Journal of Grid Computing **2:** 31–42, 2004.

Distributed Downloads of Bulk, Replicated Grid Data

Sudharshan Vazhkudai
Computer Science and Mathematics Division, Oak Ridge National Laboratory, One Bethel Valley Road, PO Box 2008 MS-6016, Oak Ridge, TN 37831, USA
E-mail: vazhkudaiss@ornl.gov

Key words: co-allocation, data grids, parallel downloads, partial transfers, peer-to-peer, scheduling

Abstract

Data-sharing scientific communities use storage systems as distributed data stores by replicating content. In such highly replicated environments, a particular dataset can reside at multiple locations and can thus be downloaded from any one of them. Since datasets of interest are significantly large in size, improving download speeds either by server selection or by co-allocation can offer substantial benefits. In this paper, we present an architecture for co-allocating Grid data transfers across multiple connections, enabling the parallel download of datasets from multiple servers. We have developed several co-allocation strategies comprising of simple brute-force, predictive and dynamic load balancing techniques as a means both to exploit rate differences among the various client–server links and to address dynamic rate fluctuations. We evaluate our approaches using the GridFTP data movement protocol in a wide-area testbed and present our results.

1. Introduction

Replicating popular content in the interest of offloading host severs is a widely used practice (FTP mirror sites, web caching [38, 18, 36], etc.). Recently, this trend is being put to extensive use in large-scale, data-sharing scientific communities. Many recent applications use resources as distributed data stores [31, 29, 11, 30, 27, 17, 20], where pieces of large datasets are replicated over several sites. For example, several high-energy physics experiments have agreed on a tiered Data Grid architecture [12, 13] in which all data (approximately 20 petabytes by 2006) is located at a single Tier 0 site; various (overlapping) subsets of this data are located at national Tier 1 sites, each with roughly one-tenth the capacity; smaller subsets are cached at smaller Tier 2 regional sites; and so on. Therefore, any particular dataset is likely to have replicas located at multiple sites.

Different replica locations are bound to have varied performance characteristics due to different architectures, and system load. Added to this is the client's network connectivity to these servers with its associated traffic. Thus, downloading data from anyone of the replica locations can result in a varied end-user experience. Downloading large datasets (with filesizes between 10 MB and 1 GB) through heavily congested links can be a significantly painful experience.

A typical Internet download between a client and a server is mired by several bottlenecks (Figure 1(a)) [14]. First, the bandwidth achievable by the client is limited by the bandwidth of the server's connection to the Internet – commonly referred as the *First-Mile* problem. The first-mile bottleneck is further compounded by simultaneous requests to a server from multiple clients. Second, the achievable bandwidth is further limited by the congestion in the link connecting the server and the client. Third, the bottleneck could be in the client's own connectivity to the Internet – the *Last-Mile*. Thus, the download speed is only as fast as the slowest link in the aforementioned setup. Sophisticated solutions are required to significantly address this issue.

One way to improve download speeds is to employ complex server selection techniques to determine the best replica location, offering high transfer rates, using a combination of server and network load details [1, 35]. In practice, however, due to the shared

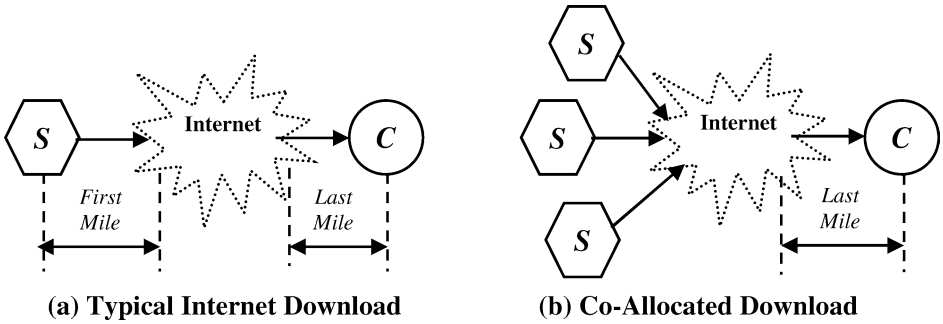


Figure 1. (a) Various bottlenecks in the Internet document download – the first mile problem, the congestion in the links connecting server and client and the last mile problem. (b) Co-allocated download model minimizes the first mile and the link congestion bottlenecks.

nature of network links the load on them can vary unpredictably. Thus, in the face of transient network conditions, downloading datasets even from the best of servers can often result in ordinary transfer rates.

A promising alternative is to download data from multiple locations, establishing multiple connections in parallel (Figure 1(b)). With this approach, instead of downloading the entire dataset from a single sever, unique partial copies of the dataset are fetched from multiple servers in parallel that are later reassembled at the client end.

This co-allocation of data transfers has several relevant properties of significant interest to us. First, it obviates the need for complex server selection. Second, due to its decentralized nature the eventual performance achieved may not be adversely affected by degradation in any of the co-allocated flows while also being resilient to server failures. Third, the client download experience can be positively amplified with the aggregate bandwidth commensurate to the summation of the individual transfer rates of each flow (in the best case). Fourth, it significantly alleviates the first-mile (slow server, serving a fast client) and the Internet congestion problem by distributing load to multiple servers and different routes (Figure 1(b)). Even in the case of a slow client served by a fast server, co-allocation can offer significant benefits due to fluctuations in network conditions.

Co-allocating data transfers across multiple replica locations can have widespread applicability beyond scientific data-sharing communities. For instance, Internet content providers rely heavily on content distribution networks [15] for managing consistent replicas of popular content on surrogate, edge-servers, closer to end-users [14]. Content distribution networks such as Akamai [1] and Speedera [28] improve download speeds by employing techniques such as request redirection [37, 16] (to select best servers) to fetch data from less congested links. Thus, parallel-downloading techniques can find significant use in such scenarios.

Peer-to-peer file sharing, where peers come together in a cooperative, decentralized manner to locate and share content, is yet another candidate for parallel downloading [6]. The enormous popularity of file sharing means that networked content sharing systems are likely to siphon much of the available Internet bandwidth. In fact, recent studies indicate that file sharing activity contributes up to 60% on any service provider network [21, 26]. Co-allocations in such cases can help improve download speeds, reduce load on certain parts of the network, alleviate loaded peers, etc.

In this paper we develop a basic architecture for co-allocating Grid data transfers and build a few techniques for downloading data in parallel, from multiple servers. We develop three techniques: (1) brute force co-allocation, (2) predictive co-allocation of flows and (3) dynamic load balancing. We apply these techniques to the GridFTP [2] data movement tool, part of the Globus Toolkit™ [9], and evaluate our approaches by conducting performance experiments in a wide-area testbed. Our results indicate a significant increase in bandwidth due to distributed downloads and denote that dynamic solutions outperform static approaches.

2. Related Work

Developing techniques for parallel downloads of Internet documents is of significant interest in the networking community and can be broadly classified into stateless and stateful approaches.

Stateless approaches to the parallel access problem rely on clients subscribing to several mirror sites to restitute the data. This approach further makes extensive use of erasure codes (error correction codes)

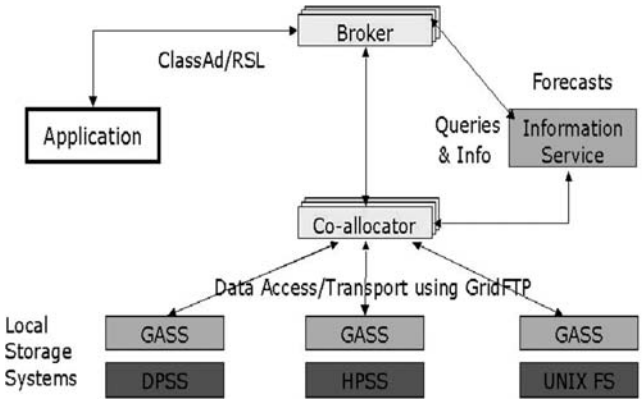


Figure 2. Resource management architecture and the role of co-allocation.

[24] to develop an n packet encoding of a k packet file, with the property that the file can be reassembled from any k packet subset of the encoding [3, 5, 4]. Pros of this approach are: obviates the need for maintaining file ranges and renegotiations on a per flow basis, fault tolerance and scalability; while the cons are: constructing an n packet encoding is nontrivial, cost of encoding and decoding can be significant for large dataset size, and clients and servers are required to agree, apriori, on common encoding schemes. Other related effort includes Rabin's [23] and Maxemchuk's [19] work on dispersing pieces of the file on different nodes in the network for fault tolerance and dispersity routing, respectively.

On the contrary, stateful techniques divide the file into disjoint sets, downloading different ranges from different servers. In [25, 10], the authors develop previous history-based and dynamic solutions, demonstrating their techniques for web-based documents of the order of several hundred kilobytes. Accurate predictions of range distributions are required for several stateful techniques, which are often quite difficult to obtain in the face of changing network conditions. In [25, 10], the authors rely on simple averages of previous transfer rates as an estimate for range calculations per flow. Work from Beck et al. demonstrated the usefulness of dynamic distributed downloads in the context of streaming applications by fetching multiple copies of file blocks in order to address jitter [22].

In our work we develop predictive and dynamic solutions similar to that of [25, 10, 22], but extend it by addressing network fluctuations. Further, we employ prediction techniques, deriving from our previous work [33, 34] on predicting data transfer rates between sources and sinks, for range calculations per flow that can significantly improve our performance and reduce renegotiations. The use of encoding schemes, and thus the stateless alternative, may not be suited for our purposes due to our concentration on large datasets, for which encoding and decoding times can be quite significant.

3. A Co-Allocation Architecture

The Globus Toolkit [9] provides a basic template for resource management [8], which can be extended to support the co-allocation of Grid data transfers. As illustrated in Figure 2, the architecture comprises of three main components: an information service, local storage systems, and broker/co-allocator.

An application requiring access to data presents a description of the data to the broker. The broker, in conjunction with information services [7], identifies possible alternatives from where the dataset in question can be fetched. This set is then presented to the co-allocation agent, which uses a combination of information services and some heuristics to map the data transfer request across multiple replica locations to download the data in parallel using GridFTP.

3.1. *GridFTP and Support for Partial Copy*

GridFTP [2] is part of the Globus Toolkit™ and is widely used as a secure, high-performance data transfer protocol. It extends standard FTP implementations with several features needed in Grid environments, such as security on control and data channels, multiple data channels for parallel streams, partial file transfers, and third party transfers. Of particular interest to us is the ability to fetch partial copies of a file. Partial copy, and several other features, is part of GridFTP's extended retrieve functionality, which is used to request

that a retrieve be done with some additional processing on the server. This command is an extensible way of providing server-side data reduction. With partial copy, a section of the file, defined by the starting offset and extent, will be retrieved from the data server.

3.2. *Allocation Mechanisms*

We now proceed to describe the co-allocation mechanisms that we have developed.

3.2.1. *Brute-Force Co-Allocation*

Brute-force co-allocation is a basic scheme that works by dividing the file size equally among available flows. Thus, if the data to be fetched is of size, S and there are n locations to fetch it from, then this technique assigns to each flow a data block of size, S/n. With this technique, although all the available servers are utilized, bandwidth differences among the various client–server links are not exploited.

3.2.2. *Predictive Co-Allocation*

To address and exploit transfer rate differences among the various co-allocated flows, we develop a predictive allocation scheme. With this technique, the block size per flow is commensurate to its predicted transfer rate, decided based on a previous history of GridFTP transfers. Thus, the file-range distribution is based on the predicted merit of the flow. If these predictions are not accurate enough, renegotiations of flow sizes might be necessary as slower links can get assigned larger portions of data, which could weigh heavily on the eventual bandwidth achieved.

In order to obtain accurate predictions of transfer rates for the various links, we derive from our previous work on forecasting GridFTP transfers. Our previous work delved into deriving accurate predictions in the face of network and system load fluctuations. We developed a series of univariate and multivariate predictors that derive forecasts from GridFTP transfers in isolation and in conjunction with network and disk load data, respectively. We use extensive regression analysis to derive predictions within 15% error [33]. Logs of previous GridFTP transfers are fed to univariate or multivariate predictors to derive forecasts. For purposes concerning co-allocations, we use a temporal variation of univariate, average predictor (moving average over time) [34].

With the predictive approach, the client divides the file into n disjoint blocks, corresponding to n servers. Each server, i, $1 \leqslant i \leqslant n$, has a predicted transfer rate of B_i to the client. In theory then, the aggregate bandwidth achievable by the client for the entire download is:

$$A = \sum_{i=1}^{n} B_i,$$

where B_i is the predicted bandwidth per flow and A is the aggregate bandwidth. Such a speedup can only be achieved when all servers keep sending data until the file is fully received – i.e., all servers being busy at all times during the entire download. In practice, however, the achieved bandwidth is limited due to network congestion in the various flows (resulting in some servers finishing earlier than others) and the client's ability to handle the bandwidth surplus. Thus, the rate-limiting factor could be anyone of the slowest links in the setup – the servers themselves, the links connecting the client and servers, or the client itself.

Assuming the client is capable of handling the bandwidth surplus, range distributions are calculated as follows. For each server i, $1 \leqslant i \leqslant n$, and for a replica size, S, the block size per flow is:

$$s_i = \frac{B_i}{A} S,$$

where s_i is the block size per flow. Thus, the block size per flow is commensurate to its transfer rate and its ratio of contribution to the achievable aggregate bandwidth. Faster servers are assigned to deliver bigger portions of the file, while slower servers are assigned smaller pieces. Ranges of partial transfers can then be formulated based on this and the whole file reassembled at the client as follows:

$$S = \sum_{i=1}^{n} s_i.$$

The time taken for the entire download is:

$$T = \operatorname*{Max}_{i=1}^{n} \frac{s_i}{B_i},$$

where T is the total download time. In this manner, this scheme addresses the transfer rate differences among the various co-allocated flows.

3.2.3. *Dynamic Co-Allocation*

Although we have addressed the rate differences in the flows and exploited it to deliver proportionate pieces of the file per flow, we do not address dynamic network variations that can cause degradation in transfer rates. In spite of careful bandwidth estimates per flow,

network traffic and system load can cause servers, previously determined as fast or slow, to behave differently. This can significantly affect the download time. Thus, an end-user is typically interested in dynamic rate adaptation – an allocation scheme that can dynamically adapt to changing network conditions.

One way to address this is to monitor the progress of predictive co-allocated flows, to perform corrective measures in case of performance degradation. For instance, if the performance in a particular flow drops below a threshold, the transfer can be migrated to an alternate location or remaining data can be equally distributed among other existing flows.

Although in theory, these are feasible alternatives, in practice, however, such techniques are quite complex to realize for the following reasons. First, we need to add additional dynamic monitoring capability to our data movement protocol to monitor each flow, which can significantly contribute to the overhead. Second, we need criteria to determine performance degradation, which can be difficult due to changing network/system conditions. Third, even if degradation could be determined, corrective measures such as transfer migration or resizing may require significant renegotiation between clients and servers, which can be more costly than the existing decrease in performance.

A promising alternative is the use of dynamic co-allocation. We develop two variations of dynamic co-allocation: (1) Conservative Load Balancing and (2) Aggressive Load Balancing.

3.2.3.1. Conservative Load Balancing. With this approach, the rate, and thus how much a server delivers, is decided dynamically instead of being based on previous history. The dataset in question is divided into k disjoint blocks of equal size and each one of the available servers is assigned to deliver, in parallel, one block initially. Once a server delivers the block, another block is requested and so on, until the entire file is downloaded.

Faster servers and servers connected to the client through less congested or faster links, will deliver quickly, thus serving larger portions of the file when compared to their slower counterparts. Thus, with this technique, the load on the co-allocated flows is automatically adjusted so that congested links and loaded or slower servers are not further burdened.

With this technique, the number of blocks per download can affect the throughput achieved. We could either have a large number of small blocks or a small number of large blocks. We study the effect of using different block counts and sizes in Section 4.

The key to achieving maximum aggregate bandwidth, as stated earlier, is to keep all available servers busy at all times. In the best case, each server is only idle for duration t, where t is the time elapsed since the server delivered the last block and until it receives a request for a new block. Neglecting client side processing and multiprogramming at both ends, this is roughly equivalent to one round-trip time, which is insignificant compared to the entire download time.

One obvious downside to this approach is the eventuality of waiting on the slowest server to deliver the final block (same as predictive allocation). An alternative is to stop the slowest flow or dynamically resize blocks to fetch the remaining data from the other servers, although we do not employ this technique.

3.2.3.2. Aggressive Load Balancing. With the previous method, although faster servers deliver quickly, we only fetch one-block size each time around. Similarly, slower servers would again be assigned to deliver blocks. To address these issues, we add the following functionality to our load balancing scheme: (1) progressively increase the amount of data requested from faster servers; and (2) reduce the amount of data requested from slower servers or stop requesting data altogether.

In order to achieve the stated effect, we develop a few heuristics. For each block delivered by each flow, we compute the rate achieved and compare it against the running maximum of all flow rates. If the rate at which a flow delivered the block is greater than the running maximum, we double the block size for that flow and reset the maximum; if it is less, we maintain the one-block size for the flow; and if the rate is significantly less than the maximum, we stop using the flow. Thus, using these techniques, we address dynamic rate changes in the various co-allocated flows.

4. Results and Analysis

We evaluated the performance of our co-allocation schemes on data collected over two distinct two-week periods during October and December 2002. In the following sections we describe the experimental setup, traces and our results.

4.1. *Testbed Configuration*

Our experiments comprised GridFTP transfers, using our co-allocation clients, between five sites in our test-

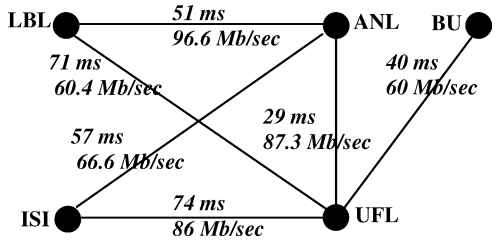


Figure 3. Network settings for our testbed sites. All sites are connected through OC-12 or OC-48 network links. For each site pair round trip times and network bottleneck bandwidths for the link between them is shown.

bed: Argonne National Laboratory (ANL), the University of Southern California Information Sciences Institute (ISI), Lawrence Berkeley National Laboratory (LBL), the University of Florida at Gainesville (UFL) and Boston University (BU). All our sites comprised of 100 Mb/sec Ethernets with high-end storage.

A prerequisite for downloading data from multiple servers is that the various links connecting the client and servers be bottleneck disjoint [25, 4]. If the client–server links share the same bottleneck then there can be little improvement due to co-allocation. From Figure 3, it is evident that the various client–server links for our setup are bottleneck-disjoint (bottleneck bandwidths were determined using iperf [32]).

4.2. *Experiment Setup*

We performed wide-area data transfer experiments using the GridFTP data movement tool. Our servers were standard GridFTP available from the Globus 2.0 Toolkit, while our clients included the various co-allocation schemes. Transfers comprised several file sizes ranging from 10 MB to 1 GB. These transfers were performed with tuned TCP buffer settings (calculated using the bandwidth delay product as shown in Figure 3) and eight parallel streams (per co-allocated flow) to achieve enhanced throughput. All our transfers were performed with co-allocation clients at either ANL or UFL. Table 1 summarizes our trace characteristics. We use a mix of fast and slow servers to study the effect therein.

4.3. *Performance*

In this section we discuss the performance of our co-allocation clients based on the data collected during October and December 2002. We evaluate four co-allocation schemes: (1) Brute-Force Co-allocation (Brute), (2) Predictive Co-allocation (Predictive), (3) Conservative Load Balancing (Conservative) and (4) Aggressive Load Balancing (Aggressive). For the two load balancing techniques, we study the effect of various block counts (Conservative-5, Conservative-10, Conservative-15, Aggressive-5, Aggressive-10 and Aggressive-15) on the bandwidth achieved. We compare each co-allocation scheme against the base case of fetching the entire file from a single server and study the bandwidth improvements therein. The bandwidth measures are averages based on two-week's worth of transfers.

Table 1. Trace characteristics for our data collected over two distinct two-week periods.

Server	GridFTP with support for partial copy
Download Client Schemes	
No co-allocation	Base client
Static	Brute and predictive
Dynamic	Conservative and aggressive
When	October and December 2002
Duration	Two weeks in each month
Client locations	ANL, UFL
File sizes	10 MB, 100 MB, 500 MB, 1 GB
Transfers per file size per co-allocation scheme	50 to 100
Total transfers (each month)	1000–1200
Prediction strategy for predictive downloads	Moving average (over time) of previous transfers
Block counts for dynamic downloads	5, 10 and 15
Number of parallel streams per co-allocated flow	8
Tuned TCP buffers per stream	Yes

4.3.1. *Impact of Client–Server Configurations on Downloads*

In Figures 4 and 5, we study the effect of slow servers (or links) with similar performance, serving a fast client. We see that all co-allocation schemes perform better than the base case of downloading the entire file from a single server. We observe that load balancing schemes (conservative and aggressive with block counts of 5) perform better than brute-force or predictive co-allocation and load balancing offers almost double the performance (for our experiments) when compared with the base case. In the case of slow servers serving fast clients there is usually residual bandwidth available that goes unused with typical downloads. With a distributed download,

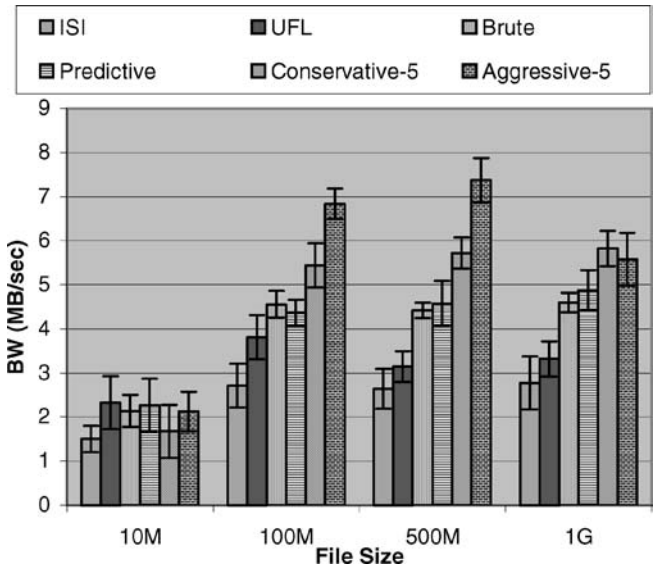


Figure 4. Servers are at ISI and UFL with client at ANL (Oct'02). First two bars in each file size denote downloading the entire file from either ISI or UFL, while others denote co-allocated downloads using the two servers. Depicts 95% confidence ranges for bandwidth.

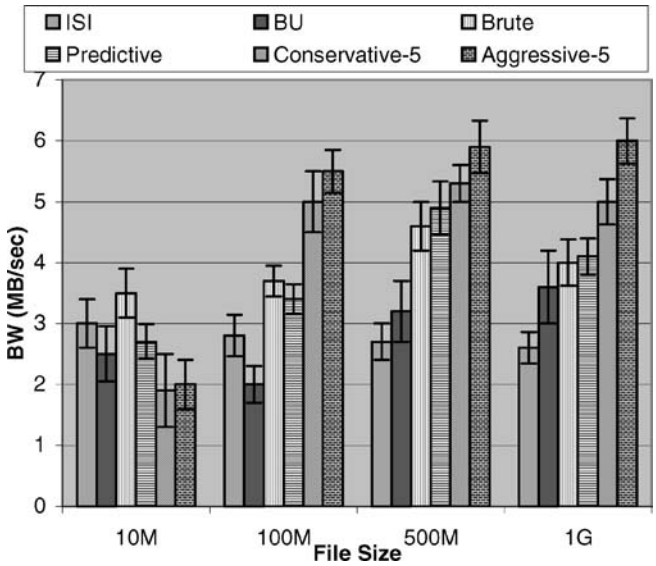


Figure 5. Servers are at ISI and BU with client at UFL (Dec'02). Depicts 95% confidence ranges for bandwidth.

this residual bandwidth is utilized to achieve enhanced throughput.

In Figures 6, 7 and 8, we use a mix of slow and fast servers and compare with base case of no co-allocation. We observe that co-allocation schemes are either better (improvements of up to 2 MB/sec) than or comparable to faster servers in isolation. The figures indicate that the gain due to co-allocation is inversely proportional to the performance gap between the servers. In Figure 6, a faster server saturates a client quickly, leaving available little residual bandwidth for other servers. Co-allocation in such cases offers very little improvement. In Figures 7 and 8, as the performance gap between the servers is low, we observe gains due to co-allocation.

4.3.2. *Shared Bottlenecks*

As mentioned earlier, in order for a distributed download to payoff, there should exist residual network bandwidth from the client to the additional servers. If not, then accessing additional sites will interfere with existing connections and contribute to the congestion.

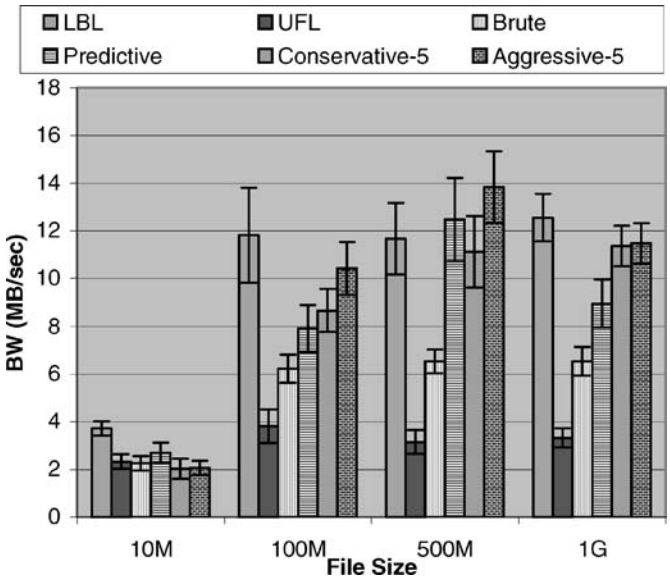


Figure 6. Servers are at LBL and UFL with client at ANL (Oct'02). Depicts 95% confidence ranges for bandwidth.

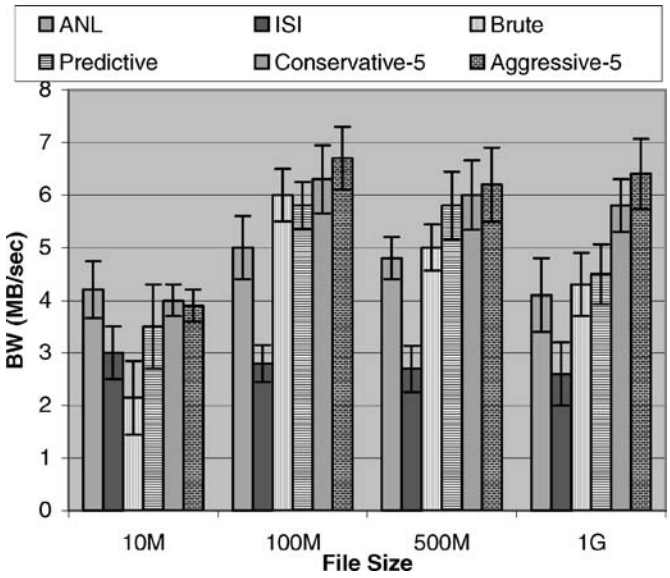


Figure 7. Servers are at ANL and ISI with client at UFL (Dec'02). Depicts 95% confidence ranges for bandwidth.

To study the effect of shared bottlenecks we choose two servers each from the LBL and ISI domains respectively. As stated earlier, in our experiments, each flow uses parallel streams and tuned TCP buffers to fully utilize the available bandwidth. Thus, adding another server (with the same bottleneck) interferes with existing connections. In Figure 9, we can see that the bandwidth achieved due to a distributed download is much less than that achieved by individual servers in isolation.

4.3.3. *Sensitivity of Schemes towards Parameters*
We analyzed the effect of file sizes, number of flows and block counts on the download performance – i.e., threshold values beyond which co-allocation offered gains or saturated. Figures 4 through 8 show that all our co-allocation schemes offer significant performance improvements (when compared with the base case) as the file size increases. For smaller file sizes we see no improvements in using co-allocation using our data movement tool. This is because, for smaller file sizes the cost of co-allocation – which involves

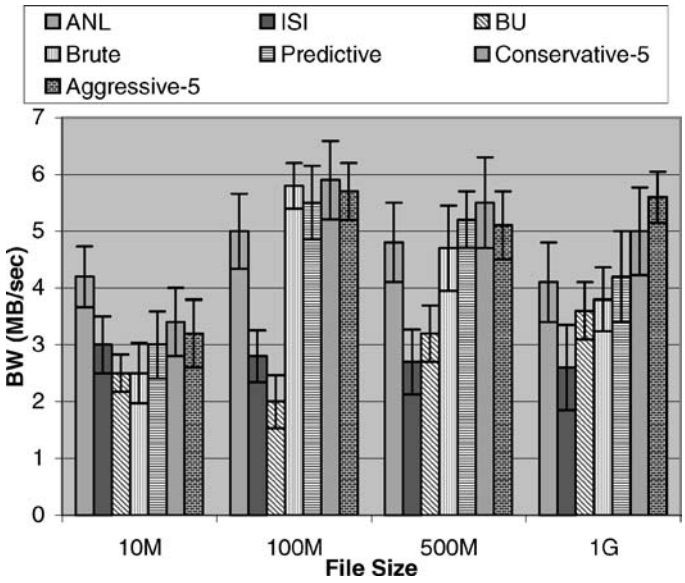
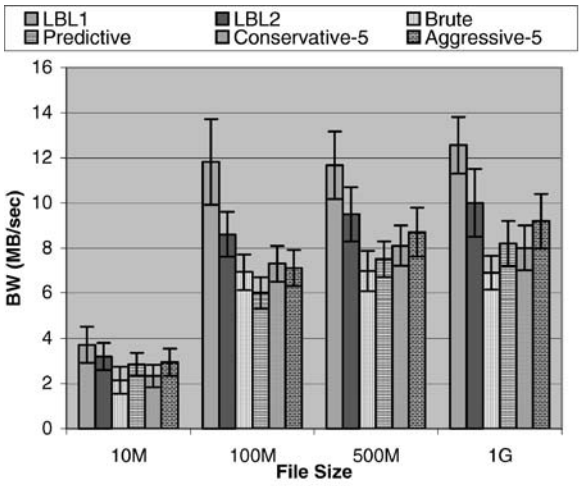
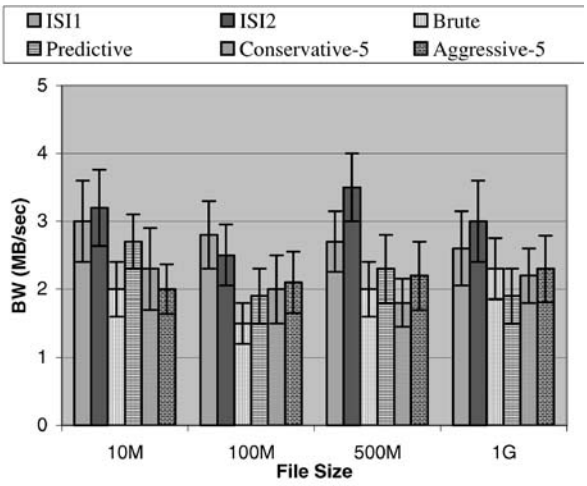


Figure 8. Servers are at ANL, ISI and BU with client at UFL (Dec'02). First three bars in each file size denote downloading the entire file from ANL, ISI or BU, while others denote co-allocated downloads using the three servers. Depicts 95% confidence ranges for bandwidth.



(a) Two servers at LBL sharing the same bottleneck to the client at ANL (Oct'02).



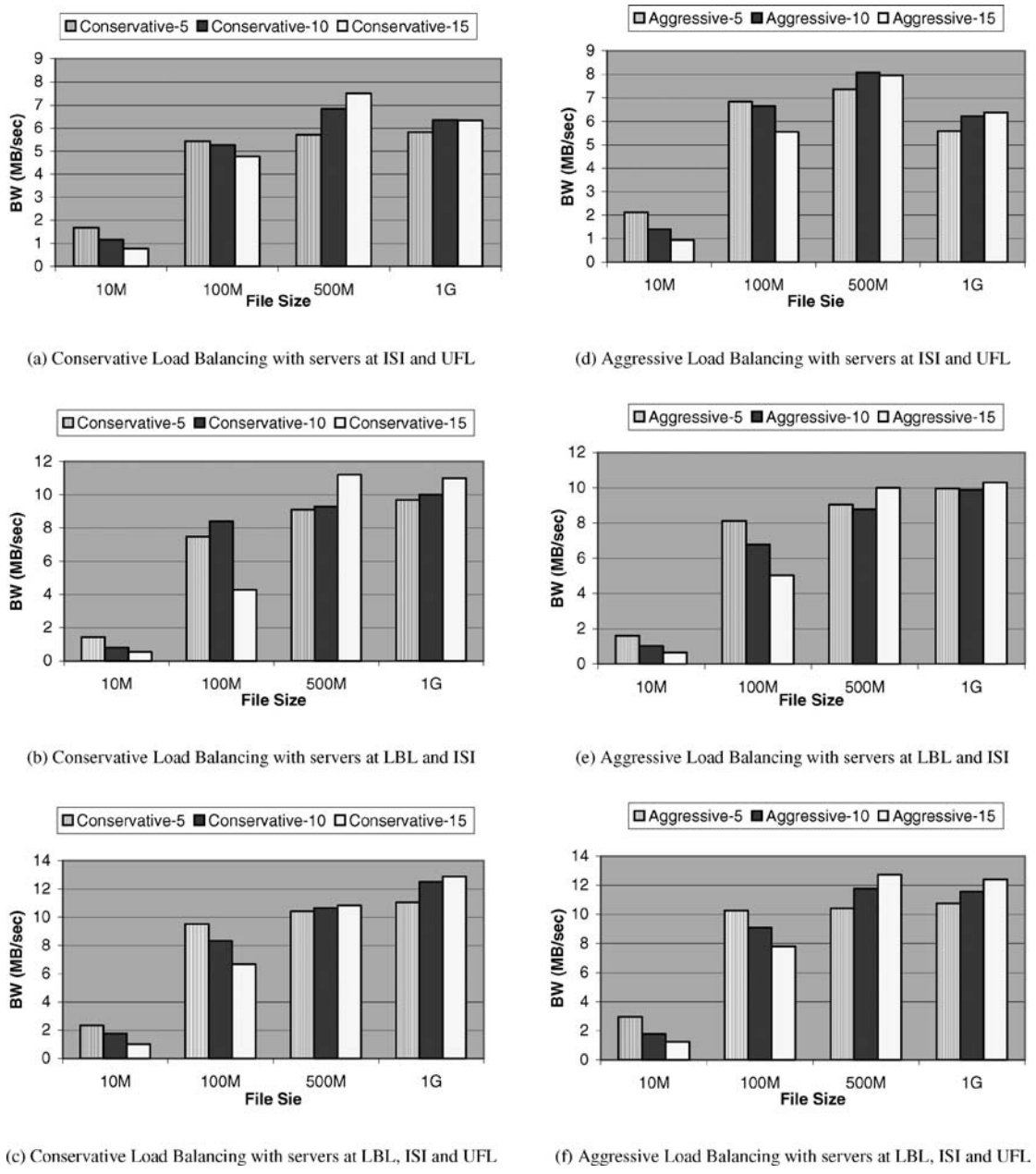
(b) Two servers at ISI sharing the same bottleneck to the client at UFL (Dec'02).

Figure 9. Servers sharing the same network bottleneck to the client. Depicts 95% confidence ranges for bandwidth.

connection establishment, negotiations, reassembly, resizing, etc. – is significantly high compared to the total download time thereby diminishing any gains.

In increasing the number of co-allocated flows (Figures 7 and 8) we observed that for our testbed and client–server configurations, download performance reached saturation at about 3 or 4 flows.

For our various load balancing techniques, we studied the effect of using different block counts (5, 10 and 15). Figure 10 compares the variations of conservative and aggressive load balancing techniques. From the figure we can infer that for smaller file sizes the load balancing schemes perform better with less number of blocks, while for larger file sizes more blocks result in better performance. For our experiments and our block counts we saw performance improvements of up to 1–2 MB/sec. With small files more blocks will result in more overhead in terms of connection establishment, reassembly, etc., when compared to the



(a) Conservative Load Balancing with servers at ISI and UFL

(d) Aggressive Load Balancing with servers at ISI and UFL

(b) Conservative Load Balancing with servers at LBL and ISI

(e) Aggressive Load Balancing with servers at LBL and ISI

(c) Conservative Load Balancing with servers at LBL, ISI and UFL

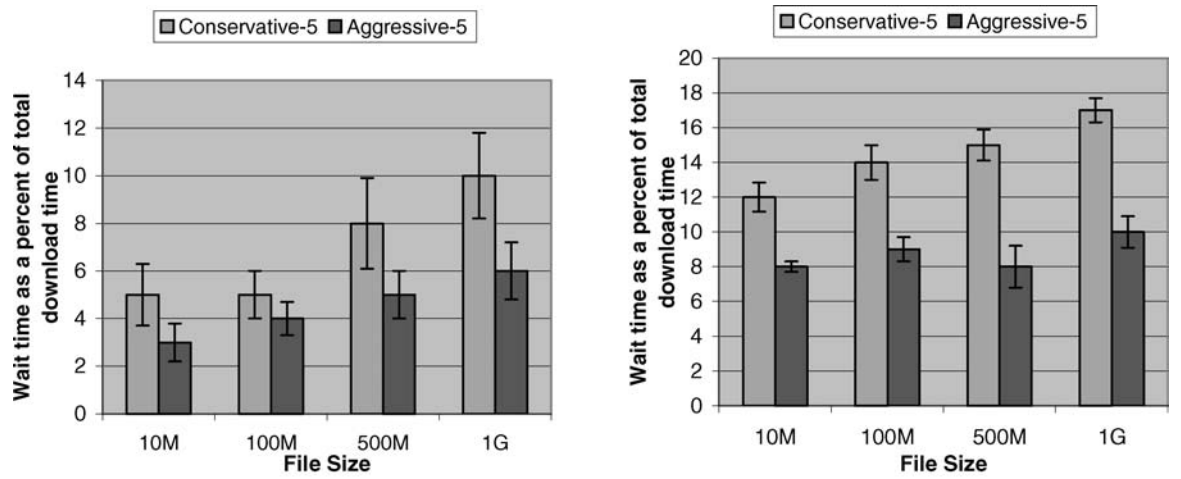
(f) Aggressive Load Balancing with servers at LBL, ISI and UFL

Figure 10. Comparison between the variants of conservative and aggressive load balancing schemes using different block counts for a client at ANL (Oct'02). Conservative-5 denotes a block count of 5.

total download time; while with large files less blocks can mean slower servers delivering bigger portions of the file.

4.3.4. *Waiting on Slow Servers*

For the load balancing schemes, we analyzed the effect of faster servers waiting on slow servers to deliver the last block. From Figure 11 we can observe that with conservative load balancing (out of the times when slower servers finished last), faster servers are idle for up to 17% of the total download time waiting for slower servers to finish delivering the last block. While aggressive balancing is not altogether devoid of this trend, we observe almost up to 40% reduc-



(a) Servers at ANL and ISI with client at UFL (Dec'02).

(b) Servers at LBL and UFL with client at ANL (Oct'02).

Figure 11. (a) ANL is the faster server having to wait on ISI. (b) LBL is the faster server having to wait on UFL. Bars denote the wait time of the faster server as a percentage of total download time. Also depicts 95% confidence for the % wait times.

tion in wait times due to a progressive increase in the amount of data fetched from faster servers. The figures also imply that using less number of blocks with larger files results in slower servers having to deliver larger pieces of data, thereby increasing the idle time of faster servers. This suggests that further techniques such as preempting flows or dynamic block sizing, to fetch more from faster servers, are worth investigating.

5. Conclusion

In this paper we have described the significance of co-allocating Grid data transfer requests across multiple servers, thus enabling parallel downloads. We have developed an architecture for downloading data from multiple servers, exploiting the partial copy feature of the GridFTP data movement tool. We have further developed several co-allocation strategies comprising of simple brute-force, predictive and dynamic load balancing techniques. We developed several techniques both as a means to address rate differences between the various flows and to address dynamic rate adaptation.

We analyzed our approaches in a wide-area testbed with a mix of fast and slow servers and observed that our techniques offered significant benefits when compared to downloading the entire file from a single server. Our results indicated an increase in performance regardless of the speed of servers and up to 2 times speedup for our testbed sites. We observed that our techniques performed better for larger file sizes and that dynamic approaches performed better than static ones. For our dynamic techniques, we found that lesser blocks worked well for smaller files and vice-versa for large files. Further, we observed that by progressively increasing the amount of data fetched from faster servers, we could reduce the waiting on slower servers to finish.

Future work includes analyzing additive/multiplicative increase and dynamic block sizing to address performance degradation in flows. Also, in our analysis we tested the effect of the number of flows on co-allocation. The natural question then is '*With how many flows should a transfer be launched?*' While this is subjective to client–server configurations, choosing an appropriate number of flows is vital to the performance achieved. One way to address this would be to start with a subset of servers ranked using our prediction strategies and applying load balancing techniques to this set. We are exploring such techniques as a means to exploit the merits inherent to both static and dynamic models.

Acknowledgments

This research was supported in part by fellowships from Argonne National Laboratory and The University of Mississippi. The fellowship from ANL (Summer of 2000 and academic years 2001 and 2002) was

due to support by the Mathematics, Information and Computational Sciences Office, Office of Advanced Scientific Computing Research, U.S. Department of Energy, under contract No. W-31-109-Eng-38. The fellowship from The University of Mississippi (Spring 2003) was due to support from the Graduate School's Doctoral Dissertation Award. This research was also supported by the U.S. Department of Energy under contract No. DE-AC05-00OR22725 with UT-Battelle, LLC. We further thank all the system administrators of our testbed sites for their valuable assistance.

References

1. "Akamai", 2002. http://www.akamai.com
2. W. Allcock et al., "High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data Grid Technologies", in *Supercomputing'01*, 2001.
3. J.W. Byers et al., "Informed Content Delivery Across Overlay Networks", in *Proceedings of ACM SIGCOMM'02*, 2002.
4. J.W. Byers, M. Luby and M. Mitzenmacher, "Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed up Downloads", in *Proceedings of IEEE INFOCOM*, 1999.
5. J.W. Byers, M. Luby and M. Mitzenmacher, "A Digital Fountain Approach to Asynchronous Reliable Multicast", *IEEE J-SAC, Special Issue on Network Support for Multicast Communication*, Vol. 20, No. 8, pp. 1528–1540, 2002.
6. J. Crowcroft and I. Pratt, "Peer to Peer: Peering Into the Future", in *Networks 2002*, 2002.
7. K. Czajkowski et al., "Grid Information Services for Distributed Resource Sharing", in *Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, IEEE Press: San Francisco, CA, 2001.
8. K. Czajkowski, I. Foster and C. Kesselman, "Resource Co-Allocation in Computational Grids", in *Proceedings of the Eigth IEEE International Symposium on High Performance Distributed Computing (HPDC-8)*, 1999.
9. I. Foster and C. Kesselman, "The Globus Project: A Status Report", in *IPPS/SPDP'98 Heterogeneous Computing Workshop*, 1998.
10. C. Gkantsidis, "Parallel Download", 2002. http://www.cc.gatech.edu/~gantsich/paralleldownload.htm
11. M. Hafeez, A. Samar and H. Stockinger, "Prototype for Distributed Data Production in CMS", in *7th International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2000)*, 2000.
12. K. Holtman, "Object Level Replication for Physics", in *4th Annual Globus Retreat*, Pittsburgh, 2000.
13. W. Hoschek et al., "Data Management in an International Grid Project", in *2000 International Workshop on Grid Computing (GRID 2000)*, Bangalore, India, 2000.
14. "Internet Bottlenecks: The Case of Edge Delivery Services", Akamai Whitepaper, 2000.
15. K. Johnson et al., "The Measured Performance of Content Distribution Networks", in *Proceedings of the 5th International Web Caching and Content Delivery Workshop*, Lisbon, Portugal, 2000.
16. J. Kangasharju, K. Ross and J.W. Roberts, "Performance Evaluation of Redirection Schemes in Content Distribution Networks", in *Proceedings of 4th Web Caching Workshop*, San Diego, 1999.
17. D. Malon et al., "Grid-enabled Data Access in the ATLAS Athena Framework", in *Computing and High Energy Physics 2001 (CHEP'01) Conference*, 2001.
18. R. Malpani, J. Lorch and D. Berge, "Making World Wide Web Caching Servers Cooperate", in *Proceedings of the Fourth International WWW Conference*, 1995.
19. N.F. Maxemchuk, "Dispersity Routing", in *Proceedings of the International Conference on Communications*, 1975.
20. H. Newman and R. Mount, "The Particle Physics Data Grid", www.cacr.caltech.edu/ppdg
21. "Peer-to-Peer File Sharing: The Effects of File Sharing on a Service Provider's Network", Sandvine Whitepaper, 2002.
22. J.S. Planck et al., *Algorithms for High Performance, Wide-Area, Distributed File Downloads*, Department of Computer Science, University of Tennessee, 2002.
23. M.O. Rabin, "Efficient Dispersal of Information for Security", *Journal of the ACM*, Vol. 38, pp. 335–348, 1989.
24. L. Rizzo, "Effective Erasure Codes for Reliable Computing", *Computer Communications Review*, 1997.
25. P. Rodriguez, A. Kirpal and W.E. Biersack, "Parallel-access for Mirror Sites in the Internet", in *Proceedings of IEEE INFOCOM*, 2000.
26. S. Saroiu, P.K. Gummadi and S. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems", in *Proceedings of Multimedia Computing and Networking (MMCN'02)*, 2002.
27. "Sloan Digital Sky Survey", 2002. http://www.sdss.org
28. "Speedera", 2002. http://www.speedera.com
29. "The Data Grid Project", 2002. http://www.eu-datagrid.org
30. "The GriPhyN Project", 2002. http://www.griphyn.org
31. "The LIGO Experiment", 2002. http://www.ligo.caltech.edu/
32. A. Tirumala and J. Ferguson, "Iperf 1.2 – The TCP/UDP Bandwidth Measurement Tool", 2001. http://dast.nlanr.net/Projects/Iperf
33. S. Vazhkudai and J. Schopf, "Predicting Sporadic Grid Data Transfers", in *11th IEEE High Performance Distributed Computing (HPDC-11)*, IEEE Press: Edinburgh, Scotland, 2002.
34. S. Vazhkudai, J. Schopf and I. Foster, "Predicting the Performance Wide-Area Data Transfers", in *16th International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE Press: Fort Lauderdale, Florida, 2002.
35. S. Vazhkudai, S. Tuecke and I. Foster, "Replica Selection in the Globus Data Grid", in *First IEEE/ACM International Conference on Cluster Computing and the Grid (CCGRID 2001)*, IEEE Press: Brisbane, Australia, 2001.
36. J. Wang, "A Survey of Web Caching Schemes for the Internet", *ACM Computer Communication Review*, 1999.
37. L. Wang, V. Pai and L. Peterson, "The Effectiveness of Request Redirection", in *Proceedings of the 5th OSDI Symposium*, 2002.
38. L. Zhang, S. Michel and S. Floyd, "Adaptive Web Caching: Towards a New Global Caching Architecture", in *Proceedings of the Third International Caching Workshop*, 1998.