARAM co-allocation strategy for data grid environments. We have designed and implemented a TCP bandwidth estimation model and Burst Mode (BM) to enhancing the original ARAM algorithm. Workloads on all selected replica servers are still adjusted according to TCP throughputs and packet loss rates, and faster servers get double or even quadruple throughputs via Burst Mode enabling. Finally, we present Cyber Transformer, a useful toolkit for data grid users. Integrated with the Information Service, Replica Location Service, and Data Transfer Service, it’s simple, friendly GUI interface makes it easy for inexperienced users to manage replicas and download files in data grid environments. This tool integrates all strategies based on co-allocation architectures including our previous and proposed algorithms.

1.3 Thesis Organization

The remainder of this research is organized as follows. Related background review and studies are presented in Chapter 2. Our new approach is outlined in Chapter 3. Experimental results and a performance evaluation of our scheme are presented in Chapter 4. Chapter 5 concludes this research article.

Chapter 2

Background Review and Related Work

2.1 Co-allocation Architecture

The architecture proposed in [29] consists of three main components: an information service, a broker/co-allocator, and local storage systems. Figure 2.1 shows co-allocation of data grid transfers, an extension of the basic template for resource management [7] provided by the Globus Toolkit. Applications specify the characteristics of desired data and pass attribute descriptions to a broker. The broker queries available resources, gets replica locations from the Information Service [6] and Replica Management Service [31], then gets lists of physical file locations.

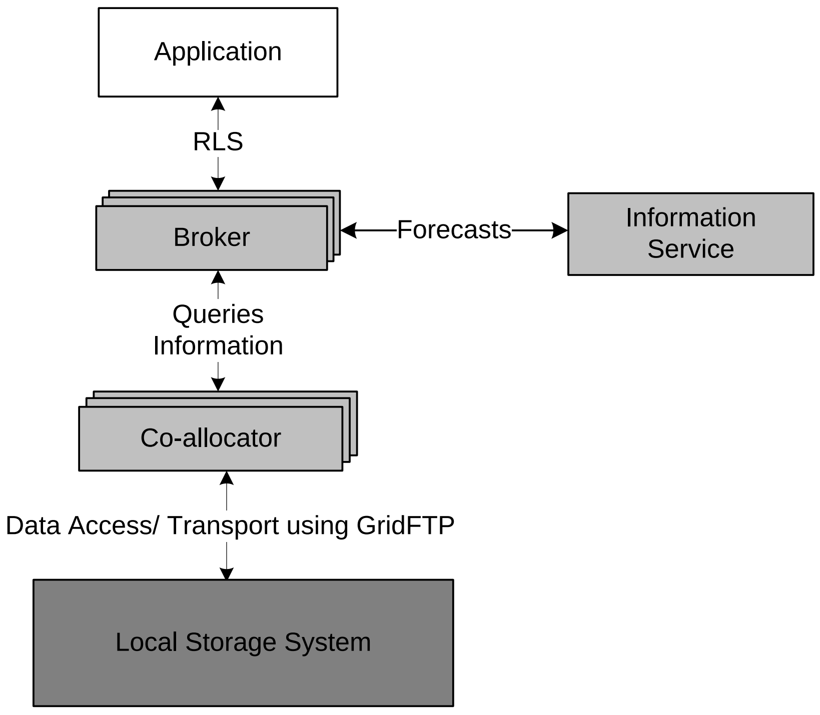


Figure 2.1: Data Grid Co-Allocation Architecture.

2.2 Brute-Force Co-Allocation

The Brute-force co-allocation scheme shown in Figure 2.2 divides file sizes equally among available flows; it does not address bandwidth differences among various client-server links.

2.3 History-Based Co-Allocation

The history-based co-allocation scheme shown in Figure 2.3 keeps block sizes per flow proportional to predicted transfer rates, and disregards the influence of network variations between client and server.

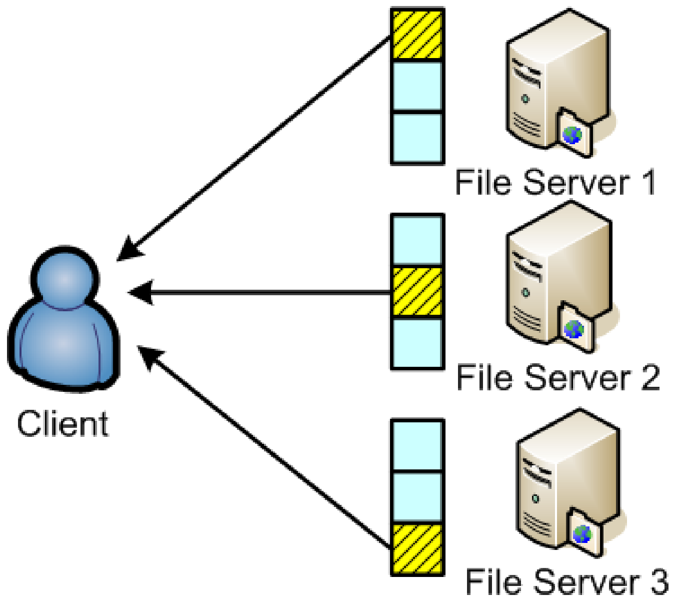


Figure 2.2: The Brute-Force Co-Allocation Process.

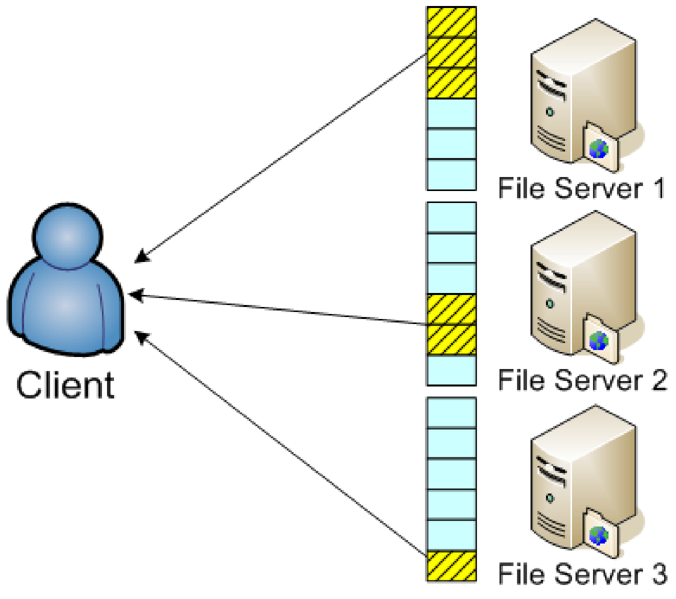


Figure 2.3: The History-Based Co-Allocation Process.

2.4 Conservative Load Balancing

The conservative load balancing scheme shown in Figure 2.4 divides requested data sets into *k* disjoint blocks of equal size. Available servers are allocated single blocks to deliver in parallel. Servers work in sequential order until all requested files are downloaded. Loadings on the co-allocated flows are automatically adjusted because the faster servers deliver larger file portions more quickly.

2.5 Aggressive Load Balancing

This method, shown in Figure 2.5, adds functions that change block size in deliveries by: (1) gradually increasing the amounts of data requested from faster servers and (2) reducing the amounts of data requested from slower servers or stopping requesting data from them altogether.

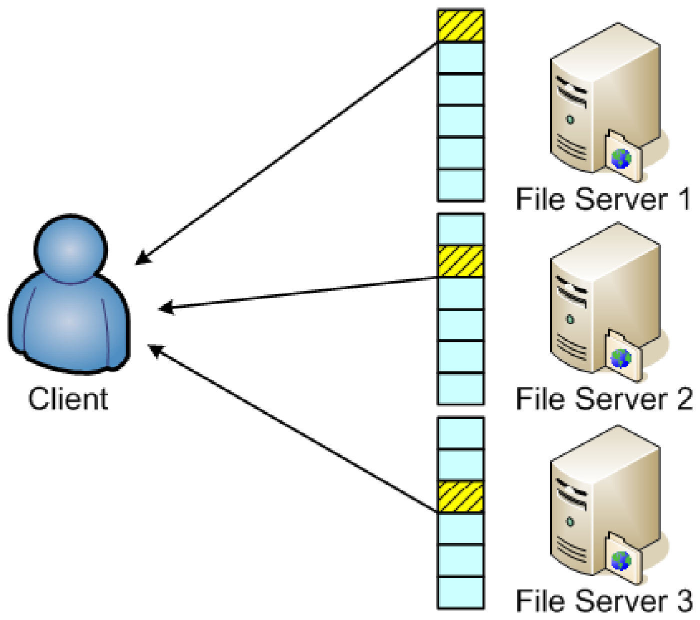


Figure 2.4: The Conservative Load Balancing Process.

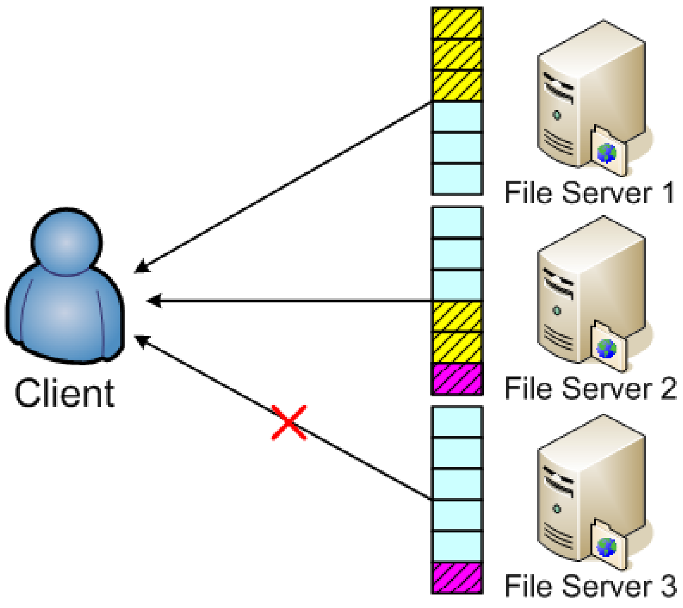


Figure 2.5: The Aggressive Load Balancing Process.

2.6 Dynamic Co-Allocation with Duplicate Assignments (DCDA)

The co-allocation strategies described above do not handle the shortcoming of faster servers having to wait for the slowest server to deliver its final block which, in most cases, wastes much time and decreases overall performance. Neither the prediction nor the heuristic approach, the DCDA scheme dynamically co-allocates duplicate assignments and copes nicely with changes in server speed performance, as shown in Figure 2.6. The DCDA scheme is based on an algorithm that uses a circular queue. Let *D* be a data set and k the number of blocks of fixed size in the data set. *D* is divided into k disjoint blocks of equal size and all available servers are assigned to deliver blocks in parallel. When a requested block is received from a server, one of the unassigned blocks is assigned to that server. The co-allocator repeats this process until all blocks have been assigned. DCDA behaves well even when server links are broken or idled. The DCDA scheme is flawed, however, in that it consumes network bandwidth by repeatedly transferring the same blocks. This wastes resources and can easily cause bandwidth traffic jams in the links between servers and clients.

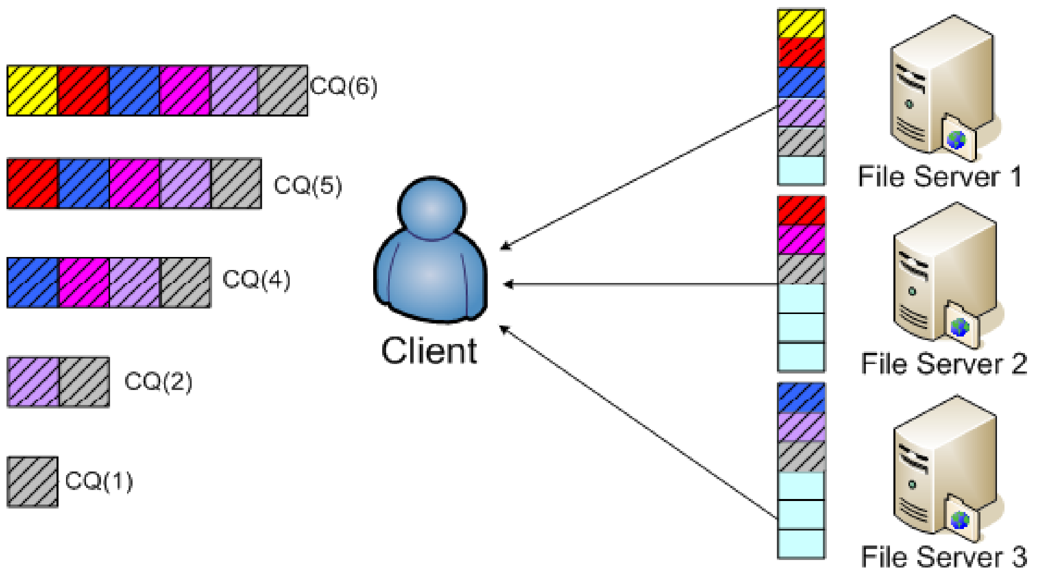


Figure 2.6: The DCDA Process

2.7 Recursively Adjusting Mechanism (RAM)

This co-allocation strategy is the most efficient approach to reducing the influence of network variations between clients and servers. However, idle times when faster servers are waiting for the slowest server to deliver its last block are still a major factor affecting overall efficiency that conservative load balancing and aggressive load balancing [18] cannot effectively avoid. In real-world networking environments, a replica server's available bandwidth might change dynamically as a result of network configuration or load variations. Previous algorithms could not adapt to these dynamisms. Therefore, the greater the degree of bandwidth variation the greater the download times needed. Thus, overall efficiency depends on several factors. Our strategy can overcome such obstacles, and improve data transfer performance. The recursively adjusting mechanism works by continuously adjusting each replica server's workload to correspond to its real-time bandwidth during file transfers. The goal is to make the expected finish times of all servers the same. As Figure 2.7 shows, when an appropriate file section is first selected, it is divided into proper block sizes according to the respective server bandwidths. The co-allocator then assigns blocks to servers for transfer. At this moment, it is expected that the transfer finish times will be consistent at $E(t_1)$. However, since server bandwidths may fluctuate during segment deliveries, actual completion times may vary (solid line, in Figure 2.7). When the quickest server finishes its work at time t_1, the next section is assigned to the servers. This allows each server to finish its assigned workload by the expected time at $E(t_2)$. These adjustments are repeated until the entire file transfer is finished.

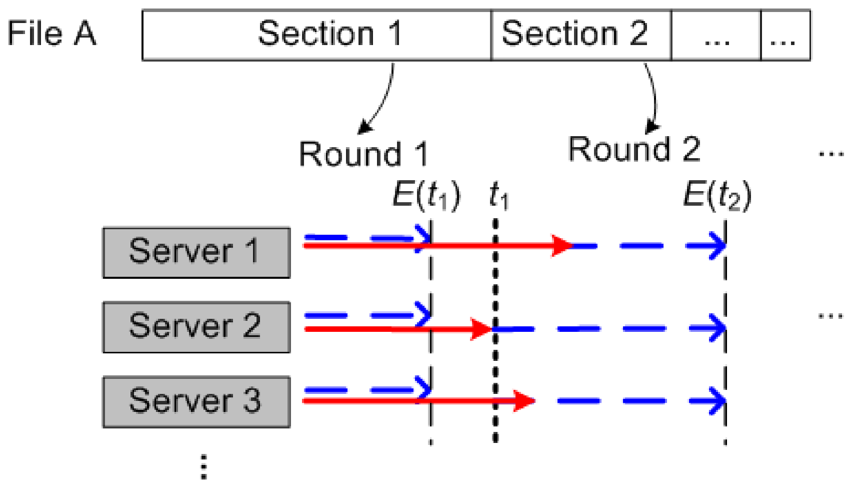


Figure 2.7: The Adjustment Process

The main purpose of this algorithm is to select appropriate data sources and download from multiple data servers to a single client resource. We proposed a recursively adjusting co-allocation scheme for parallel downloads from multiple replica servers to a single-client. This is useful in cases like downloading music file segments and playing continuous music on a single-client resource. Our algorithms are mainly aimed at transferring parallel data segments from multiple servers to multiple clients for execution of parallel numerical applications on the clients. The challenge in multiple server–multiple client scenarios is greater since server selections and data downloads on some clients can impact server selections and data transfer performance on other clients.

2.8 Dynamic Adjustment Strategy (DAS)

The DAS proposed a replica selection cost model and a replica selection service to perform replica selection, its revival the CPU load, memory usage and free space through Replica Location Service (RLS). We now propose a new data transfer strategy based on this model. It consists of three phases: (1) initial phase, (2) steady phase, and (3) completion phase.

- Initial phase: We assign equal block sizes to all GridFTP servers. In this phase, our system determines the next block size for each replica server.
- Steady phase: As job transfers are completed, servers are assigned their next jobs. Jobs sizes are determined by multiplying the client bandwidth by the weighting.
- Completion phase: To avoid the generating excessively small job sizes, we set an end condition such that if the remaining target file size is smaller than the initial block size, it is transferred immediately.

To determine the initial block size, we set an upper bound that is dependent on the relation between the client's maximum bandwidth and the number of replica sources. Though multiple replicas can be downloaded in parallel, the gathered portions of files from different links must be transferred to the client in a single link. It is clear that the client's bandwidth could be bottleneck in co-allocation architecture. Each time, our strategy dynamically adjusts a job size according to source device loading and bandwidth. The lighter the loading a source device has, the larger job size it is assigned. Figure 9 shows a flowchart illustrating this strategy.

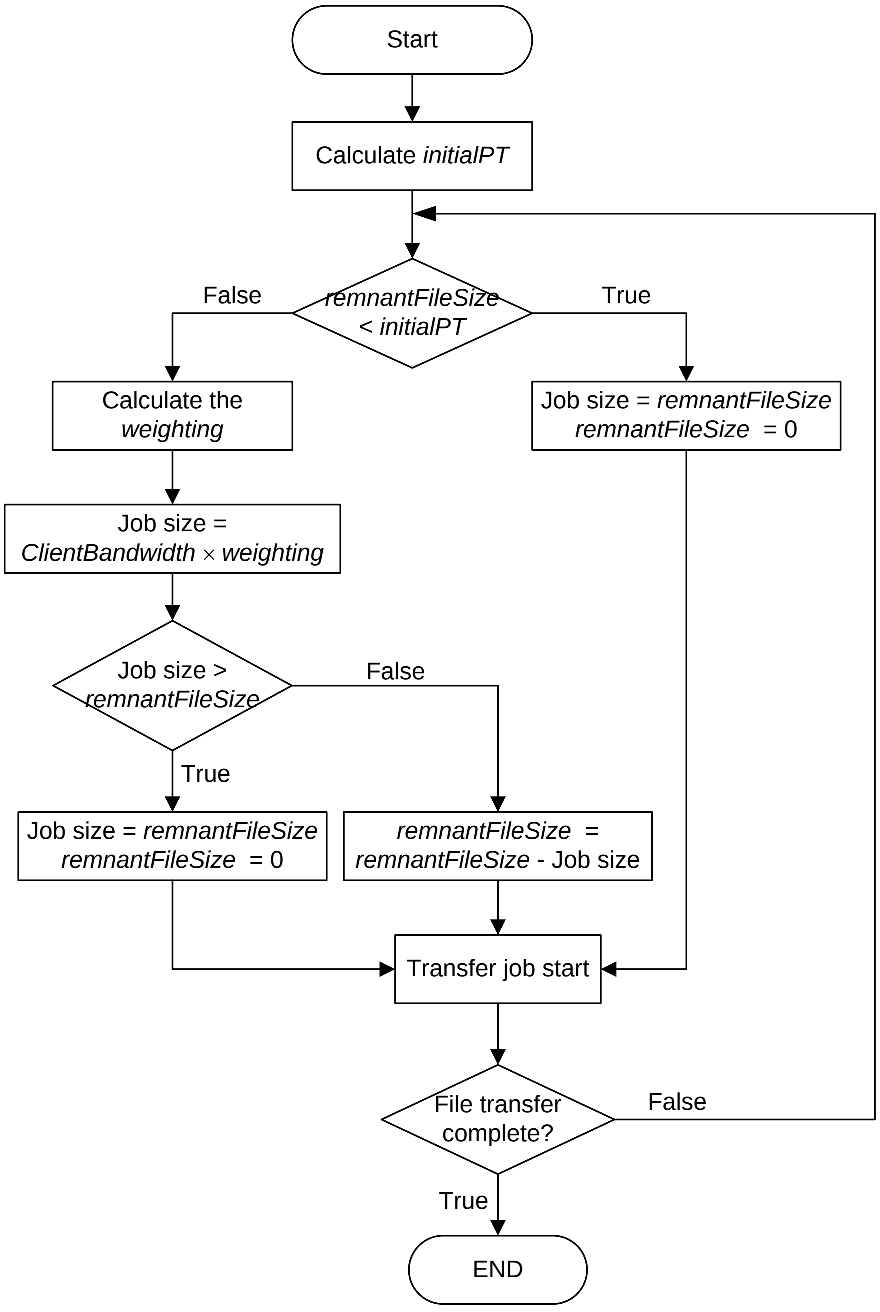

Start
Calculate *initialPT*
remnantFileSize < *initialPT*
False
True
Calculate the *weighting*
Job size = *remnantFileSize* *remnantFileSize* = 0
Job size = *ClientBandwidth* × *weighting*
Job size > *remnantFileSize*
False
True
Job size = *remnantFileSize* *remnantFileSize* = 0
remnantFileSize = *remnantFileSize* - Job size
Transfer job start
File transfer complete?
False
True
END

Figure 2.8: The flowchart of Dynamic Adjustment Strategy

Chapter 3

Our Approach

3.1 Anticipative Recursively Adjusting Mechanism (ARAM)

The recursively adjusting mechanism reduces file transfer completion times and idle times spent waiting for the slowest server. It also provides an effective scheme for reducing the cost of reassembling data blocks. However, our scheme did not consider the potential effect of server links broken or idled during file transfers. Therefore, we propose an efficient approach called the anticipative recursively adjusting mechanism (ARAM) to extend and improve upon recursive adjustment co-allocation [12]. The main idea of the ARAM is to assign transfer requests to selected replica servers according to the finish rates for previous transfers, and to adjust workloads on selected replica servers according to anticipated bandwidth statuses. In continuously adjusting selected replica server workloads, the anticipative recursively adjusting mechanism scheme measures actual bandwidth performance during data file transfers and regulates workloads by anticipating bandwidth statuses for subsequent transfers according to the finish rates for previously assigned transfers. The basic idea is to assign less work to selected replica servers on network links with greater performance variability. Links with more bandwidth variation will have smaller effective bandwidths, as well as smaller finish rates for assigned transfers. The goal is to have the expected finished times of all servers be the same. Our approach performs well, even when the links to selected replica servers are broken or idled. It also reduces the idle time wasted waiting for the slowest server. As appropriate file sections are selected, they are first divided into proper block sizes according to the respective server bandwidths, previously assigned file sizes, and transfer finish rates. Initially, the finish rate is set to 1. Next, the co-allocator assigns the blocks to selected replica servers for transfer. At this moment, it is expected that the transfer finish times will be consistent with $E(t_1)$. However, since server bandwidths may fluctuate during segment deliveries, actual completion times may differ from expected times $E(t_1)$ (solid lines in Figures 3.1 and 3.2). When the fastest server finishes at time t_1, the size of unfinished transfer blocks (italic blocks in Figures 3.1 and 3.2) is measured to determine the finish rate. Two outcomes are possible: the quickest server finish time t_1 may be slower than or equal to the expected time, $E(t_1)$, indicating that network link performance remained unchanged or declined during the transfer. In this case, the difference in transferred size between the expected time and actual completion time (italic block in Figure 3.1) is then calculated.

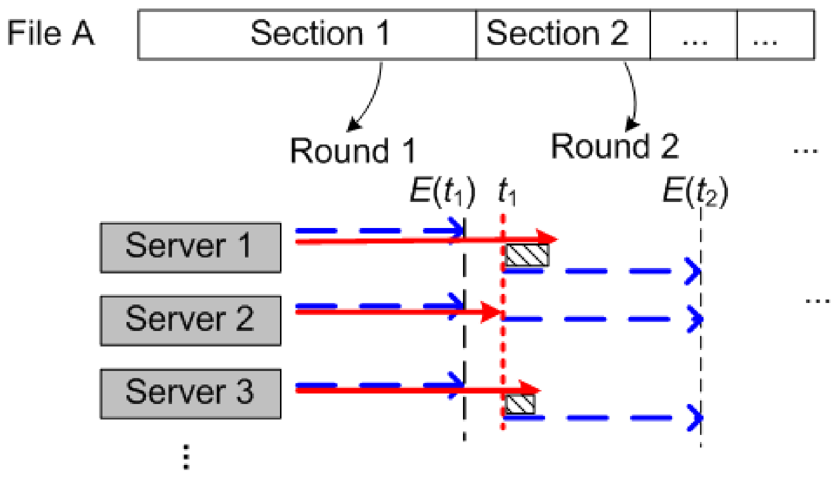


Figure 3.1: Later-than-expected-time Adjustment Process

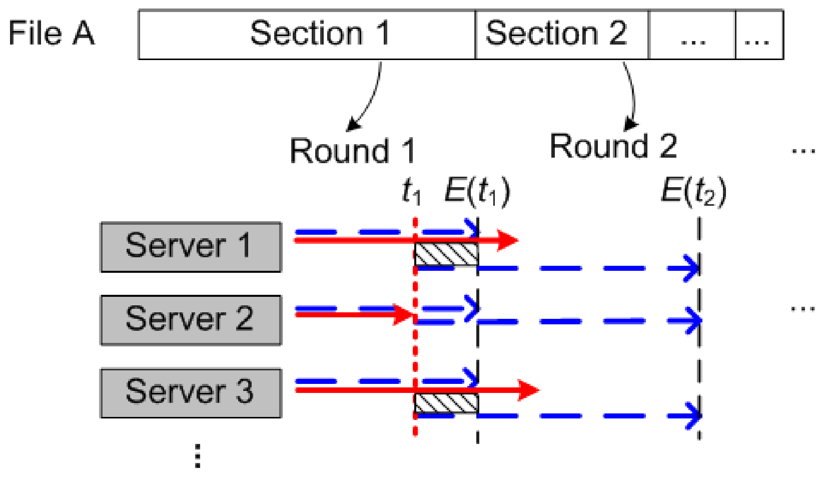


Figure 3.2: Earlier-than-expected-time Adjustment Process

The other outcome is that the quickest server finish time t_1 may be faster than the expected time, $E(t_1)$, indicating an excessively pessimistic anticipation of network performance, or an improvement in replica server network link performance during the transfer. The difference in transferred size between the expected time (italic block in Figure 3.2) and earlier time is then measured. If the anticipated network performance was excessively pessimistic, it is adjusted for the next section. The next task is to assign proper block sizes to the servers along with respective bandwidths and previous finish rates, enabling each server to finish its assigned workload by the expected time, $E(t_2)$. These adjustments are repeated until the entire file transfer is finished.

Looking more closely at ARAM, some parameter definitions are shown below.

- *A*: file requested by user
- *n:* selected replica servers
- α: rate that determines how much of the section remains to be assigned
- *Tj:* allocated time for section j
- *SEj:* allocated size for section j
- *UnassignedFileSize:* portion of file A not yet distributed for downloading
- *UnfinishedFileSize:* the size of unfinished blocks assigned in previous rounds
- *Bji:* real-time transfer rate from the selected replica server
- *rj*: transfer finish rate
- rj_{-1}: server transfer finish rate for previously assigned delivered file
- *Bj*: bandwidth available for section *j*
- *Sji*: block size per flow from *SEj* for each server *i* at time *Tj*
- *ETji*: expected time for server *i* at section *j*
- *RTji*: real finish time for server *i* at section *j*
- *TSji*: actual transfer size at real finish time *RTji*

- *rji*: job finish rate

When a user requests file *A* from the data grid environment, the replica selection server responds with a list of all available servers defined as maximum performance data sets/servers. Data sets/servers for the co-allocator to transfer the file are selected, and the target file is then transferred from the chosen replica data sets/servers.

Assume that *n* replica servers are selected and *Si* denotes server "*i*" for $1 \leqq i \leqq n$. A connection for file downloading is then built to each server.

The anticipative recursively adjusting mechanism process is as follows. A new section of a file to be allocated is first defined. The section size is shown as

$$SEj = (UnassignedFileSize\ +\ TotalUnfinishedFileSize) \times \alpha, \quad 0 < \alpha \leq 1 \tag{1}$$

where *SEj* denotes section *j* such that $1 \leqq j \leqq k$, assume *k* time is allocated for downloading and there are *k* sections, while *Tj* denotes the time allocated to section *j*. *UnassignedFileSize*, the portion of *File A* awaiting distribution for downloading is initially equal to total file size and *TotalUnfinishedFileSize* is equal to zero in the first round. α is the rate determining how much of the section remains to be assigned.

In the next step, *SEj* is divided into several blocks and assigned to "*n*" servers. Each server has a real-time transfer rate to the selected replica server of *Bji*. rj_{-1} denotes the server transfer finish rate for previously assigned files, where the initial value is 1. The block size per flow from *SEj* for each server "*i*" at time *Tj* is *Sji*:

$$Sji = \frac{SEji \times (Bji \times rj_{-1})}{\sum_{i=1}^{n}(Bji \times rj_{-1})}, 0 \leq rj_{-1} \leq 1 \tag{2}$$

$$Bj = \sum_{i=1}^{n}(Bji \times rj_{-1}) \tag{3}$$

$$ETji = \frac{Sji}{Bji} \tag{4}$$

This fulfills our requirement to minimize the time faster servers must wait for the slowest server to finish. In some cases, network variations greatly degrade transfer rates. A faster channel may finish its assigned data blocks at real finish time $RTji$, or later or earlier than expected time $ETji$. Then $TSji$ denoting the actual transfer size at real finish time $RTji$ is given by

$$TSji = Bji \times RTji \tag{5}$$

If the first finish time for $RTji$ is earlier than expected time $ETji$, the reason may be an excessively pessimistic anticipation of network performance, or the network links used for improvement during the transfer. We compare the block sizes transferred between the earliest and expected times for each server chosen. If the transferred size $TSji$ is greater than expected size Sji at the first finish time, otherwise, the first finish time for $RTji$ may be the result of the network link used remaining unchanged or deteriorating during the transfer.

$$rji = \begin{cases} \dfrac{TSji}{Sji}, RTji \geq ETji \\ 1, RTji < ETji, and\ TSji \geq Sji \end{cases} \tag{6}$$

The co-allocator then measures the bandwidth performance of each server, and estimates bandwidth statuses for the next transfer section in order to adjust workflows for the next session. At the same time, it eliminates server *UnfinishedFileSize* listings by summing them up for assignment to the next section.

After allocation, all selected replica servers continue transferring data blocks. When a faster selected replica server finishes its assigned data blocks, the co-allocator allocates an unassigned section of file *A*. Workflows are continually adjusted during the data block allocation process until the entire file has been allocated.

3.2 TCP Bandwidth Estimation Model

TCP/UDP is one of the core protocols in the Internet protocol suite. TCP provides reliable, in-order delivery of a stream of bytes, making it suitable for applications such as GridFTP file transfers. Parallel TCP sockets is a generic “hack” that improves TCP throughputs during bulk data transfers by opening several TCP connections and striping the data files over them [1]. In practice, it is often unclear how many sockets one needs to open in order to achieve satisfactory throughput, and opening too many connections may be undesirable for various reasons [1, 5, 13, 16]. The TCP Bandwidth Estimation Model [13] as a function to assessing TCP packet loss rate, such as round trip time, maximum segment size, other miscellaneous parameters, etc.

$$TCP_{BW}(p) \approx min\left(\frac{W_{max}}{RTT}, \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0\ min\left(1{,}3\sqrt{\frac{3bp}{8}}\right) p(1+32p^2)}\right) MSS \qquad (7)$$

- $TCP_{BW(p)}$: bytes transmitted per second
- *MSS*: maximum segment size
- W_{max}: maximum congestion window size
- *RTT*: round trip time
- *b*: number of transmitted data packets acknowledged by one acknowledgement (ACK) from the receiver (usually b=2)
- T_0: timeout value
- *p*: packet loss ratio, number of retransmitted packets divided by the total number of packets transmitted
- *C*: a constant value, initially set to 1.0

In equation (7), $TCP_{BW(p)}$ represents bytes transmitted per second, and three factors need to be considered: *MSS*, *RTT* and *p*. These represent overall TCP bandwidth. For TCP performance assessment, another researcher has simplified them into one.

$$BW \le \frac{MSS}{RTT}\frac{C}{\sqrt{p}} \qquad (8)$$

In equation (8), *MSS*, *RTT* and *p* are the same variables used in equation (7), *C* is a constant factor, and *BW* represents the number of bytes transmitted per second.

Thus, how the TCP Bandwidth Estimation Model measures server bandwidth makes it more reliable and fair.

3.3 k-means Algorithm

The k-means algorithm clusters n objects according to attributes into k partitions, $k < n$. It is similar to the expectation-maximization algorithm for Gaussian mixtures in that they both attempt to find natural cluster centers in data. Assuming object attributes form vector spaces, it tries to minimize total intra-cluster variance, or, the squared error function:

$$V = \sum_{i=1}^{k} \sum_{x \in s_i} \|x - m_i\|^2 \tag{9}$$

According to the k-means algorithm, where there are random k clusters S_i, i = 1, 2, ..., k, the Euclid distance of each x point to m_i in S_i , m_i is the cancroids or mean point of all the points $x \in S_i$. Equations (10) ~ (13) not only calculate Euclid distances by means of each S_i, but also recursively renew the mean point m_i depending on the cost function V. After calculations, 9 servers with different network bandwidths have been placed in three groups (k=3). The simulation results are shown in Figure 3.3.

- k: number of partitions
- x: number of points
- S_i: partition attributes form a vector space
- m_i: the mean point of all of S_i points
- $xBoolean_{ij}$: determines whether or not an x point belongs to S_i
- V: distance cost function
- d: distance between two point

$$m_i = \frac{\sum_{x \in S_i} d(x_j, m_i)}{|S_i|} \tag{10}$$

$$xBoolean_{ij} = \begin{cases} 1 \ \ if \left\| x_j - S_i \right\|^2 \leq \left\| x_j - S_k \right\|^2 & ,\forall k \neq 1 \\ 0 & ,otherwise \end{cases} \tag{11}$$

$$V = \sum_{i=1}^{k} V_i = \sum_{i=1}^{k} \left(\sum_{k,x_j \in S_i} d(x_j, m_i) \right) \tag{12}$$

$$new(m_i) = \frac{1}{|S_i|} \sum_{k,x_j \in S_i} x_j \tag{13}$$

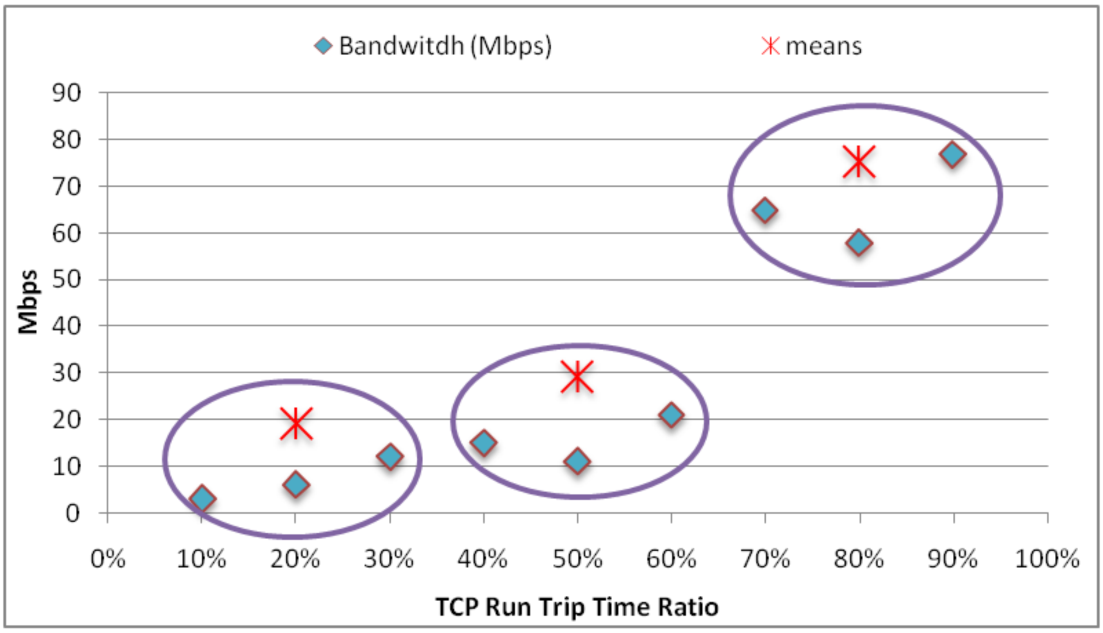


Figure 3.3: 9 Hosts Classification According to Bandwidth Using k-means Algorithm

3.4 Burst Mode

Like many network accelerator methods, and multithreading, Burst Mode (BM) first splits one huge bandwidth into small pipelines all working at the same time. Burst Mode focuses on the fastest group of servers and can differentiate among the various candidate server network bandwidths. Second, BM chooses the faster one then others (as shown in Equations 10, 11, 12, and 13). Ultimately, the BM has made single jobs into many, as shown in Figure 3.4.

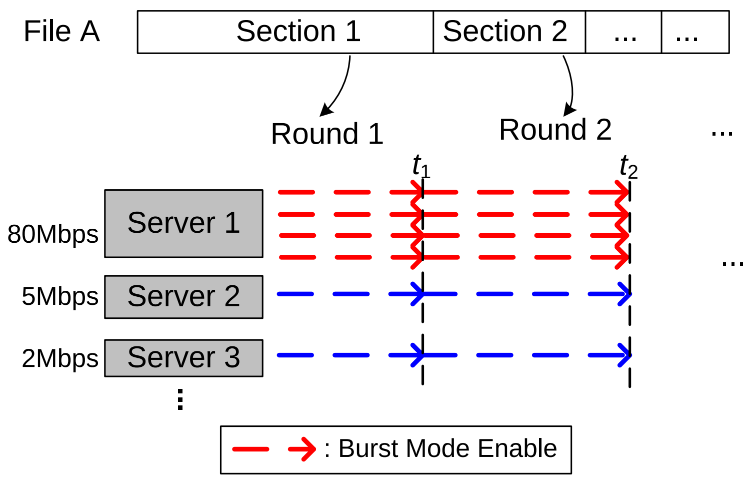


Figure 3.4: Burst Mode Enables Higher Bandwidths

The k-means simulation results showed that fewer local replica servers are high efficiency than many remote replica servers. Accordingly, the main ideas in Burst Mode (BM) are to find the fastest server group, and to make it download via multithreading. BM also deals with cutting blocks properly for various data sets.

Burst Mode function is shown below:

- N_iTCP_{BW}: candidate server bandwidth
- *FTS*: the fastest group of servers

$$N_iTCP_{BW} = \frac{MSS}{RTT}\frac{C}{\sqrt{p}} \tag{14}$$

$$FTS = S_i, MAX\{S_1, S_{2,} \cdots, S_n\}, m_i \epsilon S_i \tag{15}$$

The algorithm is listed below:

[Initialization]

Measure bandwidths and find the fastest servers using Equations 14 and 15.

BigBlockUnit set to 100MB initially

[Allocate blocks to the fastest servers and download via multithreading.]

Step 1: Group mi and rank the most powerful server FTS

Step 2: Allocate SEj and download via multithreading

Step 3: Monitor job progress statuses

```
LOOP WHEN (UnassignedFileSize and total UnfinishedFileSize are greater than BigBlockUnit (initial BigBlockUnit=100MB)) THEN
{
IF (Job finish rate is just 100% (rji=1) and UnassignedFileSize and total UnfinishedFileSize are greater than BigBlockUnit) THEN
{
Let data transfer in multiple parts between client and FTS server
$SE_j = (UnassignedFileSize + TotalUnfinishedFileSize) \times \alpha,$
$0 < \alpha \leq 1,$
$(UnassignedFileSize + TotalUnfinishedFileSize) \geq BigBlockUnit$
}
}
END LOOP;
```

3.5 Grid Network Congestion Control

Grid network congestion control is concerned with controlling traffic entry into data grid networks to prevent congestive collapse by avoiding oversubscription of any grid node processing or link capacity and taking resource reduction steps, such as reducing packet sending rates when Burst Mode is active.

The modern theory of congestion control [35, 36], describes how individuals controlling their own pack lost rate can interact to achieve an optimal network-wide rate allocation. Examples of "optimal rate" allocation are max-min fair allocation and Kelly's [35] suggestion of proportional fair allocation, although many others are possible. The mathematical expression (Equation 16) for optimal rate allocation is as follows. Let x_i be the rate of flow *i*. Let *x*, *c* and *R* be the corresponding vectors and matrix. Let *U*(*x*) be an increasing, strictly convex function, called the utility, which measures how much benefit a user obtains by transmitting at rate *x*. The optimal rate allocation will then satisfy:

$$\max_x \sum_i U(x_i)\,, Rx \leq c \tag{16}$$

3.6 Anticipative Recursively Adjusting Mechanism Plus (ARAM+)

3.6.1 Assumptions

We outline our system design model assumptions below.

- All grid nodes are installed GlobusToolkit4 previously.
- All grid nodes are supporting Simple Network Management Protocol (SNMP).
- The time for transferring, stopping/assigning processes, and calculating TCP_{BW} to selected replica servers is negligible.

3.6.2 Anticipative Recursively Adjusting Mechanism Plus (ARAM+)

The ARAM+ is not merely inherited from ARAM. It has been enhanced also in the following areas: its TCP Bandwidth Estimation Model (TCPBEM) and its Burst Mode (BM). ARAM+ continually adjusts the workloads on selected replica servers by measuring actual bandwidth performance via TCPBEM during data file transfers and, according to previous job finish rates, and adjusting alpha values for subsequent transfer sections.

Some interesting ideas have arisen from P2P networks and distributed denial-of-service (DDoS) attacks. As is well known, P2P networking is share based; it shares data and downloads in parallel, more numbers of share point get more speedup. Another typical example is DDoS attacks that occur when multiple compromised systems flood the bandwidth or resources of a targeted system. We have combined these elements in our approach. The multithreading in the Burst Mode (BM) design came from DDoS attacks, BM “floods” the target replica server bandwidth to speed up download performance. The other idea from P2P networking was applied to

ARAM+. It pre-selects many candidate replicas from various servers, then chooses appropriate servers and allocates only enough workload to fit server capacities.

Both of our previous works [24, 25, 27, 29, 30], the anticipative recursively adjusting mechanism (ARAM) and recursively adjusting mechanism (RAM) were based on co-allocation architecture and relied on tuning alpha values by hand to adapt to specific data grid situations. The ARAM+ uses the same strategies, but differs in that alpha values are tuned dynamically.

ARAM+ adapts to real-time network statuses and calculates appropriate alpha α values continually with TCPBEM $TotalTCP_{BW}$, to ensure good download flexibility and to speed up overall performance. The equations are as follows:

- $TotalTCP_{BW}$: overall bytes transmitted per second.

$$TotalTCP_{BW} = \sum_{i=1}^{N} \frac{MSS}{RTT} \frac{C}{\sqrt{p}} \tag{17}$$

$$\alpha = 1 - \left(\frac{1}{{TotalTCP_{BW}}^{0.2}}\right), 0 < \alpha \leq 1 \tag{18}$$

3.6.3 ARAM+ Algorithm

[Initialization]

Current bandwidths for all candidate servers are measured using the TCP Bandwidth Estimation Model (TCPBEM) and calculating appropriate alpha values with Equations 14 and 15.

[Allocating blocks to selected servers]

LOOP WHEN (*UnassignedFileSize* and total *UnfinishedFileSize* is greater than zero) THEN

{

IF (*UnassignedFileSize* and Total *UnfinishedFileSize* are greater than $TotalTCP_{BW}$)

```
THEN
{
IF (UnassignedFileSize and Total UnfinishedFileSize multiplied by $\alpha$ are greater than $TotalTCP_{BW}$) THEN
{
Define new section for allocation
$SEj = (UnassignedFileSize + TotalUnfinishedFileSize) \times \alpha,$
$0 < \alpha \leq 1$
}
ELSE
{
Define finial section
        $SEj = UnassignedFileSize + TotalUnfinishedFileSize$
}
}
END LOOP;
   Step 1:  Define new section for allocation SEj
   Step 2:  Monitor all selected replica servers
   Step 3:  Allocate blocks to selected replica servers, according to the $TCP_{BW}$ of the
            selected replica server, and the previous finish rates Rj-1 for the selected
            replica server (initial $R_0$=1)
   Step 4:  Monitor all download flows
LOOP WHEN (The fastest flow finishes its assigned data blocks) THEN
{
IF (First finish time for RTji is earlier than expected time ETji and transferred size
TSji is greater than expected size Sji ) THEN
{
    The $rji$=1
}
ELSE
{
    Measure the finish rate for the previously delivered file ($0 \leq rji \leq 1$)
}
$rji = \begin{cases} \frac{TSji}{Sji}, RTji \geq ETji \\ 1, RTji < ETji, and\ TSji \geq Sji \end{cases}$
}
END LOOP;
```

Chapter 4

Experimental Results

4.1 Grid Environment: Tiger Grid

The experiments in this work were conducted and evaluated on the Tiger grid, which consists over 11 clusters located at 6 educational institutions (Tunghai University—THU, National Taichung University—NTCU, Hsiuping Institute of Technology—HIT, National Dali Senior High School—DALI, Lizen High School—LZSH, and Tungs' Taichung Metro Harbor Hospital—TUNG). A logical diagram of the Tiger grid network environment is shown in Figure 4.1. The detail end-to-end transmission rate of THU to every educational unit is listed in Table 4.1. Figure 4.2 shows statuses for all machines used in the grid testbed on one monitor page.

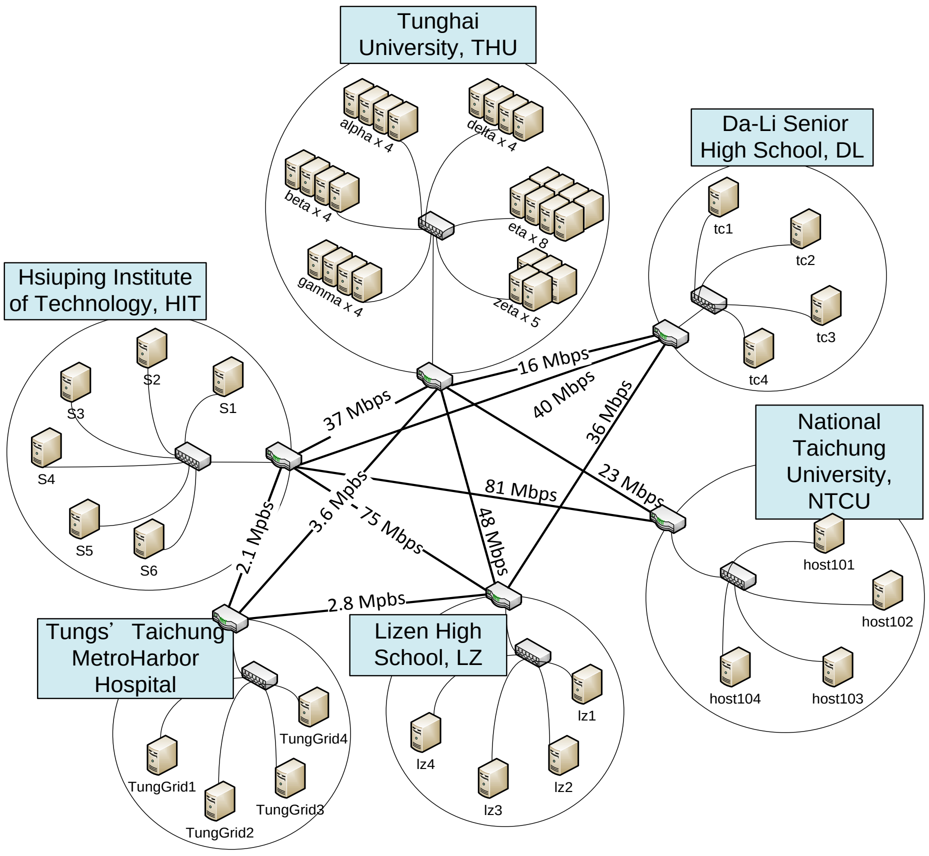


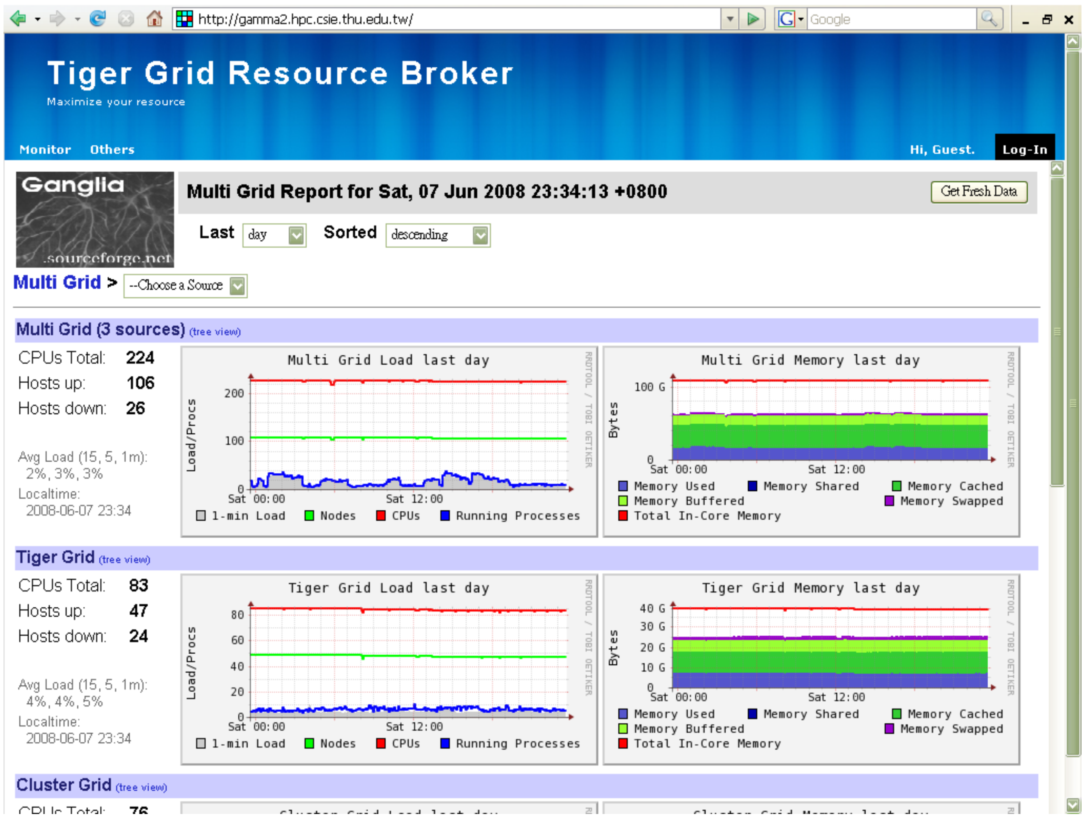
Figure 4.1: Tiger Grid Network Topology

Table 4.1: The end-to-end Measurement Using NWS in Mbps

THU to others			
Case	Bandwidth Avg	High	Low
THU → HIT	37.815	70.349	20.952
THU → DL	16.673	17.920	12.182
THU → LZ	48.139	73.466	31.678
THU → NTCU	23.432	39.824	13.176
HIT to others			
Case	Bandwidth Avg	High	Low
HIT → THU	32.487	49.384	17.913
HIT → DL	38.206	18.166	7.875
HIT → LZ	84.089	77.048	88.664
HIT → NTCU	81.391	86.995	71.530
DL to others			
Case	Bandwidth Avg	High	Low
DL → HIT	42.143	88.129	22.037
DL → THU	15.893	42.823	5.238
DL → LZ	36.631	17.830	87.351
LZ to others			
Case	Bandwidth Avg	High	Low
LZ → HIT	67.579	77.298	53.967
LZ → THU	32.324	56.313	15.099
LZ → DL	17.769	18.098	16.634
NTCU to THU, HIT			
Case	Bandwidth Avg	High	Low

NTCU → THU	23.432	39.824	13.176
NTCU → HIT	81.391	86.995	71.530

They are interconnected by the 1Gbps Taiwan Academic Network (TANET). The Tiger grid platform is built around 132 computing nodes, more than 224 CPUs with differing speeds, and total storage of more than 5 TB. All the institutions are in Taiwan, at least 10~30 km from THU. All machines have Globus 4.0.7 or above installed.


http://gamma2.hpc.csie.thu.edu.tw/

Tiger Grid Resource Broker

Maximize your resource

Monitor Others Hi, Guest. Log-In

Multi Grid Report for Sat, 07 Jun 2008 23:34:13 +0800

Last day Sorted descending

Multi Grid > --Choose a Source

Multi Grid (3 sources) (tree view)

CPUs Total: 224
Hosts up: 106
Hosts down: 26

Avg Load (15, 5, 1m): 2%, 3%, 3%
Localtime: 2008-06-07 23:34

Tiger Grid (tree view)

CPUs Total: 83
Hosts up: 47
Hosts down: 24

Avg Load (15, 5, 1m): 4%, 4%, 5%
Localtime: 2008-06-07 23:34

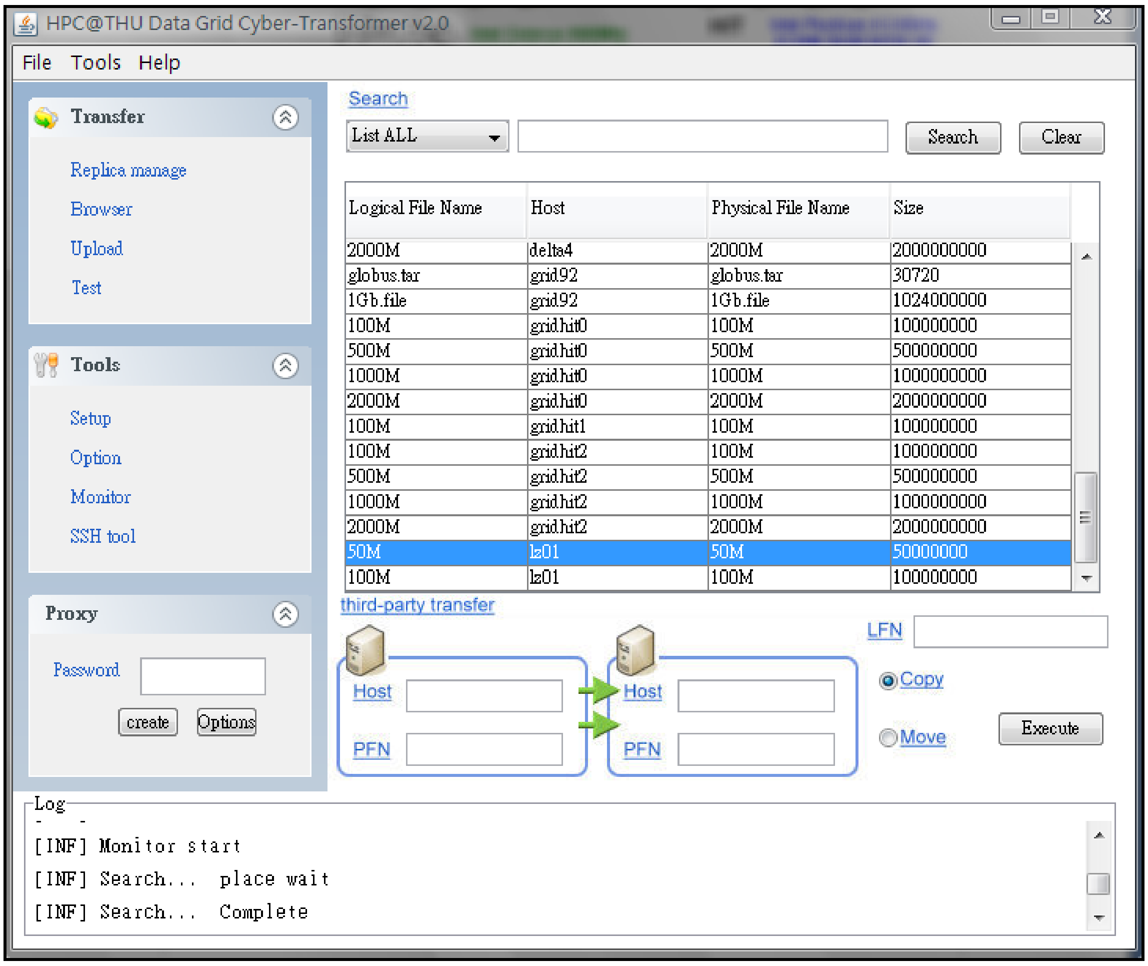
Cluster Grid (tree view)

CPUs Total: 76

Figure 4.2: Tiger Grid Rsources

4.2 Experimental Tool: Cyber Transformer

In a previous work [28], we gave experimental results for Cyber Transformer, a powerful new toolkit for replica management and data grid environment data transfers. It can accelerate data transfer rates, and also manage replicas over various sites. The friendly interface enables users to easily monitor replica sources, and add files as replicas for automatic cataloging by our Replica Location Service. Moreover, we provide a function for administrators to delete and modify replicas. Cyber Transformer can be invoked with either the logical file name of a data file or a list of replica source host names. When users search for files using logical file names, Cyber Transformer queries the Replica Location Services to find all corresponding replicas, and directs the replica sources to start parallel transfers. Cyber Transformer users can easily gather replica resources and combine them into single entities with the "strategy selection" user interface, accomplishing the task with various parallel download strategies, as shown in Figure 4.3 and 4.4



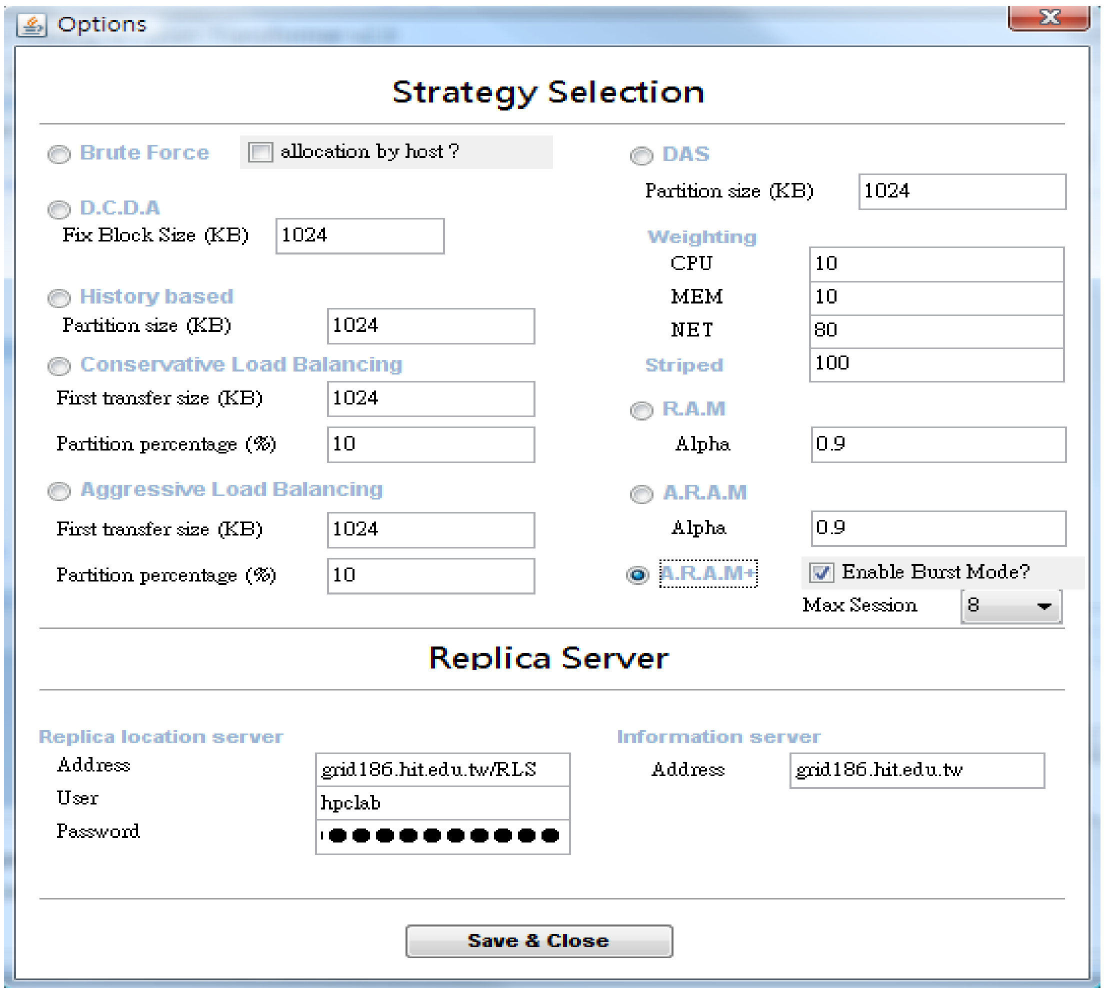
Search: List ALL

Logical File Name	Host	Physical File Name	Size
2000M	delta4	2000M	2000000000
globus.tar	grid92	globus.tar	30720
1Gb.file	grid92	1Gb.file	1024000000
100M	gridhit0	100M	100000000
500M	gridhit0	500M	500000000
1000M	gridhit0	1000M	1000000000
2000M	gridhit0	2000M	2000000000
100M	gridhit1	100M	100000000
100M	gridhit2	100M	100000000
500M	gridhit2	500M	500000000
1000M	gridhit2	1000M	1000000000
2000M	gridhit2	2000M	2000000000
50M	lz01	50M	50000000
100M	lz01	100M	100000000

[✓ Copy] Move

```
[INF] Monitor start
[INF] Search... place wait
[INF] Search... Complete
```

Figure 4.3: The GridFTP Client Tool: Cyber Transformer


Options

Strategy Selection

Brute Force — allocation by host ? ☐

D.C.D.A

Fix Block Size (KB)	1024

History based

Partition size (KB)	1024

Conservative Load Balancing

First transfer size (KB)	1024
Partition percentage (%)	10

Aggressive Load Balancing

First transfer size (KB)	1024
Partition percentage (%)	10

DAS

Partition size (KB)	1024

Weighting

CPU	10
MEM	10
NET	80
Striped	100

R.A.M

Alpha	0.9

A.R.A.M

Alpha	0.9

[✓ A.R.A.M+] — Enable Burst Mode? ✓

Max Session	8

Replica Server

Replica location server

Address	grid186.hit.edu.tw/RLS
User	hpclab
Password	●●●●●●●●●●

Information server

Address	grid186.hit.edu.tw

Figure 4.4: Parallel Download Strategy Selection

4.2.1 System Components

Cyber Transformer is implemented in the Java Cog Kit [14] library. The system stack of Cyber Transformer consists of three parts: (1) Information Monitor, (2) Replica Manager, and (3) GridFTP Browser, to simplify replica management and data transfers. With the intuitive interface, users can easily invoke the services to transfer data without delay. Figure 4.5 shows the Cyber-Transformer system components and the three main services they provided.

- Information Service: This service is invoked by the Information Monitor and provides replica sources statuses allowing users to monitor all replica source sites in the data grid. Sites status, such as CPU loading, free memory, hard disk free space, and bandwidth, are gathered by the Information Service and reported to the Information Monitor.

- Replica Management Service: This serves as middleware between users and replica databases. It enables convenient user replica searches by listing logical file names and replica source host names. Users can also easily upload files as replicas, and mark the importance of these files.
- Data Transfer Service: This is the most important Cyber-Transformer service, and is easily summoned through the GridFTP Browser. Our Dynamic Adjustment Strategy is integrated into it, and an "Option" function enables users to compensate for various data grid environment conditions by adjusting transfer factors such as machine loading, bandwidth, partition size, and stripe numbers, thus accelerating data transfer rates.

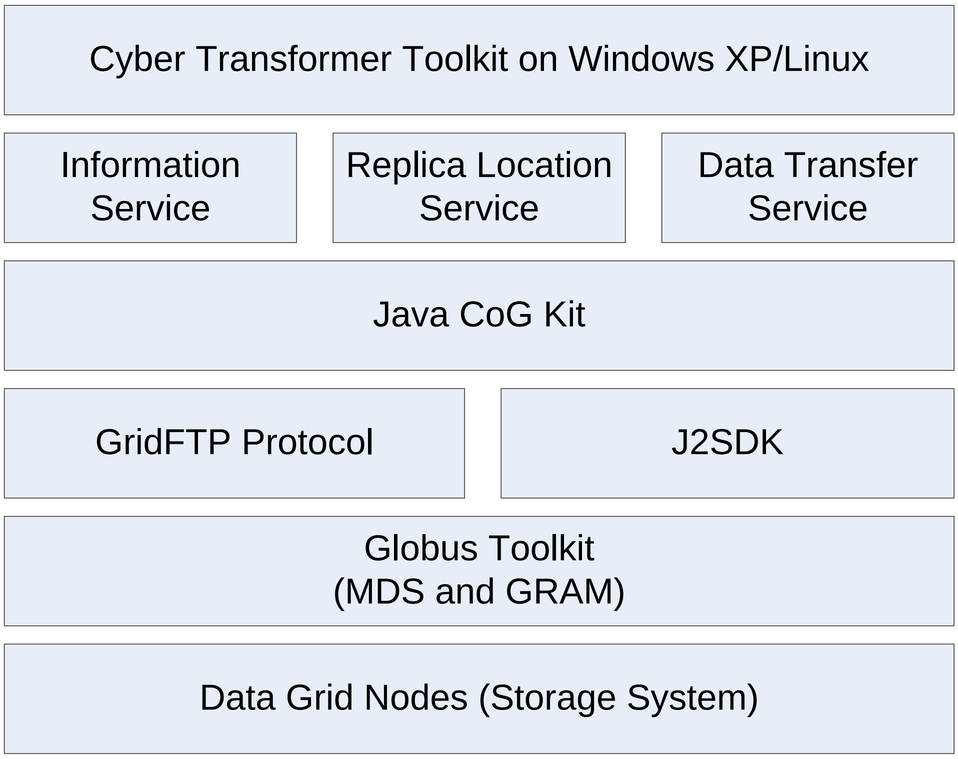

Cyber Transformer Toolkit on Windows XP/Linux
Information Service
Replica Location Service
Data Transfer Service
Java CoG Kit
GridFTP Protocol
J2SDK
Globus Toolkit (MDS and GRAM)
Data Grid Nodes (Storage System)

Figure 4.5: The System Stack of Cyber Transformer

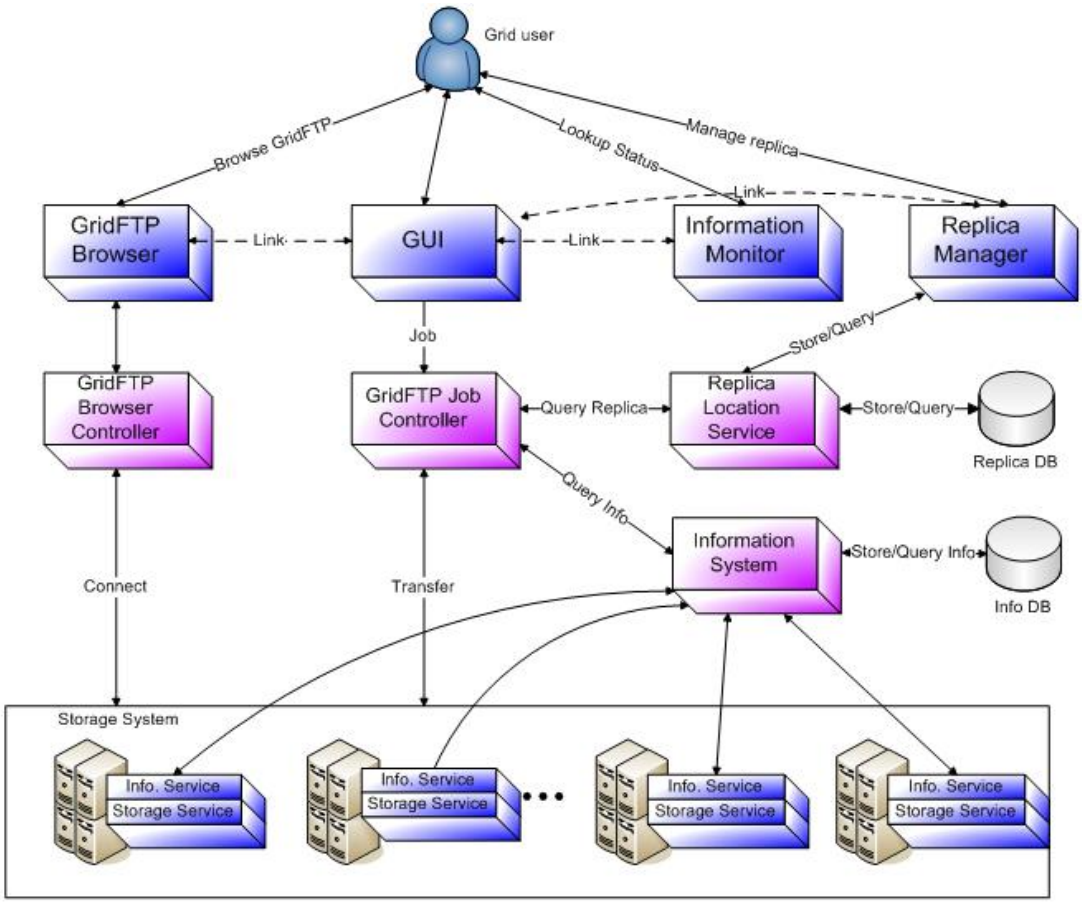


Figure 4.6: The Components of Cyber Transformer

4.2.2 System Transaction Flow

Figure 4.7 shows the Cyber Transformer transaction flow. Users must first pass the Grid Proxy Certification provided by Simple CA to get access to the Grid. They may then connect to any data grid site via the GridFTP Browser. The system automatically authenticates site certifications as connections are made. The security mechanism of our Grid environment is depicted below. Steps 4 and 5 show how users query the Replica Location Service for replica information, and the Replica Location Service reports on requests. The system ranks all replica servers according to our replica selection model [31, 32], and users can then choose the better servers for parallel downloading. The Data Transfer Service is invoked in Step 6. Information about the replicas chosen by the user is picked up by the GridFTP Job Controller. The Controller then dynamically adjusts replica transfer job sizes according to the conditions presented in the information. Job sizes are continually adjusted until all transfers have been completed. The portions from the various replica sources are then gathered into complete file. To enable users lacking deep knowledge of data grids to easily download and manage files in data grid environments, we developed a user-friendly GUI for Cyber Transformer. It is implemented in the Java CoG Kit library, and it can be executed on any operating system with JVM

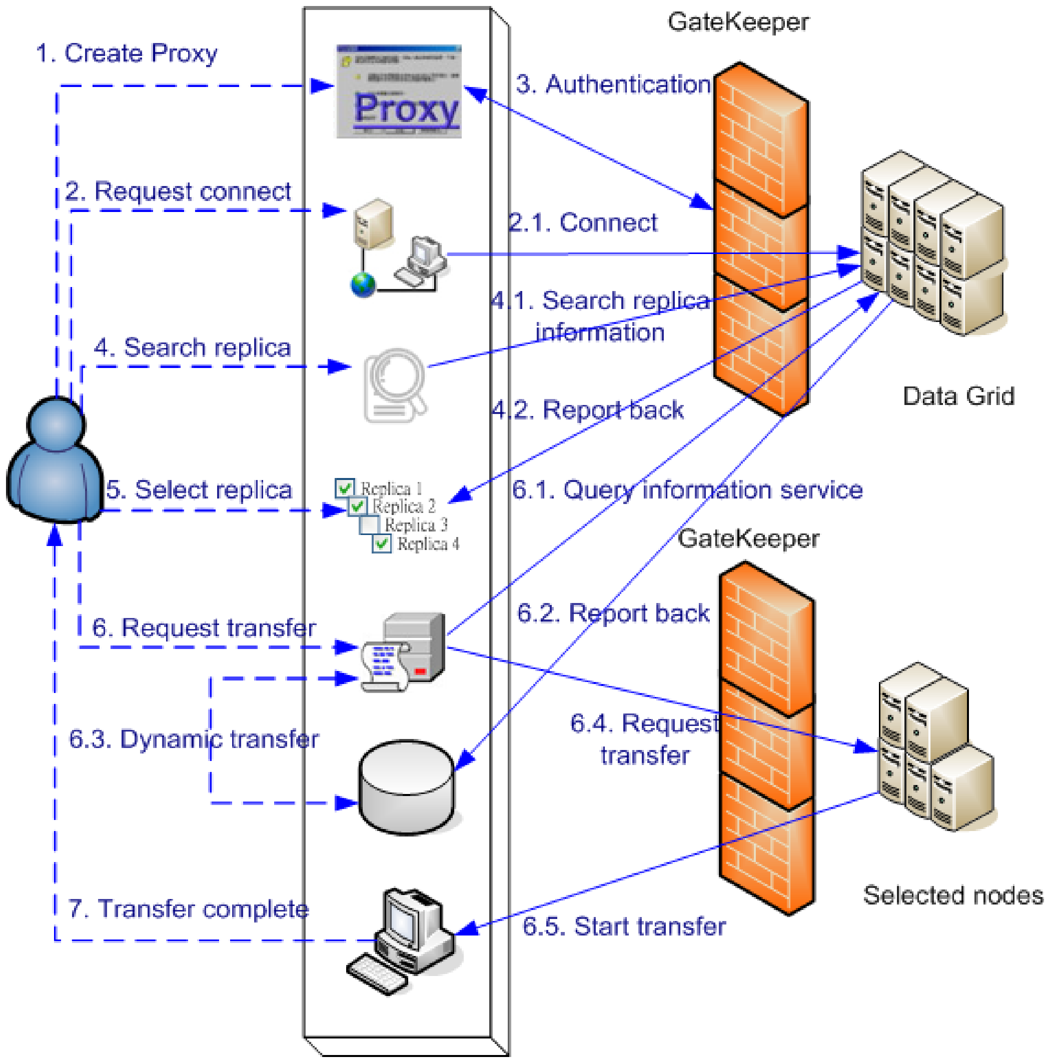


Figure 4.7: The Transaction Flow of Cyber Transformer

4.3 Experimental Results and Analyses

An experiment and a case design were devised to test Burst Mode (BM), our proposed approach to speeding up local and remote performance, and dynamically truing alpha values to adapt to variable network situations. Details of the test cases we designed are shown in Figure 4.8.

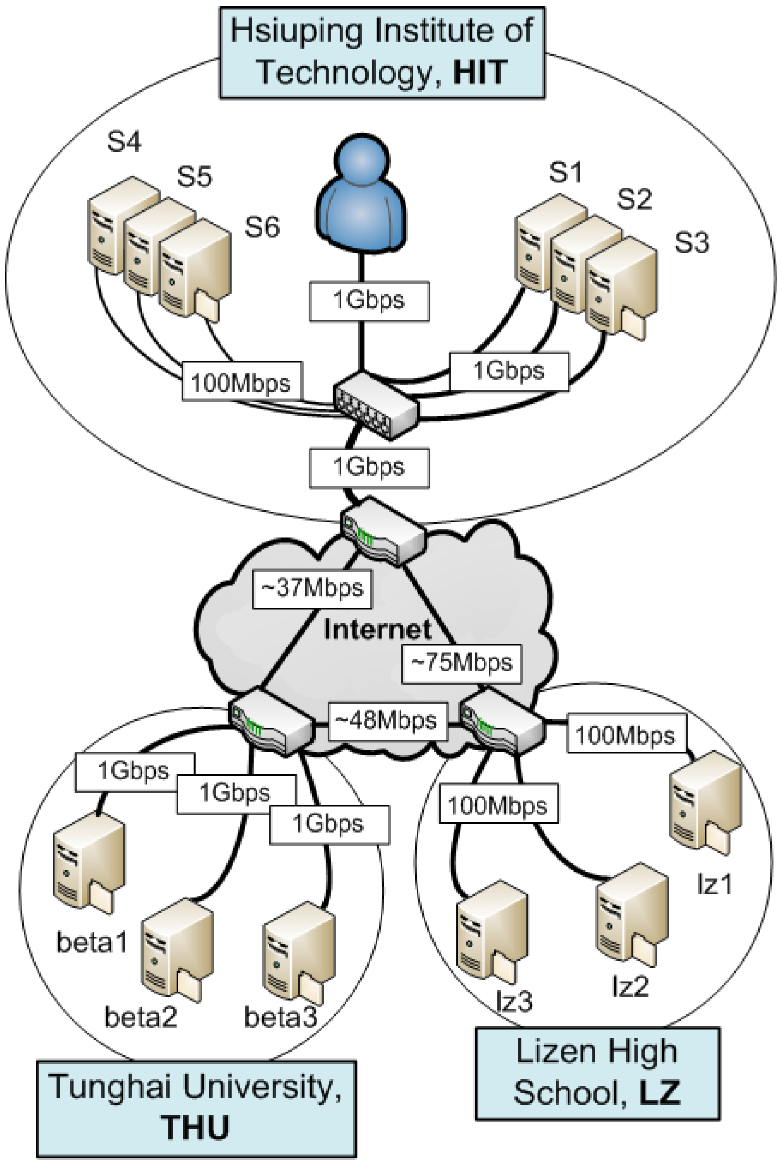


Figure 4.8: Scenarios for Our Testbed of Tiger Grid

4.3.1 Case Study──"cross-grid" vs. "local grid"

We designed two scenarios to verify the efficiency of enabling Burst Mode. All test cases are listed in Tables 4.1 and 4.2.

Table 4.2: Scenario for Replica Local or Not

Scenario	Replica Server List
ARAMplus_4: non-local	THU-S1, S2; LZ1, 2

ARAMplus_4: local-1	HIT-S1, S2; THU-beta1, beta2
ARAMplus_4: local-2	HIT-S1, S2; THU-beta1; LZ-1
ARAMplus_4: local-3	HIT-S1, S2; LZ-1, 2
ARAMplus_4: all-local	HIT-S1, S2, S3, S4

Table 4.3: Scenario for Various Replica Numbers and Selections

Scenario	Replica Server List
Rx6_non-local	LZ-1, 2, 3; THU-beta1, beta2, beta3
Rx6_local	HIT-S1, S2, S3, S4, S5, S6
Rx2_local	HIT-S1, S2
Rx2_non-local-THU	THU-S1, S2
Rx2_non-local-LZ	LZ-S1, S2

Generally, more replicas and local placement will yield better parallel file transfer performance. Our results, shown in Figure 4.9 and 4.10, show that we found more replicas remotely so user performance improvement was not obvious, even worse than the few replica found locally. However, Burst Mode function could get more performance even two copies only (refer to scenario: Rx2_local).

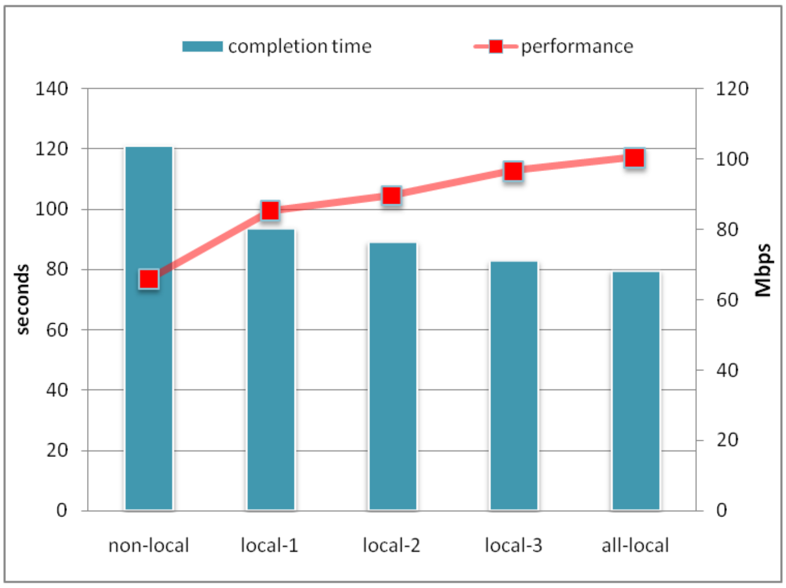


Figure 4.9: Effects of Various Replica Locations on Performance Results

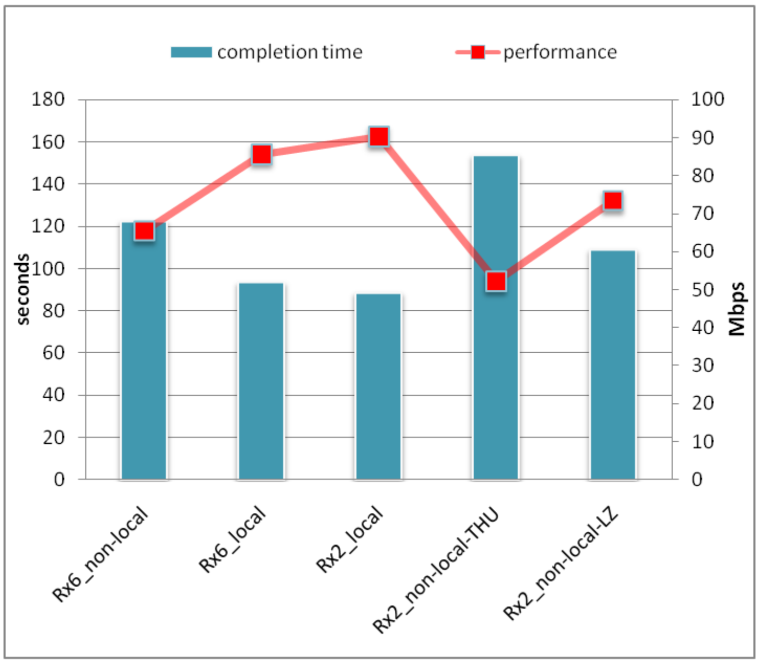


Figure 4.10: Effects of Various Replica Numbers and Selections on Performance Results

4.3.2 Case Study—RAM and ARAM vs. ARAM+

We first compare RAM [32] and ARAM [26] in different alpha value. Both RAM and ARAM were use a static variable 'α' as basis to evaluate the working load and with similar method to dispatch file transformation. The value of alpha will decide the transform blocks to each site, and it means to through the adjustment represents the value of alpha to adapt to different network environment. Testing the two scenarios with transform different file size (100MB, 1000MB) by adjust the value of alpha from 0.1 to 0.9. The results of the testing were shown as Figure 4.11 and 4.12. To change the value of alpha will affect performance when transform smaller file than huger file. Those schemes will obtain better performance when transform huge file at the value of alpha approach to 0.5 and transform small file at the value of alpha approach to 0.9. The worst performance will occurred when the value of alpha equal to 0.1. The transmission quality of ARAM scheme was more stable than RAM scheme with change the value of alpha.

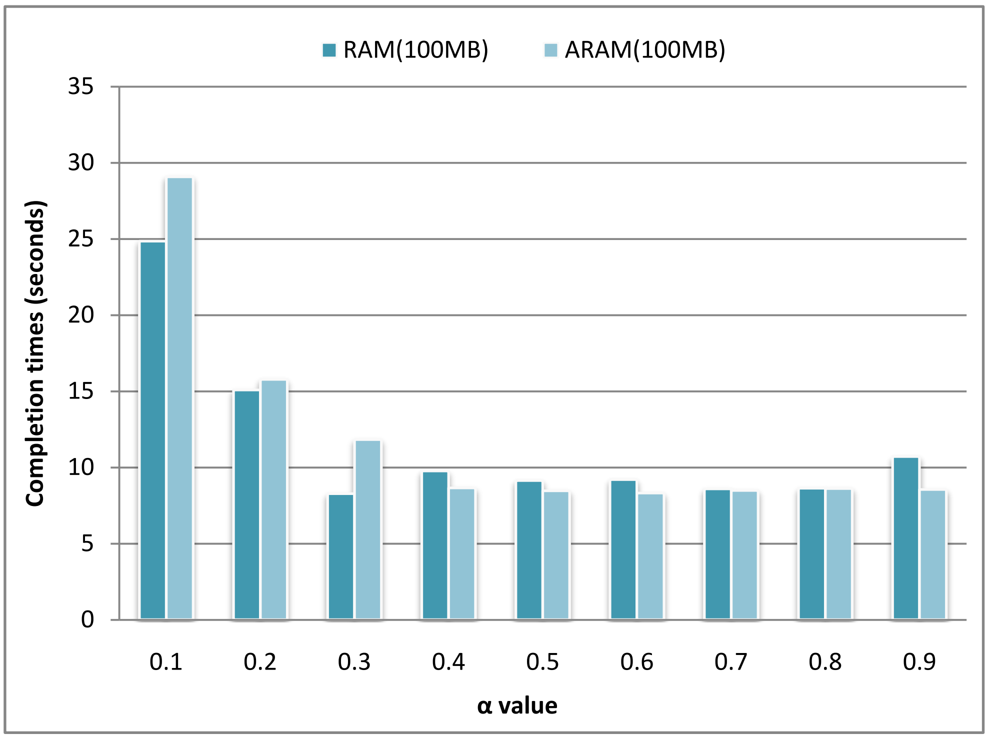


Figure 4.11: Completion Times in Different α Value Using Dataset Size 100MB

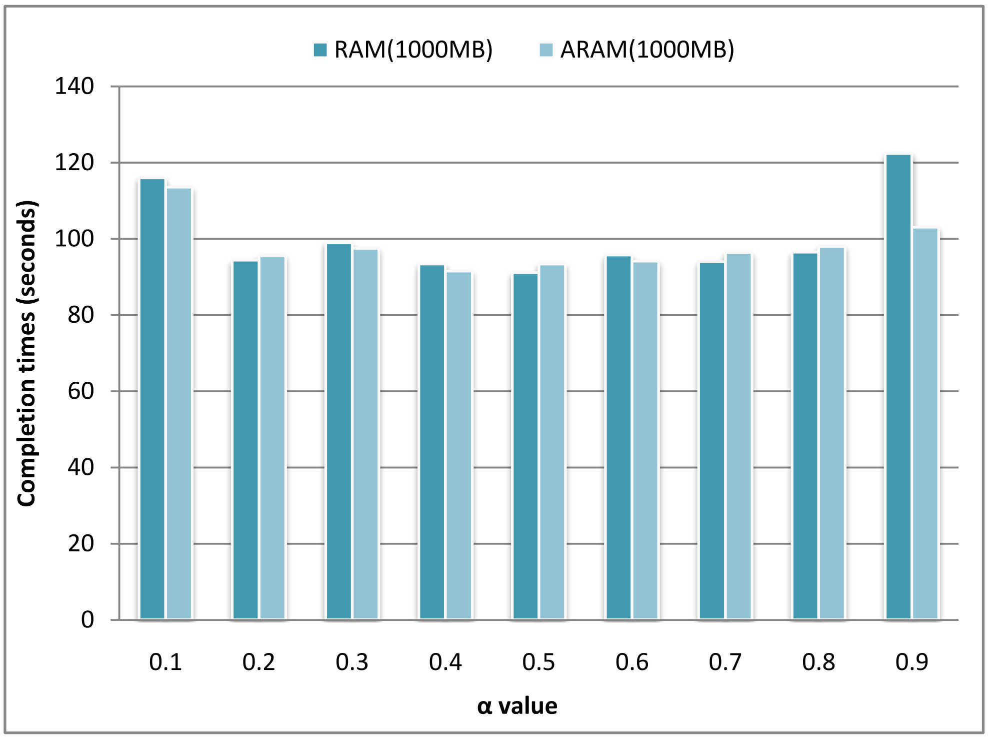


Figure 4.12: Completion Times in Different α value Using Dataset Size 1000MB

RAM and ARAM both used constant alpha values; our approach, ARAM+, relied on dynamic alpha values to adapt to data grid network link fluctuations. The case study for RAM and ARAM is listed in Table 4.4. We set the constant alpha values at 0.9, 0.5, and 0.1 for comparison with ARAM+, and replicas were selected from inside and outside regions. In order to distinguish among replica locations, these two kinds of replica selection plans are listed in Table 4.3.

Table 4.4: Replica Placement and Selection Plan

local	HIT-S1, S2, S3, S4, S5, S6
mix	HIT-S1, S2; LZ-1, 2; THU-beta1, beta2

Table 4.5: Scenario for Alpha Value Tuning

Scenario A	Scenario B
RAM(0.1)_local	RAM(0.1)_mix
ARAM(0.1)_local	ARAM(0.1)_mix
RAM(0.5)_local	RAM(0.5)_mix
ARAM(0.5)_local	ARAM(0.5)_mix
RAM(0.9)_local	RAM(0.9)_mix
ARAM(0.9)_local	ARAM(0.9)_mix
ARAM+_local	ARAM+_mix

In our next experiment, two scenarios, sets A and B, were used to accentuate the advantages of the Burst Mode method and dynamic alpha value adjustment. Overall performances in Scenario B have obviously been improved over those in Scenario A. The total amounts of TCP bandwidth in Scenario A differed slightly, but there were significant differences in Scenario B. In all these case studies, especially in Scenario B, Burst Mode yielded huge performance improvements, as shown in Figures 4.13 and 4.14.

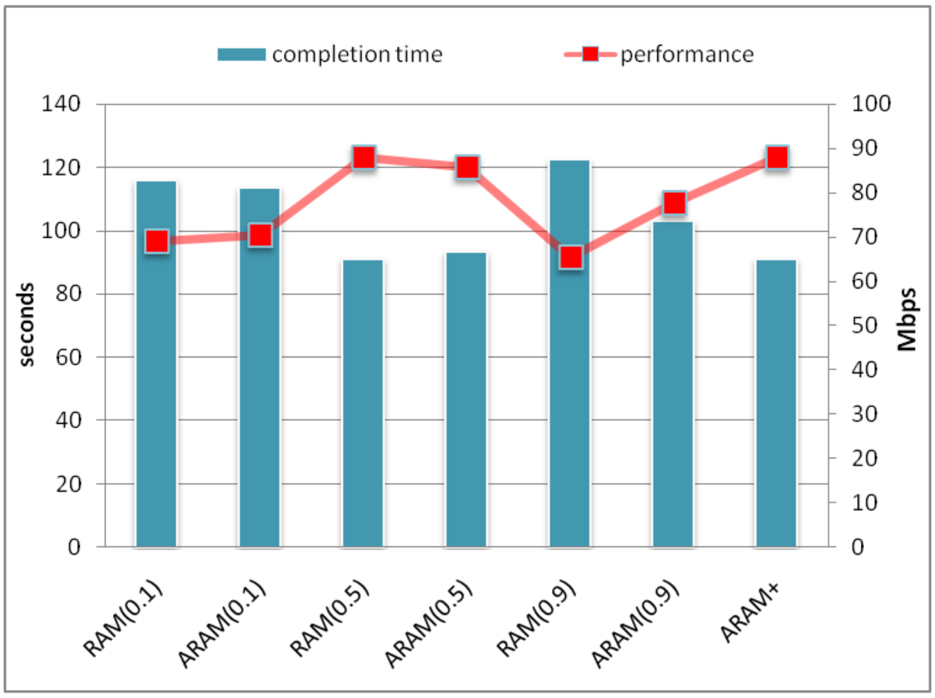


Figure 4.13: Performance Results for Scenario A

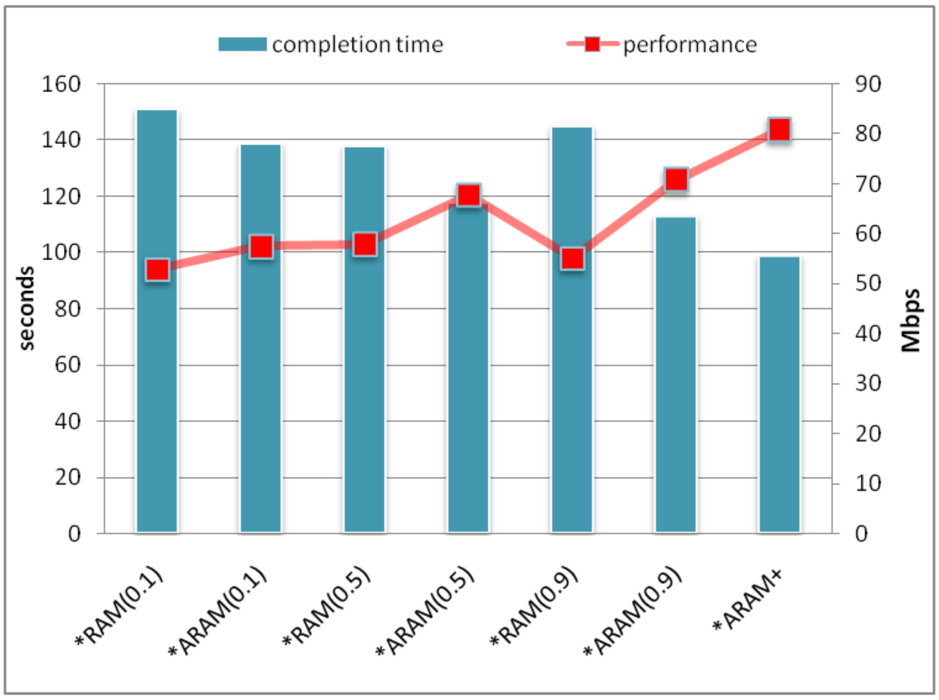


Figure 4.14: Performance Results for Scenario B

4.3.3 Case Study—Comparison of 9 Co-Allocation Schemes

To evaluate the performance of our proposed technique, we implemented the following nine co-allocation schemes: Brute-force (Brute), history-based (history), conservative load balancing (conservative), aggressive load balancing (aggressive), dynamic co-allocation with duplicate assignments (DCDA), recursively adjusting mechanism (RAM), dynamic adjustment strategy (DAS), anticipative recursively adjusting mechanism (ARAM), and anticipative recursively adjusting mechanism plus (ARAM+). Using the case setups listed in Table 4.3 for each scheme, we analyzed their performance by comparing transfer finish times and overall performance, as shown Figures 4.15 and 4.16.

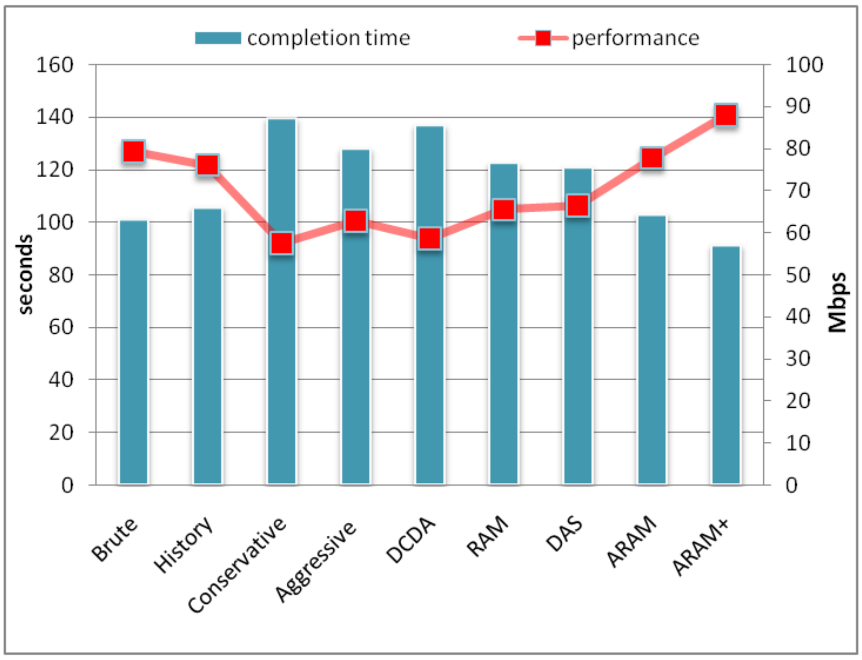


Figure 4.15: Comparing 9 Schemes on "local" Cases

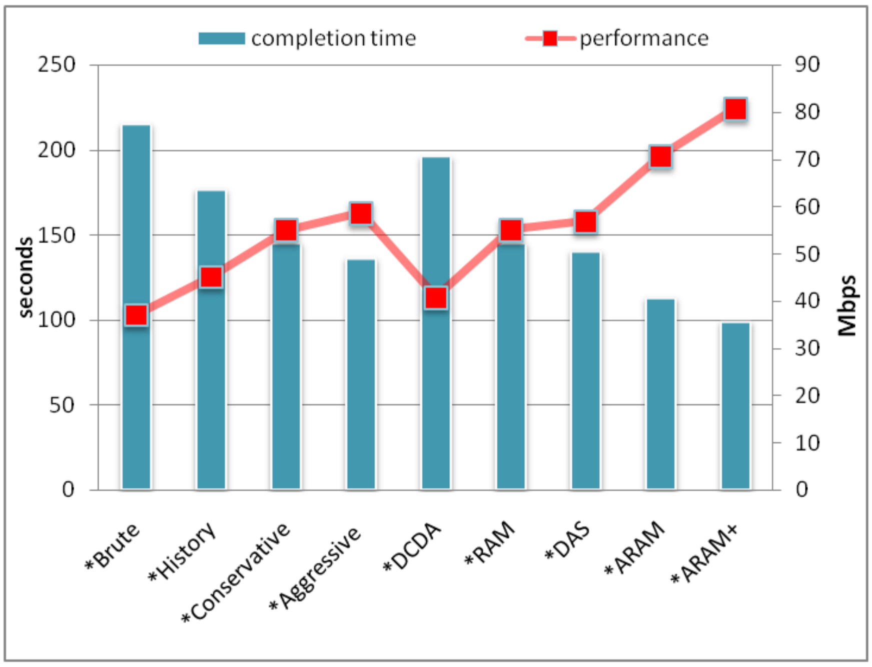


Figure 4.16: Comparing 9 Schemes on "mixed" Cases

We found that ARAM+ performed better than the others. An interesting outcome shows the Brute scheme's "local" performance differed greatly from its "mixed" performance. ARAM+ is comparable to Brute or any others. The advantages of ARAM+ are the follows:

- ARAM+ uses TCP bandwidth measurement technology, reliability and accuracy of the best.
- ARAM+ can enhance GridFTP to become multiplexing.
- ARAM+ used k-means for classifying numbers grid node. It quickly finds out the most efficient computing nodes.
- ARAM+ gives the longest amount of computing job to powerful grid node but small data set could ignore some advance option for example, dynamic α, server classification (k-mean) algorithm and congestion control.
- ARAM+ can really adapt to different grid environments, rather than to just specific experiments designed grid system.

4.3.4 Case Study━━Completion Times for Various Methods with Network Broken

In the final experimentation, we compare 9 co-allocation schemes to face normal and worst network link state. It is possible that any host’s network interrupt or any other types of network fault occur will cause file transformation failure in the heterogeneous and complex environment of the grid. The design of network fault tolerant was important to improve the usability and reliable of the full grid system. There was only DCDA, ARAM and ARAM+ has the ability to face fault in all schemes of this study. As shown in Figure 4.17, we built a disgusting environment with four replicas at grid sites. Each site with replica will disconnection a period time when transform file. Since the DCDA scheme was designed for faster sites will transform more segments in parallel to overcome network fault. The experiment results shown that when some grid node was disconnection at period time, the file can be complete transform. The ARAM and ARAM+ scheme were designed transform file base on current network status, it can avoid the situation that faster site to waiting for slower site. The overall transform file performance was still to keep in stability, as shown Figures 4.18. Finally we sorted out the advantages and disadvantages of 9 schemes and make a comparative table, as shown in table 4.6.

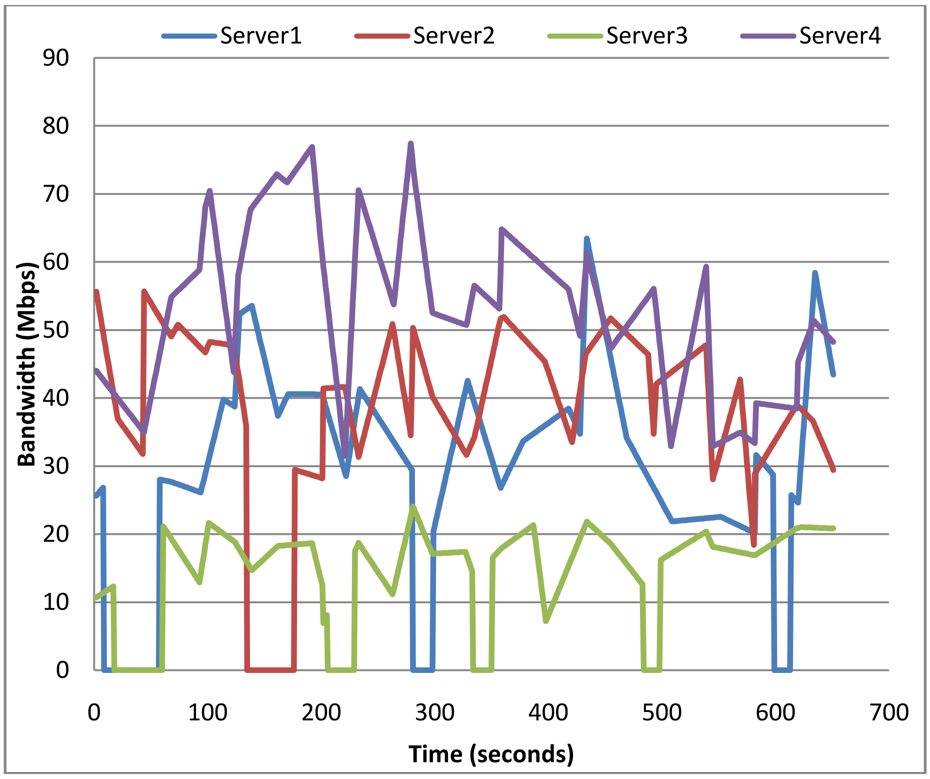


Figure 4.17: Broken Network Link Status

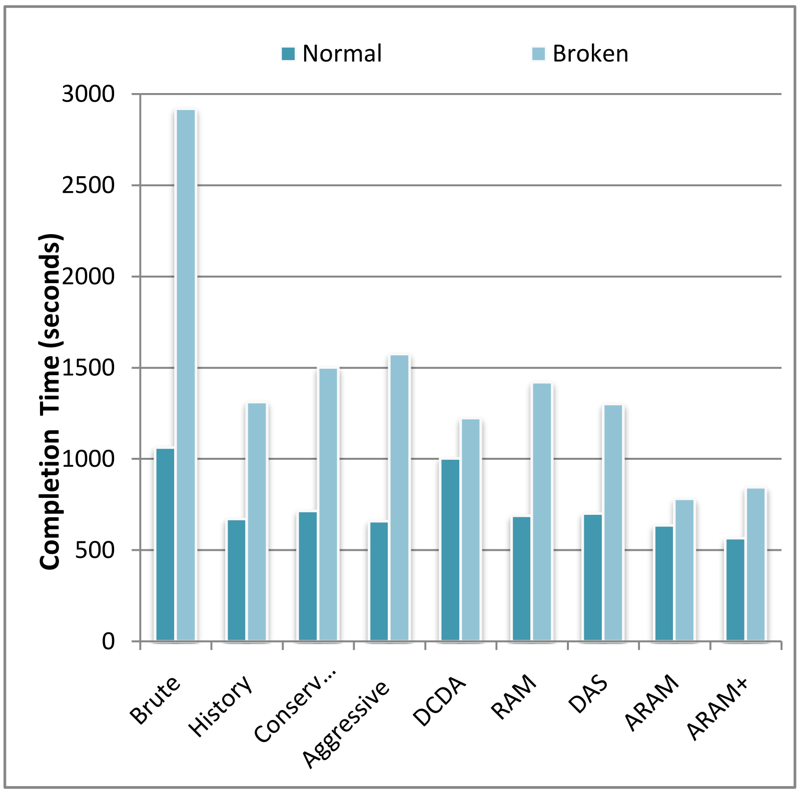


Figure 4.18: Compare 9 Schemes in Different Network Status

Table 4.6: Comparison for All Schemes

Schemes	Complexity	Overhead	Network Adaptability	Fault Tolerance
Brute-Force	Low	Non	Non	No
History-based	Low	Non	Non	No
Conservative Load Balancing	Middle	Few	Low	No
Aggressive Load Balancing	Middle	Few	Low	No
Dynamic Co-allocation with Duplicate Assignments	High	Very High	Middle	Yes
Recursively adjusting mechanism	Middle	Middle	High	No
Dynamic Adjustment Strategy	High	High	High	No
Anticipative recursively adjusting mechanism	High	Middle	High	Yes
Anticipative recursively adjusting mechanism plus	Very High	High	Very High	Yes

Chapter 5

Conclusions

Co-allocation architectures can be used to enable parallel transfers of data files from multiple replicas in data grids, which mean all replicas stored in the various grid sites. Many schemes based on the Co-Allocation Model have been proposed and used to exploit the different transfer rates among various client-server network links and to adapt to dynamic rate fluctuations by dividing data into fragments. In these schemes, the applicable piece fragments achieve more performance. In fact, some schemes can be applied to specific situations; however, most situations are not common actually. For this issue, we propose the anticipative recursively adjusting mechanism plus (ARAM+), based on ARAM. The best part is performance tuning through continual dynamic alpha value adjustment. It relies on special features to adapt to various network situations in data grid environments. The TCP Bandwidth Estimation Model was used to evaluate dynamic link states in our experiments by detecting TCP throughputs and packet lost rates between grid nodes. TCP Bandwidth Estimation Model also can be more reliable and fair than ARAM and any other schemes. Burst Mode function truly can increase transfer rates and speed up total performance especially considering congestion control. The ARAM+ not only adapts to the worst network links, but also speeds up the overall performance especially in wide-area grid networks.

Bibliography

[1] Eitan Altman, Dhiman Barman, Bruno Tuffin and Milan Vojnovic´ “Parallel TCP Sockets: Simple Model, Throughput and Validation”, *INFOCOM 2006,* April 2006.

[2] B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I.Foster, C. Kesselman, S. Meder, V. Nefedova, D.Quesnel, and S. Tuecke, “Data management and transfer in high-performance computational grid environments,” *Parallel Computing*, vol. 28, no. 5, pp.749-771, May 2002.

[3] R.S. Bhuvaneswaran, Y. Katayama, and N. Takahashi, “Dynamic Co-allocation Scheme for Parallel Data Transfer in Grid Environment,” *Proceedings of First International Conference on Semantics, Knowledge, and Grid (SKG 2005)*, pp. 17, 2005.

[4] R.S. Bhuvaneswaran, Y. Katayama, and N. Takahashi, “A Framework for an Integrated Co-allocator for Data Grid in Multi-Sender Environment,” *IEICE Transactions on Communications*, vol. E90-B, no. 4, pp. 742-749, 2007.

[5] Juerg Bolliger, Thomas Gross, and Urs Hengartner, “Bandwidth modelling for network-aware applications,” In *INFOCOM '99*, March 1999.

[6] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunszt, and M. Ripeanu, “Giggle: A Framework for Constructing Scalable Replica Location Services,” *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pp.1-17, November 2002.

[7] 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), pp.187-200, 2001.

[8] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, “Grid Information Services for Distributed Resource Sharing,” *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10’01),* pp.181-194, August 2001.

[9] K. Czajkowski, I. Foster, and C. Kesselman. “Resource Co-Allocation in Computational Grids,” *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8’99),* August 1999.

[10] I. Foster, C. Kesselman, S. Tuecke. “The Anatomy of the Grid: Enabling Scalable Virtual Organizations.” *International Journal of High Performance Computing Applications*, Vol. 15, No. 3, pp. 200-222, 2001.

[11] I. Foster, C. Kesselman, “Globus: A Metacomputing Infrastructure Toolkit,” *International Journal of High Performance Computing Applications,* Vol. 11, No. 2, pp. 115-128, 1997.

[12] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, and K. Stockinger, “Data Management in an International Data Grid Project,” *Proceedings of the First IEEE/ACM International Workshop on Grid Computing - Grid 2000, Bangalore*, pp. 77-90, India, December 2000.

[13] Thomas J. Hacker and Brian D. Athey, “The End-to-End Performance Effects of Parallel TCP Sockets on a Lossy Wide-Area Network,” *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002,* 10.1109/IPDPS.2002.1015527.

[14] Open Grid Forum, http://www.ogf.org/

[15] M. Mathis, J. Semke, J. Mahdavi and T. Ott. “The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm.” *Computer Communication Review,* volume 27, number3, July 1997.

[16] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. “Modeling TCP throughput: A simple model and its empirical validation,” *ACMSIGCOMM,* pp. 303-314, Volume 28 , Issue 4, ISSN 0146-4833, October 1998.

[17] H. Stockinger, A. Samar, B. Allcock, I. Foster, K. Holtman, and B. Tierney, “File and Object Replication in Data Grids,” *Journal of Cluster Computing,* 5(3), pp. 305-314, 2002.

[18] The Globus Alliance, http://www.globus.org/

[19] S. Vazhkudai and J. Schopf, “Using Regression Techniques to Predict Large Data Transfers,” *International Journal of High Performance Computing Applications (IJHPCA),* vol. 17, no. 3, pp. 249-268, August 2003.

[20] S. Vazhkudai and J. Schopf, “Predicting Sporadic Grid Data Transfers,” *Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11 ‘02),* pp. 188-196, July 2002.

[21] S. Vazhkudai, “Enabling the Co-Allocation of Grid Data Transfers,” *Proceedings of Fourth International Workshop on Grid Computing*, pp. 44-51, 17 November 2003.

[22] S. Venugopal, R. Buyya, and K. Ramamohanarao, “A Taxonomy of Data Grids for Distributed Data Sharing, Management, and Processing,” *ACM Computing Surveys*, vol. 38, issue 1, Article 3, March 2006.

[23] S. Vazhkudai, J. Schopf, and I. Foster, “Predicting the Performance of Wide Area Data Transfers,” *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS 2002)*, pp. 34-43, April 2002.

[24] S. Vazhkudai, S. Tuecke, and I. Foster, “Replica Selection in the Globus Data Grid,” *Proceedings of the 1st International Symposium on Cluster Computing and the Grid (CCGRID 2001)*, pp. 106-113, May 2001.

[25] C.M. Wang, C.C. Hsu, H.M. Chen, and J.J. Wu, “Efficient Multi-Source Data Transfer in Data Grids,” *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, pp. 421-424, 16-19 May 2006.

[26] Chao-Tung Yang, Yao-Chun Chi, Tsu-Fen Han and Ching-Hsien Hsu, “Redundant Parallel File Transfer with Anticipative Recursively-Adjusting Scheme in Data Grids”, *ICA3PP 2007*, pp. 242–253, 2007.

[27] C.T. Yang, I.H. Yang, and C.H. Chen, “Improve Dynamic Adjustment Mechanism in Co-Allocation Data Grid Environments,” *Proceedings of the 11th Workshop on Compiler Techniques for High-Performance Computing (CTHPC-11’05),* pp. 189-194, 17-18 March 2005.

[28] C.T. Yang, I.H. Yang, K.C. Li, and S.Y. Wang, “Improvements on Dynamic Adjustment Mechanism in Co-Allocation Data Grid Environments,” *The Journal of Supercomputing, Springer Netherlands*, vol. 40, no. 3, pp. 269-280, June 2007.

[29] C.T. Yang, C.H. Chen, K.C. Li, and C.H. Hsu, “Performance Analysis of Applying Replica Selection Technology for Data Grid Environments,” *PaCT 2005, Lecture Notes in Computer Science,* vol. 3603, pp. 278-287, Springer-Verlag, September 2005.

[30] C.T. Yang, I.H. Yang, K.C. Li, and C.H. Hsu “A Recursively-Adjusting Co-Allocation Scheme with Cyber-Transformer in Data Grids,” *Future Generation Computer Systems*, pp. 695-703, Volume 25, Issue 7, Elsevier B.V., 2008.

[31] C.T. Yang, S.Y. Wang, C.H. Lin, M.H Lee, and T.Y. Wu, “Cyber Transformer: A Toolkit for Files Transfer with Replica Management in Data Grid Environments,” *Proceedings of the Second Workshop on Grid Technologies and Applications (WoGTA’05)*, pp. 73-80, December 2005.

[32] C.T. Yang, S.Y. Wang, and C.P. Fu, “A Dynamic Adjustment Mechanism for Data Transfer in Data Grids,” *Network and Parallel Computing: IFIP International Conference, NPC 2007, Lecture Notes in Computer Science*, vol. 4672, pp. 61-70, Springer, ISSN 1611-3349, September 17-20, 2007.

[33] L. Yang, J. Schopf, and I. Foster, “Improving Parallel Data Transfer Times Using Predicted Variances in Shared Networks,” *Proceedings of the fifth IEEE International Symposium on Cluster Computing and the Grid, (CCGrid ‘05)*, pp. 734-742, 9-12 May 2005.

[34] X. Zhang, J. Freschl, and J. Schopf, “A Performance Study of Monitoring and Information Services for Distributed Systems”, *Proceedings of 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12 ‘03)*, pp. 270-282, August 2003.

[35] Frank Kelly, “Fairness and stability of end-to-end congestion Control”, *European Journal of Control 2003*, pp. 159-176, September 2003.

[36] Lefteris Mamatas, Tobias Harks and Vassilis Tsaoussidis, “Approaches to Congestion Control in Packet Networks,” *Journal of Internet Engineering* ,Vol. 1, No. 1, January 2007.