Characterizing Throughput Bottlenecks for Secure GridFTP Transfers

Gayane Vardoyan[1,2] Rajkumar Kettimuthu[1,2] Michael Link[1,2] Steven Tuecke[1,2]
[1]Computation Institute, University of Chicago, Chicago, IL 60637
[2]Argonne National Laboratory, Argonne, IL 60439
{gvardoyan, tuecke}@uchicago.edu {kettimut, mlink, foster}@mcs.anl.gov

***Abstract*—GridFTP is the de facto standard for bulk data movement in distributed science environments. It extends the legacy FTP to provide strong security, reliability, and high performance. GridFTP, like FTP, is a two-channel protocol—the control channel is used for sending commands and responses, and the data channel is used for transferring the actual data. The control channel is encrypted and integrity protected by default. The data channel is authenticated by default. Encryption and integrity protection are both supported on the data channel but are not enabled by default because of their high CPU cost and low data transfer rates. In this paper, we present an extensive experimental study on the performance implications of enabling integrity protection and encryption on the data channel. We show that in a vast number of cases involving the use of nonthreaded Globus GridFTP servers on multicore systems, throughputs of secure transfers are not comparable to those of nonencrypted and nonintegrity-protected transfers because of an inefficient use of available processors. However, in cases where a strong desire for higher security levels permits larger expenditures in processing, integrity protection and sometimes even cryptographic confidentiality can be provided without having to suffer a decline in throughput. We show that this can be accomplished through threaded Globus GridFTP server instances configured with appropriately chosen parallelism and concurrency, allowing for a more effective use of available system resources.**

I. Introduction

When large amounts of data are being reproduced, whether across local area networks (LANs) or wide area networks (WANs), the layers of security used in the transfers are considerable priorities, especially with higher levels of confidentiality of the content. However, the process of conducting integrity checks on or the encryption of data comes with a nontrivial overhead that is imposed on the processing resources of the end systems involved in managing secure data connections. In practice, it is difficult for data transfer tools, even if equipped with state-of-the-art optimization mechanisms, to do encrypted or integrity-protected data transfers without compromising the speed.

One such tool is Globus GridFTP, which implements the GridFTP [1] protocol encapsulated within a robust set of libraries and yields a modular, scalable, high-performance tool for data management [2]. These attributes of Globus GridFTP and the widespread use of the Globus Toolkit [3] [4] [5] motivated our using Globus GridFTP server instances and globus-url-copy (the command-line GridFTP client) in the experiments described in this paper.

Ample literature exists about the benefits that can be obtained from using parallel TCP streams, as well as extended theoretical analysis of the correct number of parallel sockets recommended for achieviing optimal aggregate throughput [2] [6] [7]. Although the Globus GridFTP server can, in practice, achieve high-speed integrity-protected and encrypted data transfers [8], these speeds still do not usually fare well when compared with the simply authenticated transfers [4].

In general, it is not uncommon for integrity-protected and encrypted transfers to differ from less secure transfers by an order of magnitude in throughput, especially on high-speed links [4]. This performance drawback makes it tempting to rely on TCP checksums for protecting sensitive data, as an alternative to integrity protection. However, mere TCP checksums are not a suitable substitute for integrity-protection at GridFTP (or any application) level, as emphasized by Stone and Partridge [9], who analyze the causes of the highly random failures of TCP checksums.

In this paper, we characterize the performance of Globus GridFTP for four levels of security on the data channel—no security, authentication only, integrity protection, and encryption—as hardware platforms and optimization options (concurrency, parallelism) are varied. We present an extensive experimental study on systems with varying numbers of CPU cores, and we show that even fully encrypted Globus GridFTP transfers can be done without suffering a decline in throughput, through a more effective usage of available system resources, such as the number of cores present and the application of threaded Globus GridFTP server instances.

The use of multithreading to drive encrypted transfers at high speeds comes at a cost, however: the number of cores required grows with the link speed that is to be sustained. For example, extrapolation shows that approximately 100 cores per host would be required for a 10 Gbps integrity-protected data link between two pairs of endpoints.

We offer recommendations about the system configurations as well as the Globus GridFTP server configurations in order to achieve fully encrypted and integrity protected data transfer rates similar to those of authentication-only data transfer rates.

The rest of the paper is organized as follows. Section II provides background on GridFTP. Section III covers in detail the experiments conducted and their outcomes. In Section IV, we summarize our findings.

II. GridFTP Background

The GridFTP protocol [10] extends the File Transfer Protocol (FTP) with features that permit high performance and secure and reliable data movement. It is based on the RFC 959 [11], RFC 2228 [12], and RFC 2389 [13]. Additionally, it defines a new data channel protocol called extended block mode [10]. The GridFTP protocol has been standardized through the Open Grid Forum. The standardization has led to the development of multiple interoperable implementations. The Globus implementation of GridFTP [1] is the most widely used.

The following is a summary of the key features of GridFTP.

- GridFTP allows third-party control of data transfer whereby a user can initiate, monitor, and control data transfers between two remote machines that are the source and destination for the data transfer.
- GridFTP supports Generic Security Services application program interface (GSS-API) authentication of the control channel [12] and data channel (GridFTP extensions) [10]. It also supports user-controlled levels of data integrity and/or confidentiality.
- On wide-area links, using multiple TCP streams in parallel between a single source and destination can improve the aggregate bandwidth relative to that achieved by a single stream [7], [6]. GridFTP supports such parallelism through FTP command extensions and data channel extensions.
- GridFTP supports striped data movement operations, in which a set of computers is used in a coordinated fashion to move data from one parallel file system to another. Striping and parallelism may be used in tandem.
- Some applications can benefit from transferring portions of files rather than complete files: for example, 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.
- 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 that are not widely implemented. GridFTP exploits these features and extends them to cover its new data channel protocol.

III. Experiments

We have conducted various experiments aimed at discovering the performance-hindering factors of secure Globus GridFTP transfers and determining methods for avoidance of severe degradation in throughput for such transfers. For these experiments, third-party transfers were used. Tests were performed on five pairs of endpoints—three pairs on LAN and two pairs on WAN links. For LAN tests, we used a pair of dedicated data transfer nodes (DTNs) at the National Energy Research Scientific Computing Center (NERSC), each having four cores at 3.0 GHz; a pair of dedicated data transfer nodes at the Argonne Leadership Computing Facility (ALCF), each with four cores at 2.6 GHz; and a pair of compute nodes at the San Diego Supercomputing Center (SDSC) cluster Trestles, each with 16 cores at 2.4 GHz. For WAN tests, we used one of the DTNs at NERSC and one of the DTNs at ALCF, with four cores per host; a node on Ranger cluster at the Texas Advanced Computing Center (TACC), with 16 2.3 GHz physical cores; and a node on the Pittsburgh Supercomputing Center (PSC) high-speed data conduit, with 8 physical, 16 logical 2.4 GHz cores.

In the figures that follow, test datasets are described in the format $N \times S$GB, where N is the number of files in the dataset and S is the size of each file in the dataset. Experiments were run on three datasets: 32×4 GB, 1×64 GB, and 1×32 GB. In our experiments, we varied two optimization parameters in Globus GridFTP: parallelism and concurrency. Parallelism splits a file into multiple chunks and sends those chunks simultaneously across multiple TCP streams. Concurrency [14] enables the transmission of multiple files simultaneously. For the 32X4 GB dataset, concurrency values of $\{1, 2, 4, 8, 16\}$ were tested. Parallel values were $\{1, 2, 4, 8, 16, 32\}$.

The NERSC, ALCF, and Trestles LAN experiments eliminated the effects of as many unknown and unpredictable network variations in the final results as possible. Data obtained from these transfers aided the interpretation of the more common, yet more complex and noisy disk-to-disk WAN transfers. The variety of datasets arose from an interest in tracking the effects of concurrent transfers (such as scenarios where the total data size is kept the same but gets divided into varying numbers of equivalent chunks across an experiment) on overall performance and an opportunity to compare these results with the behaviors of threads on identical datasets.

In the context of this paper, data-channel authenticated (dcau) and no security on data channel (nodcau) transfers will be referred to as less secure or nonsecure transfers (or authenticated and unauthenticated transfers, respectively); protected transfers will reference -dcsafe transfers, which entail authentication and integrity-protection; and encrypted transfers will reference -dcpriv transfers, which are authenticated and cryptographically encrypted data connections.

A. Eliminating Disk I/O Interference

To allow the effects of integrity-protected and encrypted transfers on end-system CPU load to be more apparent, and to induce throughput and disk I/O independence, we conducted "memory-to-memory" in addition to disk-to-disk experiments. Since few systems have enough RAM available to perform large memory-to-memory data transfers, we simulated such transfers using /dev/zero as source files and /dev/null as destination files (accomplished with the help of globus-url-copy -length option, a parameter specifying the amount of data to be transferred).

However, an understandable concern arises when this method is used with dcsafe and dcpriv transfers: less CPU activity may be required to integrity-protect and encrypt a completely uniform dataset (such as one comprising solely

null values) than a random dataset. To dispel this suspicion, we obtained a 32 GB (which is a sufficient amount of memory, when used with relatively smaller files, to mimic RAM) tmpfs disk and compared its performance with that of /dev/zero-/dev/null transfers. For all four security parameters, the transfer times were nearly identical, thus leading us to conclude that /dev/zero-/dev/null transfers are a permissible model for memory data transfers. Hence, for simplicity, we hereafter refer to /dev/zero-/dev/null transfers as memory-to-memory transfers.

B. Testing on Production Environments

Our early experiments occupied the relatively low-core NERSC and ALCF data transfer nodes, with memory-to-memory transfers targeted at procuring insight about the ways in which concurrency, parallelism, and threading influence throughput for each of the four security options. These tests also illustrate the behaviors of continually busy and core-power limited systems in handling additional CPU-high loads introduced by these experiments.

1) Nonthreaded Experiments: From the NERSC LAN, ALCF LAN, and ALCF/NERSC WAN data gathered, the following can be noticed:

- For the nonthreaded LAN experiments, less-secure transfers are always at least 3 Gbps faster than more secure transfers (see Figures 2(b), 2(c), and 3(a) as examples). This difference in performance persists in all cases.
- For nonthreaded protected and encrypted WAN transfers, increasing parallelism provides a minor performance boost, but this effect is usually limited only up to a parallelism of 4. Further increasing the number of parallel streams returns negligible changes to throughput (see Figure 1 as an example).
- For the nonthreaded WAN transfers, throughput from secure transfers is comparable to that of nonsecure transfers only for parallel values of 1, 2, and 4. The gap then rapidly increases between the two groups of curves, as the secure transfers cease benefitting from the increasing parallelism, and the less-secure transfers continue to benefit from it (see Figure 1).

From this information, we can deduce that for secure transfers, a compute resource bottleneck occurs at approximately 4 parallel TCP streams. Clearly, this is a limitation on the processing power, and not a network hindrance, because authenticated and nonauthenticated transfers achieve exclusively higher throughputs for corresponding secure pairs of transfers. While all nonsecure LAN transfers are capped by possibly a combination of network and CPU bottlenecks, in this case, the WAN nonsecure transfers are constrained primarily by compute resources (Figure 1). In fact, once the number of parallel streams exceeds a threshold and a sufficiently high load is introduced to the systems' cores, average throughput begins to decrease (Figure 2(c)).

2) Threaded Experiments: Experiments were run on the ALCF and NERSC LANs to determine whether combinations of concurrency and parallelism values can attain throughputs

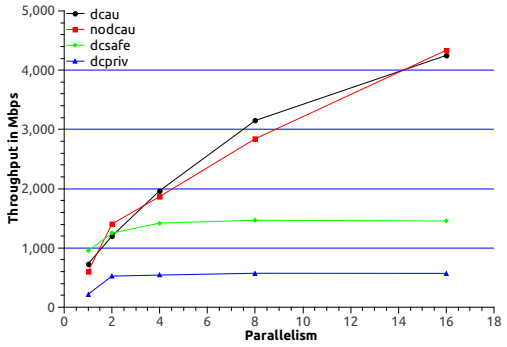
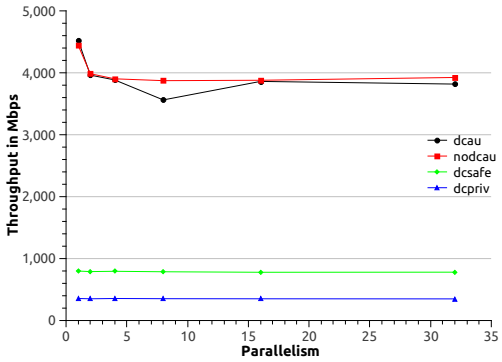
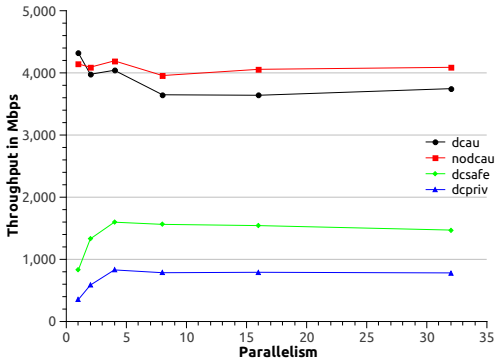


Fig. 1: Throughput between ALCF gs2 and NERSC dtn02, nonthreaded transfers. Dataset: 32x4 GB. Concurrency = 2, memory-to-memory.



(a)



(b)

Fig. 3: Comparison of threaded and nonthreaded transfers on 4-core systems. Source: ALCF gs1. Destination: ALCF gs2. Dataset: 1x32 GB. Transfers shown are memory-to-memory, with concurrency of 1. (a) shows throughput of nonthreaded LAN tests. (b) shows throughput of TCP threaded LAN tests with 8 threads, as parallelism varies.

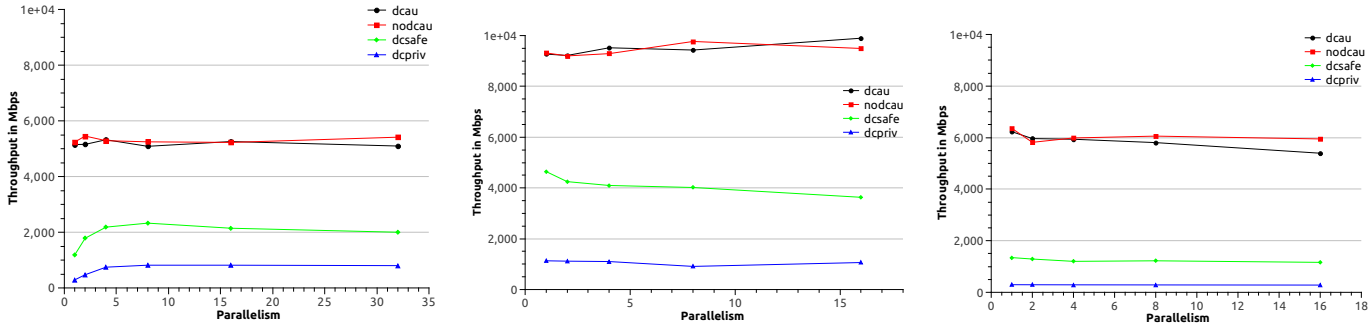


Fig. 2: Comparison of threaded and nonthreaded transfers on 4-core systems: (a) and (b) demonstrate how throughputs better than or comparable to those of threaded transfers can be obtained without using threading as an optimization parameter. Source: NERSC dtn01. Destination: NERSC dtn02. Dataset: 32x4 GB. All transfers are memory-to-memory. (a) is the result of threaded (with 8 threads) NERSC LAN transfers using TCP and concurrency of 1. (b) shows nonthreaded NERSC LAN throughputs for concurrency of 8. (c) shows nonthreaded NERSC LAN throughputs for concurrency of 1.

for integrity-protected and encrypted transfers that are higher than those reached with nonthreaded instances of globus-gridftp-server. On all endpoints, the number of threads was set to 8. An assessment of the NERSC LAN results produced the following points:

- For concurrencies of 1 and 2, secure transfers performed better when threaded. For concurrency of 1, threaded protected-transfer throughput was more than double the nonthreaded protected throughput; and it was quadruple the nonthreaded encrypted (compare Figures 2(c) and (a) for an example). For concurrency of 2, threaded encrypted received approximately 500 Mbps more than did non-threaded; and threaded integrity-protected received approximately 1 Gbps more than did nonthreaded protected transfers.
- For concurrencies of 1 and 2, nonsecure transfers had comparable performance, but threaded nonsecure transfers seemed to benefit more from increases in parallelism.
- For concurrencies of 4, 8, and 16, authenticated and unauthenticated transfers achieved higher (1–3 Gbps) throughputs than did their nonthreaded counterparts.
- For concurrencies of 4, 8, and 16, nonthreaded protected transfers received approximately 1 Gbps more than did threaded protected transfers; while threaded and non-threaded encrypted transfers performed the same. The reason concurrency improves the performance of secure transfers even when threading is not used is the following. The concurrency option enables simultaneous transfer of multiple files by creating separate GridFTP server processes. The number of concurrent GridFTP server processes created is min(concurrency value, number of files in the dataset). This results in parallel security processing on different CPUs in the system.

For a single, large file transfer on the ALCF LAN, nonsecure threaded and nonthreaded transfers produced comparable performance (Figure 3(a) and (b)). Threading with

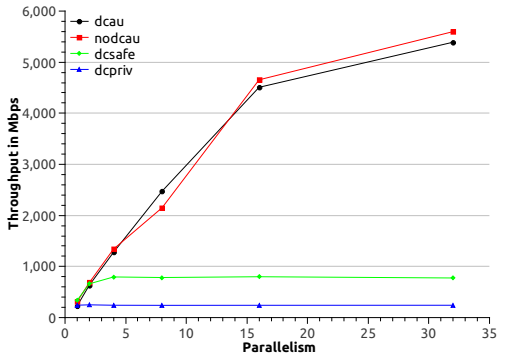
TABLE I: Comparison of threaded and non-threaded ALCF LAN and Ranger-to-Blacklight memory-to-memory transfers. Dataset 1x32GB.

	ALCF LAN		Ranger to Blacklight	
	No Threading	Threading	No Threading	Threading
auth.	(8, 3562) (1, 4524)	(16, 3640) (1, 4322)	(1, 230) (32, 5391)	(1, 308) (32, 5723)
non-auth.	(8, 3873) (1, 4442)	(8, 3957) (4, 4193)	(1, 314) (32, 5598)	(1, 313) (32, 5552)
protected	(16, 779) (1, 799)	(1, 829) (4, 1598)	(1, 337) (16, 800)	(1, 292) (16, 2683)
encrypted	(32, 349) (1, 356)	(1, 353) (4, 830)	(8, 240) (2, 250)	(1, 230) (16, 980)

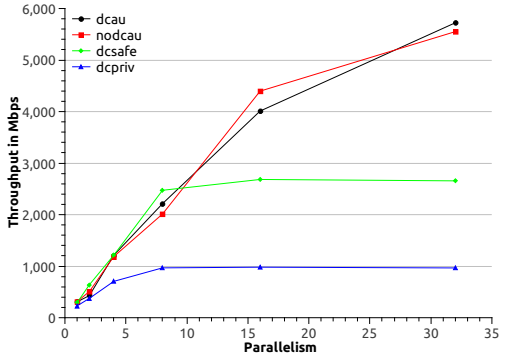
TCP doubled integrity-protected and encrypted throughputs (when comparing the maximum throughputs achieved by nonthreaded and threaded TCP experiments shown in Table I), from 799 Mbps to 1598 Mbps and from 356 Mbps to 830 Mbps, respectively. The increase in throughput for secure threaded TCP data connections with concurrency of 1 on the ALCF LAN, relative to nonthreaded transfers there, is consistent with the results of the NERSC LAN experiments. Table I presents the minimum and maximum throughputs encountered during the experiments, along with the parallel value that produced each throughput. Each cell consists of two pairs of elements (i, j), where i represents the parallel value and j is the corresponding throughput. In each cell, the top (i, j) element contains the minimum throughput, and the bottom contains the maximum throughput.

C. Quadrupling the Number of Available Cores

We now discuss the outcomes of tests conducted on systems capable of providing far more computational resources per GridFTP process than the two heavily used production environments previously mentioned. In these experiments, both the source (a Ranger node) and the destination (a Blacklight



(a)



(b)

Fig. 4: Throughput for (a) nonthreaded WAN tests and (b) threaded WAN tests using TCP and 32 threads. This figure allows a side-by-side comparison of WAN threaded and non-threaded tests on systems with 16 cores as parallelism varies. Source: Ranger. Destination: Blacklight. Dataset: 1x32 GB. Transfers had concurrency of 1 and were memory-to-memory.

node) are dedicated test systems that offer a total of 16 cores each and are RTT=68 ms (round-trip time) apart.

When comparing figures 1 and 4(a), one can again note the reoccurring pattern shared by nonthreaded memory-to-memory WAN transfers: for concurrency of 1, encrypted transfers are always capped at 500 Mbps; integrity-protected transfers are always capped by 1 Gbps; and there rarely seems to be an observable limit to authenticated and unauthenticated transfers' throughput as it climbs in a positive slope as the number of parallel streams increases.

One can also observe for the nonthreaded 1×32 GB dataset that changes in parallelism are negligible on nonthreaded encrypted transfer throughput and that transfer speed does not improve for protected nonthreaded transfers beyond 4 parallel streams. However, when the transfers are repeated with 32 threads on each end system (while using TCP), parallel values of 2, 4, and 8 become meaningful even for encrypted transfers, which are now capped by a 1 Gbps ceiling (Figure 4(b)). Protected threaded TCP transfers receive an enormous benefit from parallel value of 8, and even parallelism of 16 seems to provide a small throughput bump for both protected and encrypt transfers. Comparison of maximum values of throughput achieved with threaded TCP and nonthreaded experiments indicates that protected transfer speed is tripled, from 800 Mbps to 2683 Mbps, and encrypted transfer speed is almost quadrupled, from 250 Mbps to 980 Mbps (see Table I). The maximum throughput for the two less-secure transfer types remains approximately the same as in the nonthreaded experiment.

The reason parallelism improves the performance of secure transfers with threading is that the encryption protocol used by GridFTP (TLS/SSL) requires that data be decrypted in the same order that it was encrypted. Hence, we cannot take portions of a single data stream and process them in parallel on different CPUs. Parallel TCP streams optimization, which GridFTP uses to minimize penalties associated with TCP slow start and dropped packets, allows for parallel encryption when the GridFTP server is threaded.

Although the configurations shown in the Ranger-to-Blacklight experiment of Figure 4(b) probably do not produce the optimal results that can be achieved from this data transfer, they do illustrate the difference that a more efficient use of existing resources can make during the span of a transfer. The significant decrease in the throughput gap clearly shows that with threading TCP transfers, unused processing power is now contributing to the increase in performance.

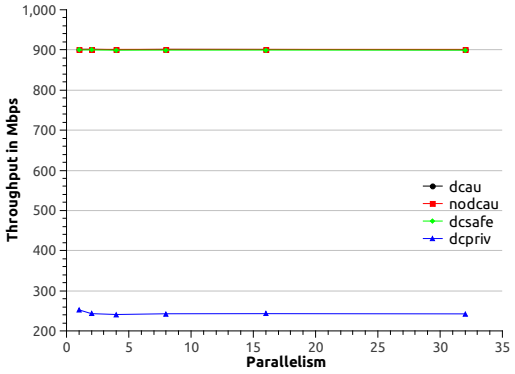
D. Octupling the Number of Available Cores

Tests run between two Trestles compute nodes (with number of threads equal to 64 for threaded transfers) described the behavior of throughput for encrypted data transfers in the presence of abundant logical cores.

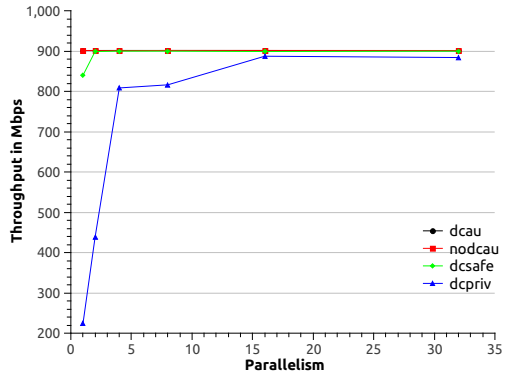
1) NonThreaded Tests: Because of 1000 Mbps NICs used on the interfaces between the Trestles compute nodes, the throughputs of authenticated, unauthenticated, and protected transfers were limited to approximately 900 Mbps (Figure 5(a)) for memory-to-memory transfers. The same is true for disk-to-disk transfers except that the integrity-protected transfer throughput limit observably decreased by 200–700 Mbps because of the extra overhead. No decrease in throughput is observed for authenticated and unauthenticated transfers because the combined CPU and I/O load of disk-to-disk transfers is still above the 1000 Mbps NIC limit (Figure 6).

However, this network bottleneck did not interfere with the display of encrypted data transfer throughput for standard memory-to-memory and disk-to-disk transfers. One can note from the encrypted-transfer curves in Figures 5(a) and 6 that despite the existence of 32 logical and mostly idle cores on both endpoints, the utilization of compute power is extremely feeble in the absence of threading.

2) Threaded Tests: For threaded TCP transfers on a 32-core NIC-bound system, authenticated, unauthenticated, and protected transfers still achieve throughputs equal to or greater than the network (in this case NIC) bottleneck. However, a rapid increase in performance is observed for encrypted transfers as the number of parallel streams increases from 1 to 4, and a further gain of almost 100 Mbps in throughput is attained as the parallel streams increase to 16 (Figure 5(b)).



(a)



(b)

Fig. 5: Throughputs shown for (a) nonthreaded test and (b) threaded (TCP with 64 threads) test on 32-core systems vs. parallelism. Source: Trestles node 1. Destination: Trestles node 2. Dataset for (a): 1x64 GB; (b): 1x64 GB for dcau, nodcau, dcsafe; 1x16 GB for dcpriv. Memory-to-memory, concurrency=1. Comparison of threaded and nonthreaded tests on systems with 32 logical cores. Protected nonthreaded throughputs match authenticated and unauthenticated throughputs, while encrypted threaded throughput nearly reaches nonsecure transfer performance.

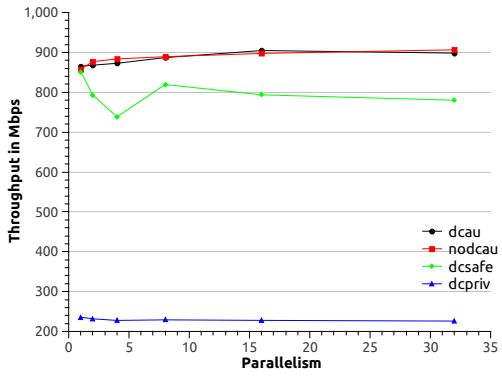


Fig. 6: Throughput for nonthreaded disk-to-disk Trestles test with variation in parallelism. Concurrency of 1, dataset 1x32 GB.

IV. Conclusion

The following are conclusions made as a result of analysis of empirical data collected from experimentation on both production and test environments:

- For systems with few cores, threaded transfers are beneficial for concurrent values of 1 and 2, where they maintain transfer speeds for authenticated and unauthenticated transfers yet increase throughput for protected and encrypted transfers.
- Threading with TCP is useful when the number of cores is 16 (can be physical or hyperthreaded), but parallel values of higher than 8 are not advantageous. For secure transfers, throughput evens out after 16 or more parallel TCP streams are used.
- For systems with 32 cores or higher, threading with TCP is advantageous for improving encrypted-transfer throughputs, even if such a system experiences a network throughput cap lower than its compute-power limit.

Acknowledgment

We thank Ian Foster for his valuable input and contributions.

References

[1] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, S. Tuecke, S. O. T. Memo, L. Liming, and S. Tuecke, "GridFTP: Protocol Extensions to FTP for the Grid," 2001.

[2] 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*, 2005.

[3] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of High Performance Computing Applications*, pp. 115–128, 1997.

[4] R. Kettimuthu, L. Lacinski, M. Link, K. Pickett, S. Tuecke, and I. Foster, "Instant GridFTP." in *9th Workshop on HighPerformance Grid and Cloud Computing*, 2012.

[5] W. Allcock, A. Chervenak, I. Foster, L. Pearlman, V. Welch, and M. Wilde, "Globus Toolkit Support for Distributed Data-Intensive Science," 2001.

[6] T. J. Hacker, B. D. Athey, and B. Noble, "The End-to-End Performance Effects of Parallel TCP Sockets on a Lossy Wide-Area Network," in *Proceedings of the 16th IPDPS*, 2002.

[7] L. Qiu, Y. Zhang, and S. Keshav, "On Individual and Aggregate TCP Performance," in *Network Protocols, 1999. (ICNP '99) Proceedings. Seventh International Conference on*, 1999, pp. 203 – 212.

[8] J. Bresnahan, R. Kettimuthu, M. Link, and I. Foster, "Harnessing Multicore Processors for High-Speed Secure Transfer," in *High-Speed Networks Workshop*, 2007.

[9] J. Stone and C. Partridge, "When the CRC and TCP Checksum Disagree," in *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '00, 2000.

[10] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke, "Data Management and Transfer in High-Performance Computational Grid Environments," *Parallel Computing*, 2002.

[11] J. Postel and J. Reynolds, "File Transfer Protocol (FTP)." *Internet Engineering Task Force, RFC 959*, 1985.

[12] M. Horowitz and S. Lunt, "FTP Security Extensions, RFC 2228," 1997.

[13] P. Hethmon and R. Elz, "Feature Negotiation Mechanism for the File Transfer Protocol RFC 2389," 1998.

[14] R. Kettimuthu, A. Sim, D. Gunter, B. Allcock, P.-T. Bremer, J. Bresnahan, A. Cherry, L. Childers, E. Dart, I. Foster, K. Harms, J. Hick, J. Lee, M. Link, J. Long, K. Miller, V. Natarajan, V. Pascucci, K. Raffenetti, D. Ressman, D. Williams, L. Wilson, and L. Winkler, "Lessons Learned from Moving Earth System Grid Data Sets Over a 20 Gbps Wide-Area Network," in *Proceedings of the 19th ACM International Symposium on HPDC*, 2010, pp. 316–319.