FTProfiler: A New Profiling Tool for GridFTP Servers

Huong Luu[1], Rajkumar Kettimuthu[2,3], Marianne Winslett[1]
[1] Department of Computer Science, University of Illinois at Urbana-Champaign, USA
[2]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, USA
[3]Computation Institute, University of Chicago/Argonne National Laboratory, Chicago, USA

***Abstract*— GridFTP is a high-performance, secure, and reliable data transfer protocol that is being widely used in data transmission in Grid computing. A GridFTP server needs to achieve high throughput as it sends and receives data to and from multiple sources, each with its own configuration. Profiling tools can potentially help GridFTP administrators gain insight into the system's activities and identify configuration tradeoffs as well as potential bottlenecks. This paper presents a new profiling tool, called FTProfiler, which is built upon standard system profiling tools OProfile and Sar, to provide a more complete view of the system and simplify the profiling process for new GridFTP servers. FTProfiler calls OProfile and Sar, analyzes their profiling results and generates a detailed report. Through a case study, we show how FTProfiler can help administrators understand the effects of system parameters such as the TCP buffer size, block size, and parallel TCP streams on server performance and load, thus simplifying the process of detecting bottlenecks and tuning for performance.**

Keywords – Profiling GridFTP, Profiling WAN data movement, Profiling high-speed transfers

I. INTRODUCTION

GridFTP [1] extends the standard FTP [2] protocol to provide a high-performance, secure, reliable data transfer protocol optimized for high-bandwidth wide-area networks. The Globus GridFTP implementation [3] has been widely used for data transfer in the Grid community. It provides a modular and extensible data transfer system architecture suitable for wide area and high-performance environments. To get the maximum performance from the GridFTP server, a variety of parameters need to be tuned. Profiling tools can potentially help GridFTP administrators gain insight into the system's activities, identify configuration tradeoffs, and understand their impact on server load.

Currently two good profiling tools, *OProfile* [21] and *Sar* [23], can provide information on different aspects of the system behavior without imposing much overhead. In this paper we present FTProfiler, which leverages *OProfile* and *Sar* to provide a more complete view of the GridFTP system and simplify the profiling process on new servers. Through our case studies, we demonstrate the use of FTProfiler to profile system, to identify the potential bottleneck and to understand the effects of system parameters on server behavior, including TCP buffer size, I/O block-size, and the number of parallel streams or otherwise called parallelism.

In the remainder of the paper, we provide background on GridFTP in Section 2, discuss the design of FTProfiler in Section 3 and present the profiling case studies in Section 4. We describe related work in Section 5 and summarize in Section 6.

II. GRIDFTP

FTP is a widely implemented and well-understood IETF-standard protocol. It provides a well-defined architecture for protocol extensions and supports dynamic discovery of the extensions supported by a particular implementation. Many extensions for FTP have been defined through the IETF. The FTP protocol also separates control and data channels, enabling third-party transfers, that is, the transfer of data between two end hosts, mediated by a third host. The GridFTP protocol is based on FTP and thus also benefits from the FTP protocol advantages mentioned above. In addition, it extends the FTP protocol to provide a high-performance, secure, and reliable data transfer protocol optimized for high-bandwidth wide-area networks.

The following is a summary of key GridFTP features.

Third-party control of data transfer. To manage large datasets for distributed communities, we must provide authenticated third-party control of data transfers between storage servers. A third-party operation allows a user or application at one site to initiate, monitor and control a data transfer operation between two other sites: the source and destination for the data transfer.

Authentication, data integrity, and data confidentiality. GridFTP supports Generic Security Services (GSS)-API authentication of the control channel (RFC 2228) and data channel (GridFTP extensions), and supports user-controlled levels of data integrity and/or confidentiality. Data channel authentication is of particular importance in third party transfers, since the IP address of the host connecting for the data channel will be different than that of the host connected on the control channel, and there must be some way to verify that it is the intended party.

Striped data transfer. Data may be striped or interleaved across multiple servers, as in a parallel file system or DPSS disk cache [19]. Thus, GridFTP defines protocol extensions that support the transfer of data partitioned among multiple servers.

Parallel data transfer. On wide-area links, using multiple TCP streams in parallel between a single source

and destination can improve aggregate bandwidth relative to that achieved by a single stream [17, 20]. GridFTP supports such parallelism via FTP command extensions and data channel extensions. A GridFTP implementation can use long virtual round trip times to achieve fairness when using parallelism or striping [18]. Note that striping and parallelism may be used in tandem, i.e., users may have multiple TCP streams open between each of the multiple servers participating in a striped transfer.

Partial file transfer. Some applications can benefit from transferring portions of files rather than complete files, such as analyses that require access to subsets of massive files. FTP allows transfer of the remainder of a file starting at a specified offset. GridFTP supports requests for arbitrary file regions.

Automatic negotiation of TCP buffer/window sizes. Using optimal settings for TCP buffer/window sizes can dramatically improve data transfer performance. However, manually setting TCP buffer/window sizes is an error-prone process, particularly for non-experts, and is often simply not done. GridFTP extends the FTP command set and data channel protocol to support both manual setting and automatic negotiation of TCP buffer sizes for large files and for large sets of small files.

Support for reliable and restartable data transfer. Reliable transfer is important for many applications that manage data. Fault recovery methods are needed to handle failures such as transient network and server outages. The FTP standard includes basic features for restarting failed transfers, but these are not widely implemented. GridFTP exploits these features and extends them to cover its new data channel protocol.

The Globus implementation of GridFTP provides all these key features along and is highly extensible. Its modular architecture enables a standard GridFTP-compliant client to access any storage system that implements its data storage interface [5], including the HPSS archival storage system [6], SRB [7], the PVFS parallel file system [8], the GPFS parallel file system [9], and POSIX [10] file systems. Its eXtensible I/O interface [11] allows GridFTP to target high-performance wide-area communication protocols such as UDT [12], FAST TCP [13], and RBUDP [14]. Globus GridFTP is optimized to handle a variety of types of datasets, from a single, huge file to datasets comprising lots of small files [15, 16].

III. FTPROFILER OVERVIEW

In general, GridFTP server performance profiling needs to be done in a way that minimizes interference with the production jobs running on the server. Thus the steps of preparation and generating reports are usually done on a separate machine, rather than the server itself. For this reason, FTProfiler runs on the client machine where the transfer is initiated for a client-server transfer or a third party transfer between servers. During execution, FTProfiler first remotely accesses the server to start the underlying tools, including *OProfile* and *Sar*, then runs the test workload on the client machine. After finishing the run, FTProfiler again accesses the server to turn off the running profiling tools and post-process the results, which are then sent back to the client machine. Based on the results received, FTProfiler generates a report that covers the essential information. The user has the option of viewing additional information from the profiling result files. The working scenario of FTProfiler is shown in Figure 1.

FTProfiler uses *OProfile* and *Sar* as underlying tools because they provide different aspects of system profiling with low overhead. *OProfile* uses the hardware performance counters of the CPU to profile the entire system. *OProfile* shows how time is spent in different parts of the system, such as the kernel, kernel modules, interrupt handlers, shared libraries and applications. On the other hand, *Sar* is capable of profiling the memory usage, including memory and swap space utilization and other performance metrics for each processor. FTProfiler processes the results from both tools to provide essential information to the user, such as CPU utilization, peak and average load.

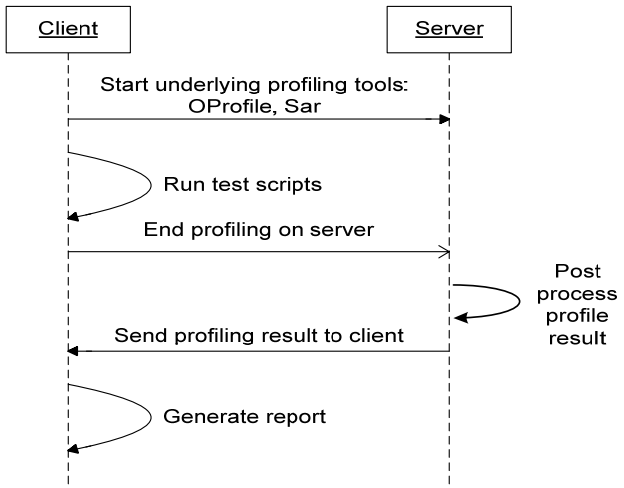

Figure 1: Working scenario for FTProfiler

IV. CASE STUDY

The goal of our experiments was to perform a performance study to see the effects of different parameters to GridFTP performance. In this process, we demonstrate the use of FTProfiler and other tools to detect the bottleneck of the transfers. To this end, we used five servers in multiple locations, to model both local and distant data transfers. For the local case, the client and server are nodes in the Breadboard cluster at Argonne National Laboratory, each with 4GB of main memory,

approximately 300GB of storage per node, connected by a Myri-10G 10Gbps network with a 1Gb/s network interface card (NIC) on each machine.

For the distant case, we used dedicated GridFTP servers at different TeraGrid sites, namely the servers Pople at the Pittsburgh Supercomputing Center, Abe at the National Center for Supercomputing Applications, and Ranger at the Texas Advanced Computing Center. These servers can utilize the TeraGrid backbone network (10 – 30 Gb/sec) to the fullest for a better transfer rate. However, Pople uses 1 Gb/s NIC while Abe and Ranger have 10 Gb/s NICs.

We had the privileges to adjust server configurations on Breadboard nodes, but not elsewhere. Our experiments tune the most common options for performance: TCP buffer size, parallel TCP streams, and buffer size. We transfer 16GB of data with files of size 800MB, 1GB, and 4GB.

A. *TCP buffer size*

The TCP buffer size option (*-tcp-bs*) specifies the size of the TCP buffer to be used by the underlying FTP data channels. This parameter has long been believed to be critical to achieve good performance over the WAN. The optimal value for the TCP buffer size is believed to be the Bandwidth * Delay Product (BDP), which depends on the available Bandwidth and Round Trip Time (RTT) between two destinations. However, in most modern operating systems, TCP buffers are automatically tuned which means the buffer size is automatically adjusted based on the changing network conditions. This raises the question: will manually changing the TCP buffer size hurt or help, in the presence of auto-tuning? We investigated this question through a performance study of the effect of TCP buffer size parameter on GridFTP transfers.

For the case where the RTT between the client and the server is on the order of few milliseconds or less, the TCP buffer size does not affect the transfer rate much, hence it will not be the source of a bottleneck. Thus we tested the effect of this parameter in the long distance case. We transferred data between TeraGrid sites and the server on the Breadboard cluster. We tested with different TCP buffer sizes: the default option of GridFTP (which uses autotuning), a manually calculated BDP value and fixed values of 4MB, 8MB, 16MB and 32MB. The 4MB, 8MB, 16MB and 32MB values are recommended for the optimal transfer rate among TeraGrid servers, and are actually used by TeraGrid for this purpose. In all tests we have conducted, parameters other than the one being tested are set to the default values.

To calculate BDP, we used Iperf [24] to measure bandwidth and ping to determine the RTT between sites. To assess GridFTP servers' performance, we measured the transfer rate of each configuration multiple times, eliminated any outliers, then calculated the average value. The transfers were performed at different times of day to take into account the effects of loads created by other users in a shared environment like TeraGrid. Based on our measurements, the BDP values are approximately 500KB (Abe and BB), 2MB (Pople and Abe, Pople and BB), 3MB (Ranger and Abe, Ranger and Pople) and 4.5MB (Ranger and Pople).

One disadvantage of using BDP rather than other options is that BDP needs to be calculated for every connection that we want to improve, while auto-tuning and fixed buffer sizes do not require that. Manually choosing the TCP buffer size might worsen the performance because too-large buffers possibly overload the receiver's TCP window and create congestion on the server. This is a serious problem called the "buffer bloat" problem [28, 29], discussed by Jim Gettys, in which excessively large buffers in the network communication system eventually lead to network congestion, destroy congestion avoidance in the transport protocols and cause poor performance.

On the other hand, sometimes the TCP settings are too small, thus limiting the server's performance and under-utilizing the network. Hence manual tuning is generally not recommended over auto-tuning.

As shown in Figure 2, with auto-tuning, the choice of TCP buffer size is not as important as it was in the past. Among Teragrid servers, the difference between the best and the worst transfer rates is about 10% except for the Ranger to Pople case, where the default auto-tuning gives a 46% improvement compared to the worst buffer size, 16MB. The Breadboard server worked best with the auto-tuning option and the performance quickly dropped when TCP buffer size increased which might indicate the bottleneck in TCP settings. Figure 2 also shows that the transfer rates in different directions for the Ranger server are not the same; in fact, the rate from Abe to Ranger is double the reverse direction. This could mean that Ranger has different network configurations for each direction. Ranger seems to prioritize its configuration for incoming traffic over outgoing.

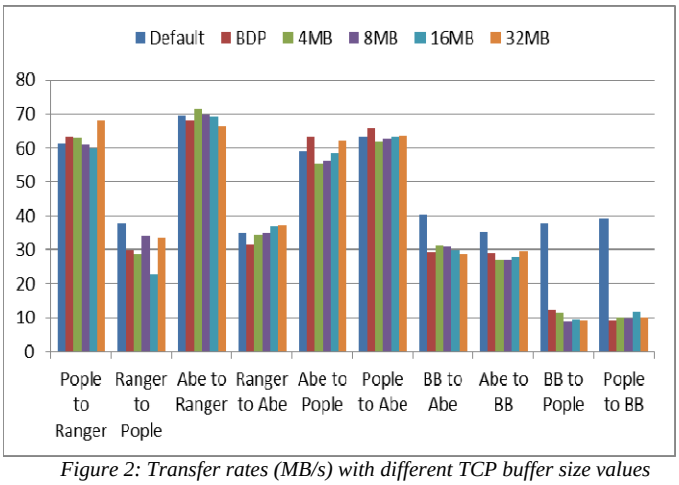

Figure 2: Transfer rates (MB/s) with different TCP buffer size values

	With old TCP settings				With new TCP settings			
	Peak load	Transfers to I/O devices per second	Network packets received per second	Network packets transmitted per second	Peak load	Transfers to I/O devices per second	Network packets received per second	Network packets transmitted per second
Default	23.7%	4.85	11675	15529	34.9%	9.43	45372	61977
BDP	20.6%	2.11	5734	7664	20.4%	10.78	38332	52539
4MB	20.5%	2.87	7271	9673	28.2%	11.63	38725	53191
8MB	20.6%	3.40	8352	11083	34.5%	12.00	39021	53618
16MB	21.4%	3.99	9524	12635	34.5%	10.51	33798	46433
32MB	20.1%	4.49	10527	13947	34.0%	10.92	34483	47385

Table 1: GridFTP server profiling result using different TCP settings

To explain the low performance of transfers between Breadboard and Abe/Pople in Figure 2, we used FTProfiler to gain more insight into this problem. The profiling result for the Breadboard server, shown in the left half of Table 1, suggested that the current TCP settings under-utilize the server with a very low transfer rate to the physical I/O device (server) as well as the network transfer rate. This indicates that network setting could potentially be the bottleneck in this case.

To understand the impact of TCP settings, we examined the settings on each server; the main differences between servers' settings are listed in Table 2. As we can see, the TCP settings in Breadboard are quite small and hence, the transfer rate from/to Breadboard server is low because of this limitation. It also explains why Breadboard server performance decreased as we increased the TCP buffer size, as shown in Figure 2.

We adjusted the settings of the Breadboard server to improve its performance, using the sysctl command [25]. We changed the Breadboard server auto-tuning settings to be the values suggested in TCP tuning manuals [22]. The suggested values are very close to Abe's except that netdev_max_backlog becomes 30,000 instead of 250,000. After changing the TCP settings, the transfer rates using different buffer sizes all improved greatly, as much as five times faster, as shown in Figure 3. Further, once the TCP settings were set appropriately, the effect of the buffer size is small. The Breadboard profiling result with new settings, shown in the right half of Table 1, also confirmed this.

In conclusion, auto-tuning does a good job in achieving good performance without having to manually select a value for the TCP buffer size. However, we might need to adjust other TCP settings to make sure that we can fully take advantage of auto-tuning.

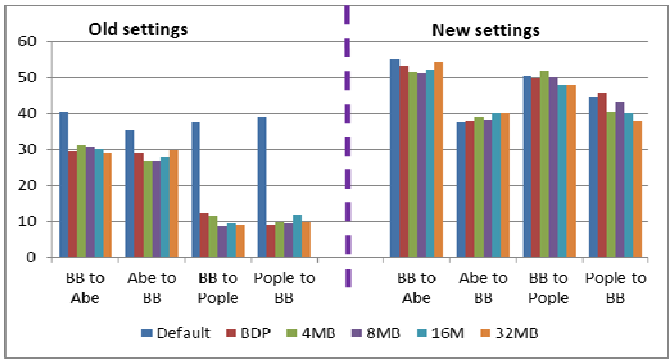

Figure 3: Transfer rate (MB/s) for different TCP buffer sizes, after adjusting Breadboard's other TCP settings

B. *Parallelism*

This parameter specifies the number of TCP streams running in parallel to be used in the transfer. It is one of the most commonly tuned parameters to achieve good performance because it improves the aggregate bandwidth and makes better use of the available bandwidth. The default value for parallelism is 1. In these tests, we auto-tune the TCP buffer size and use the new settings for Breadboard's other TCP parameters, as described in the previous section. Other parameters are set to default.

The experiment results are presented in Figure 4. In general, using parallel streams improves the transfer rate quite significantly, especially when the RTT between servers is large. For Abe and Breadboard transfers whose RTT is only 6.5 ms, increasing the number of parallel streams does not help. It is interesting to observe that even though Ranger does not have a good outgoing rate using the TCP buffer size option, we can actually improve it significantly with the parallelism option. And the transfer from Abe to Pople requires further investigation, but our

Variable: meaning	Abe	Pople	Ranger	Breadboard
net.core.rmem_max/wmem_max: Set max size of TCP receive/transmit window.	16MB	2MB	32MB	128KB
net.core.rmem_default/wmem_default: Set default size of TCP receive/transmit window.	112KB	256KB	64KB	122KB
net.ipv4.tcp_rmem: Set min, default, max receive window.	4KB/ 85KB/ 16MB	4KB/ 85KB/ 16MB	4KB/ 85KB/ 32MB	4KB/ 85KB/ 4MB
net.ipv4.tcp_wmem: Set min, default, max transmit window.	4KB/ 64KB/ 16MB	4KB/ 16KB/ 256KB	4KB/ 12MB/ 32MB	4KB/ 16KB/ 4MB
net.ipv4.tcp_mem: Set min, default, max allocatable TCP buffer space.	1.5MB/ 2MB/ 3MB	48KB/ 64KB/ 96KB	768KB/ 1MB/ 32MB	367KB/489KB/734KB
net.core.netdev_max_backlog: Maximum number of packets in the receiver's queue.	250,000	2500	400,000	1000

Table 2: Servers' TCP settings

privileges only allow in-depth investigation on Breadboard.

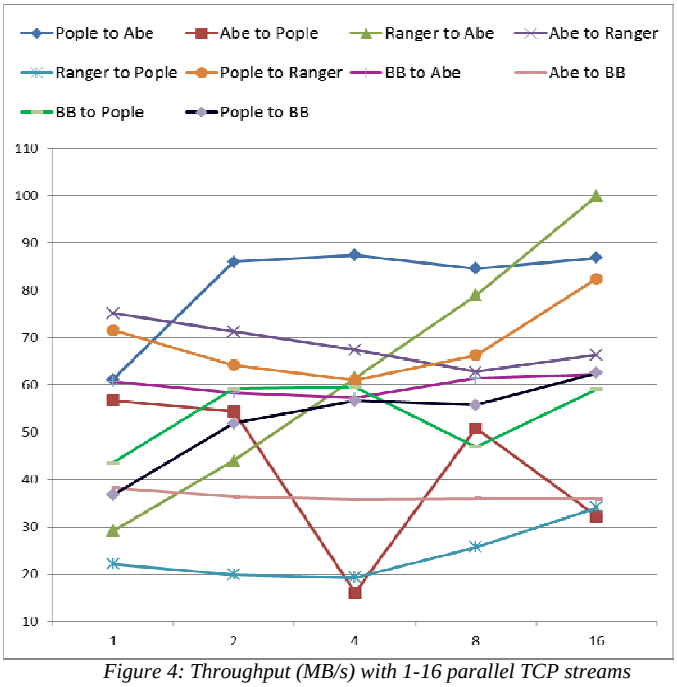

Figure 4: Throughput (MB/s) with 1-16 parallel TCP streams

For transfers from Breadboard to Pople, increasing parallelism (P) significantly helped to improve the transfer rate for small P, but this trend reversed for P=8. By looking at the profiling result shown in Table 3, we see that P=4 improves total throughput without increasing kernel load. So for this transfer, we could recommend to use P=4 to maximize the server performance.

	P =1	P = 2	P = 4	P = 8	P = 16
Transfer rate (MB/s)	45.86	58.36	59.06	47.01	58.33
Peak CPU utilization (%)	22.2	39.1	41.7	38.7	36.4
Kernel load (%)	45.22	48.60	46.73	53.94	59.26

Table 3: Server profiling for Breadboard to Pople transfer case

C. *Block size:*

This parameter specifies the size of the buffer that the underlying IO system uses when posting read requests to the disk. This parameter gives the user more control, as each underlying IO system that GridFTP uses has its own optimal IO buffer size value. As this parameter is only applicable to read requests, it affects only the data sender. We must distinguish between two cases – sending from the client or server – to make sure we adjust the right value.

When the client sends data to the server, we specify the block size value using the *–bs* option in *globus-url-copy* command. In the third party transfer case, in which a client initiates the transfer between two servers or when the server sends data to the client, we adjust the server's configuration by adding the block size option when starting the server. The default value is 256KB. Other parameters are set to default.

As shown in Figure 5, the experiments' result shows that the adjustment does not significantly improve the performance. In fact the default value for block size (256KB) performs slightly better overall than other values.

Intuitively, as long as the block size is big enough to keep the rest of the system fed with data, it will not be the bottleneck. We, therefore suspect that it might become the bottleneck if there is more load, for example with 16 parallel streams. However, the impact of increasing the block size is still small, for 16-way parallel transfers from a Breadboard client to the Abe server.

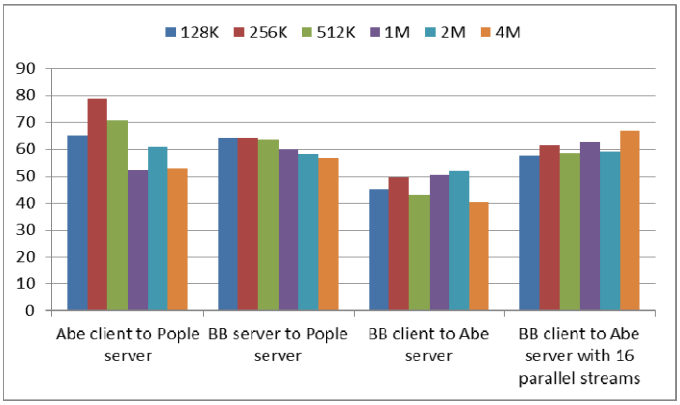

Figure 5: Effects of adjusting block size

D. *Putting everything together: How well do we do?*

In general, there are two types of GridFTP transfer: one that involves disk activity and another one that does not, which is memory-to-memory transfer. The latter type is usually used to identify if the performance bottleneck lies in the network configuration or in disk I/O. In this section, we first perform memory-to-memory transfers between servers and then by comparing the memory-to-memory transfer rate to the best disk-to-disk transfer rate, we find out how efficiently we have utilized the network for disk related transfer.

	Abe	**Breadboard**	**Pople**	**Ranger**
Abe		110.9	113.9	552.95
Breadboard	111		105.98	111.33
Pople	115.94	84.3		115.32
Ranger	434.47	NA	111.78	

Table 4: Memory to memory transfers between servers, in MB/s

As shown in Table 4, memory-to-memory transfers from or to Breadboard and Pople were able to saturate the network and reach close to the bandwidth limit (bounded by 1 Gb/s NIC (approximately 125 MB/s)). Abe and Ranger servers have 10 Gb/s NIC. That's why the transfer rates between Abe and Ranger are 550MB/s and 434MB/s. Obviously, we were not able to saturate the network link in this case. The throughput is only 35-45% of the available capacity. It could be due to network bottlenecks or because the end systems are not powerful enough to drive a 10Gb/s network link. The transfer rate from Pople to Breadboard

was about two thirds of the NIC limit, which might indicate bottlenecks in network configuration.

Next, we compared the best achievable disk-to-disk transfer rate (using all performance-improving parameter settings discussed so far) to the memory-to-memory transfer rate, to identify the disk I/O overhead. Our preliminary results in Table 5 indicated that we were not using the network efficiently. FTProfiler can help us gain insight into where the bottleneck might be and improve the performance optimization.

	Best disk to disk transfer rate (MB/s)	Memory to memory transfer rate	Percentage utilization
Abe to BB	40.32	110.90	36%
Abe to Pople	56.76	113.90	50%
Abe to Ranger	75.13	552.95	14%
BB to Abe	62.25	111.00	56%
BB to Pople	59.53	105.98	56%
Pople to Abe	86.82	115.94	75%
Pople to BB	45.41	84.30	54%
Pople to Ranger	82.41	115.32	71%
Ranger to Abe	99.99	434.47	23%
Ranger to Pople	37.84	111.78	34%

Table 5: Percentage of network utilization for best disk to disk transfers

We first measured the GridFTP transfer rate from Breadboard clients to the Breadboard server. We measured the local disk-to-disk transfer because on the Breadboard system we have exclusive access to the nodes used in the test, so that the transfer rate is not affected by the load created by other clients. The memory to memory transfer rate was 112MB/s. However, the disk-to-disk transfer rate between Breadboard nodes is only about 60MB/s. We also measured the memory to disk and disk to memory transfer rates, which were 50MB/s and 80MB/s, respectively.

We expected the transfer rate to be close to the rate measured by a standard disk benchmark such as FIO [26] or Bonnie [27]. We used Bonnie to measure Breadboard nodes' disk performance, focusing on block sequential IO. However, the rate reported by Bonnie is 100MB/s and 91MB/s for output and input respectively, which is much higher than the GridFTP disk IO performance.

To understand where the bottleneck might be, we profiled both source and destination server for all transfer cases between memory and disk. The results are shown in Table 6. In this table, /mbcache refers to the filesystem meta information block cache. */nfs* is the file system module, */sunrpc* is the protocol for making remote procedure calls. */tcp_cubic* module provides the cubic congestion control protocol and */tg3* module is the Ethernet NIC driver.

We discovered several interesting patterns. First, /nfs is heavier on the destination server with disk I/O because it has to allocate disk blocks to write data to. /mbcache is disk I/O related only; /sunrpc runs mainly on the source server with disk-related transfer while /tcp_cubic only appears on the source server, and /tg3 is needed in all cases. This observation suggests that we should focus on improving these related modules for disk transfer cases.

	/mbcache	/nfs	/sunrpc	/tcp_ cubic	/tg3
Mem-to-mem: Log on source	0	0	0	0.148	2.403
Mem-to-mem: Log on destination	0	0	0	0	3.389
Mem-to-disk: Log on source	0	0	0	0.087	2.008
Mem-to-disk: Log on destination	0.004	**2.423**	0.364	0	2.615
Disk-to-mem: Log on source	0.005	0.428	**1.048**	0.014	2.341
Disk-to-mem: Log on destination	0	0	0	0	2.943
Disk-to-disk: Log on source	0.006	0.412	**1.209**	0.038	2.703
Disk-to-disk: Log on destination	0.005	**2.395**	0.240	0	2.387

Table 6: Percentage of load on server for important modules

V. Related Work

The work we present in this paper is similar to that of George Kola et al. [4], who performed a full system profiling and comparison between GridFTP and NeST servers, the very commonly used data servers at that time. Based on the profiling result, they discussed the configuration tradeoffs of parallel streams and the block-size parameter for server performance and server load. Their experiments were carried out with GridFTP 2.4.3. Since then, Globus Alliance has released several other versions of GridFTP with many improvements and new features. Our experiments are performed with Globus Toolkit 5.0.0 and GridFTP server 3.19, and thus provide a more current view of GridFTP.

VI. Summary and Future Work

We have developed FTProfiler, a profiling tool for GridFTP. This tool was motivated by the need to understand the effects of various system parameters in performance tuning and detecting bottlenecks. We have demonstrated the use of this tool using a variety of GridFTP transfers both in the LAN and WAN settings. We have shown the effect of tuning parameters such as TCP buffer size, parallel streams and block size on the performance of GridFTP.

Our performance study has shown that the auto-tuning feature in modern operating systems is doing a good job in adjusting the TCP buffer size automatically based on the changing network conditions. However, to be effective, auto-tuning requires good settings for other TCP parameters. After changing the other TCP settings of the Breadboard server, data transfer rates with auto-tuning were as much as five times faster. We also found that using

parallel streams helps to improve the transfer rate while block size does not show a clear effect on the performance.

Further, we have compared the performance of disk-to-disk transfers with memory-to-memory transfers and identified some bottlenecks in the GridFTP system, using the detailed analysis provided by the FTProfiler tool.

In future, we plan to extend our study to profile striped servers and analyze the impact of small file optimizations such as pipelining, parallel transfers, and on-the-fly tarring of files. We also intend to do a detailed study of the pros and cons of using TCP versus UDT in WAN settings, using FTProfiler.

This work was supported in part by NSF grant CCF 0938064 and the Google Summer of Code program.

REFERENCES

1. Allcock, W. GridFTP: Protocol Extensions to FTP for the Grid. Global Grid Forum GRD-R-R.020, 2003.
2. Postel, J. and Reynolds, J. File Transfer Protocol. Internet Engineering Task Force, RFC 959, 1985.
3. W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, “The Globus Striped GridFTP Framework and Server,” SC’05, ACM Press, 2005
4. Kola, G., Kosar, T., Livny, M., "Profiling Grid Data Transfer Protocols and Servers", In Proceedings of 10th European Conference on Parallel Processing (Europar 2004)
5. Kettimuthu, R., Link, M., Bresnahan, J., Allcock, W., “Globus Data Storage Interface (DSI) - Enabling Easy Access to Grid Datasets,” 1st DIALOGUE Workshop: Applications-Driven Issues in Data Grids, Aug. 2005.
6. Watson, R.W. and Coyne, R.A. The Parallel I/O Architecture of the High-Performance Storage System (HPSS). IEEE MSS Symposium, 1995.
7. Baru, C., Moore, R., Rajasekar, A. and Wan, M., The SDSC Storage Resource Broker. 8th Annual IBM Centers for Advanced Studies Conference, Toronto, Canada, 1998.
8. Carns, H., Ligon III, W.B., Ross, R.B., and Thakur, R., "PVFS: A Parallel File System For Linux Clusters", Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, GA, October 2000
9. General Parallel File System (GPFS), 2004. www-1.ibm.com/servers/eserver/clusters/software/gpfs.html.
10. POSIX 1003.1e draft specification "http://www.suse.de/~agruen/acl/posix/posix_1003.1e-990310.pdf"
11. Allcock, W., Bresnahan, J., Kettimuthu, R. and Link, J., The Globus eXtensible Input/Output System (XIO): A Protocol-Independent I/O System for the Grid. Joint Workshop on High-Performance Grid Computing and High-Level Parallel Programming Models held in conjunction with International Parallel and Distributed Processing Symposium, 2005.
12. Gu, Y. and Grossman, R.L., UDT: An Application Level Transport Protocol for Grid Computing. Second International Workshop on Protocols for Fast Long-Distance Networks, 2003.
13. Jin, C., Wei, D.X. and Low, S.H., FAST TCP: motivation, architecture, algorithms, performance. IEEE Infocom, 2004.
14. He, E., Leigh, J., Yu, O. and DeFanti, T.A., Reliable Blast UDP: Predictable High Performance Bulk Data Transfer. IEEE Cluster Computing, 2002.
15. Bresnahan, J., Link, M., Kettimuthu, R., Fraser, D., and Foster, I., "GridFTP Pipelining," in Teragrid 2007 Conference, Madison, WI, 2007.
16. Kettimuthu, R., Sim, A., Gunter, D. Allcock, W., Bremer, P., Bresnahan, J., Cherry, A., Childers, L., Dart, E., Foster, I., Harms, K., Hick, J., Lee, J., Link, M., Long, J., Miller, K., Natarajan, V., Pascucci, V., Raffenetti, N., Ressman, D., Williams, D., Wilson, L., Winkler, L., "Lessons learned from moving Earth System Grid data sets over a 20 Gbps widearea network", 19th ACM International Symposium on High Performance Distributed Computing (HPDC), 2010
17. Hacker, T., Athey, B. and Noble, B., The end-to-end performance effects of parallel tcp sockets on a lossy wide-area network.16th IEEECS /ACM International Parallel and Distributed Processing Symposium, 2002.
18. Hacker, T.J., Noble, B.D. and Athey, B.D., Improving Throughput and Maintaining Fairness using Parallel TCP. IEEE InfoCom, 2004.
19. Johnston, W., Greiman, W., Hoo, G., Lee, J., Tierney, B., Tull, C. and Olson, D., High-Speed Distributed Data Handling for On-Line Instrumentation Systems. ACM/IEEE SC97: High Performance Networking and Computing, 1997
20. Qiu, L., Zhang, Y. and Keshav, S., On Individual and Aggregate TCP Performance. 7th International Conference on Network Protocols, 1999.
21. OProfile: http://oprofile.sourceforge.net/news/
22. Linux TCP Tuning: http://fasterdata.es.net/fasterdata/host-tuning/linux/
23. Sar manual page: http://linux.die.net/man/1/sar
24. Iperf project: http://sourceforge.net/projects/iperf/
25. Sysctl manual page: http://linux.die.net/man/8/sysctl
26. FIO benchmark: http://linux.softpedia.com/get/System/Filesystems/fio-7881.shtml
27. Bonnie benchmark: http://www.textuality.com/bonnie/
28. Buffer bloat wiki: http://www.bufferbloat.net/projects/bloat
29. Gettys, J., “Buffer bloat: dark buffers in the Internet”, talk at Bell Labs. April 3, 2011.