A Technique for Moving Large Data Sets over High-Performance Long Distance Networks

Bradley W. Settlemyer, Jonathan D. Dobson, Stephen W. Hodson,
Jeffery A. Kuehn, Stephen W. Poole, Thomas M. Ruwart
Oak Ridge National Laboratory
One Bethel Valley Road
P.O. Box 2008
Oak Ridge, 37831-6164
{settlemyerbw,dobsonjd,hodsonsw,kuehn,spoole}@ornl.gov,tmruwart@ioperformance.com

***Abstract*—In this paper we look at the performance characteristics of three tools used to move large data sets over dedicated long distance networking infrastructure. Although performance studies of wide area networks have been a frequent topic of interest, performance analyses have tended to focus on network latency characteristics and peak throughput using network traffic generators. In this study we instead perform an end-to-end long distance networking analysis that includes reading large data sets from a source file system and committing the data to a remote destination file system. An evaluation of end-to-end data movement is also an evaluation of the system configurations employed and the tools used to move the data. For this paper, we have built several storage platforms and connected them with a high performance long distance network configuration. We use these systems to analyze the capabilities of three data movement tools: BBcp, GridFTP, and XDD. Our studies demonstrate that existing data movement tools do not provide efficient performance levels or exercise the storage devices in their highest performance modes.**

I. Introduction

The time spent manipulating large data sets is often one of the limiting factors in modern scientific research. In fields as diverse as climate research, genomics, and petroleum exploration, massive data sets are the rule rather than the exception. With multi-Terabyte and Petabyte data sets becoming common, previously simple tasks, such as transferring data between institutions becomes a challenging problem. The push to Exascale computing promises to increase the difficulty level in at least two ways. First, with projected system memory sizes greater than 32 Petabytes, typical Exascale data sets will be significantly larger than the total amount of storage in existing file systems. Further, because of the staggering operational costs of leadership-class Exascale machines, we expect that only a small number of such machines will initially exist, increasing the number of users interested in remote data analysis.

In this project we examine the performance of three data movement tools: BBcp, GridFTP, and XDD. Both BBcp and GridFTP are popular data movement tools in the HPC community. However, both tools are intended for a wide audience and are not optimized to provide maximum performance. In constructing our test bed we have focused on technologies and performance constraints that are applicable to Exascale performance levels. In particular, we have focused on analyzing the impacts of distance (i.e. network latency) in transferring large data sets and the importance of achieving device-level performance from each component participating in the data transfer.

A. Related Work

The most popular tool for moving large data sets is likely GridFTP [1], a component of the Globus toolkit [2]. The Globus GridFTP client, *globus-url-copy*, provides support for high performance networking with the UDT protocol and also supports parallel file system aware copying via the use of transfer striping parameters. Our tool, XDD, currently provides less security than GridFTP, and focuses on disk-aware I/O techniques to provide high levels of transfer performance.

Another popular high performance transfer tool is BBcp [3]. BBcp is designed to securely copy data between remote sites and provides options for restarting failed transfers, using direct I/O to bypass kernel buffering, and an ordered mode for ensuring data is both read and written in strict serial order. Our tool, XDD, attempts to copy many of the features available in BBcp, while also providing a disk-aware implementation.

Efficient use of both dedicated and shared 10 Gigabit Ethernet network links has been a popular area of study for several research teams. Marian, et al., examined the congestion algorithm performance of TCP flows over high latency dedicated 10 Gigabit Ethernet network connections and found that modern congestion control algorithms such as HTCP and CUBIC provide high levels of performance even with multiple competing flows [4]. Wu, et al., and Kumazoe, et al., studied the impacts of congestion control on shared 10 Gigabit Ethernet links [5], [6]. Both efforts found that congestion control algorithms strongly impacted performance at long distances. XDD provides a configurable number of threads and supports modified TCP window sizes to provide performance in shared network scenarios; however, XDD is primarily designed as an high-end computing (i.e. Exascale)

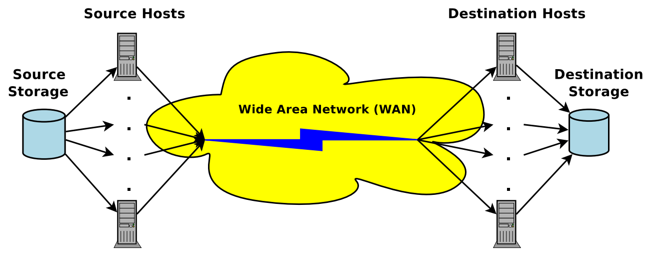

Fig. 1. Hardware components participating in the impedance matching for a long distance file transfer.

data movement tool where circuit switched dedicated network links are more likely to be available.

II. High Performance Data Transfers

Fundamentally, a high performance file transfer is a matter of tuning each component to provide high levels of individual performance and matching the performance of each connected hardware component. The five primary hardware components required for a file transfer are:

1) Source storage devices,
2) Source transfer hosts,
3) Wide area network (WAN),
4) Destination transfer hosts, and
5) Destination storage devices.

This problem formulation can be thought of as special type of an impedance matching problem; where each component must provide data throughput compatible with the connected components. Many of the difficulties of a high performance file transfer are ensuring that each component is both executing in its fastest performance mode and not interfering with the performance of the adjacent components.

Figure 1 shows the direction of data flow for a long-distance file transfer. In the simplest scenario, a system consisting of two hosts with local hard drives and Internet connections may use SCP or an FTP client to transfer a file. However, the size of data sets employed for HPC use cases typically require dedicated data transfer hosts with large amounts of memory, high performance network interface cards (NICs) and access to high speed storage arrays. Further, the interconnection network is often a high-speed, circuit-switched network that allows dedicated (or nearly so) connections between the HPC connection sites. In terms of leadership computing facilities, it is likely to be the case that a small number of leadership-class machines will exist, with many scientists from multiple institutions generating massive data sets requiring analysis.

A. XDD Architecture

XDD is designed to provide the software infrastructure required to move large data sets with high levels of performance and reliability. One advantage of using the XDD software package as a platform for developing our end-to-end data mover is that XDD was designed specifically to drive disks and disk arrays at their maximum bandwidth. Our enhanced version of XDD provides several options to facilitate better file transfer impedance matching: configurable device access schemes, configurable numbers of threads, and configurable I/O scheduling policies.

1) Device Access Schemes: An end-to-end data transfer operation is accomplished by executing *matched pairs* of XDD instances with one instance running on a destination host and another instance running on a source host. The destination instance acts as a server that binds to the network, receives data from the source, and commits the data to storage. The source XDD instance connects to the destination host, reads data from the source file system, and transmits the read data over the network. Data movement is always from the source host to the destination host. In order to ensure that I/O device accesses are efficient, XDD allows the use of direct I/O to access storage devices, file pre-allocation support, easy configuration of I/O buffer sizes, and TCP window size configuration.

Enabling file access with direct I/O is one of the most important ways XDD achieves higher levels of performance than existing file transfer packages. POSIX file I/O in the Linux operating system requires the use of a buffer cache that requires data to be copied at least once in kernel space before being read into application memory or written to disk. Although memory performance is typically much faster than disk performance, the cost of copying a 10TB file into kernel memory and then into application memory is far from free.

Another important component of direct I/O is file pre-allocation. With buffered I/O, the kernel commits file data to disk in serial order ensuring the file grows over time. With direct I/O, it is possible for the file to be constructed out of order, resulting in file fragmentation (at least for many file systems). By pre-allocating the destination file, XDD ensures that the file is mapped into logically contiguous disk regions, avoiding file fragmentation and the resultant low performance when the destination file is itself read.

To further ensure high performance disk access, XDD also allows the configuration of the I/O request size. Large I/O requests typically provide higher levels of file access performance by ensuring that all disks in an array are simultaneously active and reducing the number of context switches. However, the I/O requests must also be small enough that both network and disk accesses can be overlapped and pipe-lined to the degrees necessary to achieve high performance levels.

2) Thread Count: A second critical factor in matching network performance with storage system performance is ensuring adequate degrees of parallelism. Our experiments indicate that long distance TCP-based networking generally supports large numbers of threads well, and at smaller TCP window sizes requires large numbers of threads to achieve high levels of throughput. XDD allows the user to control threading behavior at a fairly fine-grained level.

For a file transfer, an XDD process will spawn a single *Target Thread* that opens the file, and then spawns a user specified number of *QThreads*. If the XDD instance is invoked as a file transfer destination, the QThreads will each bind to a network port and wait for a connection from the source-side process. After opening the file in read-only mode, the source-side XDD process will spawn the specified number of QThreads, which will each connect to one of the listening threads. XDD avoids the use of poll and select, which are sometimes costly when a large number of threads are used.

3) I/O Scheduling Policies: Originally, XDD we provided a single scheduling option called *strict ordering*. Strict ordering simply assigned file offsets to each thread statically in a round-robin ordering. So, in conjunction with a 1MB I/O request size and four QThreads, thread zero would be responsible for transferring a single Megabyte beginning at offsets 0, 4, 8, 12, and so on until reaching the file end. Similarly, thread one would transfer a Megabyte from file offsets 1, 5, 9, 13, etc. While this scheme proved very easy to reason about from a software correctness point of view, the lack of effective thread synchronization during large file transfers eventually caused the disk arrays to waste time seeking between tracks due to out of order I/O requests.

Our first attempt at optimizing the I/O access scheme was *serial ordering*. Serial ordering ensures that all disk accesses are performed in serial order by forcing threads to wait until all preceding requests are completed. The manager thread, or Target Thread, dispatches the I/O requests in serialized order to the QThreads. The QThreads immediately enter a synchronization point and are released from the synchronization point in disk access order. The serial ordering policy ensures that all file accesses are contiguous, and in the case of accessing a single hard disk drive, often results in optimal performance. However, the synchronization overhead of serial ordering is not free, and a small delay exists between each I/O request dispatched to the storage system.

The *loose ordering* scheduling policy is our attempt to develop an I/O request scheduling policy that relaxes synchronization costs in order to achieve higher levels of file access performance. As I/O requests are dispatched to the QThreads, the threads again enter a single synchronization point and exit the synchronization in request order. The difference between serial ordering and loose ordering is that in serial ordering, the next QThread to run releases from the synchronization point only after the preceding thread has completed its I/O access, whereas in loose ordering, threads release from the synchronization point immediately *before* issuing their I/O operation. When the executing threads wait only on I/O (i.e. there are enough cores to execute all of the running threads) loose ordering can increase the issue rate to the disk array controller and ensure that the I/O requests are issued in an order that is mostly serial. The small number of out of order requests can typically be re-ordered by the disk array controller, ensuring that storage media access occurs in strictly serialized fashion and achieves the highest possible performance levels.

Finally, XDD offers a *no ordering* policy that imposes no synchronization overhead. This is useful on the destination side of file transfers, where data is usually arriving in a mostly serial order due to the source-side scheduling policy. In this mostly-serialized case, the disk array controller can often perform the re-ordering necessary to ensure that storage media access occurs in the highest streaming performance modes.

III. Methodology

A. Storage Infrastructure

Our storage testbed consisted of six Infortrend EonStor S16F-R1430 disk arrays equipped with two controllers and 16 Hitachi DeskStar E7K500 disks. The controllers were configured to provide RAID level 5 in a 7+1 configuration with 64KiB stripe size. Each controller was connected into a storage area network (SAN) with a single 4Gbps Fibre Channel connection. The storage network fabric was a Brocade Silkworm 4100 switch. For our configuration it was necessary to split the 6 shelves (12 controllers) into 4 file systems.

B. Host Infrastructure

Our host configurations used standard commodity parts and the Linux operating system. All four systems were identical with the exception that the machines named pod7, pod9, and pod10 had dual Quad-Core AMD Opteron 8382 processors, whereas our fourth system, pod11, had dual Quad-Core AMD Opteron 2358 SE processors. All four systems had 32GB of main memory, a Myricom Myri-10G Dual-Protocol network interface card (NIC), and two dual port QLogic ISP2432-based 4Gb Fibre Channel host bus adapters. The 10G NICs were connected to the long haul network described later, and each host had three Fibre Channel connections into the Fibre Channel switch as described in Section III-A.

The host systems ran Fedora 13 version of the Linux operating system using kernel version 2.6.33.3-85. To aggregate the three storage units into a single file system per host we constructed a single volume group striped across each physical volume with a 64KiB stripe size. We then constructed a local XFS file system on each host using the default file system parameters with the exception that we used 4KiB block sizes.

C. Network Infrastructure

The source and destination nodes were interconnected by the Department of Energy's UltraScience Net (USN), an ultra-scale network testbed for large-scale science [7]. The USN facility included a dedicated OC192 connection that connected Oak Ridge National Laboratory (ORNL) to Fermilab in Chicago, Pacific Northwest National Laboratory in Seattle, and the Stanford Linear Accelerator Center in Sunnyvale, California. The network was constructed with two lambdas; however, since we configured the network to both begin and end at Oak Ridge, we were only able to achieve single OC192 connection speeds (9.6Gbps). Table I shows the round-trip-time (RTT) measured with the ping utility for each of the wide area network configurations.

D. Data Movement Software

Several software packages have been built for high performance data movement. In this section we focus on describing two of the most popular tools, BBcp and GridFTP, and our own data movement software package, XDD.

TABLE I
UltraScience Net (USN) Latency Measurements

Network Loop Details	Distance (mi)	RTT (ms)
ORNL	0.2	0.28
ORNL-Chicago	1400	26.8
ORNL-Chicago-Seattle	6600	128
ORNL-Chi.-Sea.-Sunnyvale	8600	163

1) BBCP: Originally built to transfer large data sets as part of the BaBar Collaboration [8], BBCP was developed by the the SLAC National Accelerator Laboratory (formerly the Stanford Linear Accelerator Center). The major advantage of BBCP versus traditional file transfer tools such as FTP and SCP is increased performance for large data transfers, particularly over large distances. BBCP supports a host of data transfer options included multiple I/O threads, network compression, and an append mode that allows a previously cancelled transfer to be resumed. Additionally, BBcp provides support for an *ordered* transfer mode that ensure file data is read and written in serial order, and an *un-buffered* mode that support direct I/O file access as long as the file size is a multiple of 8KiB.

2) GridFTP: GridFTP is a core service within the Globus Grid Toolkit, and provided by Globus' GridFTP servers and the *globus-url-copy* GridFTP client. GridFTP provides features for cluster-to-cluster end-to-end file transfers including file striping support, load balancing across independent numbers of source and destination hosts, multiple network protocols, and a data source plugin architecture that may be effective for improving file I/O performance. GridFTP allows the use of the Grid Security Infrastructure (GSI) to provide certificate based authentication for all file transfers.

3) XDD: Although not designed to transfer file data over the network, the efficient disk access routines in XDD made it an excellent starting point for building a disk-aware file transfer utility. We added a very simple networking implementation that transforms XDD into a point-to-point data mover. In general, we have attempted to copy the user interface of BBcp where possible.

IV. Long Distance Data Transfer Performance

As we described in Section II, sustained performance for a long distance data transfer requires efficient performance from each hardware component of the transfer and a mechanism for coupling the highest performance modes for each device into a high performance implementation. The remainder of this section provides a detailed performance analysis of the hardware components and the three file transfer software packages.

A. Network Performance

Table I shows the distance and roundtrip latency measurements for each of the USN network loop configurations. To measure network bandwidth over each loop configuration we ran Iperf for 60 seconds while varying both the number of

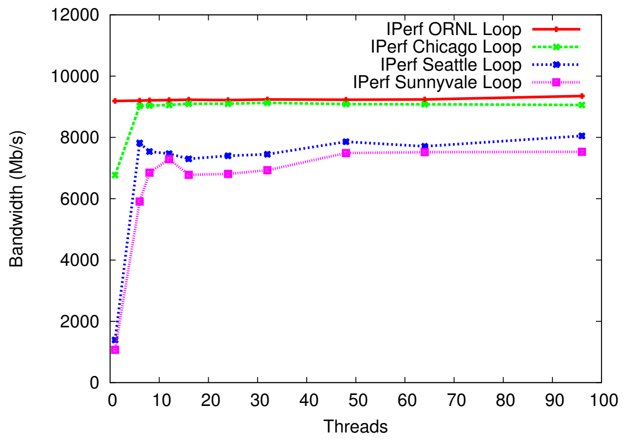

Fig. 2. USN network loop bandwidth measurements with IPerf using a 16MiB TCP Window size.

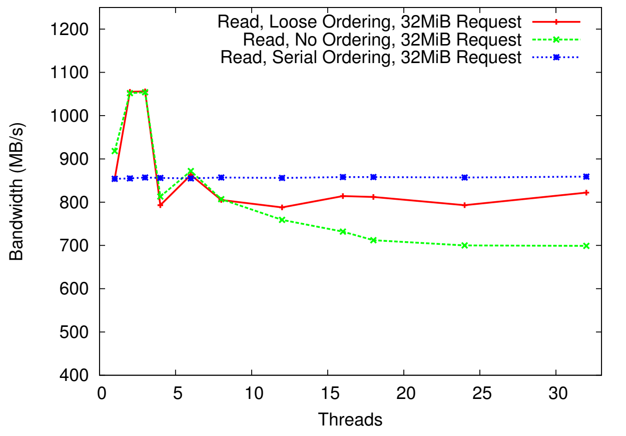

Fig. 3. Disk array read performance.

TCP flows. Figure 2 shows the network bandwidth in Megabits per second (Mbps) using a 16MiB TCP window size. We can see that with low latencies, network bandwidth can be maxed out by a single flow; however, as the latency is increased it becomes necessary to increase the number of flows to achieve maximum throughput.

B. Disk Array Read Performance

XDD was originally designed as a tool for measuring I/O system throughput. We have used the performance measurement features within XDD to profile reading 200GiB of random data from pod10 with a 32MiB request size to measure the impacts of I/O thread count and thread scheduling strategies on disk array read bandwidth. Figure 3 shows the performance of reading data for each the thread scheduling strategies. Although small numbers of reading threads provides high levels of file read bandwidth, without thread ordering large numbers of I/O threads cause the disk read performance to decrease dramatically. For the source side of our file transfers, we have elected to use the loose-ordering algorithm.

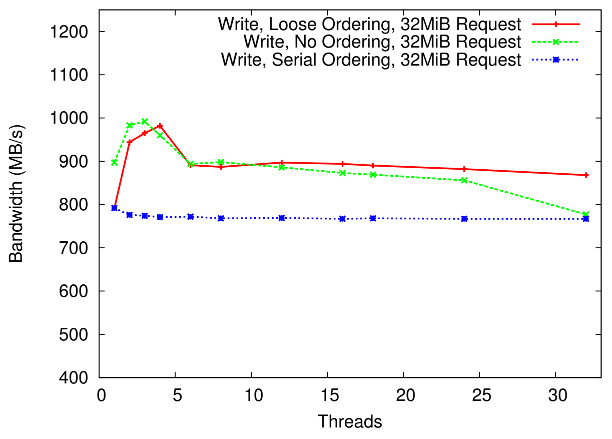

Fig. 4. Disk array write performance.

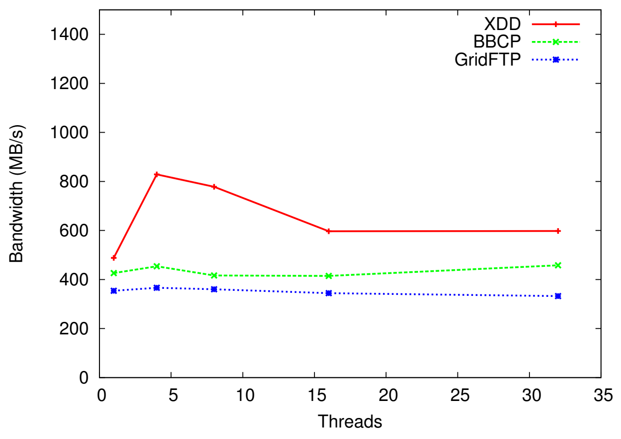

Fig. 5. Comparison of transfer performance over the ORNL loop.

C. *Disk Array Write Performance*

We used the same configuration to measure the file read performance from disk, reading the 200GiB data file with varying request sizes and thread counts. Again we measured only direct I/O, as buffered I/O file system bandwidths were much lower. Figure 4 shows that the loose-ordering produces the highest possible performance, while serial ordering generates the most stable disk performance. However, because we are planning to use loose ordering on the source side of the file transfer, it is not necessary to impose any ordering on the destination side because data should arrive at the destination mostly in order.

D. *End-to-End File Transfer Performance*

Having examined the performance of each of the individual components in the end-to-end file transfer, we note that we expect to be disk limited when using a single source host and single destination host. Our 9.6Gbps USN network can theoretically provide 1200MB/s of data throughput; however, our disk arrays are only barely capable of 1000MB/s. Further, we know that long distance network loops will require multiple flows, and the disk arrays perform best with multiple flows as well. However, the implementation of data movement from disk to network is critical to achieving performance.

1) Low Latency Networks: Figure 5 shows the end-to-end file transfer performance of a 200GiB file copied from pod7 to pod9 over the USN looped through the ORNL cross connection for XDD, BBCP, and GridFTP. For each tool we used a 32MiB request size. Although we tried to further improve BBCP performance by using ordered and un-buffered configuration options, we achieved the highest observed performance by setting only the buffer size to 32MiB. The Globus GridFTP client provides far fewer optimization options, and so we were only able to set the request sizes. More interestingly, even though XDD is the only transfer tool capable of effectively leveraging the file system's direct I/O capabilities, the performance difference between proper impedance matched settings (4 threads achieving 829MB/s) and improperly matched transfer settings (e.g. 16 threads achieving 596MB/s) are nearly 30% different. Clearly proper impedance matching is important.

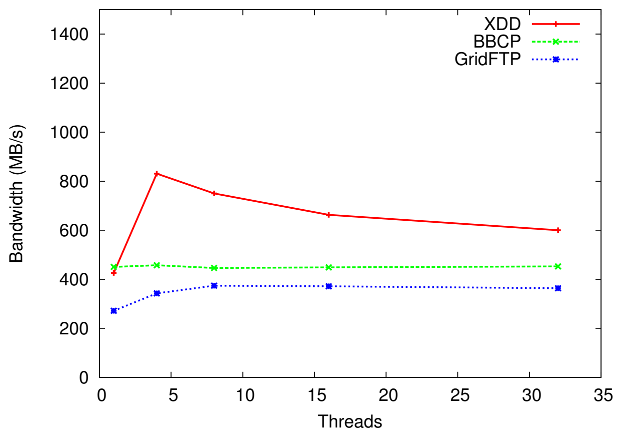

Fig. 6. Comparison of transfer performance over the Chicago loop.

2) Increasing Latency: Figures 6, 7, and 8 show the end-to-end transfer performance over the Chicago, Seattle, and Sunnyvale network loops respectively. Again we see that the request scheduling approach and explicit buffer management used in XDD provides better transfer performance than BBCP and GridFTP. We also again note that choosing the best parameters is critical to maximizing the file transfer performance.

3) Multiple Hosts: Finally, while an efficient implementation will be critical in moving data at extreme performance levels, scalability beyond single endpoint transfers will be necessary as networking technology moves to 40Gb Ethernet and 100Gb Ethernet. We have implemented both multi-host and multi-NIC capabilities within XDD. Figure 9 shows the performance using two source endpoints and two destination endpoints to transfer a 200GiB file over each of the USN network loops. XDD is able to saturate the network by adding flows at each endpoint. For this experiment we did not use a parallel file system, instead we replicated files on the local source file systems, and transferred half of the file to each

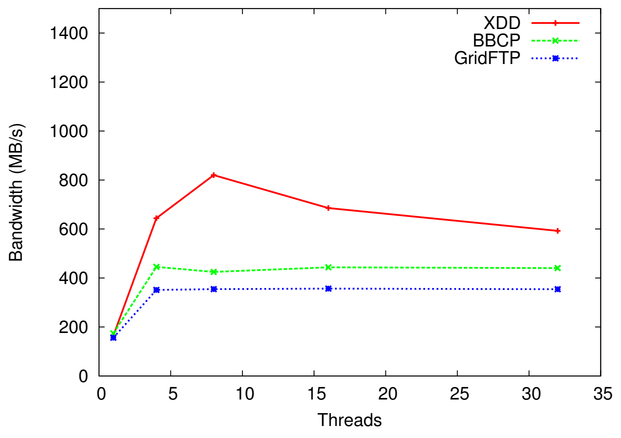

Fig. 7. Comparison of transfer performance over the Seattle loop.

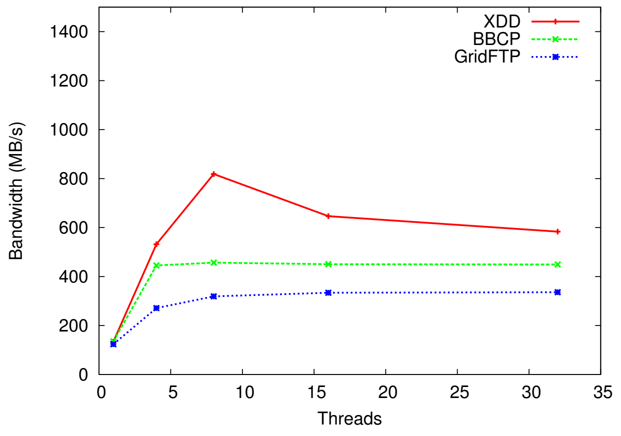

Fig. 8. Comparison of transfer performance over the Sunnyvale loop.

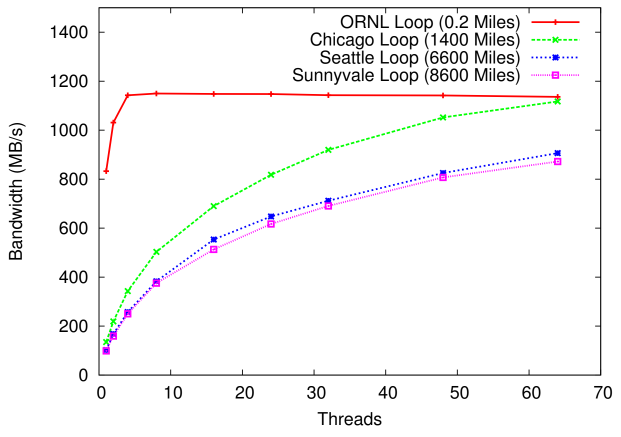

Fig. 9. XDD 2 Hosts transfer performance over dedicated WAN link.

destination endpoint. Although BBCP does not offer multi-host capabilities, GridFTP offers extremely flexible multi-host support. However, due to the network bottleneck limiting the performance of this test, we did not perform testing with GridFTP.

V. Discussion

Our experience developing file transfer capabilities for XDD has demonstrated that an impedance matching model is an effective scheme for high performance end-to-end file transfers. Leveraging the highest performance modes of each of the constituent devices is critical in achieving high performance file transfers. Earlier tools, such as BBcp and GridFTP, provide adequate performance for many types of data transfers; however, with XDD we have focused on pushing file transfer performance as far toward device speeds as possible. We believe that the types of techniques described in this paper are critical to achieve the I/O performance goals set forth for Exascale computing initiatives.

Acknowledgments

This work was supported by the Department of Defense (DoD) and used resources at the Extreme Scale Systems Center, located at Oak Ridge National Laboratory (ORNL) and supported by DoD. This research also used resources at the National Center for Computational Sciences at ORNL, which is supported by the U.S. Department of Energy Office of Science under Contract No. DE-AC05-00OR22725.

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*, ser. SC ’05. Washington, DC, USA: IEEE Computer Society, 2005.

[2] T. G. Alliance, “The Globus Toolkit,” http://www.globus.org/toolkit/.

[3] A. Hanushevsky, “BBCP,” http://www.slac.stanford.edu/~abh/bbcp/.

[4] T. Marian, D. Freedman, K. Birman, and H. Weatherspoon, “Empirical characterization of uncongested optical lambda networks and 10gbe commodity endpoints,” in *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, 28 2010-july 1 2010, pp. 575 –584.

[5] Y. Wu, S. Kumar, and S.-J. Park, “On transport protocol performance measurement over 10gbps high speed optical networks,” in *Proceedings of the 2009 Proceedings of 18th International Conference on Computer Communications and Networks*, ser. ICCCN ’09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–6.

[6] K. Kumazoe, M. Tsuru, and Y. Oie, “Performance of high-speed transport protocols coexisting on a long distance 10-gbps testbed network,” in *Proceedings of the first international conference on Networks for grid applications*, ser. GridNets ’07. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007, pp. 2:1–2:8.

[7] N. Rao, W. Wing, S. Carter, and Q. Wu, “UltraScience Net: network testbed for large-scale science applications,” *Communications Magazine, IEEE*, vol. 43, no. 11, pp. S12 – S17, nov. 2005.

[8] A. Hanushevsky and M. Nowak, “Pursuit of a scalable high performance multi-petabyte database,” in *16th IEEE Symposium on Mass Storage Systems*. IEEE Computer Society, 1999, pp. 169–175.