J Supercomput (2007) 40: 269–280
DOI 10.1007/s11227-006-0022-3

Improvements on dynamic adjustment mechanism in co-allocation data grid environments

Chao-Tung Yang · I-Hsien Yang · Kuan-Ching Li · Shih-Yu Wang

Published online: 31 March 2007

Abstract Several co-allocation strategies have been coupled and used to exploit rate differences among various client-server links and to address dynamic rate fluctuations by dividing files into multiple blocks of equal sizes. However, a major obstacle, the idle time of faster servers having to wait for the slowest server to deliver the final block, makes it important to reduce differences in finishing time among replica servers. In this paper, we propose a dynamic co-allocation scheme, namely *Recursive-Adjustment Co-Allocation* scheme, to improve the performance of data transfer in Data Grids. Our approach reduces the idle time spent waiting for the slowest server and decreases data transfer completion time.

Keywords Data Grid · Dynamic · Recursive · GridFTP · Co-allocation · Data transfer

1 Introduction

In Data Grid environments, access to distributed data is typically as important as access to distributed computational resources [1, 2, 4, 7, 9]. Distributed scientific and

C.-T. Yang (✉) · I.-H. Yang · S.-Y. Wang
High-Performance Computing Laboratory, Department of Computer Science and Information Engineering, Tunghai University, Taichung City 40704, Taiwan R.O.C.
e-mail: ctyang@thu.edu.tw

I.-H. Yang
e-mail: g922906@thu.edu.tw

S.-Y. Wang
e-mail: g932813@thu.edu.tw

K.-C. Li
Department of Computer Science and Information Engineering, Providence University, Taichung 43301, Taiwan R.O.C.
e-mail: kuancli@pu.edu.tw

engineering applications require transfers of large amounts of data between storage systems, and access to large amounts of data generated by many geographically distributed applications and users for analysis and visualization, among others. Unfortunately, the lack of standard protocols for transferring and accessing data in Grids has led to a fragmented Grid storage community. Users wishing to access various storage systems are forced to use multiple protocols, and it is difficult to transfer data efficiently between these various storage systems [2, 9].

Replicating popular content in distributing servers is widely used in practice [11, 13, 15]. Recently, large-scale, data-sharing scientific communities such as those described in [1, 4] used this technology to replicate their large datasets over several sites. Downloading large datasets from several replica locations may result in varied performance rates, because the replica sites may have different architectures, system loadings, and network connectivity. Bandwidth quality is the most important factor affecting transfers between clients and servers since download speeds are limited by the bandwidth traffic congestion in the links connecting the servers to the clients [17, 18].

The co-allocation of data transfers enables the clients to download data from multiple locations by establishing multiple connections in parallel [6, 13]. This can improve the performance compared to the single-server cases and alleviate the internet congestion problem [13]. Several co-allocation strategies were provided in the previous work [13]. An idle-time drawback remains since faster servers must wait for the slowest server to deliver its final block. Therefore, it is important to reduce the differences in finish time among replica servers. In this paper, we propose a dynamic co-allocation scheme based on co-allocation Grid data transfer architecture called *Recursive-Adjustment Co-Allocation* that can reduce the idle time spent waiting for the slowest server and improves data transfer performance. Experimental results show that our approach is superior to previous methods and achieved the best overall performance.

The remainder of this paper is organized as follows. Related studies are presented in Sect. 2 and the co-allocation architecture is introduced in Sect. 3. Our research approaches are outlined in Sect. 4, and experimental results and a performance evaluation of our scheme are presented in Sect. 5. Section 6 concludes this research paper.

2 Related work

Data grids consist of scattered computing and storage resources located in different countries/regions yet accessible to users [7]. In this study we used the grid middleware Globus Toolkit [8, 10, 12] as the data grid infrastructure. The Globus Toolkit provides solutions for such considerations as security, resource management, data management, and information services. One of its primary components is MDS [5, 8, 10, 12, 20], which is designed to provide a standard mechanism for discovering and publishing resource status and configuration information. It provides a uniform and flexible interface for data collected by lower-level information providers in two modes: static (e.g., OS, CPU types, system architectures) and dynamic data (e.g., disk availability, memory availability, and loading). And it uses GridFTP [1, 8, 12], a reliable, secure, and efficient data transport protocol to provide efficient management and transfer of terabytes or petabytes of data in a wide-area, distributed-resource environment.

As datasets are replicated within Grid environments for reliability and performance, clients require the abilities to discover existing data replicas, and create and register new replicas. A Replica Location Service (RLS) [3, 15] provides a mechanism for discovering and registering existing replicas. Several prediction metrics have been developed to help replica selection. For instance, Vazhkudai and Schopf [14, 16, 17] used past data transfer histories to estimate current data transfer throughputs.

In our previous work [19], we proposed a replica selection cost model and a replica selection service to perform replica selection. In [13], the author proposes a co-allocation architecture for co-allocating Grid data transfers across multiple connections by exploiting the partial copy feature of GridFTP. It also provides *Brute-Force*, *History-Base*, and *Dynamic Load Balancing* for allocating data block. *Brute-Force Co-Allocation* works by dividing file sizes equally across available flows without addressing bandwidth differences among the various client-server links. The *History-based Co-Allocation* scheme keeps block sizes per flow proportional to predicted transfer rates.

The *Conservative Load Balancing* dynamic co-allocation strategy divides requested datasets into "*k*" disjoint blocks of equal size. Available servers are assigned single blocks to deliver in parallel. When a server finishes delivering a block, another is requested, and so on, till the entire file is downloaded. The loadings on the co-allocated flows are automatically adjusted because the faster servers will deliver more quickly providing larger portions of the file. The *Aggressive Load Balancing* dynamic co-allocation strategy presented in [13] adds functions that change block size de-liveries by: (1) progressively increasing the amounts of data requested from faster servers, and (2) reducing the amounts of data requested from slower servers or ceasing to request data from them altogether.

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. In most cases, this wastes much time and decreases overall performance. Thus, we propose an efficient approach called *Recursive-Adjustment Co-Allocation* and based on a co-allocation architecture. It improves dynamic co-allocation and reduces waiting time, thus improving overall transfer performance.

3 Co-allocation architecture

The co-allocation of data transfers enables the clients to download data from multiple locations by establishing multiple connections in parallel [6, 13]. Figure 1 shows the co-allocation of data transfers in Grid, which is an extension of the basic template for resource management [6] provided by Globus Toolkit. The architecture consists of three main components: an information service, broker/co-allocator, and local storage systems. Applications specify the characteristics of desired data and pass the attribute description to a broker. The broker queries available resources and gets replica locations from information services [5] and replica management services [13, 15], and then gets a list of physical locations for the desired files.

The candidate replica locations are passed to a replica selection service [13, 15], which was presented in a previous work [19]. This replica selection service provides estimates of candidate transfer performance based on a cost model and chooses ap-

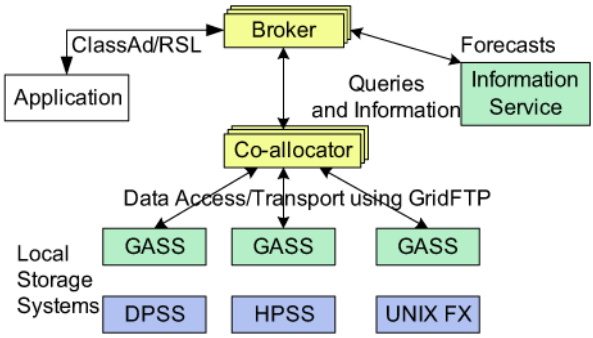

Fig. 1 Data Grid co-allocation architecture [13, 15]

propriate amounts to request from the better locations. The co-allocation agent then downloads the data in parallel from the selected servers. In this research, we use GridFTP [1, 8, 12] to enable parallel data transfers. GridFTP is a high-performance, secure, reliable data transfer protocol optimized for high-bandwidth wide-area networks. Among its many features are security, parallel streams, partial file transfers, third-party transfers, and reusable data channels. Its partial file transfer ability allows files to be retrieved from data servers by specifying the start and end offsets of file sections.

4 Dynamic co-allocation strategy

Dynamic co-allocation, described above, is the most efficient approach to reducing the influence of network variations between clients and servers. However, the idle time of faster servers awaiting the slowest server to deliver the last block is still a major factor affecting overall efficiency, which *Conservative Load Balancing* and *Aggressive Load Balancing* [13] cannot effectively avoid. The approach proposed in the present paper, a dynamic allocation mechanism called "*Recursive-Adjustment Co-Allocation*" can overcome this, and thus, improve data transfer performance.

Our *Recursive-Adjustment Co-Allocation* can continuously adjust each replica server's workload to correspond to its real-time bandwidth during file transfers. The goal is to make the expected finish time of all servers the same. As Fig. 2 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 the blocks to servers for transfer. At this moment, it is expected that the transfer finish time will be consistent at $E(T_1)$. However, since server bandwidths may fluctuate during segment deliveries, actual completion time may be dissimilar (solid line, in Fig. 2). Once the quickest server finishes its work at time T_1, the next section is assigned to the servers again. This allows each server to finish its assigned work-load by the expected time at $E(T_2)$. These adjustments are repeated until the entire file transfer is finished.

The *Recursive-Adjustment Co-Allocation* process is illustrated in Fig. 3. When a user requests file A, the replica selection service responds with the subset of all available servers defined by the maximum performance matrix. The co-allocation service gets this list of selected replica servers. Assuming n replica servers are selected, S_i denotes server "i" such that $1 \leqq i \leqq n$. A connection for file downloading is then built to each server.

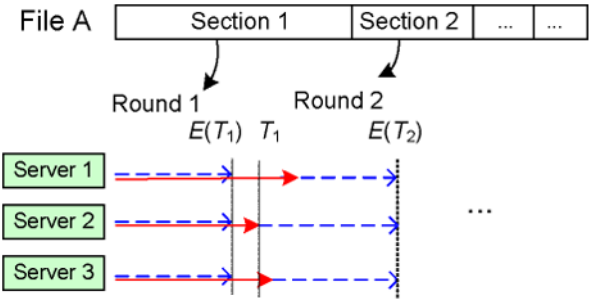

Fig. 2 The adjustment process

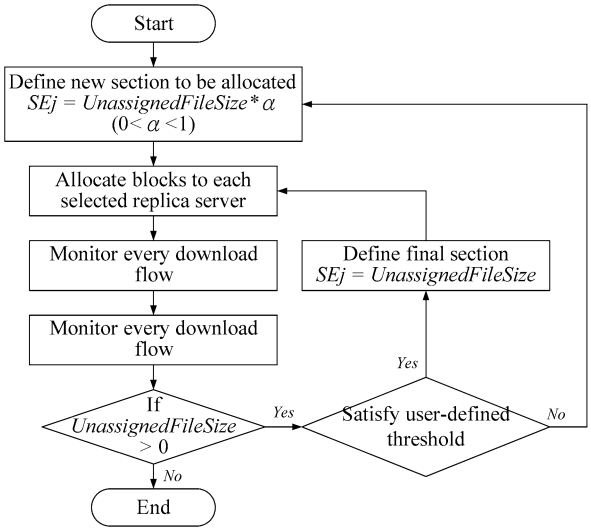

Fig. 3 The flowchart of recursive-adjustment co-allocation

The *Recursive-Adjustment Co-Allocation* process is as follows. A new section of a file to be allocated is first defined. The section size, "SE_j," is:

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

where SE_j denotes the section j such that $1 \leqq j \leqq k$, assuming we allocate k times for the download process, and thus, there are k sections, while T_j denotes the time section j allocated. *UnassignedFileSize* is the portion of file A not yet distributed for downloading; initially, *UnassignedFileSize* is equal to the total size of file A. α is the rate that determines 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 client of B_i, which is measured by the Network Weather Service (NWS) [18]. The block size per flow from SE_j for each server "i" at time T_j is:

$$S_i = (SE_j + \sum_{i=1}^{n} UnFinishSize_i) \times B_i / \sum_{i=1}^{n} B_i - UnFinishSize_i \tag{2}$$

where $UnFinishSize_i$ denotes the size of unfinished transfer blocks that is assigned in previous rounds at server "i." $UnFinishSize_i$ is equal to zero in first round. Ideally, depending to the real time bandwidth at time T_j, every flow is expected to finish its workload in future.

This fulfills our requirement to minimize the time faster servers must wait for the slowest server to finish. If, in some cases, network variations greatly degrade transfer rates, $UnFinishSize_i$ may exceed $(SE_j + \sum_{i=1}^{n} UnFinishSize_i) * B_i / \sum_{i=1}^{n} B_i$, which is the total block size expected to be transferred after T_j. In such cases, the co-allocator eliminates the servers in advance and assigns SE_j to other servers.

After allocation, all channels continue transferring data blocks. When a faster channel finishes its assigned data blocks, the co-allocator begins allocating an unassigned section of file A again. The process of allocating data blocks to adjust expected flow finish time continues until the entire file has been allocated.

Our approach gets new sections from whole files by dividing unassigned file ranges in each round of allocation. These unassigned portions of the file ranges become smaller after each allocation. Since adjustment is continuous, it would run as an endless loop if not limited by a stop condition.

However, when is it appropriate to stop continuous adjustment? We provide two monitoring criteria, *LeastSize* and *ExpectFinishedTime*, to enable users to define stop thresholds. When a threshold is reached, the co-allocation server stops dividing the remainder of the file and assigns that remainder as the final section. The *LeastSize* criterion specifies the smallest file we want to process, and when the unassigned portion of *UnassignedFileSize* drops below the *LeastSize* specification, division stops. *ExpectFinishedTime* criterion specifies the remaining time transfer is expected to take. When the expected transfer time of the unassigned portion of a file drops below the time specified by *ExpectFinishedTime*, file division stops. The expected rest time value is determined by:

$$UnAssignedFileSize / \sum_{i=1}^{n} B_i \tag{3}$$

5 Experimental results and analyses

In this section, we evaluate four co-allocation schemes: (1) *Brute-Force (Brute)*, (2) *History-based (History)*, (3) *Conservative Load Balancing (Conservative)* and (4) *Recursive-Adjustment Co-Allocation (Recursive)*. We analyze the performance of each scheme by comparing their transfer finish time, and the total idle time faster servers spent waiting for the slowest server to finish delivering the last block. Figure 4 shows a snapshot of our GridFTP client tool.

In our example, we assumed that a client site at Tunghai University (THU), Taichung city, Taiwan, was fetching a file from three selected replica servers: one at Providence University (PU), one at Li-Zen High School (LZ), and one at Da-Li High School (DALI). We monitored the bandwidth variations from THU to each server using NWS [18] probes. Network environment variations of each connection are shown in Fig. 5.

We assign $\alpha = 0.5$ and experiment it over several file sizes, such as 500 MB, 1000 MB, 2000 MB, and 4000 MB. We set the *LeastSize* limit threshold to 100 MB, which result in 12, 15, 17, and 19 block numbers. As mater of comparison, we use

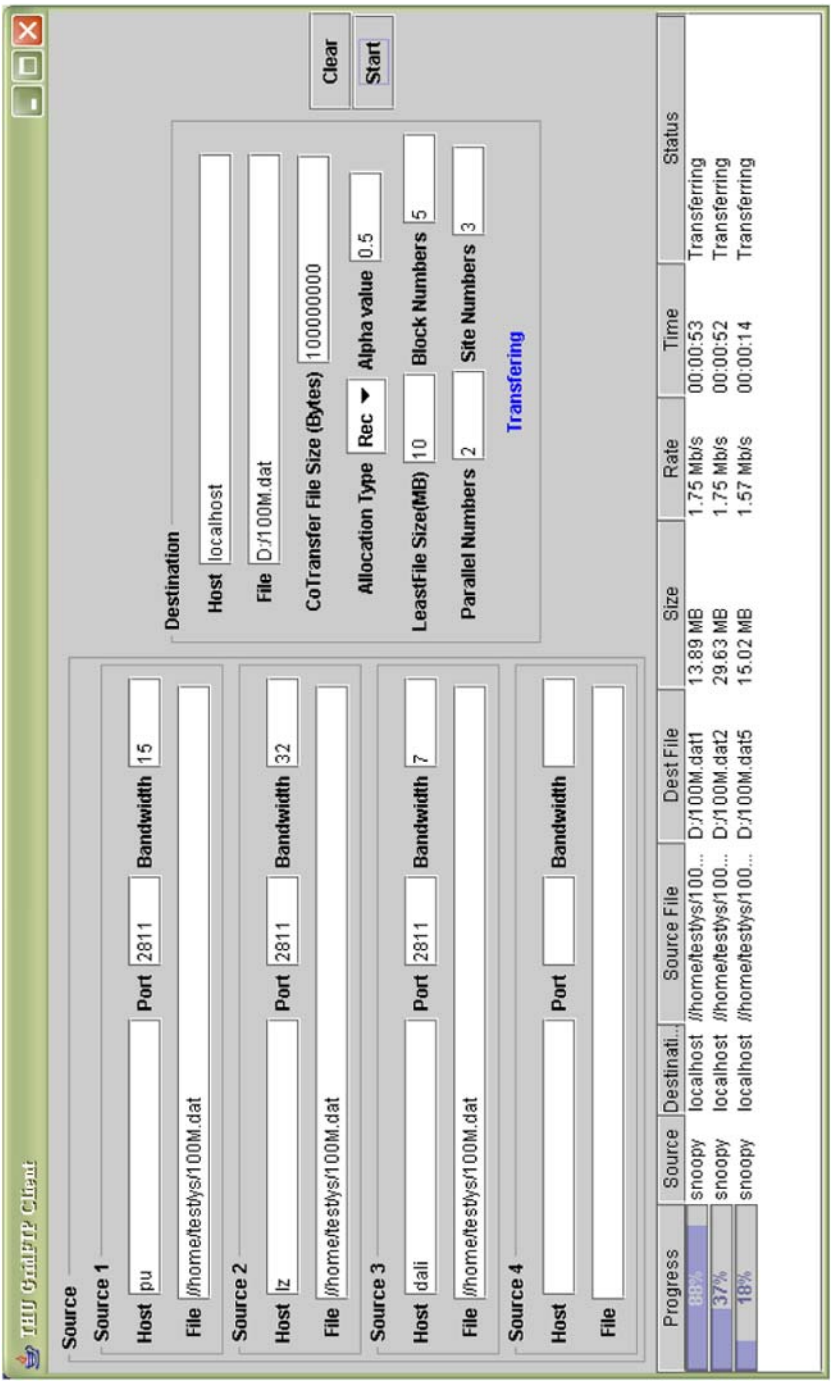

Fig. 4 Our GridFTP client tool

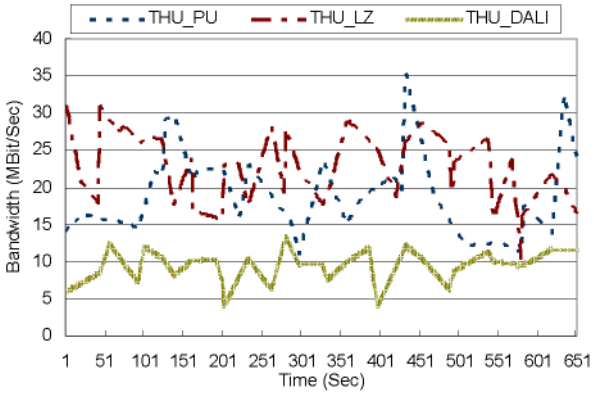

Fig. 5 Network variation between client and each server

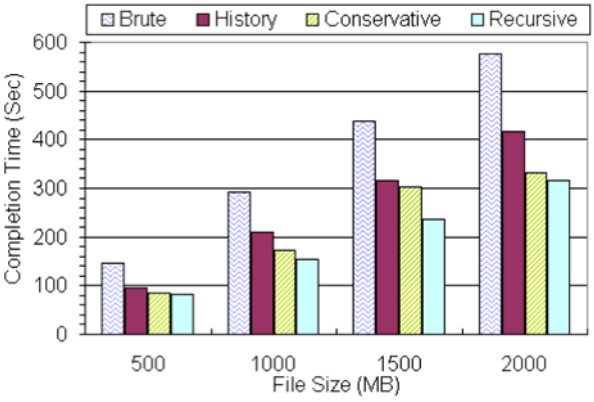

Fig. 6 Completion times for various methods

the equal block numbers above to calculate the performance of each size, when using the *Conservative Load Balancing*. In Fig. 6, we show the cost time of each scheme that transfers different file sizes. Obviously, Fig. 6 shows that our approach reduces the time efficiently when compared with the other three schemes.

For each of schemes, we analyzed the effect of faster servers waiting for the slowest server to deliver the last block. In Fig. 7, we calculate the total waiting idle time with different file sizes, and it shows that our *Recursive-Adjustment Co-Allocation* scheme offers significant performance improvements in every file size case when compared with other schemes. This result is due to our approach reduces the difference of each server's finished time efficiently.

For the *Recursive-Adjustment* technique, we study the effects of various α values on the block numbers and the total idle times. Figure 8 shows for an assigned file size of 10 MB to LeastSize, the total idle time increased and the total block number decreased as the α value increased. When the α value was greater then 0.7, the wait time grew rapidly, and although the wait time performance was good when the α value was less than 0.4, it resulted in a great increase in block numbers, which may cause high co-allocation costs. This experiment indicates that the assigned α value should be neither too large nor too small.

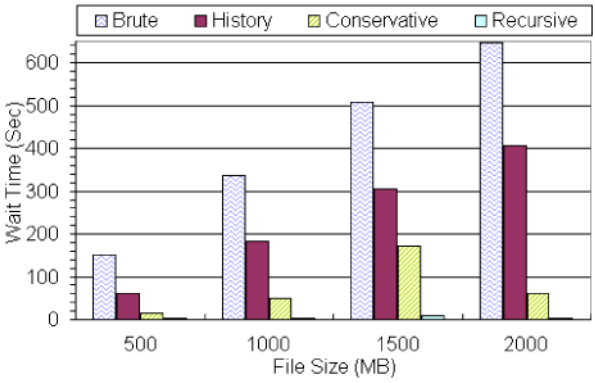

Fig. 7 Idle times for various methods

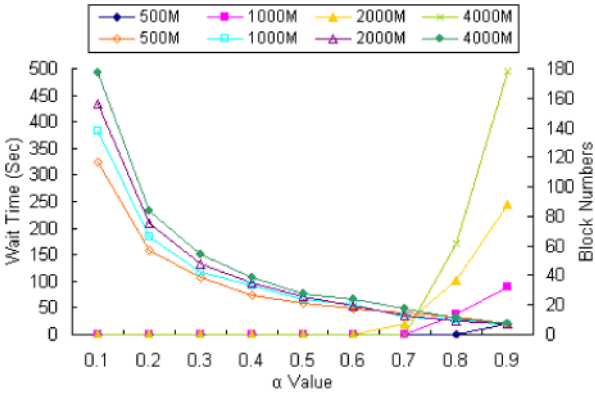

Fig. 8 Idle times for various α values and block numbers for various α values

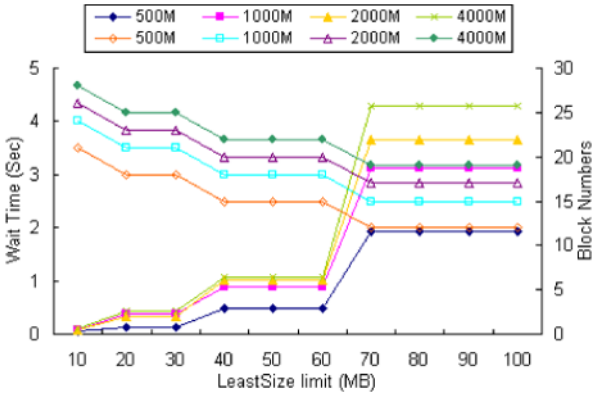

Fig. 9 Idle times for various *LeastSize* values and block numbers for various *LeastSize* values

Figure 9 shows that the *LeastSize* threshold value in our *Recursive-Adjustment* method is also an important factor affecting total wait time and block numbers. In this experiment, we set the α value to 0.5 and tested various *LeastSize* values. The results indicate that decreasing the *LeastSize* threshold value effectively reduces the total wait time. Although this results in more block numbers, the increase is not excessive. Figure 9 also indicates we may infer that the *Recursive-Adjustment* scheme performs

better with smaller *LeastSize* threshold values for most file sizes because smaller size final blocks are less influenced by network variations.

6 Conclusions

Data Grids enable the sharing, selection, and connection of a wide variety of geographically distributed computational and storage resources for addressing large-scale data-intensive scientific application needs in, for instance, high-energy physics, bioinformatics, and virtual astrophysical observatories, among others. Data sets are replicated in Data Grids and distributed among multiple sites. Unfortunately, datasets of interest are significantly large in size, which may lead to access efficiency overheads. Using the parallel-access approach to downloading data from multiple servers reduces transfer time and increases resilience to servers.

The co-allocation architecture provides a coordinated agent for assigning data blocks. A previous work showed that the dynamic co-allocation scheme leads to performance improvements, but cannot handle the idle time of faster servers, that must wait for the slowest server to deliver its final block. In this paper, we proposed the *Recursive-Adjustment Co-Allocation* scheme to improve data transfer performances using the co-allocation architecture in [13]. In our approach, the workloads of selected replica servers are continuously adjusted during data transfers, and we provide a function that enables users to define a final block threshold, according to their data grid environment. Experimental results show the effectiveness of our proposed technique in improving transfer time and reducing overall idle time spent waiting for the slowest server.

Acknowledgements This paper is supported in part by National Science Council, Taiwan R.O.C., under grants no. NSC93-2213-E-029-026, NSC94-2213-E-029-002, and NSC95-2221-E-029-004.

References

1. Allcock B, Bester J, Bresnahan J, Chervenak A, Foster I, Kesselman C, Meder S, Nefedova V, Quesnel D, Tuecke V (2002) Data management and transfer in high-performance computational grid environments. Parallel Comput 28(5):749–771
2. Allcock B, Bester J, Bresnahan J, Chervenak A, Foster I, Kesselman C, Meder S, Nefedova V, Quesnel D, Tuecke S (2001) Secure, efficient data transport and replica management for high-performance data-intensive computing. In: Proc of the eighteenth IEEE symposium on mass storage systems and technologies, 2001, pp 13–28
3. Chervenak A, Deelman E, Foster I, Guy L, Hoschek W, Iamnitchi A, Kesselman C, Kunszt P, Ripeanu M (2002) Giggle: a framework for constructing scalable replica location services. In: Proc of SC 2002, Baltimore, MD, 2002
4. Chervenak A, Foster I, Kesselman C, Salisbury C, Tuecke S (2001) The data grid: towards an architecture for the distributed management and analysis of large scientific datasets. J Netw Comput Appl 23:187–200
5. Czajkowski K, Fitzgerald S, Foster I, Kesselman C (2001) 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, pp 181–194
6. Czajkowski K, Foster I, Kesselman C (1999) Resource co-allocation in computational grids. In: Proceedings of the eighth IEEE international symposium on high performance distributed computing (HPDC-8'99), August 1999

7. Donno F, Gaido L, Ghiselli A, Prelz F, Sgaravatto M (2002) DataGrid prototype 1. In: TERENA Networking Conference, June 2002. http://www.terena.nl/conferences/tnc2002/Papers/p5a2-ghiselli.pdf
8. Global Grid Forum. http://www.ggf.org/
9. Hoschek W, Jaen-Martinez J, Samar A, Stockinger H, Stockinger K (2000) Data management in an international data grid project. In: First IEEE/ACM international workshop on grid computing—grid 2000, Bangalore, India, December 2000
10. IBM Red Books, Introduction to Grid Computing with Globus. IBM Press, www.redbooks.ibm.com/redbooks/pdfs/sg246895.pdf
11. Stockinger H, Samar A, Allcock B, Foster I, Holtman K, Tierney B (2002) File and object replication in data grids. J Clust Comput 5(3):305–314
12. The Globus Alliance. http://www.globus.org/
13. Vazhkudai S (2003) Enabling the co-allocation of grid data transfers. In: Proceedings of fourth international workshop on grid computing, November 2003, pp 41–51
14. Vazhkudai S, Schopf J (2003) Using regression techniques to predict large data transfers. Int J High Perform Comput Appl (IJHPCA) 17:249–268
15. Vazhkudai S, Tuecke S, Foster I (2001) Replica selection in the globus data grid. In: Proceedings of the 1st international symposium on cluster computing and the grid (CCGRID 2001), May 2001, pp 106–113
16. Vazhkudai S, Schopf J (2002) Predicting sporadic grid data transfers. In: Proceedings of 11th IEEE international symposium on high performance distributed computing (HPDC-11 '02) July 2002, pp 188–196
17. Vazhkudai S, Schopf J, Foster I (2002) Predicting the performance of wide area data transfers. In: Proceedings of the 16th international parallel and distributed processing symposium (IPDPS 2002), April 2002, pp 34–43
18. Wolski R, Spring N, Hayes J (1999) The network weather service: a distributed resource performance forecasting service for metacomputing. Future Gener Comput Syst 15(5-6):757–768
19. Yang C-T, Chen C-H, Li K-C, Hsu C-H (2005) Performance analysis of applying replica selection technology for data grid environments. In: PaCT 2005, lecture notes in computer science, vol 3603, Springer, September 2005, pp 278–287
20. Zhang X, Freschl J, Schopf J (2003) 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, pp 270–282

Chao-Tung Yang received a BS degree in computer science and information engineering from Tunghai University, Taichung, Taiwan in 1990, and the MS degree in computer and information science from National Chiao Tung University, Hsinchu, Taiwan in 1992. He received the PhD degree in computer and information science from National Chiao Tung University in July 1996. He won the 1996 Acer Dragon Award for outstanding PhD Dissertation. He has worked as an associate researcher for ground operations in the ROCSAT Ground System Section (RGS) of the National Space Program Office (NSPO) in Hsinchu Science-based Industrial Park since 1996. In August 2001, he joined the faculty of the Department of Computer Science and Information Engineering at Tunghai University, where he is currently an associate professor. His researches have been sponsored by Taiwan agencies National Science Council (NSC), National Center for High Performance Computing (NCHC), and Ministry of Education. His present research interests are in grid and cluster computing, parallel and high-performance computing, and internet-based applications. He is both member of the IEEE Computer Society and ACM.

I-Hsien Yang received the BS degree in business administration at National Chung-Hsing University, Taichung, Taiwan, in July 1996. He received the MS degree in computer science and information engineering at Tunghai University, Taichung, Taiwan, in July 2005. He also works at Knowledge and Service Information Corporation in Taichung City, Taiwan. His research interests include data grid, grid computing, and cluster computing.

Kuan-Ching Li received the PhD and MS in Electrical Engineering and Licenciatura in Mathematics from University of Sao Paulo, Brazil in 2001, 1996 and 1994, respectively. After he received his PhD, he had been post-doc scholar in the Department of Electrical and Computer Engineering, University of California—Irvine, California, USA and the Department of Electrical Engineering—Systems, University of Southern California, California, USA. Professor Li is currently an Associate Professor in the Department of Computer Science and Information Engineering at the Providence University, Taiwan. Prior to join Providence in February 2003, he was a research scientist at the University of California—Irvine. His research interests include cluster and grid computing, parallel software design, and life sciences computing.