Maximizing the Performance of Scientific Data Transfer by Optimizing the Interface Between Parallel File Systems and Advanced Research Networks

Nicholas Mills[a,*], F. Alex Feltus[b], Walter B. Ligon III[a]

[a] *Holcombe Department of Electrical and Computer Engineering, Clemson University, Clemson, SC*
[b] *Department of Genetics and Biochemistry, Clemson University, Clemson, SC*

Abstract

The large amount of time spent transferring experimental data in fields such as genomics is hampering the ability of scientists to generate new knowledge. Often, computer hardware is capable of faster transfers but sub-optimal transfer software and configurations are limiting performance. This work seeks to serve as a guide to identifying the optimal configuration for performing genomics data transfers. A wide variety of tests narrow in on the optimal data transfer parameters for parallel data streaming across Internet2 and between two CloudLab clusters loading real genomics data onto a parallel file system. The best throughput was found to occur with a configuration using GridFTP with at least 5 parallel TCP streams with a 16 MiB TCP socket buffer size to transfer to/from 4–8 BeeGFS parallel file system nodes connected by InfiniBand.

Keywords: Software Defined Networking, High Throughput Computing, DNA sequence, Parallel Data Transfer, Parallel File System, Data Intensive Science

1. Introduction

Solving scientific problems on a high-performance computing (HPC) cluster will happen faster by taking full advantage of specialized infrastructure such as parallel file systems and advanced software-defined networks. The first step in a scientific workflow is the transfer of input data from a source repository onto the compute nodes in the HPC cluster. When the amount of data is small enough it is possible to store datasets within the local institution for quick retrieval. However, for certain domains ranging from genomics to social analytics the amount of data is becoming too large to store locally. In such cases the data must be optimally transferred from an often geographically distant central repository.

In genomics, our primary area of interest, the velocity of data accumulation due to high-throughput DNA sequencing should not be underestimated, as data accumulation is accelerating into the Exascale. As of this writing there are six quadrillion base pairs in the sequence read archive database at NCBI [1] with each A,T,G,C stored as at least two bytes. These data represent publicly mineable DNA datasets across the Tree of Life from viruses to rice to humans to elephants. New powerful genetic datasets such as The Cancer Genome Atlas [2] contain deep sequencing from tumors of over 11,000 patients; sequences of over 2,500 human genomes from 26 populations have been determined (The 1000 Genomes Project [3]); genome sequences have been produced for 3,000 rice varieties from 89 countries (The 3000 Rice Genomes Project [4]). These raw datasets are but a few examples that aggregate into petabytes with no end of growth in sight. In fact, if DNA sequencing technology continues to improve in terms of resolution and cost, one could view DNA sequencing as an Internet of Things application with associated computational challenges across the DNA data life cycle [5].

Domain scientists who are not experts in computing often resort to transferring these large datasets using FTP over the commodity Internet at frustratingly slow speeds. Even if a fast network is available, large transfers can take on the order of several days to complete if the correct transfer techniques are not used. The HPC and Big Data communities have developed several tools for fast data transfers but these tools are often not utilized. Several factors contribute to the lack of utilization by scientists of the proper tools. First, awareness of the proper tools in the community is lacking to the point where researchers result to familiar but technically inferior tools such as FTP. Old habits are hard to break. Second, a lack of adequate documentation means even when researchers are pointed to the right tools they are lost in a maze of configuration options. Manuals give an overview of various parameters but fail to explain what parameter values are suitable for high-performance data transfers.

An end-to-end data transfer involves at least three major components: a high-performance storage system, a high-performance network, and the software to tie it all

*Corresponding author
Email addresses: nlmills@g.clemson.edu (Nicholas Mills), ffeltus@clemson.edu (F. Alex Feltus), walt@clemson.edu (Walter B. Ligon III)

together. The various research communities have all examined optimized performance parameters for their respective components. Unfortunately, our experience has been that independently optimizing a single component leads to slower performance in the system as a whole. This work attempts to fill the gaps by suggesting optimized transfer parameters based on experimentally measured end-to-end transfer performance. Where possible the experimentally-determined parameters are supported with appropriate theory.

1.1. Storage component

When considering storage requirements it is important to remember that data transfer is only the first step in an HPC workflow. Presumably there will later be a significant amount of computation performed on the data in parallel. While technologies such as large single-node flash storage arrays may provide the highest raw storage throughput during a transfer, those arrays will quickly become a bottleneck during the computation phase of the workflow as multiple compute nodes compete for access to the centralized storage component. Instead, a parallel file system (PFS) provides a good compromise between raw storage bandwidth and parallel processing scalability.

A PFS is a special case of a clustered file system, where the data files are stored on multiple server nodes. Storing a file on n nodes where the nodes can be accessed simultaneously theoretically allows for a speedup of n times over the single-node case. In practice, the achievable speedup is often much less than n and depends on both the parallel file system implementation and the access patterns of the application performing I/O. It is therefore advantageous to evaluate multiple parallel PFSs to find the one most suited to a particular workload.

In this paper some of the more popular PFSs are evaluated. Among the file systems compared are BeeGFS [6], Ceph [7], GlusterFS [8], and OrangeFS [9]. BeeGFS is later used for file transfer tests once it is determined to have the best performance in a benchmark. Lustre is another popular PFS that was not evaluated because it is not supported for the Ubuntu operating system [10].

1.2. Network component

Access to a fast network with plenty of excess capacity is essential for high performance data transfers. The Science DMZ model [11] is well suited to data transfers because of its support for the long-lived elephant flows typical of large transfers. In particular the lack of firewalls in the model prevents slowdown caused by packet processing overhead. The network infrastructure for our experiments is provided as a part of the CloudLab environment [12, 13, 14].

CloudLab is a platform for exploring cloud architectures and evaluating design choices that exercise hardware and software capabilities. CloudLab is a great benefit to researchers because it allows them to quickly and easily test different modern hardware and software configurations without the usual troubles associated with reinstalling the operating system and re-configuring the network [15]. CloudLab allows the allocation of *bare metal* machines with no virtualization overhead. Thus, CloudLab is a realistic experimental environment for moving real datasets within and between sites before moving algorithms and procedures into production.

A major capability of CloudLab is the ability to connect clusters together over Internet2 using software-defined networking. Our experiments use two of these clusters: the Apt cluster in the University of Utah's Downtown Data Center in Salt Lake City, and the Clemson cluster of Clemson University in Anderson, South Carolina [16].

Communication between the Apt and Clemson CloudLab clusters occurs over the Internet2 network using the Advanced Layer-2 Service (AL2S) [17]. AL2S allows nodes in the two clusters to transparently communicate as if they were connected to the same switch. That is, they appear to be on the same IP subnetwork. The only discernible difference is the relatively high communication latency of 26 ms when compared to a more typical latency of 180 μs.

Configuring a new path over AL2S would normally involve multiple technical and administrative hurdles. The experimenter would need to contact the appropriate network administrator and convince them to set up a new path between sites. The network administrator would need to configure the new path with Internet2 via the GlobalNOC [18]. The delay involved in this process could be significant. For a production network that may run for several years or more such a delay may not matter. But for a temporary experimental network intended to run on the order of weeks or less the large initial setup delay can be significant.

A major advantage of CloudLab for multi-site experiments is that the CloudLab software configures new links over AL2S automatically at the beginning of each experiment. In most cases this process is seamless and requires only a couple minutes of waiting. A researcher need only specify the two sites to be connected and the CloudLab infrastructure handles the rest.

1.3. Software component

Two software tools drive experiments. The first tool, XDD [19], is used to perform benchmarks of parallel file systems. As a benchmark tool XDD makes use of its knowledge of disks to achieve maximum performance by queueing multiple large I/O requests simultaneously. In our experiments XDD uses multiple threads to access different regions of the same file in parallel.

The second tool, GridFTP [20], is a well-known application for performing large data transfers. GridFTP enhances the original FTP protocol with extensions designed to support faster data transfers. It also has a wide variety of networking configuration options. GridFTP uses a client-server model where the server provides access to the

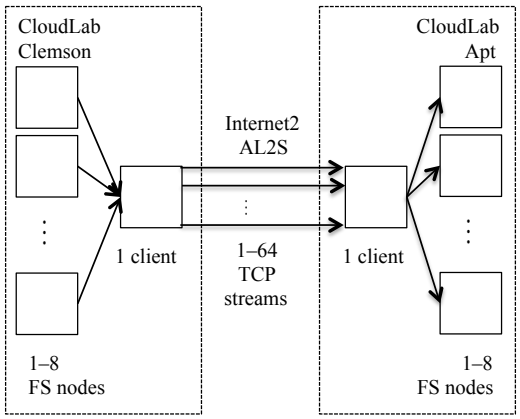


Figure 1: Dataset transfer in the CloudLab environment. Data are read in parallel from 1–8 file system (FS) nodes to the Clemson client. The Clemson client transfers the data in parallel over Internet2 Advanced Layer-2 Service (AL2S) to the Apt client using 1–64 TCP streams. The Apt client writes the data in parallel to 1–8 file system nodes.

host file system, and the client can either push or pull data relative to the server.

2. Materials and methods

2.1. Test dataset

The test dataset used for file transfer experiments is composed of DNA sequencing data in Sequence Read Archive format. There are 345 files in the dataset that range in size from 643 MB to 11 GB. The median file size is 2.4 GB. Over a quarter of the files are less than 1 GB, and less than 11% are greater than 4.5 GB.

2.2. Testbed configuration

As seen in Figure 1, a data transfer experiment is composed of nodes in the Apt CloudLab cluster connected to nodes in the Clemson CloudLab cluster over Internet2's Advanced Layer-2 Service. The round-trip time between these clusters across AL2S is approximately 52 ms. Depending on the experiment both clusters have 1–8 parallel file system nodes, and the number of parallel file system nodes is the same in both clusters. Communication with the parallel file systems uses either TCP or InfiniBand protocols. There is one file system client node per cluster using GridFTP to transfer data in parallel with 1–64 TCP streams. The node hardware at each cluster is as follows:

1. *Apt:* an Intel Xeon E5-2450 8-core CPU at 2.1 GHz with 16 GiB of main memory. Each node has four 500 GB SATA 2.6 hard disk drives at 7200 RPM. The network interface card is a Mellanox MX254A with one port running 56 Gbit/s InfiniBand for communication with the local parallel file system and the other port running 10 Gbit/s Ethernet for communication with the Clemson cluster over Internet2.

Table 1: Network tuning parameters.

Parameter	Value
Low-latency TCP tuning	enabled
Packet queue size	30,000
TCP buffer max	32 MiB
TCP congestion control	H-TCP
TCP selective ACK	disabled
TCP timestamps option	disabled

2. *Clemson:* two Intel Xeon E5-2660 10-core CPUs at 2.2 GHz with 256 GiB of ECC main memory. Each node has two 1 TB SATA 3.0 hard disk drives at 7200 RPM. A QLogic QLE7340 40 Gbit/s InfiniBand HCA is used to communicate with the parallel file system on the other Clemson nodes, and an Intel 82599ES 10 Gbit/s Ethernet NIC is used to communicate with the Apt cluster over Internet2.

The total amount of storage available at Apt and Clemson is similar, but the Apt nodes have twice the number of disk drives. The Clemson nodes have 2.5 times the number of cores as the Apt nodes and 16 times as much main memory.

In order to realize the configuration of Figure 1 the CloudLab graphical configuration tool, Jacks, was used to configure the two clusters and the connections between them. Jacks produces a Resource Specification (RSpec) file that can be used to create identically-configured instances of the same experiment. The RSpec for our experiments is available at [21].

2.3. Installed software

All the bare metal nodes in the experiment are installed with Ubuntu 14.04.1 LTS running Linux kernel v3.13.0-68-generic. For file transfers GridFTP client version 9.19 and server v10.4 were installed from the Ubuntu distribution repository. For file system benchmarks a modified version of XDD is used [22]. The various parallel file systems used are BeeGFS release 2015.03, Ceph v10.2.2-1trusty, GlusterFS v3.7, and OrangeFS v2.9.3.

2.4. Node network configuration

The operating system's network layer is configured based on recommendations from the ESnet Fasterdata Knowledge Base [23], which provides advice for tuning hosts for maximum network performance. All nodes are configured via the Linux *sysctl* interface according to Table 1.

InfiniBand drivers are installed from the operating system's package repository. CPU frequency scaling is disabled to prevent measurement errors caused by the processor speed changing dynamically. An InfiniBand subnet manager is running on the InfiniBand switch.

The CloudLab infrastructure attempts to reserve bandwidth across the experimental network using traffic shaping in the Linux kernel. The traffic shaper is configured to begin dropping packets when the outbound flow rate exceeds a certain threshold. This traffic shaping was disabled

Table 2: XFS mount options for transfers with BeeGFS.

Option	Value
allocsize	131072k
logbsize	256k
logbufs	8
atime	off
barrier	off
diratime	off
largeio	on

on all nodes in the experiment in order to help achieve the maximum possible throughput.

2.5. File system configuration

The parallel file systems used to store data for experiments must in turn store their data on the local file system of each node. XFS [24] is configured as the local file system for each experiment. For file systems other than BeeGFS the default XFS creation and mount options are used. For BeeGFS the default mount options are used in the file system benchmark tests, but in dataset transfer tests the optimized mount options shown in Table 2 are used. These tuning parameters follow the recommendations of the BeeGFS tuning guide [25].

The local file system in turn stores its data on a block device. This device can either be a virtual device created by the logical volume manager or a raw disk device. A virtual disk device is created by using the Linux Logical Volume Manager to stripe data across the available data partitions on all disks. Striping, also known as RAID 0, combines disk storage and in theory increases throughput by allowing disks to be accessed in parallel transparent to the application program. GlusterFS and OrangeFS always use a virtual disk device. Ceph always uses a raw disk device. BeeGFS uses a striped virtual disk in the file system benchmark test and a raw disk device in all other tests.

Further configuaration details depend on the parallel file system being used. The file systems were configured as follows:

1. *BeeGFS:* BeeGFS was installed with one data server per node. The first node runs the management daemon and metadata server in addition to the data server. A single client mounts the file system. The network transport is switched between InfiniBand and TCP depending on the experiment. A connection filter file is used to limit communication of the BeeGFS servers to the experimental network interface (either Ethernet or InfiniBand). The filter file prevents the client and servers from communicating over the slower management interfaces. In order to achieve the maximum amount of parallelism the BeeGFS administration tool *beegfs-ctl* is used to tell the file system to stripe every file across all available storage devices on all storage nodes.
2. *Ceph:* The *ceph-deploy* tool was used to install and configure Ceph on four nodes using all available data partitions. A Ceph file system was created with a single metadata server running on the first file system node. In order to save space the pool size was changed so that only one copy of every object is stored. The number of placement groups for all pools was set to 360. A single client mounts the file system.
3. *GlusterFS:* GlusterFS was installed and a striped GlusterFS volume using TCP as a transport was created on four nodes. A single client mounts the file system.
4. *OrangeFS:* OrangeFS was installed with one data server per node. There is a single metadata server on the first node. The network transport used was TCP. The file system was mounted on one client node by installing the kernel module included in the OrangeFS distribution. The configuration file was generated with the standard *pvfs2-genconfig* script with default configuration options. This default configuration stripes file data across all storage nodes.

Several experiments using BeeGFS or OrangeFS require changing the number of server nodes over the course of the experiment, an expensive operation whose procedure varies depending on the type of server. The number of storage nodes is typically reduced from 8 down to 1. For BeeGFS, the built-in administration tool is run from the client to migrate data off the node being removed onto the remaining nodes, and the BeeGFS server process is stopped on the node to prevent new data from being placed there. In the case of OrangeFS the file system is created from scratch each time because there is no built-in migration tool. Unfortunately this technique requires all the data to be copied to the new OrangeFS file system.

2.6. Software configuration

File system benchmark tests use XDD to measure the throughput when reading and writing a parallel file system. XDD has a plethora of configuration options, but for these experiments the chief options are the target of the I/O operation and its size, the block size, and the number of parallel I/O threads. In the file system benchmarks the only option that changes is the operation (read or write). The target is always a single file filled with random data, the block size is always 4 MiB, and the number of I/O threads is always 4.

The dataset transfer tests use GridFTP to transfer data. GridFTP has both a client and a server, so there are both client options and server options. The server is always run in anonymous mode (no authentication) with a block size equal to the TCP socket buffer size and the number of threads equal to the number of parallel TCP streams. The client is run in *fast* mode with a block size equal to the TCP socket buffer size. The number of parallel data connections and the exact value of the buffer sizes depend on the particular experiment.

2.7. Experiments

All experimental measurements are in units of time. The average throughput is calculated by dividing the total number of bytes transferred by the wall time. For XDD the time value is output as the end of the benchmark. For GridFTP the time is calculated using the elapsed run time output by the standard UNIX *time* tool.

The first experiment in Figure 2 was a benchmark of the BeeGFS, Ceph, GlusterFS, and OrangeFS file systems on the Apt cluster. BeeGFS was configured to use a striped virtual disk for storage and TCP for the network. All file systems used four nodes for the server plus one node as a client. The client node mounted the parallel file system and used XDD to benchmark read and write throughput when operating on a single approximately 864-gigabyte random test file. The read and write performance of each file system was tested 3 times. The test file was deleted between runs of the write test.

The second experiment in Figure 3 was a comparison between BeeGFS and OrangeFS for transfers of the actual dataset. For this experiment BeeGFS was configured to use an XFS file system on the raw disk devices while OrangeFS was configured the same as the first experiment. Files in the test dataset were transferred using GridFTP from Clemson to Apt with 4 parallel TCP streams and varying the number of servers from 1 to 8 with a 4 MiB TCP socket buffer. Each test was run 3 times. Test files were deleted from the Apt cluster between runs.

The third experiment in Figure 4 compares the performance of a GridFTP dataset transfer from Clemson to Apt with BeeGFS when using InfiniBand versus TCP for the local cluster network. The number of servers is varied from 1 to 8 and the number of streams is 1–16, 32, and 64. When the number of servers is varied the number of streams is fixed at 4, and when the number of streams is varied the number of servers is fixed at 5. The Internet2 connection always uses TCP with a socket buffer size of 4 MiB. Each test was run 3 times. Test files were deleted from the Apt cluster between runs.

The fourth experiment in Figure 5 compares 4 MiB and 16 MiB TCP buffers when transferring with 8 BeeGFS servers running InfiniBand and 1–16, 32, 64 streams. Test files were deleted from the Apt cluster between runs.

The final experiment in Figure 6 uses BeeGFS over InfiniBand and a 16 MiB TCP buffer size. The throughput is measured with 1–16, 32, 64 streams and 2–6, 8 servers. Test files were deleted from the Apt cluster between runs.

3. Experimental results

In this study we wanted to identify optimal data transfer parameters for parallel data streaming across AL2S and between two CloudLab sites loading real genomics data onto a parallel file system. The key parameters modified were the file system type, number of file system nodes, network transport protocol, and number of parallel TCP streams. These parameters are of high interest for genomics and other domain scientists across the technical skill spectrum who need to move massive data sets.

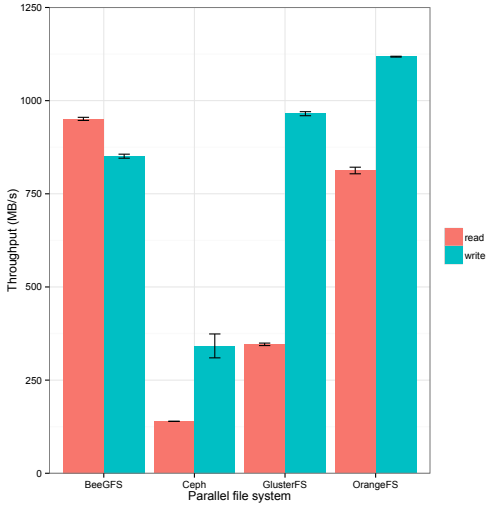


Figure 2: Comparing parallel file system read/write benchmarks. A single client performs I/O to a single file on 4 servers using 4 parallel application threads. Benchmark tool is XDD. Error bars show the standard error of the mean. MB/s = millions of bytes per second.

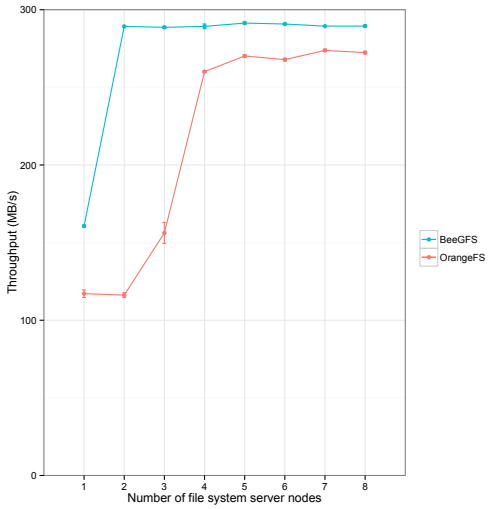


Figure 3: Comparing transfer rates between BeeGFS and OrangeFS parallel file systems. For a dataset transfer with 4 parallel TCP streams using GridFTP. TCP socket buffer sizes set to 4 MiB. Error bars show the standard error of the mean. MB/s = millions of bytes per second.

3.1. Parallel file system configuration

The first choice that had to be made with respect to parallel file transfers was the parallel file system implementation to use for storage of the dataset. Because actual transfer tests can take a long time to run a faster and more simple benchmark comparison was performed. The first test compared the BeeGFS, Ceph, GlusterFS, and OrangeFS parallel and distributed file systems when reading

and writing a single large file.

According to the benchmark results in Figure 2, BeeGFS has the fastest read throughput, and its write throughput is faster than Ceph but slower than GlusterFS and OrangeFS. Ceph has both the slowest read throughput and the slowest write throughput. The read throughput of GlusterFS is faster than Ceph but slower than BeeGFS and OrangeFS, and its write throughput is faster than BeeGFS and Ceph but slower than OrangeFS. The read throughput of OrangeFS is faster than Ceph and GlusterFS but slower than BeeGFS. OrangeFS has the fastest write throughput. For all file systems except for BeeGFS write throughput is greater than read throughput. For BeeGFS read throughput is greater than write throughput.

After examining the results of the file system benchmark it appeared that no file system has both the best read throughput and the best write throughput. BeeGFS has the best read throughput but the third-best write throughput. OrangeFS has the best write throughput but the second-best read throughput. Finding the highest-performing file system for the purpose of parallel file transfers required further testing.

In order to decide between BeeGFS and OrangeFS another experiment was conducted to compare the throughput of the two file systems during an actual transfer of the test dataset. Figure 3 shows the results of this transfer test. In this test BeeGFS has consistently higher throughput than OrangeFS from 1 to 8 server nodes. BeeGFS reaches 99% of its maximum throughput at 2 nodes while OrangeFS requires 5 nodes. Additionally, OrangeFS shows a slight drop in throughput from 1 to 2 nodes before climbing steadily at 3 and 4 nodes. Based upon the results of this test we concluded that BeeGFS was the best file system for parallel file transfers.

3.2. Network configuration

The preferred parallel file system, BeeGFS, has the ability to communicate over the network using both TCP and InfiniBand. The experimental hardware used for InfiniBand has a at least a 4x higher theoretical throughput (40 Gbps) compared to the hardware used for TCP (10 Gbps), but it was desired to see if the difference during an actual dataset transfer was large enough to justify the substantially higher complexity and cost of InfiniBand when compared to 10 Gb Ethernet.

Figure 4a shows the results of comparing the InfiniBand and TCP transports for a dataset file transfer with BeeGFS. In this test the number of Internet2 TCP streams is held constant at 4 while the number of servers changes. The results show that using TCP as a transport outperforms InfiniBand for 1 to 8 servers.

Figure 4b also compares InfiniBand and TCP transports, but in this experiment the number of servers is held constant at 5 while the number of parallel TCP streams varies. For 1 to 8 streams the throughput with TCP is higher than the throughput with InfiniBand. However, at 9 streams the throughput with InfiniBand overtakes the throughput with TCP and remains higher all the way to 64 streams. Altogether the maximum InfiniBand throughput of 880 MB/s at 16 streams is over 40% higher than the maximum TCP throughput of 601 MB/s at 13 streams. Because of the higher peak throughput with InfiniBand compared to TCP, InfiniBand was used for the remainder of the experiments.

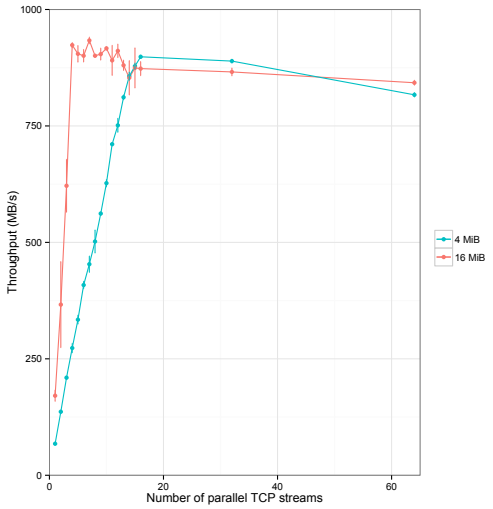


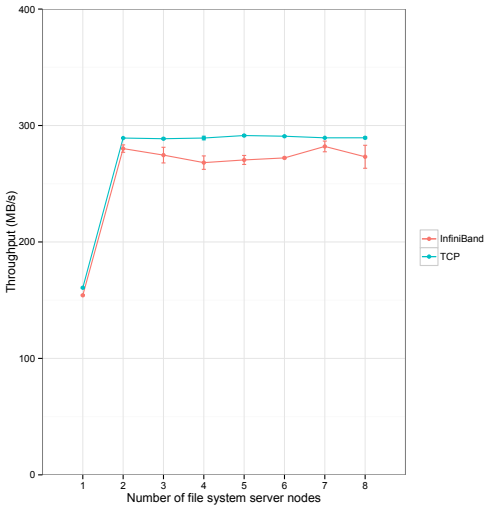
Figure 5: Comparing dataset transfer rates with 4 MiB and 16 MiB TCP socket buffer sizes. For a dataset transfer with GridFTP. Parallel file system is BeeGFS over InfiniBand on 8 server nodes. Error bars show the standard error of the mean. MB/s = millions of bytes per second. 1 MiB = 1,048,576 bytes.

The last network configurations investigated before running a full parameter sweep were the TCP send and receive socket buffer sizes. The size of these buffers influences the TCP window size which can have a large effect on the performance of long distance transfers [26]. As previous tests all used a buffer size of 4 MiB, a larger size of 16 MiB was chosen because it was believed to be much closer to the bandwidth-delay product of the network.

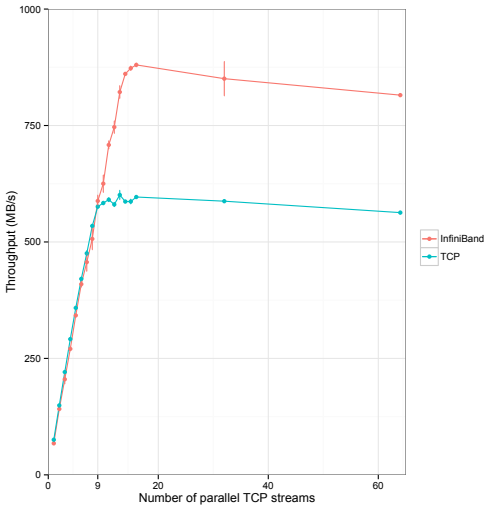
Figure 5 shows a comparison between transfers with 4 MiB and 16 MiB TCP socket buffer sizes. From 1 to 13 streams the throughput with 16 MiB buffer size is significantly larger than the throughput with 4 MiB buffer size. The maximum throughput with 4 MiB buffer size of 898 MB/s occurs at 16 streams, while the maximum throughput with 16 MiB buffer size of 934 MB/s occurs at 7 streams. Future experiments used the 16 MiB buffer size, because based on these results it is possible to reach a higher throughput with fewer streams.

3.3. Parameter sweep of streams and server nodes

Up to this point BeeGFS was chosen for the parallel file system, InfiniBand for the network transport, and 16 MiB for the TCP socket buffer size. The major parameters left to investigate are the number of parallel TCP streams



(a) Number of parallel TCP streams held constant at 4.



(b) Number of server nodes held constant at 5.

Figure 4: Comparing InfiniBand and TCP for dataset transfers. Datasets transferred with GridFTP. Parallel file system is BeeGFS. TCP socket buffer sizes set to 4 MiB. Error bars show the standard error of the mean. MB/s = millions of bytes per second. 1 MiB = 1,048,576 bytes.

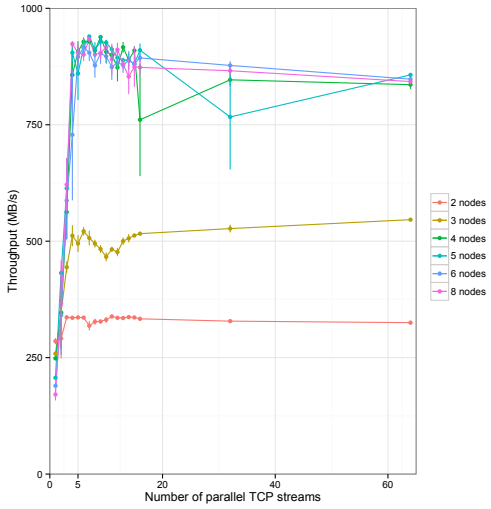


Figure 6: Dataset transfer while varying the number of TCP streams and file system server nodes. For a dataset transfer using GridFTP. Parallel file system is BeeGFS over InfiniBand. TCP socket buffer sizes set to 16 MiB. Error bars show the standard error of the mean. MB/s = millions of bytes per second.

and the number of parallel file system data server nodes. Figure 6 shows the results of sweeping these parameters.

The curves for 4–8 nodes are more or less overlapping. The only exceptions may be 4 servers, 16 streams and 5 nodes, 32 streams. These two points have a large amount of error. At 3 nodes the curve has the same general shape but the throughput is less than the curve for 4–8 nodes. The throughput at 2 nodes is less than the throughput at 3 nodes. The highest average throughput of 940 MB/s occurs at 5 nodes and 7 streams.

4. Discussion

4.1. File system comparison

The goal of the test results in Figure 2 was to predict the throughput of various parallel file systems during a dataset transfer by benchmarking their performance during a read/write benchmark. The benchmark leaves out any Internet2 network traffic and focuses only on the local performance of the parallel file system and its client. Because there is no long distance communication involved in this test we believe the benchmark approximates the upper bound of the throughput during an actual dataset transfer. We expected the throughput during an actual dataset transfer to be significantly lower than the throughput of the corresponding benchmark. Our experience has been that for a long distance data transfer the performance of the transfer as a whole is lower than the performance of the components of that transfer.

Before discussing the actual results we first explain what we perceive as limitations in the test itself. First, because the full dataset transfer test was not run for every file system, we cannot prove that lower throughput on the benchmark correlates to lower throughput in an actual file transfer. We believe the benchmark is still useful as a guide for determining the parallel file system to use for subsequent tests. Second, the benchmark test by design runs only within a local cluster with no long distance network access. We have chosen to run the benchmark test on the Apt cluster in Utah. In contrast, an actual file transfer involves both the Apt cluster and the CloudLab cluster at Clemson. As described in section 2.2, these two clusters have different hardware characteristics, especially when it comes to disk storage and main memory. We

suspect, however, that the relative performance levels of the parallel file systems would be the same on both clusters. Third, out of necessity the benchmark uses different software from the actual file transfer. However, both the benchmark tool (XDD) and the transfer tool (GridFTP) are optimized for high disk throughput. Last, and perhaps most important, the benchmark uses a single large 863 GB file whereas an actual transfer uses 345 files ranging in size from 0.6 GB to 11 GB and an aggregate size of 863 GB. Parallel file systems are known to have vastly different performance with large files compared to small files [27]. Parallel file systems also have different performance characteristics with a large number of files compared to a small number of files. However, we believe the number of files during an actual transfer to be small enough and the median file size of 2.4 GB to be large enough to avoid causing performance issues.

4.1.1. BeeGFS

BeeGFS is notable for having the highest read throughput of the file systems tested. Interestingly, it was also the only file system whose read throughput was higher than its write throughput. The higher read throughput could be a result of read-ahead by the BeeGFS server nodes.

4.1.2. Ceph

Ceph had the lowest throughput of any file system in this benchmark. The read throughput of Ceph was 207 MB/s less than the throughput of the next-slowest file system for read, GlusterFS. The write throughput of Ceph was 509 MB/s less than the next-slowest file system for write, BeeGFS. It's possible that changing the default configuration of Ceph by storing only one copy of each object caused a drop in performance. However, the fact remains that there was not enough storage to contain the default of three copies of each object. It's also possible that Ceph, as an object store, may not be well suited to the type of workload tested by our experiment.

4.1.3. GlusterFS

GlusterFS has the second-highest write throughput after OrangeFS, but its read throughput is the second-lowest after Ceph. GlusterFS also has the biggest gap between read and write throughput—the read throughput is only 36% of the write throughput.

4.1.4. OrangeFS

OrangeFS is notable for having the highest write throughput and the second-highest read throughput. The write throughput alone is impressive for reaching close to 90% of the maximum throughput on a 10 gigabit/s network. OrangeFS may be the only file system to hit a network bottleneck instead of a disk bottleneck for writes.

4.2. BeeGFS vs. OrangeFS

The goal of the BeeGFS vs. OrangeFS comparison test of Figure 3 was to determine which of the two file systems supported faster data transfers. This experiment varies the number of parallel file system server nodes, and at the time the experiment was designed we believed the number of nodes to be the most important factor in determining the throughput of the file transfer. However, later tests seem to show that the number of nodes is the least important variable.

Both BeeGFS and OrangeFS are layered on top of a local file system on each node. We chose to use the XFS file system [24] in both cases. For the XFS file system used in OrangeFS tests we used the default mount options. However, for the XFS file system used in BeeGFS tests we used the optimized mount options in Table 2 that were suggested by the BeeGFS documentation [25]. The different local file system mount options could be responsible for the relative performance of BeeGFS and OrangeFS.

A limitation of the BeeGFS vs. OrangeFS comparison test may be that the disk configuration was different for the two tests, although both file systems were configured for the best performance. BeeGFS does not make use of any RAID configuration while OrangeFS uses striped RAID. Striped RAID is currently the only option for OrangeFS because it is not possible to configure OrangeFS to use multiple data storage targets. RAID 0 striping was not used for BeeGFS in this test and all further tests because during a benchmark it was discovered to have lower performance than the configuration that used the drives independently. For unknown reasons the throughput of a transfer with OrangeFS drops from 1 to 2 nodes. This drop could be explained by an increase in network overhead when contacting more than one server.

4.3. Network configuration

After selecting BeeGFS as the parallel file system there were two major choices for the network transport. The first choice was the very popular and well-supported TCP. The second choice was InfiniBand which has a 4–5.6x higher theoretical maximum throughput but is significantly more expensive than the 10 GigE used for TCP. It was desired to determine if the higher theoretical throughput of InfiniBand translated to an actual increase in throughput during a data transfer.

Another possible protocol that could have been tested is IP over InfiniBand (IPoIB). As the name implies, IPoIB allows IP (and its associated protocols such as TCP) to be carried over InfiniBand hardware. However, the native IB protocol offers benefits such as lower latency and support for remote direct memory access. As long as the software is capable of supporting native IB protocol there is little reason to use legacy IPoIB.

The results of the test that varied the number of servers shown in Figure 4a suggest that InfiniBand in fact has lower throughput compared to TCP. But the test that varied the number of streams in Figure 4b showed that after

8 streams InfiniBand has higher throughput than TCP. It is possible that some overhead causes InfiniBand to be slower than TCP up to 8 streams. After 8 streams, the higher theoretical throughput of InfiniBand allows it to surpass TCP.

Figure 5 shows the results of increasing the socket buffer size to 16 MiB and compares it to the results with a 4 MiB socket buffer size. A larger socket buffer size allows for a larger TCP window, and a larger TCP window potentially allows for a more continuous stream of data flowing through the network. The fact that throughput increases with the socket buffer size suggests that the initial size of 4 MiB was too small for the test network.

4.4. Parameter sweep

The parameter sweep in Figure 6 explores the effects of the number of parallel TCP streams and the number of parallel file system server nodes on throughput when the socket buffer size is 16 MiB. It was expected that when the number of storage nodes decreased the throughput curve would lower as the storage system would not have enough available throughput to feed the data transfer. The curves for 4–8 nodes are similar, but as predicted at 3 nodes the performance drops sharply by about 345 MB/s at 4 streams. There is another large drop of about 176 MB/s from 3 to 2 nodes at 4 streams.

4.5. Maximizing throughput

By examining the results in Figure 5 a pattern emerges with respect to the maximum throughput, the TCP buffer size, and the number of parallel streams. In Figure 5 when the buffer size is 4 MiB the peak throughput occurs at 16 streams. Similarly, when the buffer size is 16 MiB the peak (within error) occurs at 4 streams. In other words, changing the buffer size by a factor of 4 changed the number of streams by a factor of 1/4.

Further insight can be made with additional background information. The maximum amount of data that can be "in flight" in a network is its bandwidth-delay product, or BDP. As the name suggests, the BDP is calculated by multiplying the link speed of the network by the round-trip delay time (RTT). The experimental network has a bandwidth of 10 gigabits/s and an RTT of approximately 52 ms as measured by the *ping* tool. The BDP is therefore 10 gigabits/s $\times$ 0.052 s = 0.52 gigabits. Converting the units yield a value of 62 MiB.

Therefore, in order to expect the maximum throughput from the experimental network we must ensure the amount of data available to be transmitted is at least 62 MiB. The amount of data that can be transmitted is the TCP window size, estimated by the TCP buffer size, times the number of streams. This formula matches with the experimental data: the maximum throughput occurs at 4 MiB $\times$ 16 = 64 MiB and 16 MiB $\times$ 4 = 64 MiB. This relation between BDP, buffer size, and RTT that gives maximum throughput is given in Equation 1.

$$\text{BDP} \leq \text{buffer} \times \text{streams} \tag{1}$$

In Equation 1, the BDP is usually a property of the network and should not change. An exception to this rule may be when there are multiple paths connecting nodes in the network. In that case it can be expected that each path might have a different BDP. It is far more useful to manipulate the other two parameters. The TCP socket buffer size is relatively easy to set on Linux systems. A possible constraint might be the maximum buffer size defined by the system administrator. The number of parallel TCP streams can be easy or difficult to change depending on the application. For GridFTP changing the number of TCP streams is as simple as setting a command line option, but other applications may need to be completely redesigned to incorporate thread-level parallelism. Additionally, the number of practical parallel TCP streams is likely bounded by the number of physical CPU cores.

A final constraint that occurs during a real data transfer is that the storage system must be able to keep up with the network throughput. A network that is capable of 100 gigabits/s is not going to be fully utilized when it is connected to a storage system capable of only 1 gigabit/s. The 2- and 3-node results of Figure 6 show what happens to throughput when there is not enough storage bandwidth available. For a large dataset (such as those seen in genomics) the storage system must be able to sustain a high throughput. This fact is one of the reasons our transfers were run "at scale" with real data—using small test files would have not so much tested the full storage throughput as the throughput of the cache.

5. Conclusion

A wide variety of tests we performed narrowed the optimal data transfer parameters for parallel data streaming across AL2S and between two CloudLab clusters loading real genomics data onto a parallel file system. The best throughput was found to occur with GridFTP using at least 5 parallel TCP streams with a 16 MiB TCP socket buffer size to transfer to/from 4–8 BeeGFS parallel file system nodes connected by InfiniBand. An attempt at generalizing the results was made in Equation 1, where the TCP buffer size and number of parallel streams were related to the bandwidth-delay product of the network.

There are a few experimental thrusts available for further optimization. First, it is possible that several of the network settings outlined in Table 1 contributed negatively to the throughput of the transfer tests. In particular we are concerned about the TCP timestamps, low-latency, and selective acknowledgement options because their settings appear contrary to common sense for a fast network with a high RTT. Although the settings were based on the ESnet recommendations, the exact values were copied from an HPC cluster that we later discovered was tuned for local low-latency message passing type communication.

Second, an important limitation to Equation 1 may be that it is appropriate only for networks with very little loss. In the case of loss, it is likely that more streams should be

favored over a larger buffer size, as a greater number of parallel streams will quicken the recovery of the transfer in the face of loss.

Third, our experiments are designed such that all data flow through the clients which act as data transfer nodes (DTNs). This configuration is common in Science DMZs, but in CloudLab there is no technical requirement since all nodes are capable of communicating with each other. Using n times as many data transfer nodes has the potential of unlocking n times the performance of the single-node case as long as the parallel file system is capable of supplying enough storage bandwidth. Fortunately, parallel file systems are in fact designed for just such a situation.

5.1. Future work

It is important to note that dataset transfers are only a means to an end and that the actual objective is to run the genomic analysis workflow. As such, future work will focus not only on transfers but also on analysis, possibly both at the same time.

The effects of packet loss on the transfer throughput need to be more closely examined. Future work will use the Linux socket statistics tool, *ss*, to record loss events such as TCP retransmissions. The loss will be controlled with Linux traffic shaping. We will then attempt to fit this loss into our model.

6. Acknowledgements

This work was supported by the National Science Foundation [grant number 1447771].

References

[1] NCBI. Sequence read archive [online] (Aug. 2016). URL: http://www.ncbi.nlm.nih.gov/Traces/sra/.

[2] R. N. The Cancer Genome Atlas, J. N. Weinstein, E. A. Collisson, G. B. Mills, K. R. M. Shaw, B. A. Ozenberger, K. Ellrott, I. Shmulevich, C. Sander, J. M. Stuart, The cancer genome atlas pan-cancer analysis project, Nature genetics 45 (10) (2013) 1113–1120. URL: http://dx.doi.org/10.1038/ng.2764.
URL http://dx.doi.org/10.1038/ng.2764

[3] The 1000 Genomes Project Consortium, A global reference for human genetic variation, Nature 526 (7571) (2015) 68–74. URL: http://dx.doi.org/10.1038/nature15393.
URL http://dx.doi.org/10.1038/nature15393

[4] L. Zhi-Kang, G.-Y. Zhang, K. L. McNally, W.-S. Wang, J. Li, N. A. Alexandrov, J. A. et al., The 3,000 rice genomes project, GigaScience 3 (1) (2014) 1–6, iD: ref1. URL: http://dx.doi.org/10.1186/2047-217X-3-7.
URL http://dx.doi.org/10.1186/2047-217X-3-7

[5] F. A. Feltus, J. R. B. III, J. Deng, R. S. Izard, C. A. Konger, W. B. L. III, D. Preuss, K.-C. Wang, The widening gulf between genomics data generation and consumption: A practical guide to big data transfer technology, Bioinformatics and biology insights 9 (Suppl 1) (2015) 9.

[6] J. Heichler. An introduction to beegfs [online] (Nov. 2014). URL: http://www.beegfs.com/docs/Introduction_to_BeeGFS_by_ThinkParQ.pdf.

[7] S. A. Weil, Ceph: reliable, scalable, and high-performance distributed storage, Ph.D. thesis, University of California Santa Cruz (2007).

[8] A. Davies, A. Orsaria, Scale out with glusterfs, Linux Journal 2013 (235).

[9] P. Carns, W. B. L. III, R. B. Ross, R. Thakur, Pvfs: A parallel file system for linux clusters, in: Proceedings of the 4th annual Linux Showcase and Conference, 2000, pp. 391–430.

[10] Lustre software release 2.x [online]. URL: http://doc.lustre.org/lustre_manual.xhtml#installinglustre.

[11] E. Dart, B. Tierney, E. Pouyoul, J. Breen, Achieving the science dmz, presentation, Joint Techs, Baton Rouge, LA, Jan 2012 (Jan. 2012). URL: http://www.internet2.edu/presentations/jt2012winter/ScienceDMZ-Tutorial-Jan2012-1.pdf.
URL http://www.internet2.edu/presentations/jt2012winter/ScienceDMZ-Tutorial-Jan2012-1.pdf

[12] R. Ricci, E. Eide, The CloudLab Team, Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications, USENIX ;login: 39 (6). URL: https://www.usenix.org/publications/login/dec14/ricci.
URL https://www.usenix.org/publications/login/dec14/ricci

[13] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, A. Joglekar, An integrated experimental environment for distributed systems and networks, in: Proceedings of the Fifth Symposium on Operating Systems Design and Implementation, USENIX Association, Boston, MA, 2002, pp. 255–270.

[14] Cloudlab [online] (Sep. 2016). URL: http://cloudlab.us/.

[15] Cloudlab overview [online]. URL: http://cloudlab.us/files/cloudlab-overview.pdf.

[16] Cloudlab hardware [online] (Sep. 2016). URL: http://docs.cloudlab.us/hardware.html.

[17] Internet 2 layer 2 services [online] (Sep. 2016). URL: http://www.internet2.edu/products-services/advanced-networking/layer-2-services/.

[18] Globalnoc [online]. URL: http://globalnoc.iu.edu/i2network/index.html.

[19] B. W. Settlemyer, J. D. Dobson, S. W. Hodson, J. A. Kuehn, S. W. Poole, T. M. Ruwart, A technique for moving large data sets over high-performance long distance networks, in: 2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST), 2011, pp. 1–6. doi:10.1109/MSST.2011.5937236.

[20] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, The globus striped gridftp framework and server, in: Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference, 2005, pp. 54–54. doi:10.1109/SC.2005.72.

[21] N. Mills. Cloudlab transfer experiment rspec [online] (Sep. 2016). URL: https://www.cloudlab.us//p/CloudLab/nlmpfs8xfer.

[22] Xdd [online]. URL: https://github.com/nlmills/xdd/commit/cdfb9a5a1ba14ca398f621508f7386c9d756322e.

[23] ESnet. Linux tuning [online] (Aug. 2016). URL: http://fasterdata.es.net/host-tuning/linux/.

[24] C. Hellwig, Xfs: the big storage file system for linux, ; login:: the magazine of USENIX & SAGE 34 (5) (2009) 10–18.

[25] Beegfs tuning and advanced configuration [online] (Sep. 2016). URL: http://www.beegfs.com/wiki/TuningAdvancedConfiguration.

[26] N. S. V. Rao, D. Towsley, G. Vardoyan, B. W. Settlemyer, I. T. Foster, R. Kettimuthu, Sustained wide-area tcp memory transfers over dedicated connections, in: High Performance Computing and Communications (HPCC), 2015, pp. 1603–1606. doi:10.1109/HPCC-CSS-ICESS.2015.86.

[27] P. Carns, S. Lang, R. Ross, M. Vilayannur, J. Kunkel, T. Ludwig, Small-file access in parallel file systems, in: Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on, 2009, pp. 1–11. doi:10.1109/IPDPS.2009.5161029.