Accelerating Data Movement Leveraging End-system and Network Parallelism

Jun Yi[+], Raj Kettimuthu[+*], Venkatram Vishwanath[+*]
Mathematics and Computer Science Division, Argonne National Laboratory[+]
Computation Institue, University of Chicago[*]
{jyi, kettimut, venkatv}@mcs.anl.gov

Abstract

Data volumes produced by simulation, experimental and observational science is rapidly increasing. This data needs to be moved from its source to another resource for analysis, visualization and archival purposes. The destination resource could be either local or remote. The data intensive science is critically dependent upon the high-performance parallel file and storage end systems to read/write and high-speed networks to move their enormous data between local and remote computing and storage facilities. 100 Gigabit per second networks such as DOE's Advanced Network Initiative (ANI), Internet2's 100G network represent a major step forward in wide area network performance. Effective utilization of these networks requires substantial and pervasive parallelism, at the file system, end system, and network levels. Additional obstacles such as heterogeneity and time-varying conditions of network and end system arise that, if not adequately addressed, will render high performance storage and network systems extremely underperformed. In this paper, we propose a data movement system that dynamically and adaptively adjusts end systems and networks parallelisms in response to changing conditions of end systems and networks to sustain high-throughput for data transfers. We evaluate our system in multiple settings and show that (1) in a homogeneous configuration, the design can achieve better throughput for light and medium workload than GridFTP and achieve comparable throughput for heavy workload, (2) and in a heterogeneous configuration, the design can achieve several factors higher throughput for all workloads than GridFTP.

1 Introduction

Scientific experiments (e.g., climate modeling and prediction, biomedical imaging, geosciences, and high-energy physics) [10] are expected to generate, analyze, and distribute data volumes on the order of petabytes. As data volumes increase, experiments may have to use external computing facilities (e.g., supercomputing, data centers) to process the data, which will necessitate high-performance file and storage systems and additional fast wide area network (WAN) transfers. The U.S. Department of Energy (DOE) and Internet2 are building 100 Gbps networks to address the bandwidth requirements of scientific applications. These initiatives represent a major step forward in wide area network performance and arrive just in time for moving big data worldwide. The introduction of high-throughput storage area network (SAN) and parallel file systems (e.g. GPFS, PVFS, Lustre) enables fast parallel reads/writes of big files.

However, sustaining end-to-end data transfer rates close to network speeds is a challenging task. Moving data across the wire at 100 Gbps is not the same as reliably transferring 45 terabytes in an hour (or 1.08 petabytes/day) end-to-end, as simple math might suggest should be easily attainable. Many obstacles contribute to the degraded performance, span-

ning from the heterogeneity of end systems and network configurations to dynamically changing conditions of end systems and networks. We propose to exploit the parallelism of end systems and adapt to the load on storage systems and networks to achieve high end-to-end throughput for data transfers.

Depending on the configuration of end hosts, storage systems and networks, bottleneck for a given data movement could vary. For example, an endpoint may be connected to a parallel file system that can constantly provide 100MBps to 1GBps disk I/O rates for single-file reads/writes but that endpoint may have a 1Gbps bottleneck for wide area communication. Some endpoints may be connected by a 100Gbps wide area connection but the NICs at the end hosts can drive only 10Gbps. In this case more nodes are needed at each end to increase the end-to-end transfer throughput and saturate the available network capacity. In some cases, one endpoint may have a 10Gbps NIC and the other endpoint may have only a 1Gbps NIC, even though the network in between and the storage systems at both ends can handle 10Gbps. Using multiple hosts at the end with nodes having 1Gbps NIC will improve the end-to-end throughput in such cases. Also, the load on the end systems and networks will vary over time. The disk read throughput at sender may no longer be able to saturate the network or the disk write throughput at the receiver may start to hold down the network. The number of nodes at each end of the transfer and the number of disk and network threads with a node may have to be varied dynamically to optimize the performance as well as resource usage.

In this paper, we present a system that exploits the parallelism of end systems and networks, and dynamically adapts to changing conditions of both the network and end systems in order to accelerate end-to-end big data transfers.

The remaining of the paper is organized as follows. We discuss related work in Section 2. We provide some background in Section 3. We present our design and implementation in Section 4. In Section 5, we discuss our experiments and results. We conclude this paper in Section 6.

2 Related Work

Efficient data movement is not a new problem. The Distributed Parallel Storage System (DPSS) [9] provide high-speed random access to large data sets through collection of distributed disk servers that operate in parallel. Logistical networking [3] uses storage depots in the network to accelerate data distribution. Phoebus [12] and Kangaroo [11] that make opportunistic use of disks in intermediate nodes to improve end-to-end performance have been proposed. Peer-to-peer systems such as BitTorrent [6] enable clients to download pieces of file from multiple sources while simultaneously uploading pieces to others. Weigle and Chien [13] propose algorithms for computing efficient communication schedules for M-to-N communication problem. GridFTP [1] uses parallel TCP streams to workaround the limitations of TCP in high-bandwidth high-latency environments. The striping capability in GridFTP enables M-to-N communication but it has a limitation that M and N must be statically configured. Managed GridFTP [5] provides a framework to dynamically change the number of data movers at each end but the server does not have the ability to automatically increase or decrease the number of data movers based on the state of the system. Bhat et al. describe a self-managing data streaming service that allows users to specify self-managing behaviour. The service combines rule-based approach and model-based approach to enforce the behaviour. [4] presents a threaded buffer management algorithm for real time streaming of the simulation data but it uses a logitical networking based approach for data transfer. Our system continuously monitors the transfer environment and adaptively changes the number of nodes involved in the transfer, number of processes per node and number of disk and network I/O threads per processes to achieve optimal performance. There has been a number of studies [2, 7, 8, 14] on automatically configuring the number of parallel streams but these studies focus on optimizing network performance for single node transfers whereas our system considers multi-node transfers and optimizing disk I/O as well.

3 Background

We present some basic methods to increase the end-to-end throughput for big data transfer in this section. We mainly use multiple threads, multiple TCP streams, and multiple nodes to adjust the network and end systems parallelism. Other factors (e.g., file read/write block sizes, send/receive buffer sizes, transfer protocols, thread/process-core affinity, direct memory access, and network interrupt coalescing) are system-specific. We assume these are set for optimal performance before the system is in operation. So, these factors not the concerns of this paper.

Increasing the number of threads to read/write (in blocking mode) a file in a production file system can significantly improve the throughput (as shown in Figure 1). This is attributable mainly to two factors: (1) a file is partitioned into blocks and striped over multiple storage nodes and the blocks residing in different storage nodes can be read/written in parallel and (2) the round-trip time from the issue of a read/write request to the completion of the request via SAN is longer than actual file block read/write time, therefore issuing a read/write request after the completion of previous request will not saturate the file system. This trend on a single node continues until the node becomes the bottleneck due to NIC limitations or overwhelming context-switching overhead. Using multiple nodes to read/write from the file system resumes this trend until the file system becomes the bottleneck.

Similarly, increasing the number of TCP streams on a single node increases the network transfer throughput (as shown in Figure 2). This trend levels off when the node becomes a bottleneck due to overwhelming context-switching overhead. Using multiple nodes for network transfer continues this trend until the wide area network becomes saturated.

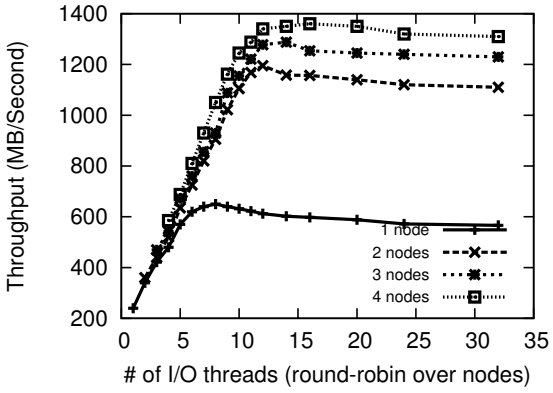

Figure 1: Disk read throughput of a 32GB file on GPFS using nodes with 10Gbps NIC.

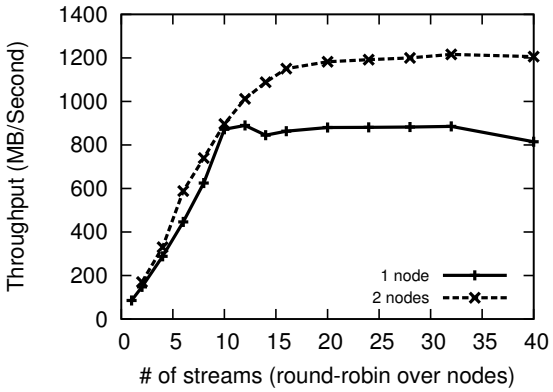

Figure 2: Memory-to-memory transfer of 32GB between ALCF and NERSC using nodes with 10Gbps NIC.

4 Design

4.1 Assumption

In designing the high-throughput data transfer system, we have been guided by assumptions that offer both challenges and opportunities. We lay out our assumptions in detail:

- The nodes that are responsible for data transfers are heterogeneous and may be dynamically added or removed. Some nodes may have faster memory access and/or network interfaces. Exploiting all the available resources is critical to increase throughput.

- The workload consists of big file transfers over

wide and local areas. Connection setup times are negligible comparing to transfer durations.

- Files are stored in high-performance parallel file systems, where files are heavily striped over many disks. One write/read thread may not be able to saturate the file system.
- Wide area networks have large latency-bandwidth product and may be lossy, where one TCP connection may not be able to achieve the highest throughput.
- The bottleneck of the end-to-end transfer may lie in the file systems, end hosts, or the network for a given transfer. The bottleneck might change during the data transfer.
- End users may not have the expertise to understand the system and thus the system must be able to tune the performance automatically without any input from the user.

4.2 Architecture

Our system consists of a single mover master, numerous movers, multiple transfer front-ends, and transfer clients, as shown in Figure 3. A mover is a user-level process that may consists of multiple I/O threads and network transfer threads. A node may provide multiple movers based on its computation, I/O, and network capacity. Movers are divided into clusters. Movers of the same cluster have the same access right to a specific file system and usually physically coupled with or close to the file system storage devices. Each mover is listening on a freely chosen port for potential incoming connections. TCP connections are established between movers (which is called *pairing*) and actual data is transferred only between movers. Each file is divided into fixed-size chunks. Each chunk is identified with a pair (file offset, chunk size). Transfer managers assign chunks to movers for transfer. When a mover is assigned to a specific transfer request, it will transfer the file data of the request until the file transfer completes. When a mover completes or is going to complete its chunk, it asks for more chunks. When no more chunks are available, the source mover disconnects with its peer destination mover (which is called dis-pairing) and then both of them are ready to serve other transfer requests.

The mover master maintains all mover metadata using a mover pool. Each dis-paired mover (after transfer completion) registers itself to the mover master, including address (IP address and listening port), cluster identification, and configurations (e.g., the number of I/O threads and allowed number of network connections). Having a single master vastly simplifies our design and enables the master to make sophisticated mover pairings using global knowledge. However, we must minimize its involvement in managing transfer processes so that it does not become a bottleneck. We introduce transfer managers, which are responsible for managing actual transfers. A transfer manager accepts requests from clients, and requests for idle movers from the mover master, pairs movers, and assigns file chunks to movers.

Lets look at the flow of interactions for a simple file transfer with reference to Figure 3. First, a transfer request (with source and destination cluster identification and file paths) is issued by the client. The request is assigned to a transfer manager, which is usually the transfer manager closest to the source cluster for smaller communication latency between transfer manager and source movers. The transfer manager requests for idle movers from the mover master at both the source and destination clusters. When a pair of movers are returned from the mover master, the transfer manager instructs the destination mover to establish connection to the source mover. Upon the connection establishment, the source mover asks for chunks of the file to transfer. The transfer manager allocates chunks to requesting movers till the end of the file. When there are no more chunks available, the allocated movers are dis-paired and returned to the mover pool at the mover master, which then can serve other transfer requests.

4.3 On Demand Chuck Allocation

Since movers may have different capacity and end-system and network conditions may change unpredictably, movers request for chunks on-demand. By default, each chunk is 1G bytes and the size of a block

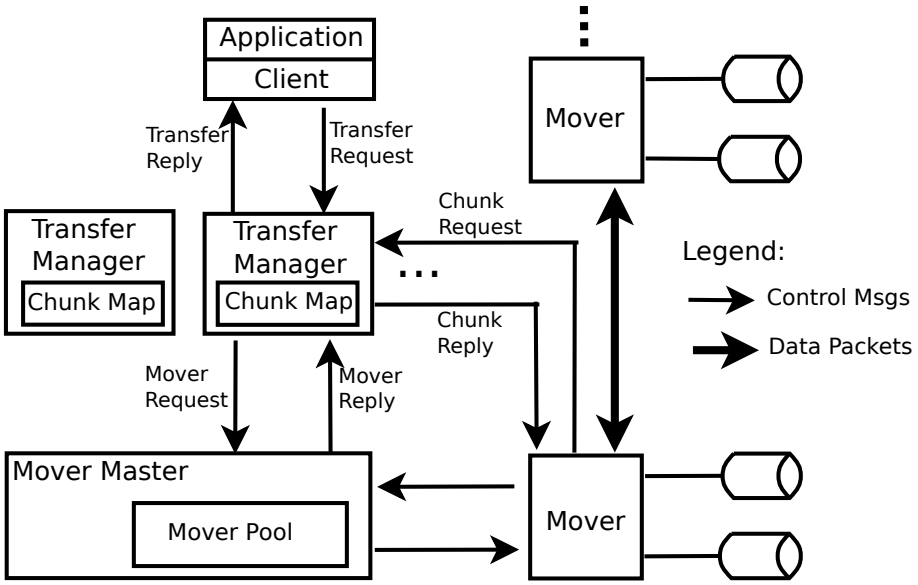

Figure 3: Architecture of our data movement system.

read is 4M bytes. The I/O threads of a mover synchronize locally (with the same node) to read blocks within the allocated chunks, which avoids frequent communication between the mover and the transfer manager and therefore avoids the transfer manager from becoming a bottleneck. Each mover starts to request for more chunk only when the number of bytes remaining in the currently allocated chunk is less than a given threshold. The threshold should be set as the maximum bytes that the mover can read in the largest round-trip time of a chunk request and reply. By default, the threshold is 256 MB. Once a transfer manager receives a chunk request from a mover, the next unassigned chunk is allocated to this mover.

4.4 Automatic Performance Tuning

Movers periodically report its performance information to transfer managers. Since a transfer is managed by only one transfer manager, the transfer manager can collect the performance data of all movers involved in the transfer. Using the global knowledge

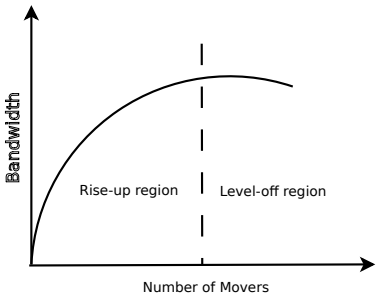

Figure 4: Automatic Performance Tuning.

and the historic performance data of the transfers, the transfer manager can be aware of potential opportunity to improve throughput in time. The transfer manager plots a graph, as shown in Figure 4, based on the history data. The graph is divided into two portions: the rise-up and level-off portions. The graph is dynamically updated according to the newest performance report, reflecting the changing conditions of network and end system in time. When a transfer lies in the rise-up portion, the transfer manager on

behalf of the transfer can request more movers from the mover master, otherwise, a pair of movers should be released and return to the mover pool.

5 Experiments

We perform experiments in two typical settings: a local area network (LAN) and a wide area network (WAN). Nodes for LAN and WAN transfers are set up both homogeneously and heterogeneously. Each node in the LAN has a 1 Gbps and 10 Gbps Myrinet interface. Data is transferred in WAN from Argonne in Illinois and NERSC in California. Hosts are either 2.1 GHz 8 core SMP processors with at least 8 GB memory and 2 GB swap space. In all tests, we set the TCP buffer size, file block read/write size to the optimal values specific to each host. We used files and memory content of size 32 GB. We use multiple nodes for both source and destination. We compare the performance of our system with manually tuned GridFTP.

5.1 LAN Performance

We vary the number of nodes in the sender and receiver clusters for the LAN transfers, and vary the number of movers at each node as well to utilize the available resources effectively. Each mover has one I/O thread and one network thread for each TCP connection over a 10 Gbps interface. We benchmarked the maximum disk read/write performance and network bandwidth and the results are shown in Table 1. Note that the maximum bandwidth achievable by a single file read/write is slightly lower than the aggregated bandwidth achievable by simultaneous multiple file reads/writes since a single file read/write usually can not saturate a production parallel file system.

Figure 5 shows that as the number of nodes (where 10 Gbps interface is used for each node) per cluster or the number of movers per node increases, the throughput of the single file transfer increases rapidly mainly due to the fact that a single mover can not saturate either the high-performance file system or the high-speed network. As the number of movers increases per node beyond the number of processor per node, the aggregate bandwidth per node starts to level off since it reaches the maximum read/write bandwidth achievable by a single node. Moreover, the achievable maximum bandwidth by a single node is less than the maximum bandwidth by multiple nodes mainly due to the context-switching overhead among multiple movers at a single node.

Table 1: Maximum disk read/write of a single file and network bandwidth over a LAN

Operations	# of nodes	Max BW (MBps)
Read	1	708
	2	1286
	4	1912
	8	1888
	16	1868
Write	1	520
	2	790
	4	890
	8	950
	16	980
Net	1	1220
	2	2050
	4	4024
	8	7980
	16	15200

We also set up the GridFTP servers that stripe over the 4 nodes at both the sender and receiver side. Among them, 2 nodes use 1 Gbps interface and 2 nodes use 10 Gbps interface for data transfers. We report the maximum bandwidth achieved by GridFTP with such a configuration and compare it with our system with the same nodes at both the sender and receiver side, as shown in Figure 6. We observed that the throughput of GridFTP is almost constant. The reason lies in the static striping mode of GridFTP, which statically and equally distributes file chunks/blocks to each node, no matter how different the capacities of these nodes are. As a result, the throughput of the entire transfer is same as the throughput obtained from the least-capable node. In the aforementioned heterogeneous configuration, the

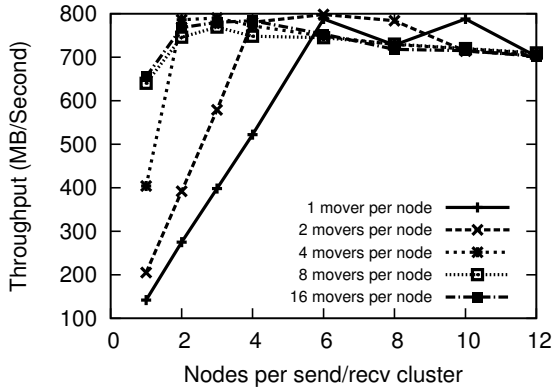

Figure 5: Disk-to-disk throughput with varying parallelisms for single file transfer over LAN.

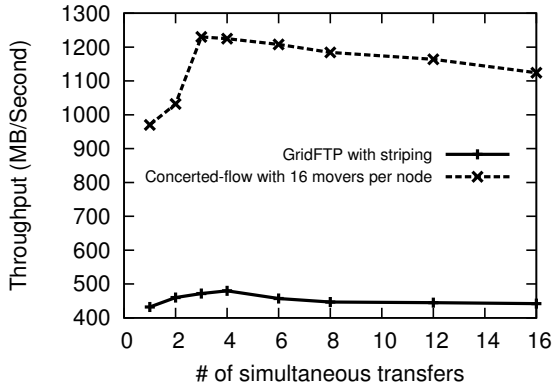

Figure 6: Disk-to-Disk throughput for multiple file transfers over LAN.

two nodes with 1 Gbps become a bottleneck for multiple file transfers while the other two nodes with 10 Gbps is not fully utilized. All transfers need to transfer half of their file chunks over these two slow nodes. In contrast, the movers in our system dynamically ask for more chunks upon the completion of previous chunks. Movers on the two nodes with 10 Gbps interface transfer proportionally much more data than the other two nodes with 1 Gbps interface. Therefore, the system maintains a high overall throughput.

Table 2: Maximum disk read/write of a single file and network bandwidth over the homogeneous WAN

Operations	# of nodes	Max BW (MBps)
net	1	690
net	2	1200

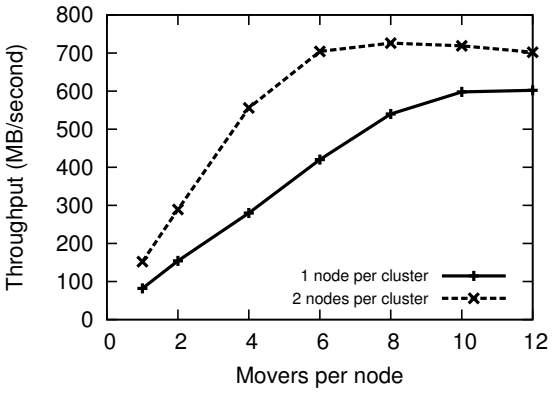

Figure 7: Disk-to-disk throughput with varying parallelisms for single file transfer over a homogeneous WAN.

5.2 WAN Performance

5.2.1 Homogeneous Configuration

We transfer files from two DTNs at ALCF to two DTNs at NERSC. Each node has a 10 Gbps interface. We benchmark the maximum network bandwidth as in Table 1, the disk read/write performance is similar to Table 1 since these nodes share the same file system.

We vary the number of movers at each node and the throughput, as observed by the end-user, for a single file transfer increases dramatically, as shown in Figure 8. The figure shows that a single node can transfer up to 660 MBps with multiple movers. The bandwidth can boost up to 1200 MBps with two nodes.

We then compare the performance of GridFTP and our system in the aforementioned homogeneous configuration over a WAN from ALCF to NERSC. We

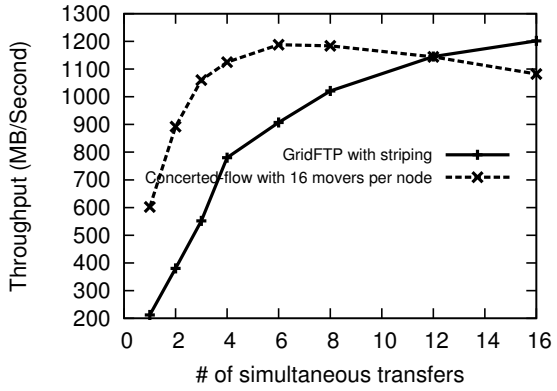

Figure 8: Disk-to-disk throughput for multiple file transfers over the homogeneous WAN.

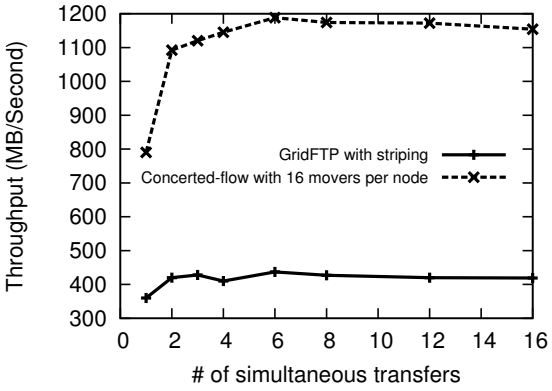

Figure 9: Disk-to-Disk throughput for multiple file transfers over heterogeneous WAN.

transfer files between NERSC and ALCF using two nodes at each side. Each node is connected to the network with a 10 Gbps interface. We observed that the achieved throughput of our system in our experiments is several factors greater than finest-tuned GridFTP for light and medium workload since that GridFTP does not use all available resources to maximize the throughput. However, when the workload is heavy, the achieved throughput of our system is slightly lower than the finest-tuned GridFTP. We are investigating the reason for this and ways to improve the performance of our system under heavy workload.

5.2.2 Heterogeneous Configuration

We also compare the performance of GridFTP and our system in a heterogeneous configuration over WAN. Both source and destination cluster consist of 4 nodes, two nodes with 1 Gbps interfaces and the other two with 10 Gbps interfaces. Each node have 16 movers for our system. Transfers are striped over the 4 nodes for GridFTP. We vary the number of simultaneous transfers (a transfer batch) the cluster undertaking and report the aggregate bandwidth, which is calculated as the ratio of the aggregate transfer file sizes to the batch completion time.

Like Figure 6, Figure 9 shows that our system has much higher aggregate bandwidth due to the on-demand chuck allocation.

6 Conclusion

High-bandwidth, high-latency long-haul optical networks are becoming increasingly available to researchers and scientists. But high end-to-end performance is often elusive for various reasons. When considering sustained end-to-end performance, rather than instantaneous network performance, numerous additional issues arise. The file systems at the endpoints may require considerable parallelism in order to store/retrieve data rapidly. Network conditions and load on the end systems may vary dynamically. Bottlenecks may exist at any component along the end-to-end path. Most instantaneous or lasting end-to-end transfer bottlenecks can be overcome by increasing the level of hardware or software parallelism at these bottleneck. Therefore, exploiting parallelism of end systems and networks dynamically and optimally can accelerate end-to-end transfers. We designed a prototype that exploits available parallelisms dynamically and adaptively. The design overcomes some limitation of the GridFTP design. The preliminary results show that (1) in a homogeneous configuration, the design can achieve better throughput for light and medium workload than GridFTP and

achieve approximately same throughput for heavy workload, (2) and in a heterogeneous configuration, the design can achieve several factor higher throughput for all workloads than GridFTP.

References

[1] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. The globus striped gridftp framework and server. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC '05, pages 54–, Washington, DC, USA, 2005. IEEE Computer Society.

[2] M. Balman and T. Kosar. Dynamic adaptation of parallelism level in data transfer scheduling. In *CISIS*, pages 872–877, 2009.

[3] M. Beck, T. Moore, and J. S. Plank. An end-to-end approach to globally scalable network storage. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '02, pages 339–346, New York, NY, USA, 2002. ACM.

[4] V. Bhat, S. Klasky, S. Atchley, M. Beck, D. McCune, and M. Parashar. High performance threaded data streaming for large scale simulations. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, GRID '04, pages 243–250, Washington, DC, USA, 2004. IEEE Computer Society.

[5] J. Bresnahan, M. Link, R. Kettimuthu, and I. Foster. Managed gridftp. In *Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, IPDPSW '11, pages 907–913, Washington, DC, USA, 2011. IEEE Computer Society.

[6] B. Cohen. Incentives Build Robustness in BitTorrent. In *In Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*, 2003.

[7] T. Ito, H. Ohsaki, and M. Imase. Automatic parameter configuration mechanism for data transfer protocol gridftp. In *Proceedings of the International Symposium on Applications on Internet*, SAINT '06, pages 32–38, Washington, DC, USA, 2006. IEEE Computer Society.

[8] T. Ito, H. Ohsaki, and M. Imase. Gridftp-apt: Automatic parallelism tuning mechanism for data transfer protocol gridftp. In *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, CCGRID '06, pages 454–461, Washington, DC, USA, 2006. IEEE Computer Society.

[9] W. E. Johnston, W. Greiman, G. Hoo, J. Lee, B. Tierney, C. Tull, and D. Olson. High-speed distributed data handling for on-line instrumentation systems. In *Proceedings of the 1997 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '97, pages 1–19, New York, NY, USA, 1997. ACM.

[10] R. Kettimuthu, A. Sim, D. Gunter, B. Allcock, P.-T. Bremer, J. Bresnahan, A. Cherry, L. Childers, E. Dart, I. Foster, K. Harms, J. Hick, J. Lee, M. Link, J. Long, K. Miller, V. Natarajan, V. Pascucci, K. Raffenetti, D. Ressman, D. Williams, L. Wilson, and L. Winkler. Lessons learned from moving earth system grid data sets over a 20 gbps wide-area network. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 316–319, New York, NY, USA, 2010. ACM.

[11] S.-C. Son. The kangaroo approach to data movement on the grid. In *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing*, HPDC '01, pages 325–, Washington, DC, USA, 2001. IEEE Computer Society.

[12] M. Swany. Improving throughput for grid applications with network logistics. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, SC '04, pages 23–, Washington, DC, USA, 2004. IEEE Computer Society.

[13] E. Weigle and A. A. Chien. The composite endpoint protocol (cep): scalable endpoints for terabit flows. In *Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2 - Volume 02*, CCGRID '05, pages 1126–1134, Washington, DC, USA, 2005. IEEE Computer Society.

[14] E. Yildirim, M. Balman, and T. Kosar. Dynamically tuning level of parallelism in wide area data transfers. In *Proceedings of the 2008 international workshop on Data-aware distributed computing*, DADC '08, pages 39–48, New York, NY, USA, 2008. ACM.