Enabling the Co-Allocation of Grid Data Transfers

Sudharshan Vazhkudai
Computer Science and Mathematics Division, Oak Ridge National Laboratory
vazhkudaiss@ornl.gov

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, history-based 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.

Keywords: *Data Grids, Co-allocation, Partial Transfers, Scheduling, Peer-to-peer.*

1. Introduction

Replicating popular content in the interest of offloading host severs is a widely used practice (FTP mirror sites, web caching [ZMF98, MLB95, Wang99] etc.). Recently, this trend is being put to extensive use in large-scale, data-sharing scientific communities where pieces of large datasets are replicated over several sites [LIGO02, DataGrid02, HSS00, GriPhyN02, SDSS02, MMR+01, NM02]. For example, several high-energy physics experiments have agreed on a tiered Data Grid architecture [Holtman00, HJS+00] 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.

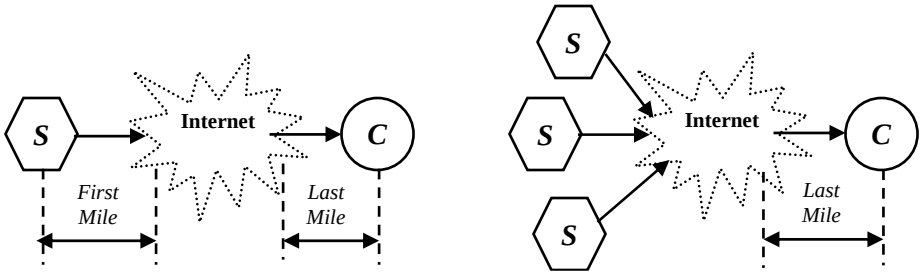
Different replica locations are bound to offer varied performance rates due to different architectures, system load and network connectivity. Thus, downloading large datasets (10 MB – 1 GB) from anyone of the replica locations can result in a varied end-user experience.

A typical Internet download between a client and a server is mired by several bottlenecks (Figure 1a) [Akamai00]. First, the bandwidth achievable by the client is limited by the bandwidth of the server's connection to the Internet (*First-Mile*), compounded further by simultaneous client requests. 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 (*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 [Akamai02, VTF01]. In practice, however, due to the shared 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 1b). 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. 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 1b). 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.



(a) Typical Internet Download **(b) Co-Allocated Download**

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.

Co-allocating data transfers across multiple replica locations can have widespread applicability beyond scientific data-sharing communities. For instance, Internet content distribution networks [JCD+00] that manage consistent replicas of popular content on surrogate, edge-servers, closer to end-users [Akamai00] or peer-to-peer systems that achieve file sharing in a decentralized manner [CP02] can significantly benefit from parallel downloading. Content distribution networks [Akamai02, Speedera02] cater to much of the Internet content provider traffic and attempt to improve download speeds by employing techniques such as request redirection [WPP02, KRR00] to fetch data from less congested links; while peer-to-peer downloads siphon much of the available Internet bandwidth—up to 60% on any service provider network [Sandvine02, SGG02]. 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) history-based co-allocation of flows and (3) dynamic load balancing. We apply these techniques to the GridFTP [AFN+01] data movement tool, part of the Globus Toolkit™ [FK98], 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 makes extensive use of erasure codes [Rizzo97] 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 [BCM+02, BLM02]. 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 [Rabin89] and Maxemchuk's [Maxemchuk75] 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 [RKB00, Gkantsidis02], 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 [RKB00, Gkantsidis02], 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 [PAD+02].

In our work we develop history-based and dynamic solutions similar to that of [RKB00, Gkantsidis02, PAD+02], but extend it by addressing network fluctuations. Further, we employ prediction techniques, deriving from our previous work [VSF02] 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 [FK98] provides a basic template for resource management [CFK99], 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 [CFF+01], 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.

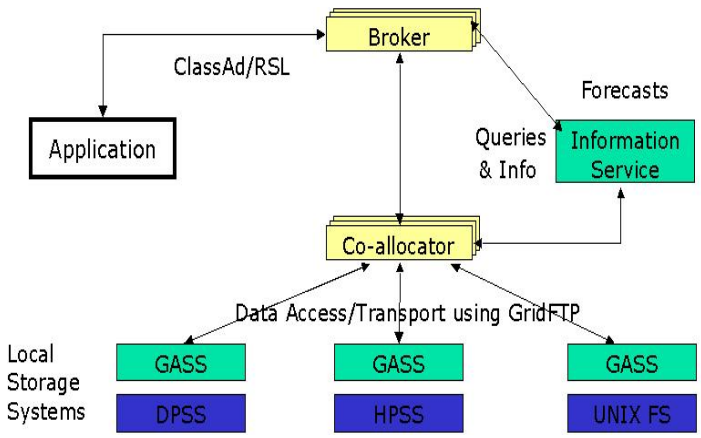


Figure 2. Resource management architecture and the role of co-allocation. Co-allocator combines broker decisions and information services to map data transfer requests onto storage systems using GridFTP and Globus Access to Secondary Storage (GASS).

3.1. GridFTP and Support for Partial Copy

GridFTP [AFN+01] is part of the Globus Toolkit™ and is widely used as a secure, high-performance data transfer protocol in Grids with features such as security, 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 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. 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 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. History-based Co-Allocation

To address and exploit transfer rate differences among the various co-allocated flows, we develop a history-based 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. 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 (within 15% error) in the face of network and system load fluctuations. For purposes concerning co-allocations, we use a temporal variation of average predictor (moving average over time) [VSF02].

With the history-based approach, the client divides the file into "n" disjoint blocks, corresponding to "n" servers. Each server, "i", $1 \le i \le n$, 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:

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

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 are 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.

Assuming the client is capable of handling the bandwidth surplus, range distributions are calculated as follows. For each server "i", 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. 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. Despite careful bandwidth estimates per flow, network traffic and system load can cause servers, previously determined as fast or slow, to behave differently. Thus, an end-user is typically interested in dynamic rate adaptation.

One way to address this is to monitor the progress of history-based 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.

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 history-based 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.

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 testbed: 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.

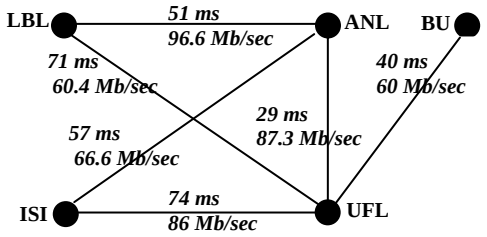


Figure 3. Network settings for our testbed sites. All sites are connected through OC-12 or OC-48 network links. For each site pair round trip times and network bottleneck bandwidths for the link between them is shown.

A prerequisite for downloading data from multiple servers is that the various links connecting the client and servers be bottleneck disjoint [RKB00, BLM02]. 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 [TF01]).

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 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. 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. We evaluate four co-allocation schemes: (1) Brute-Force (Brute), (2) History-based (History), (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, Aggrssive-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 (up to 1200 transfers each month).

4.3.1. Impact of Client-Server Configurations

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 perform better than brute-force or history-based co-allocation and load balancing offers almost double the performance 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, this residual bandwidth is utilized to achieve enhanced throughput.

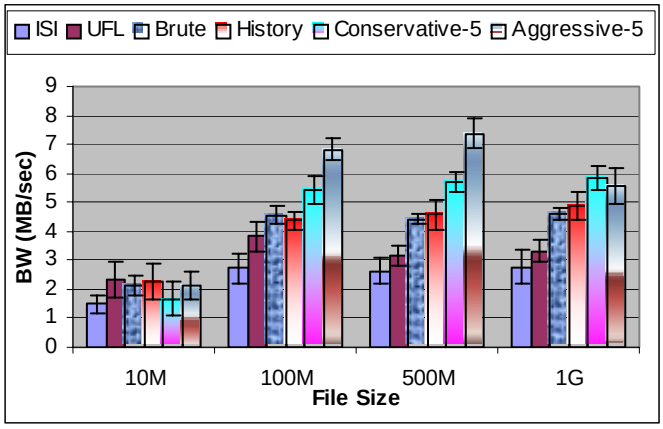


Figure 4. Servers are at ISI and UFL with client at ANL (Oct'02). First two bars in each file size denote downloading the entire file from either ISI or UFL, while others denote co-allocated downloads using the two servers. Depicts 95% confidence ranges.

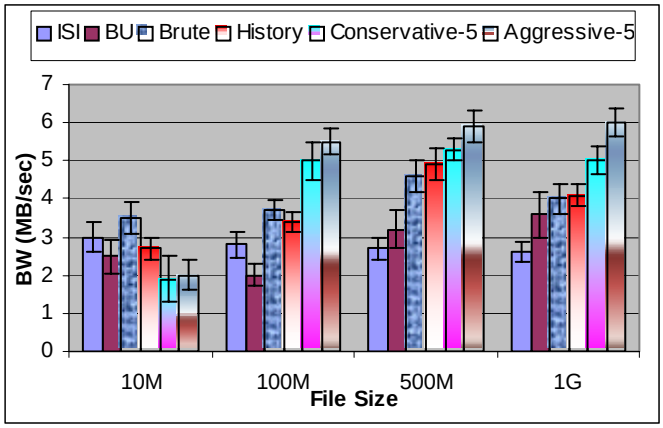


Figure 5. Servers are at ISI and BU with client at UFL (Dec'02). Depicts 95% confidence ranges.

In Figures 6, 7 and 8, we use a mix of slow and fast servers to study its effect on download. We observe that co-allocation schemes are either better (gains up to 2 MB/sec) 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 and no gain due to co-allocation. In Figures 7 and 8, as the performance gap between the servers is low, we observe gains due to co-allocation.

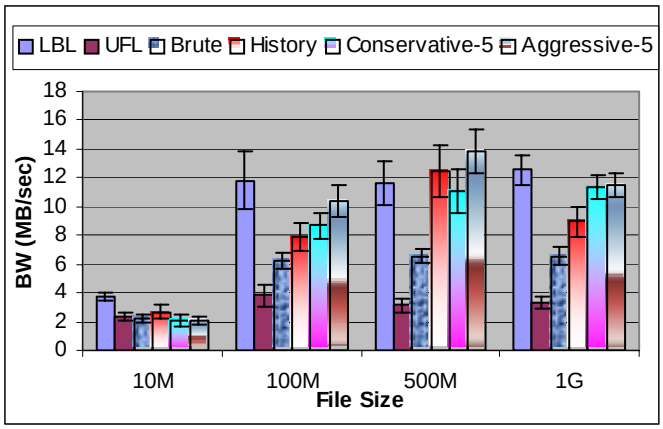


Figure 6. Servers are at LBL and UFL with client at ANL (Oct'02). Depicts 95% confidence ranges.

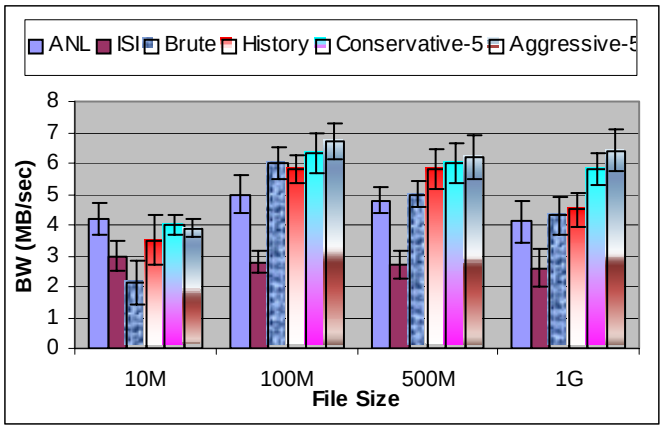


Figure 7. Servers are at ANL and ISI with client at UFL (Dec'02). Depicts 95% confidence ranges.

4.3.2. 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. A low value for the performance ratio, R, where R is:

R = Co-allocation Cost / Total Time to Download,

results in gains due to co-allocation. The cost of co-allocation involves connection establishment, negotiations, reassembly, resizing, etc. For smaller files, this co-allocation cost is high compared to the total download time.

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. While this is subjective to client-server

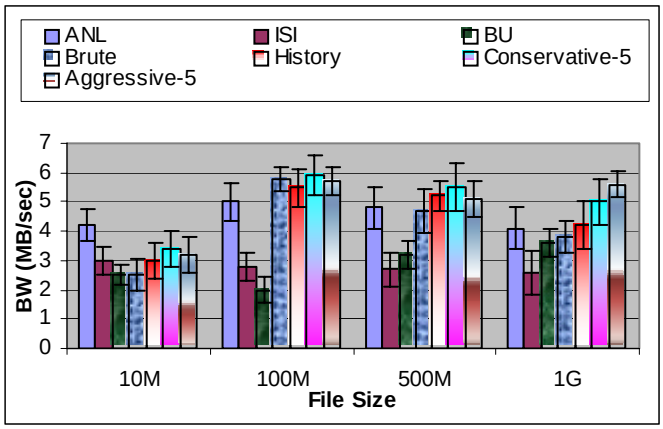


Figure 8. Servers are at ANL, ISI and BU with client at UFL (Dec'02). Depicts 95% confidence ranges for bandwidth.

configurations, choosing an appropriate number of flows is vital to the performance achieved.

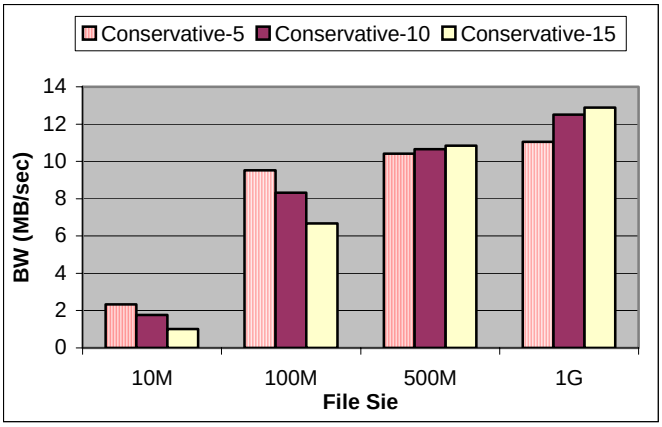
For our various load balancing techniques, we studied the effect of using different block counts (5, 10 and 15). Figure 9 compares the variations of conservative and aggressive load balancing techniques. From the figure we can infer that for smaller file sizes the load balancing schemes perform better 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 total download time; while with large files less blocks can mean slower servers delivering bigger portions of the file.

4.3.3. Waiting on Slow Servers

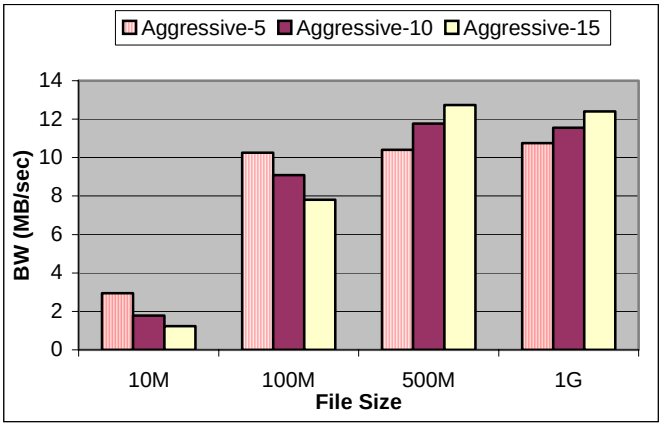
For the load balancing schemes, we analyzed the effect of faster servers waiting on slow servers to deliver the last block. From Figure 10 we can observe that with conservative load balancing (out of the times when slower servers finished last), faster servers are idle for up to 17% of the total download 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% reduction in wait times due to a progressive increase in the amount of data fetched from faster servers. The figure also implies 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.

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

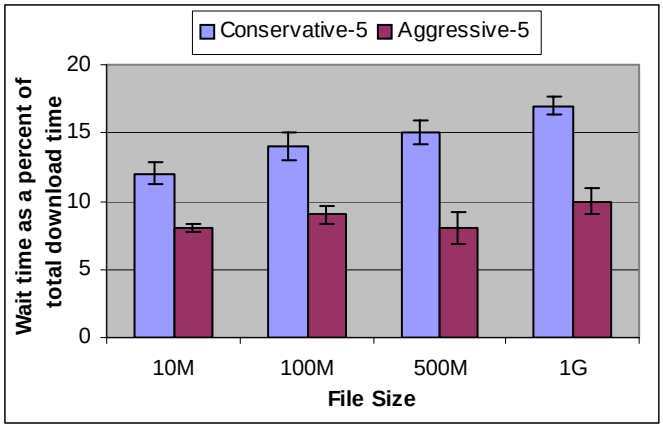


(a) Conservative Balancing with servers at LBL, ISI and UFL

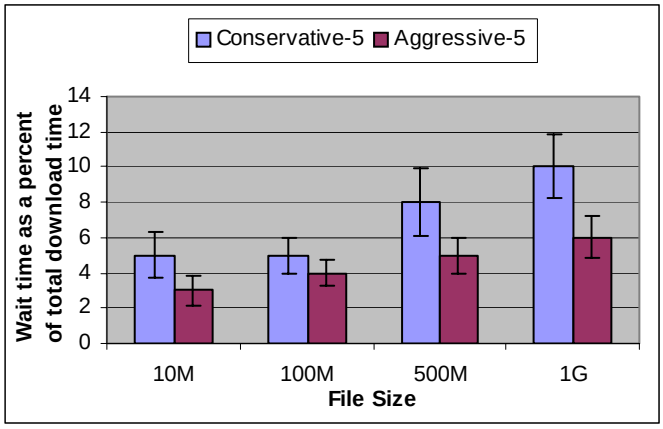


(b) Aggressive Balancing with servers at LBL, ISI and UFL

Figure 9. 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.



(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 10. (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.

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, history-based 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 x 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 dynamic techniques, 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 dynamic block sizing to address performance degradation.

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

[Akamai00] *Internet Bottlenecks: The Case of Edge Delivery Services*. 2000, Akamai Whitepaper.

[Akamai02] Akamai, http://www.akamai.com, 2002.

[DataGrid02] The Data Grid Project, http://www.eu-datagrid.org, 2002.

[GriPhyN02] The GriPhyN Project, http://www.griphyn.org, 2002.

[LIGO02] The LIGO Experiment, http://www.ligo.caltech.edu/, 2002.

[Sandvine02] *Peer-to-Peer File Sharing: The Effects of File Sharing on a Service Provider's Network*. 2002, Sandvine Whitepaper.

[SDSS02] Sloan Digital Sky Survey, http://www.sdss.org, 2002.

[Speedera02] Speedera, http://www.speedera.com, 2002.

[AFN+01] Allcock, W., et al. *High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data Grid Technologies*. in *Supercomputing'01*. 2001.

[BCM+02] Byers, J.W., et al. *Informed Content Delivery Across Overlay Networks*. in *Proceedings of ACM SIGCOMM'02*. 2002.

[BLM02] Byers, J.W., M. Luby, and M. Mitzenmacher, *A Digital Fountain Approach to Asynchronous Reliable Multicast.* IEEE J-SAC, Special Issue on Network Support for Multicast Communication, 2002. **20**(8): p. 1528-1540.

[CP02] Crowcroft, J. and I. Pratt. *Peer to Peer: peering into the future*. in *Networks 2002*. 2002.

[CFF+01] Czajkowski, K., et al. *Grid Information Services for Distributed Resource Sharing*. in *Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*. 2001. SanFrancisco, CA: IEEE Press.

[CFK99] Czajkowski, K., 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.

[FK98] Foster, I. and C. Kesselman. *The Globus Project: A Status Report*. in *IPPS/SPDP '98 Heterogeneous Computing Workshop*. 1998.

[Gkantsidis02] Gkantsidis, C. Parallel Download, http://www.cc.gatech.edu/~gantsich/parallel_download.htm, 2002.

[HSS00] Hafeez, M., 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 (ACAT2000)*. 2000.

[Holtman00] Holtman, K. *Object Level Replication for Physics*. in *4th Annual Globus Retreat*. 2000. Pittsburgh.

[HJS+00] Hoschek, W., et al. *Data Management in an International Grid Project*. in *2000 Internationsl Workshop on Grid Computing (GRID 2000)*. 2000. Bangalore, India.

[JCD+00] Johnson, K., et al. *The Measured Performance of Content Distribution Networks*. in *Proceedings of the 5th International Web Caching and Content Delivery Workshop*. 2000. Lisbon, Portugal.

[KRR00] Kangasharju, J., K. Ross, and J.W. Roberts. *Performance Evaluation of Redirection Schemes in Content Distribution Networks*. in *Proceedings of 4th Web Caching Workshop*. 1999. San Diego.

[MMR+01] Malon, D., et al. *Grid-enabled Data Access in the ATLAS Athena Framework*. in *Computing and High Energy Physics 2001 (CHEP'01) Conference*. 2001.

[MLB95] Malpani, R., J. Lorch, and D. Berge. *Making World Wide Web Caching Servers Cooperate*. in *Proceedings of the Fourth International WWW Conference*. 1995.

[Maxemchuk75] Maxemchuk, N.F. *Dispersity Routing*. in *Proceedings of the International Conference on Communications*. 1975.

[NM02] Newman, H. and R. Mount. *The Particle Physics Data Grid, www.cacr.caltech.edu/ppdg*.

[PAD+02] Planck, J.S., et al., *Algorithms for High Performance, Wide-Area, Distributed File Downloads*. 2002, University of Tennessee, Department of Computer Science.

[Rabin89] Rabin, M.O., *Efficient Dispersal of Information for Security.* Journal of the ACM, 1989. **38**: p. 335-348.

[Rizzo97] Rizzo, L., *Effective Erasure Codes for Reliable Computing.* Computer Communications Review, 1997.

[RKB00] Rodriguez, P., A. Kirpal, and W.E. Biersack. *Parallel-access for Mirror Sites in the Internet*. in *Proceedings of IEEE INFOCOM*. 2000.

[SGG02] Saroiu, S., 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.

[TF01] Tirumala, A. and J. Ferguson. *Iperf 1.2 - The TCP/UDP Bandwidth Measurement Tool, http://dast.nlanr.net/Projects/Iperf*. 2001.

[VSF02] Vazhkudai, S., J. Schopf, and I. Foster. *Predicting the Performance Wide-Area Data Transfers*. in *16th International Parallel and Distributed Processing Symposium (IPDPS)*. 2002. Fort Lauderdale, Florida: IEEE Press.

[VTF01] Vazhkudai, S., 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)*. 2001. Brisbane, Australia: IEEE Press.

[Wang99] Wang, J., *A Survey of Web Caching Schemes for the Internet.* ACM Computer Communication Review, 1999.

[WPP02] Wang, L., V. Pai, and L. Peterson. *The Effectiveness of Request Redirection*. in *Proceedings of the 5th OSDI Symposium*. 2002.

[ZMF98] Zhang, L., S. Michel, and S. Floyd. *Adaptive Web Caching: Towards a New Global Caching Architecture*. in *Proceedings of the Third International Caching Workshop*. 1998.