Journal of Network and Computer Applications 32 (2009) 834–845

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Enhancement of anticipative recursively adjusting mechanism for redundant parallel file transfer in data grids ☆

Chao-Tung Yang [a,*], Ming-Feng Yang [a,1], Wen-Chung Chiang [b]

[a] High-Performance Computing Laboratory, Department of Computer Science, Tunghai University, Taichung, Taiwan, ROC
[b] Department of Information Networking Technology, Hsiuping Institute of Technology, Taichung County, Taiwan, ROC

ARTICLE INFO

Article history:
Received 4 July 2008
Received in revised form
16 December 2008
Accepted 8 February 2009

Keywords:
Co-allocation
Data grid
Recursively adjusting
Parallel file transfer
Burst Mode

ABSTRACT

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 paper, 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.

1. Introduction

An increasing number of scientific applications, e.g., arising from Genomics, Proteomics, and Bioinformatics require exchanges of large volumes of data to support computation (Allcock et al., 2002; Czajkowski et al., 1999, 2001; Foster et al., 2001; Hoschek et al., 2000; Open Grid Forum; Stockinger et al., 2002; The Globus Alliance). 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 (Chervenak et al., 2001, 2002; Czajkowski et al., 1999, 2001; Foster and Kesselman, 1997; Yang et al., 2005, 2008; Zhang et al., 2003; Vazhkudai and Schopf, 2002, 2003; Yang et al., 2006). 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 (Vazhkudai, 2003) 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 (Yang et al., 2007b). Parallel downloading (Vazhkudai et al., 2002, 2001; Wang et al., 2006; Yang et al., 2007a) 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 (Vazhkudai, 2003; Venugopal et al., 2006; Vazhkudai et al., 2002). Several co-allocation strategies were addressed in previous works (Mathis et al., 1997; Yang et al., 2007a), but drawbacks remain, such as faster servers having to wait for the slowest one to deliver its final block. As shown in Mathis et al. (1997) and Padhye et al. (1998), 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.

☆ This work is supported in part by the National Science Council, Taiwan, ROC, under Grant nos. NSC 96-2221-E-029-019-MY3 and NSC 97-2622-E-029-003-CC2.
* Corresponding author. Tel.: +886 4 23590415; fax: +886 4 23591567.
E-mail addresses: ctyang@thu.edu.tw (C.-T. Yang), orson@mail.hit.edu.tw (M.-F. Yang), wcchiang@mail.hit.edu.tw (W.-C. Chiang).
[1] Computer Center, Hsiuping Institute of Technology, Taichung County, Taiwan, ROC.

1084-8045/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jnca.2009.02.002

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 (Yang et al., 2007a) 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.

In this paper, 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, its 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.

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

2. Background review and related work

2.1. Co-allocation architecture

The architecture proposed in Vazhkudai (2003) consists of three main components: an information service, a broker/co-allocator, and local storage systems. Fig. 1 shows co-allocation of data grid transfers, an extension of the basic template for resource management (Vazhkudai et al., 2001; Vazhkudai and Schopf, 2002) 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 (Czajkowski et al., 1999, 2001) and Replica Management Service (Czajkowski et al., 2001), then gets lists of physical file locations.

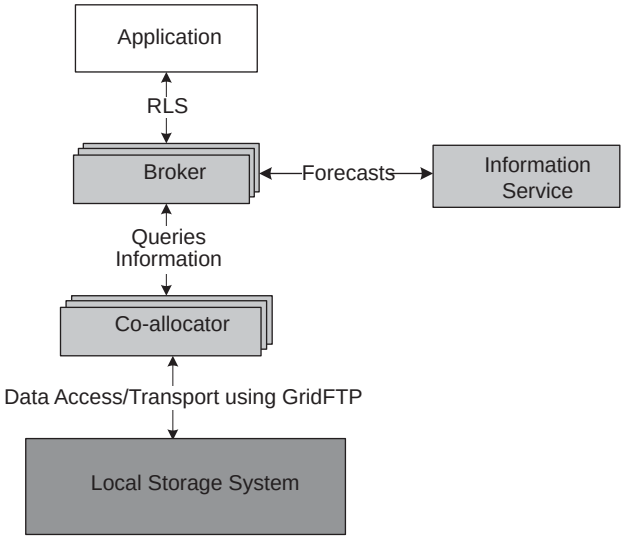

Fig. 1. Data grid co-allocation architecture.

2.1.1. Brute-force co-allocation

The Brute-force co-allocation scheme shown in Fig. 2 divides file sizes equally among available flows; it does not address bandwidth differences among various client–server links.

2.1.2. History-based co-allocation

The history-based co-allocation scheme shown in Fig. 3 keeps block sizes per flow proportional to predicted transfer rates, and disregards the influence of network variations between client and server.

2.1.3. Conservative load balancing

The conservative load balancing scheme shown in Fig. 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.

2.1.4. Aggressive load balancing

This method, shown in Fig. 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

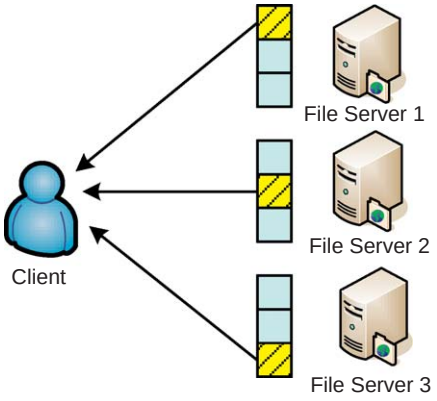

Fig. 2. The Brute-force co-allocation process.

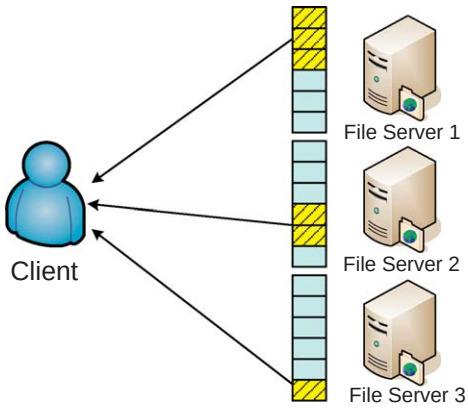

Fig. 3. The history-based co-allocation process.

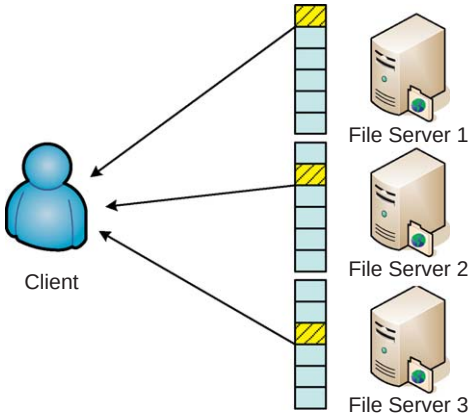

Fig. 4. The conservative load balancing process.

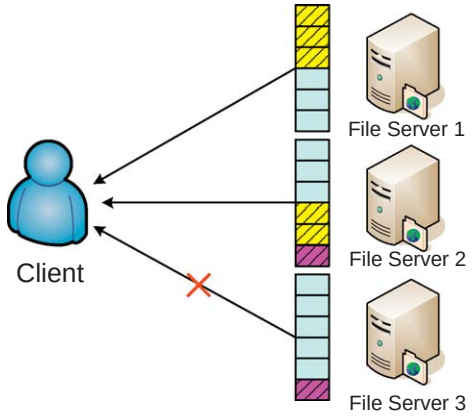

Fig. 5. The aggressive load balancing process.

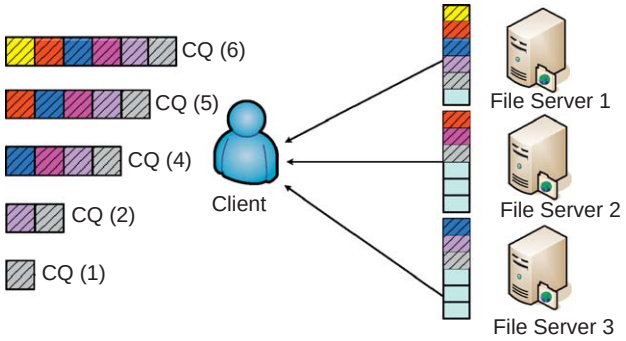

Fig. 6. The DCDA process.

data requested from slower servers or stopping requesting data from them altogether.

2.1.5. *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 (Bhuvaneswaran et al., 2005, 2007) and copies nicely with changes in server speed performance, as shown in Fig. 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.

2.1.6. *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 (Vazhkudai, 2003; The Globus Alliance), 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 Fig. 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 Fig. 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.

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.

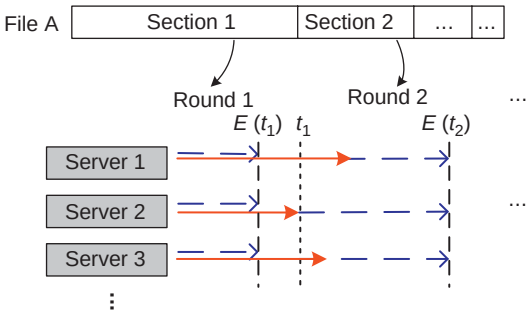

Fig. 7. The adjustment process.

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.

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 to extend and improve upon recursively adjusting co-allocation mechanism (Yang et al., in press). 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 Figs. 8 and 9). When the fastest server finishes at time t_1, the size of unfinished transfer blocks (italic blocks in Figs. 8 and 9) 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 Fig. 8) is then calculated.

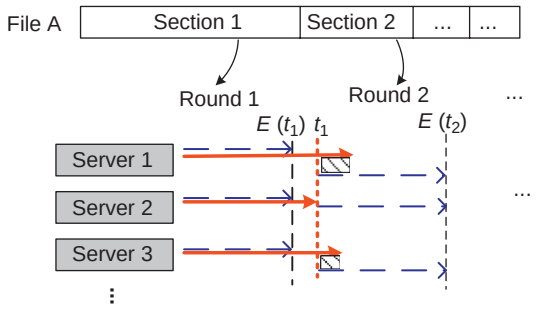

Fig. 8. Later-than-expected-time adjustment process.

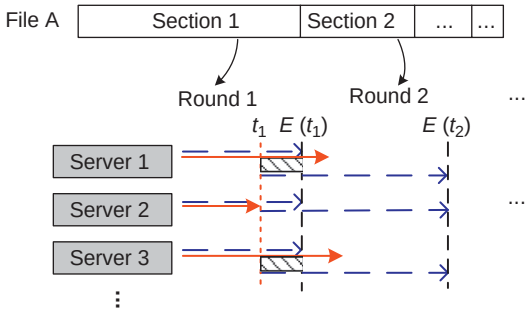

Fig. 9. Earlier-than-expected-time adjustment process.

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 Fig. 9) 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
- T_j: allocated time for section j
- SE_j: allocated size for section j
- *UnassignedFileSize*: portion of file A not yet distributed for downloading
- *UnfinishedFileSize*: size of unfinished blocks assigned in previous rounds
- B_{ji}: real-time transfer rate from the selected replica server
- r_j: transfer finish rate
- r_{j-1}: server transfer finish rate for previously assigned delivered file
- B_j: bandwidth available for section j
- S_{ji}: block size per flow from SE_j for each server i at time T_j
- ET_{ji}: expected time for server i at section j
- RT_{ji}: real finish time for server i at section j
- TS_{ji}: actual transfer size at real finish time RT_{ji}
- r_{ji}: 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 S_i 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

$$SE_j = (UnassignedFileSize + TotalUnfinishedFileSize)\alpha, \quad 0 < \alpha \leqslant 1 \tag{1}$$

where SE_j denotes section j such that $1 \leqq j \leqq k$, assume k time is allocated for downloading and there are k sections, while T_j 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, SE_j is divided into several blocks and assigned to "n" servers. Each server has a real-time transfer rate to the selected replica server of B_{ji}. r_{j-1} denotes the server transfer finish rate for previously assigned files, where the initial value is 1. The block size per flow from SE_j for each server "i" at time T_j is S_{ji}:

$$S_{ji} = \frac{SE_{ji}(B_{ji} \times r_j - i)}{\sum_{i=1}^{n}(B_{ji} \times r_j - i)}, \quad 0 \leqslant r_j - i \leqslant 1 \tag{2}$$

$$B_j = \sum_{i=1}^{n}(B_{ji} \times r_j - i) \tag{3}$$

$$ET_{ji} = \frac{S_{ji}}{B_{ji}} \tag{4}$$

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 RT_{ji}, or later or earlier than expected time ET_{ji}. Then TS_{ji} denoting the actual transfer size at real finish time RT_{ji} is given by

$$TS_{ji} = B_{ji} \times RT_{ji} \tag{5}$$

If the first finish time for RT_{ji} is earlier than expected time ET_{ji}, 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 TS_{ji} is greater than expected size S_{ji} at the first finish time, otherwise, the first finish time for RT_{ji} may be the result of the network link used remaining unchanged or deteriorating during the transfer:

$$r_{ji} = \begin{cases} \dfrac{TS_{ji}}{S_{ji}}, & RT_{ji} \geqslant ET_{ji} \\ 1, & RT_{ji} < ET_{ji}, \text{and } TS_{ji} \geqslant S_{ji} \end{cases} \tag{6}$$

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 (Altman et al., 2006). 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 (Altman et al., 2006; Bolliger et al., 1999; Hacker and Athey, 2002; Padhye et al., 1998). The TCP Bandwidth Estimation Model (Hacker and Athey, 2002) as a function to assessing TCP packet loss rate, such as round trip time, maximum segment size, other miscellaneous parameters, etc.

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

- $TCP_{BW(p)}$: 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 Eq. (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:

$$BW \leqslant \frac{MSS}{RTT}\frac{C}{\sqrt{p}} \tag{8}$$

In Eq. (8), *MSS*, *RTT*, and *p* are the same variables used in Eq. (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, $i = 1, 2, \ldots, k$, 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 $x \in S_i$. Eqs. (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, 10 servers with different

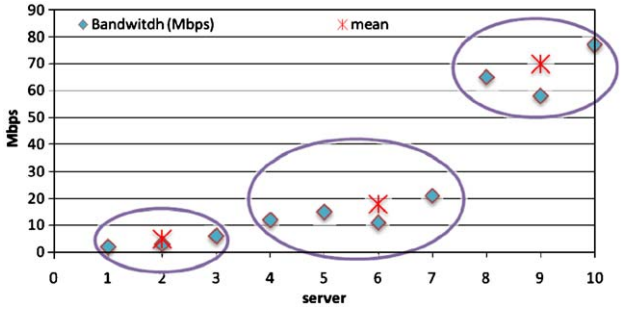

Fig. 10. 10 hosts classification according to bandwidth using *k*-means algorithm.

network bandwidths have been placed in three groups ($k = 3$). The simulation results are shown in Fig. 10:

- 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

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

$$xBoolean_{ij} = \begin{cases} 1 \text{ if } \|x_j = S_i\|^2 \leqslant \|x_j = S_k\|^2 & \forall k \neq 1 \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

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

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

3.4. Burst Mode

Like many network accelerator methods, and multithreading, Burst Mode 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 Eqs. (10)–(13)). Ultimately, the BM has made single jobs into many, as shown in Fig. 11.

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 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:

- $N_i\,TCP_{BW}$: candidate server bandwidth
- FTS: the fastest group of servers

$$N_i\,TCP_{BW} = \frac{MSS}{RTT}\frac{C}{\sqrt{p}} \tag{14}$$

$$FTS = S_i \max\{S_1, S_2, \ldots, S_n\}, m_i \in S_i \tag{15}$$

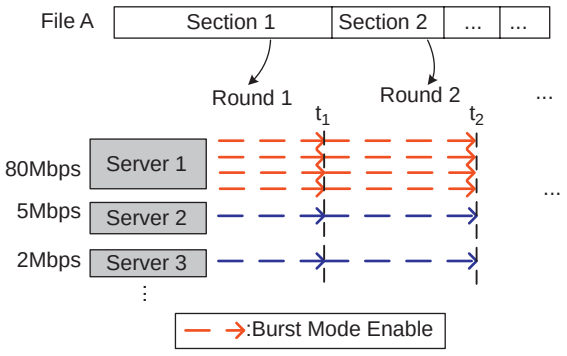

Fig. 11. Burst Mode enables higher bandwidths.

The algorithm is listed below:

[Initialization]
Measure bandwidths and find the fastest servers using Eqs. (14) and (15).
BigBlockUnit set to 100 MB initially
[Allocate blocks to the fastest servers and download via multithreading.]
Step 1: Group m_i and rank the most powerful server *FTS*
Step 2: Allocate SE_j and download via multithreading
Step 3: Monitor job progress statuses
LOOP WHEN (*UnassignedFileSize* and total *UnfinishedFileSize* are greater than *BigBlockUnit* (initial *BigBlockUnit* = 100 MB))
THEN
{
IF (Job finish rate is just 100% ($r_{ji} = 1$) and *UnassignedFileSize* and total *UnfinishedFileSize* are greater than *BigBlockUnit*)
THEN
{
Let data transfer in multiple parts between client and *FTS* server
$SE_j = (UnassignedFileSize + TotalfinishedFileSize)\alpha$, $0 < \alpha \leqslant 1$ ($UnassignedFileSize + TotalUnfinishedFileSize$) $\geqslant BigBlockUnit$
}
}
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 (Kelly, 2003; Mamatas et al., 2007), 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 (2003) suggestion of proportional fair allocation, although many others are possible. The mathematical expression (Eq. (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:

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

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 two areas: its TCP Bandwidth

Estimation Model (TCPBEM) and its Burst Mode. 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 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 (Vazhkudai et al., 2001; Wang et al., 2006; Yang et al., 2005, 2007b, in press), the anticipative recursively adjusting mechanism 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

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

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

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 Eqs. (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 α are greater then $TotalTCP_{BW}$)
THEN
{
Define new section for allocation
$SE_j = (UnassignedFileSize + TotalUnfinishedFileSize)\alpha$, $0 < \alpha \leqslant 1$
}
ELSE
{
Define finial section
$SE_j = UnassignedFileSize + TotalUnfinishedFileSize$
}
}
END LOOP;
Step 1: Define new section for allocation SE_j
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 R_{j-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 RT_{ji} is earlier then expected time ET_{ji} and transferred size TS_{ji} is greater than expected size S_{ji}) THEN
{
The $r_{ji} = 1$
}
ELSE
{
Measure the finish rate for the previously delivered file ($0 \leqslant r_{ji} \leqslant 1$)
}

$$r_{ji} = \begin{cases} \dfrac{TS_{ji}}{S_{ji}}, & RT_{ji} \geqslant Et_{ji} \\ 1, & RT_{ji} < ET_{ji}, \text{ and } TS_{ji} \geqslant S_{ji} \end{cases}$$

}
END LOOP;

4. Experimental

4.1. Our grid environment: Tiger grid

The experiments in this work were conducted and evaluated on the TigerGrid, which consists of more than 100 processors distributed over 10 clusters located at 5 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 Fig. 12. Fig. 13 shows statuses for all machines used in the grid testbed on one monitor page.

They are interconnected by the 1 Gbps Taiwan Academic Network (TANET). The Tiger grid platform is built around 60 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 km from THU. All machines have Globus 4.0.7 or above installed.

We performed wide-area data transfer experiments using Cyber Transformer, our GridFTP GUI client tool, on our co-allocation testbed at Tunghai University (THU), Taichung City, Taiwan, and fetched files from replica servers at National Da-Li Senior High School (DL), Li-Zen High School (LZ), Tungs' Taichung Metro Harbor Hospital (TUNG), and Hsiuping Institute of Technology School (HIT). These institutions are all in Taichung, Taiwan, 10–30 km from THU.

4.2. Our experimental tool: Cyber Transformer

In a previous work Yang et al. (2006), we gave experimental results for Cyber Transformer, a powerful new toolkit for replica

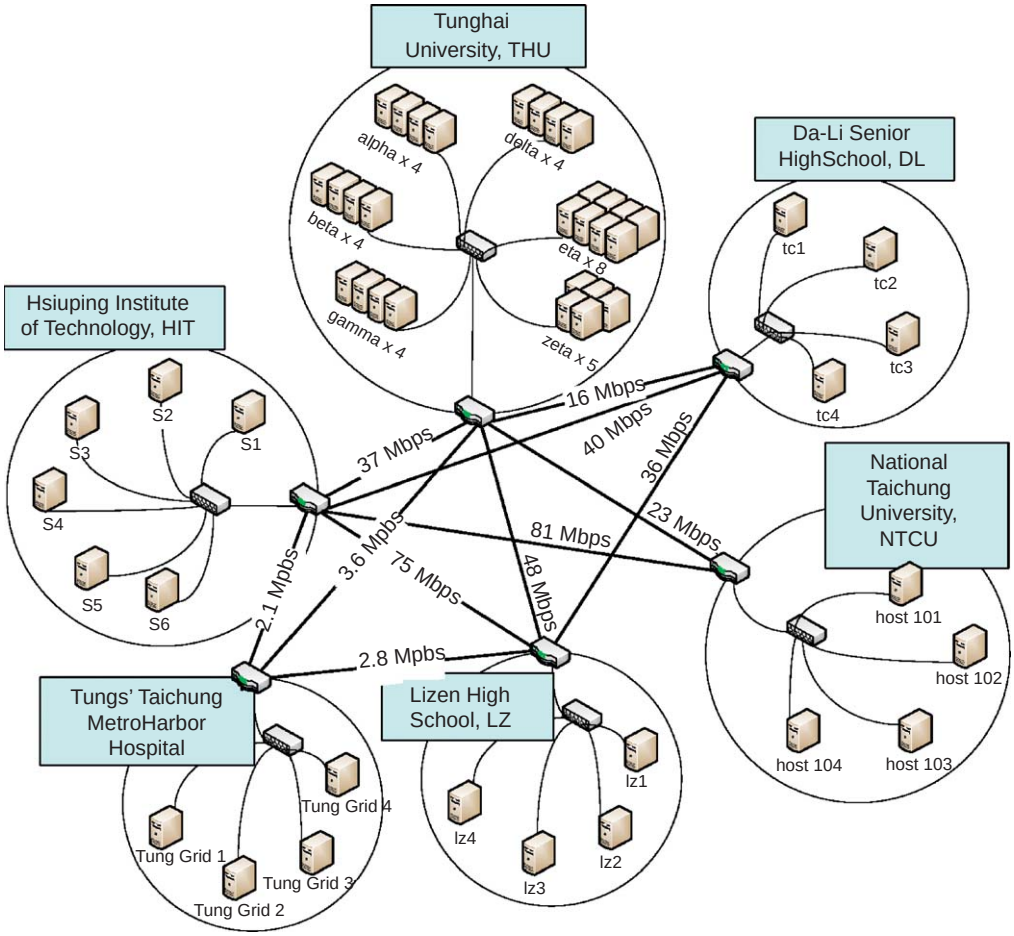

Fig. 12. Tiger grid network.

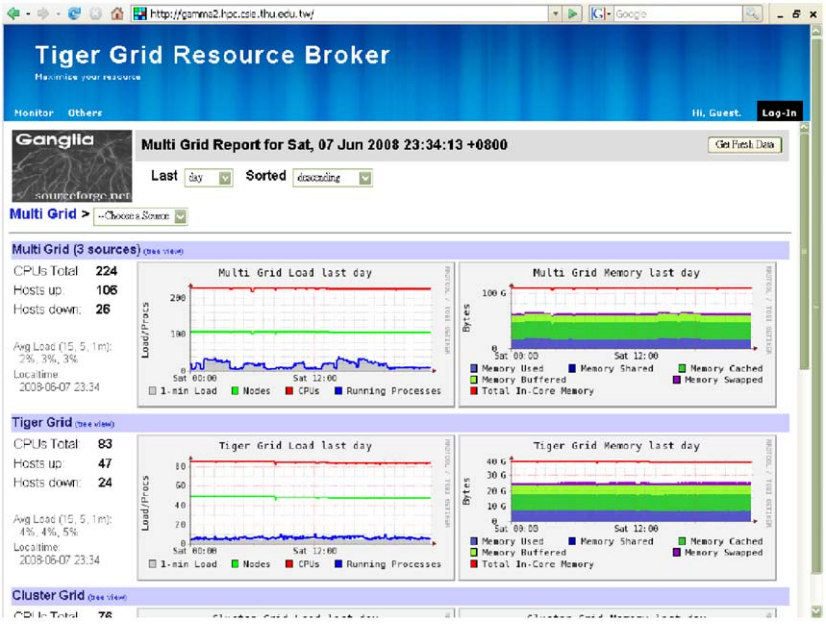

Fig. 13. Tiger grid resources.

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

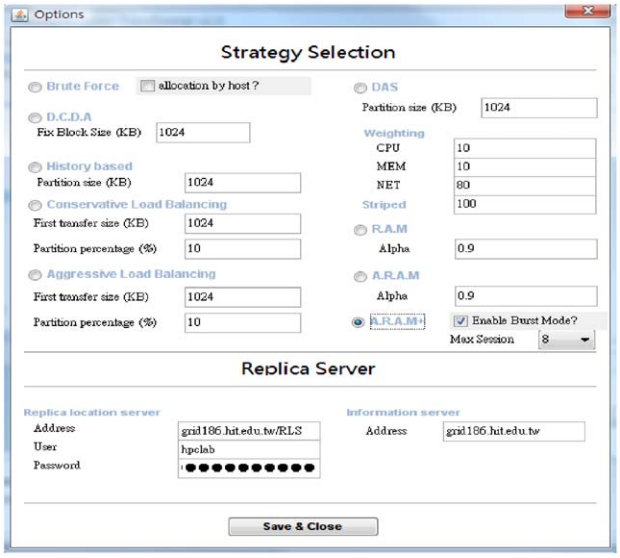

Fig. 14. Parallel download strategy selection.

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 Fig. 14.

4.3. Experimental results and analyses

An experiment and a case design were devised to test Burst Mode, our proposed approach to speeding up local and remote performance, and dynamically adjusting alpha values to adapt to variable network situations. Details of the test cases we designed are shown in Fig. 15.

4.3.1. Case study—"cross-grid" vs. "local grid" replica selects and transfers

We designed two scenarios to verify the efficiency of enabling Burst Mode. All test cases are listed in Tables 1 and 2.

Generally, more replicas and local placement will yield better parallel file transfer performance. Our results, shown in Figs. 16 and 17, 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).

4.3.2. Case study—RAM and ARAM vs. ARAM+

RAM (Yang et al., 2007c) and ARAM (Yang et al., 2007a) 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. 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 3.

In our next experiment, two scenarios, sets A and B, are listed in Table 4 and 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 Figs. 18 and 19.

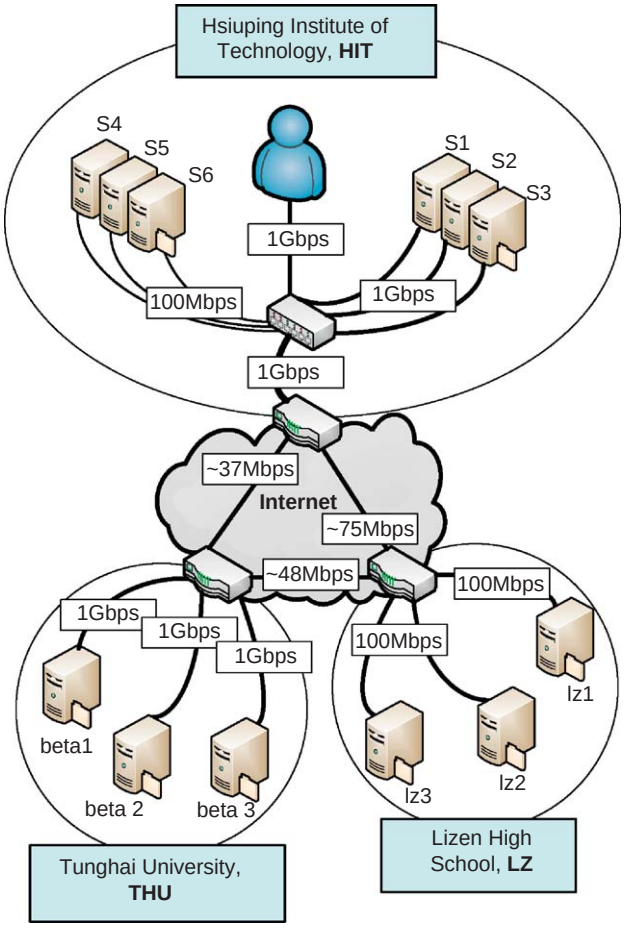

Fig. 15. Scenarios for our test-bed of Tiger grid.

Table 1
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-beta 1; LZ-1
ARAMplus_4: local-3	HIT-S1, S2; LZ-1, 2
ARAMplus_4: all-local	HIT-S1, S2, S3, S4

Table 2
Scenario for various replica numbers and selections.

Scenario	Replica server list
R × 6_non-local	LZ-1, 2, 3; THU-beta 1, beta 2, beta 3
R × 6_local	HIT-S1, S2, S3, S4, S5, S6
R × 2_local	HIT-S1, S2
R × 2_non-local-THU	THU-S1, S2
R × 2_non-local-LZ	LZ-S1, S2

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-

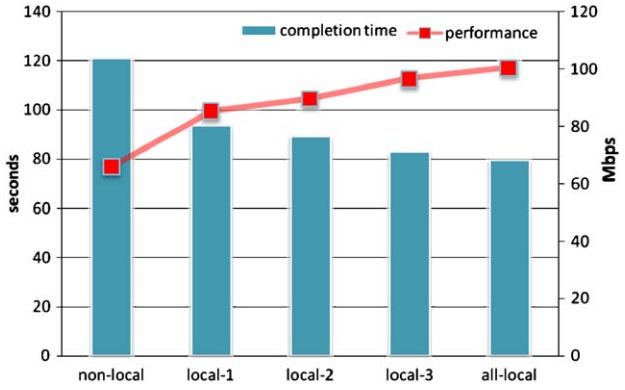

Fig. 16. Effects of various replica locations on performance results.

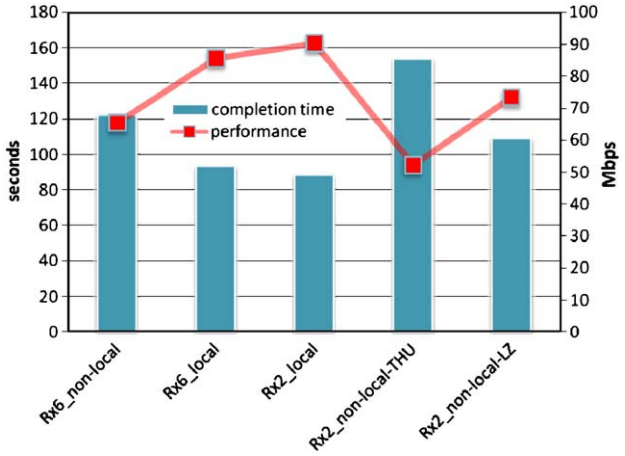

Fig. 17. Effects of various replica numbers and selections on performance results.

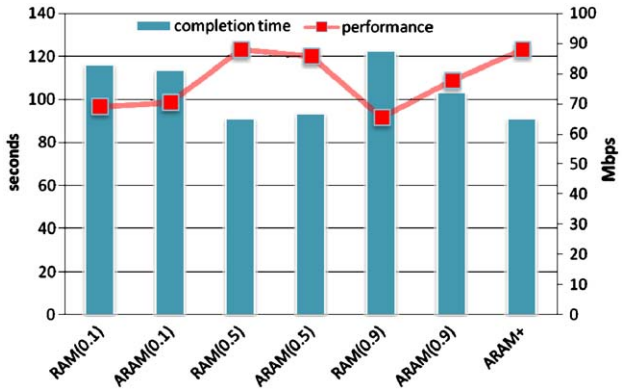

Fig. 18. Performance results for scenario A.

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 3 for each scheme, we analyzed their performance by comparing transfer finish times and overall performance, as shown Figs. 20 and 21.

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

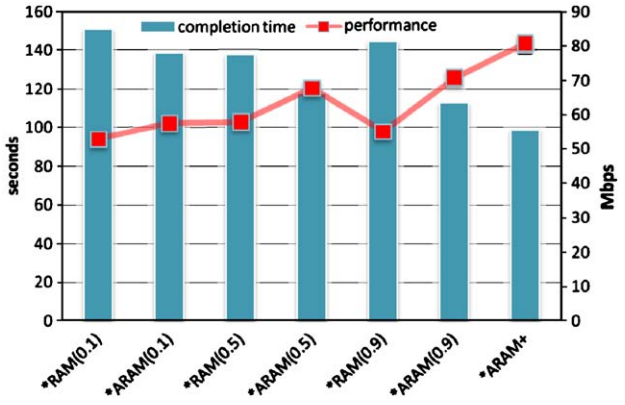

Fig. 19. Performance results for scenario B.

Table 3
Replica placement and selection plan.

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

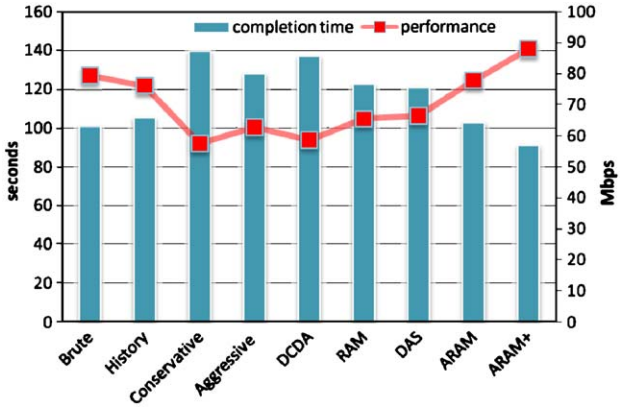

Fig. 20. Comparing 9 schemes on "local" cases.

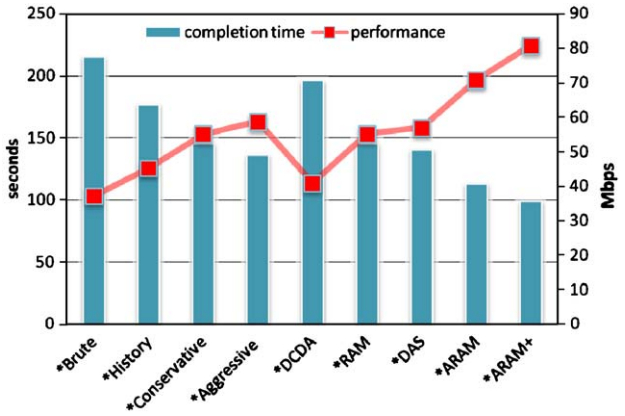

Fig. 21. Comparing 9 schemes on "mixed" cases.

comparable to Brute or any others. The advantages of ARAM+ are the following:

- ARAM+ uses TCP bandwidth measurement technology, reliability and accuracy of the best.
- ARAM+ can enhance GridFTP to become multiplexing.

Table 4
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

- 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.

5. Conclusion

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 scheme. 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.

References

Allcock B, Bester J, Bresnahan J, Chervenak A, Foster I, Kesselman C, et al. Data management and transfer in high-performance computational grid environments. Parallel Computing 2002;28(5):749–71.

Altman Eitan, Barman Dhiman, Tuffin Bruno, Vojnovic Milan. Parallel TCP sockets: simple model, throughput and validation. In: INFOCOM 2006, April 2006.

Bhuvaneswaran RS, Katayama Y, Takahashi N. Dynamic co-allocation scheme for parallel data transfer in grid environment. In: Proceedings of first international conference on semantics, knowledge, and grid (SKG 2005), 2005. p. 17.

Bhuvaneswaran RS, Katayama Y, Takahashi N. A framework for an integrated co-allocator for data grid in multi-sender environment. IEICE Transactions on Communications 2007;E90-B(4):742–9.

Bolliger Juerg, Gross Thomas, Hengartner Urs. Bandwidth modelling for network-aware applications. In: INFOCOM '99, March 1999.

Chervenak A, Foster I, Kesselman C, Salisbury C, Tuecke S. The data grid: towards an architecture for the distributed management and analysis of large scientific datasets. Journal of Network and Computer Applications 2001;23(3):187–200.

Chervenak A, Deelman E, Foster I, Guy L, Hoschek W, Iamnitchi A, et al. Giggle: a framework for constructing scalable replica location services. In: Proceedings of the 2002 ACM/IEEE conference on supercomputing, November 2002. p. 1–17.

Czajkowski K, Foster I, Kesselman C. Resource co-allocation in computational grids. In: Proceedings of the eighth IEEE international symposium on high performance distributed computing (HPDC-8 '99), August 1999.

Czajkowski K, Fitzgerald S, Foster I, Kesselman C. Grid information services for distributed resource sharing. In: Proceedings of the tenth IEEE international symposium on high-performance distributed computing (HPDC-10 '01), August 2001. p. 181–94.

Foster I, Kesselman C. Globus: a metacomputing infrastructure toolkit. International Journal of High Performance Computing Applications 1997;11(2):115–28.

Foster I, Kesselman C, Tuecke S. The anatomy of the grid: enabling scalable virtual organizations. International Journal of High Performance Computing Applications 2001;15(3):200–22.

Hacker Thomas J, Athey Brian D. The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network, parallel and distributed processing symposium. In: Proceedings international, IPDPS 2002, 10.1109/IPDPS.2002.1015527.

Hoschek W, Jaen-Martinez J, Samar A, Stockinger H, Stockinger K. Data management in an international data grid project. In: Proceedings of the first IEEE/ACM international workshop on grid computing-grid 2000, Bangalore, India, December 2000.

Kelly Frank. Fairness and stability of end-to-end congestion control. European Journal of Control 2003:159–76.

Mamatas Lefteris, Harks Tobias, Tsaoussidis Vassilis. Approaches to congestion control in packet networks. Journal of Internet Engineering 2007;1(1):2.

Mathis M, Semke J, Mahdavi J, Ott T. The macroscopic behavior of the TCP congestion avoidance algorithm. Computer Communication Review 1997;27(3).

Open Grid Forum. 〈http://www.ogf.org/〉.

Padhye J, Firoiu V, Towsley D, Kurose J. Modeling TCP throughput: a simple model and its empirical validation. In: ACMSIGCOMM, September 1998.

Stockinger H, Samar A, Allcock B, Foster I, Holtman K, Tierney B. File and object replication in data grids. Journal of Cluster Computing 2002;5(3):305–14.

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

Vazhkudai S. Enabling the co-allocation of grid data transfers. In: Proceedings of fourth international workshop on grid computing, 17 November 2003. p. 44–51.

Vazhkudai S, Schopf J. Predicting sporadic grid data transfers. In: Proceedings of 11th IEEE international symposium on high performance distributed computing (HPDC-11 '02), July 2002. p. 188–96.

Vazhkudai S, Schopf J. Using regression techniques to predict large data transfers. International Journal of High Performance Computing Applications (IJHPCA) 2003;17(3):249–68.

Vazhkudai S, Tuecke S, Foster I. Replica selection in the globus data grid. In: Proceedings of the first international symposium on cluster computing and the grid (CCGRID 2001), May 2001. p. 106–13.

Vazhkudai S, Schopf J, Foster I. Predicting the performance of wide area data transfers. In: Proceedings of the 16th international parallel and distributed processing symposium (IPDPS 2002), April 2002. p. 34–43.

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

Wang CM, Hsu CC, Chen HM, Wu JJ. Efficient multi-source data transfer in data grids. In: Proceedings of the sixth IEEE international symposium on cluster computing and the grid (CCGRID '06), 16–19 May 2006. p. 421–4.

Yang CT, Chen CH, Li KC, Hsu CH. Performance analysis of applying replica selection technology for data grid environments. In: PaCT 2005, Lecture Notes in Computer Science, vol. 3603. Berlin: Springer; 2005. p. 278–87.

Yang CT, Yang IH, Chen CH, Wang SY. Implementation of a dynamic adjustment mechanism with efficient replica selection in co-allocation data grid environments. Proceedings of the 21st Annual ACM Symposium on Applied Computing (SAC 2006) – Distributed Systems and Grid Computing Track, France, April 23–27, 2006. pp. 797–804.

Yang CT, Chi YC, Han TF, Hsu CH. Redundant parallel file transfer with anticipative recursively-adjusting scheme in data grids. Distributed and Parallel Computing: 7th International Conference on Algorithms and Architectures for Parallel Processing, ICA3PP 2007, Lecture Notes in Computer Science, vol. 4494, 2007a. p. 242–53.

Yang CT, Yang IH, Li KC, Wang SY. Improvements on dynamic adjustment mechanism in co-allocation data grid environments. Journal of Supercomputing 2007b;40(3):269–80.

Yang CT, Wang SY, Fu CP. A dynamic adjustment mechanism for data transfer in data grids. In: Network and parallel computing: IFIP international conference, NPC 2007, Lecture Notes in Computer Science, vol. 4672, September 17–20. Berlin: Springer; 2007c. p. 61–70, ISSN 1611-3349.

Yang CT, Yang MF, Chiang WC. Implementation of a cyber transformer for parallel download in co-allocation data grid environments. Proceedings of the 7th International Conference on Grid and Cooperative Computing (GCC2008) and Second EchoGRID Conference, October 24–26, 2008, in Shenzhen, Guangdong, China. pp. 242–53.

Yang CT, Yang IH, Wang SY, Hsu CH, Li KC. A recursively-adjusting co-allocation scheme with cyber-transformer in data grids. Future Generation Computer Systems, in press (available online 21 January 2007).

Yang L, Schopf J, Foster I. Improving parallel data transfer times using predicted variances in shared networks. In: Proceedings of the fifth IEEE international symposium on cluster computing and the grid (CCGrid '05), 9–12 May 2005. p. 734–42.

Zhang X, Freschl J, Schopf J. A performance study of monitoring and information services for distributed systems. In: Proceedings of 12th IEEE international symposium on high performance distributed computing (HPDC-12 '03), August 2003. p. 270–82.