mdtmFTP and Its Evaluation on ESNET SDN Testbed

Liang Zhang, Wenji Wu, Phil DeMar
Fermilab
{liangz, wenji, demar}@fnal.gov

Eric Pouyoul
ESNET
lomax@es.net

ABSTRACT

To address the high-performance challenges of data transfer in the big data era, we are developing and implementing mdtmFTP: a high-performance data transfer tool for big data. mdtmFTP has four salient features. First, it adopts an I/O centric architecture to execute data transfer tasks. Second, it more efficiently utilizes the underlying multicore platform through optimized thread scheduling. Third, it implements a large virtual file mechanism to address the lots-of-small-files (LOSF) problem. Finally, mdtmFTP integrates multiple optimization mechanisms, including—zero copy, asynchronous I/O, pipelining, batch processing, and pre-allocated buffer pools—to enhance performance. mdtmFTP has been extensively tested and evaluated within the ESNET 100G testbed. Evaluations show that mdtmFTP can achieve higher performance than existing data transfer tools, such as GridFTP, FDT, and BBCP.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Applications;
C.2.4 [**Distributed Systems**]: Client/server

General Terms

Algorithms, Performance, Design

Keywords

Multicore, data transfer, high-speed networking.

1. Introduction

Big data has emerged as a driving force for scientific discoveries [1]. Large scientific instruments (e.g., colliders, light sources, and telescopes) generate exponentially increasing volumes of data. Currently, Large Hadron Collider (LHC) experiments generate hundreds of petabytes of data per years. The aggregated amount of climate science data is projected to exceed 100 exabytes by 2020. To enable scientific discovery, science data must be collected, indexed, archived, shared, and analyzed, typically in a widely distributed, highly collaborative manner [2-7]. At present, computing facilities for large-scale science, such as ALCF, OLCF, and NERSC, offer the types of computing and storage resources needed to process and analyze science data. The efficient movement of science data from their sources into processing and storage facilities and ultimately to user analysis is critical to the success of any such endeavor. Data transfer is now an essential function for science discoveries, particularly within big data environments.

Within the DOE research community, the emergence of distributed, extreme-scale science applications is generating significant performance challenges regarding data transfer [2-7]. First, it is becoming essential to transfer data at the highest possible throughputs to deal with exponentially growing volumes of science data. Second, DOE is in the process of deploying extreme-scale supercomputer facilities to support its extreme-scale science applications. To maximize utilization of these very high cost computing facilities, ultra-high-throughput data transfer capabilities will be required to move data in and out of them.

To date, several data transfer tools (e.g., GridFTP [8-9] and BBCP [10]) have been developed to support bulk data movement. Advanced data transfer features, such as transfer resumption, partial transfer, third-party transfer, and security, have been implemented in these tools and services. There have also been numerous enhancements to speed up data transfer performance. Parallelism at all levels (e.g., multi-stream parallelism [8] and multi-path parallelism [12-15]) is now widely implemented in bulk data movement, providing significant improvement in aggregate data transfer throughput.

Although significant improvements have been made in the area of bulk data transfer, the currently available data transfer tools will not be able to successfully address the high-performance challenges of data transfer in big data era for the following reasons:

- Existing data transfer tools are unable to fully exploit multicore hardware under the default OS support, especially on NUMA systems.
- Existing data transfer tools are unable to effectively address *the lots of small files (LOSF)* problem [16]. The state-of-the-art solutions to the LOSF problem—pipelining, concurrency, and tar-based solution—are either inefficient, or do not scale well.

To address these challenges, we have developed and implement mdtmFTP: A High-performance Data Transfer Tool in Big Data Era. mdtmFTP has been extensively tested and evaluated within the ESNET 100G testbed. Our evaluations show mdtmFTP can achieve higher performance than existing data transfer tools.

DOE's Advanced Scientific Computing Research (ASCR) office has funded Fermilab to work on Multicore-Aware Data Transfer Middleware (MDTM) [11]. mdtmFTP is the latest outcome of this research effort. mdtmFTP software is available at http://mdtm.fnal.gov.

2. mdtmFTP

mdtmFTP is a high-performance data transfer tool that builds upon the MDTM middleware (Figure 1). It has the following salient features:

- It adopts a pipelined I/O-centric architecture to execute data transfer tasks. Dedicated I/O threads are spawned to perform network and disk I/O operations.
- It utilizes MDTM middleware services to make optimal use of the underlying multicore system.
- It implements a *large virtual file* mechanism to address the LOSF problem.
- Zero copy, asynchronous I/O, pipelining, batch processing, and buffer pools mechanisms are applied to optimize performance.

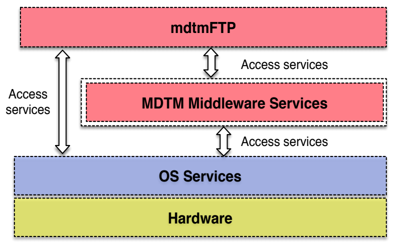

Figure 1 The MDTM Architecture

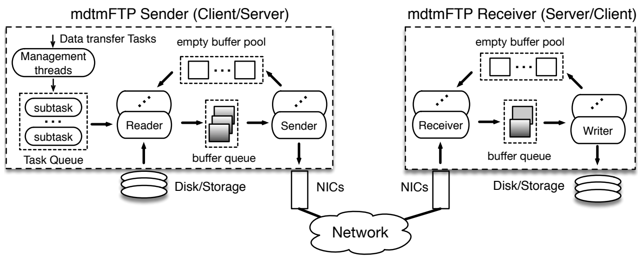

Figure 2 A Pipelined I/O Centric Design

2.1 A pipelined I/O centric design

Multicore has become the norm of high-performance computing. In order to take full use of the underlying multicore processing capability, mdtmFTP adopts a pipelined I/O centric design. A data transfer task is carried out in a pipelined manner across multiple cores. Dedicated I/O threads are spawned to perform network and disk I/O operations in parallel (Figure 2).

mdtmFTP handles two types of I/O device, storage/disk(s) and NIC(s). Depending on a device's I/O capability, one or multiple threads are spawned for each I/O device. Typically, four types of I/O threads will be spawned:

- Disk/storage reader threads to read data from disks or storage systems.
- Disk/storage writer threads to write data to disks or storage systems.
- Network sender threads to send data to networks via NIC.
- Network receiver threads to receives data from network via NIC.

In addition to the I/O threads, mdtmFTP spawns management threads to handle user requests, and management-related functions.

mdtmFTP calls MDTM middleware scheduling service to schedule cores for its threads. For each I/O thread, MDTM middleware first selects a core near the I/O device (e.g., NIC or disk) the thread uses, and then pins the thread to the chosen core. This strategy has two benefits: (1) it enforces I/O locality on NUMA systems; (2) it avoids I/O thread migrations, thus providing core affinity for I/O operations. Therefore, mdtmFTP performance can be significantly improved. Typically, an I/O thread is dedicated with a single core. No other threads will be scheduled to a core that an I/O thread has been assigned to.

In addition, MDTM middleware partitions system cores into two zones – MDTM zone and non-MDTM zone. mdtmFTP runs in the MDTM-zone while other applications are confined within the non-MDTM-zone. This strategy reduces other applications' interference to mdtmFTP, thus resulting in optimum data transfer performance.

High-performance data transfer involves a significant amount of memory buffer operations. To avoid costly memory allocation/deallocation in the critical data path of data transfer, mdtmFTP pre-allocates multiple data buffers, and manages them in a data buffer pool. Data buffers are pinned and locked so as to avoid being paged to the swap area and memory migration. Data buffers are recycled and reused.

mdtmFTP executes data transfers in a pipelined manner. In the sender, management threads first preprocess data transfer requests. A data transfer task is typically split into multiple subtasks. A subtask can comprise file segments, a group of files, or file folders. Subtasks are then put in a task queue. Disk/storage reader threads keep fetching subtasks from the task queue. For each subtask, data will be first fetched from storage/disk(s) into empty buffers. Filled data buffers are temporarily put in a buffer queue. Concurrently, network sender threads continue fetching filled data buffers from the buffer queue, and send data to the network in parallel on multiple TCP streams. In the receiver, data will be received into empty buffers via network receiver threads; afterwards, the filled buffers will be passed over to storage/disk writer threads to store data into storage/disk(s).

2.2 MDTM middleware service

To achieve higher scalability and efficiency, most existing OS schedulers use a distributed run-queue model, in which the scheduler maintains one run queue per core. The scheduler applies a thread-independent scheduling policy, which schedules threads independently, regardless of application types and dependencies. Periodically, the scheduler balances the load across cores to facilitate load balance. In the case of NUMA systems, the balancing is across all NUMA nodes. When data transfer applications run on multicore systems, dynamic load balancing may result in frequent thread migration, or leading to high-latency inter-node communications, which would significantly degrade the overall data transfer performance. Furthermore, I/O devices (e.g., NIC and storage) on NUMA systems are connected to processor sockets in a NUMA manner. This results in NUMA effects for transfer between I/O devices and memory banks, as well as CPU I/O access to I/O devices. Investigations show that I/O throughputs can be significantly improved if applications can be placed on cores near the I/O device they use (i.e., I/O locality) [19, 20]. However, existing OSes have very limited supports for such IO locality. Processes/threads may end up being scheduled on cores that are distant from the I/O devices they use, leading to high-latency inter-node I/O operations and incurring extra communication overheads. Bulk data transfers involve significant network and disk I/O operations. Using default OS scheduling can lead to significant inter-node I/O operations and severely degrade the overall data transfer performance.

MDTM middleware has been developed to address these problems. It is a user-space resource scheduler that harnesses multicore parallelism to scale data movement toolkits at multicore systems.

MDTM middleware is implemented as a system daemon. Periodically, the daemon collects, monitors, and caches information about the multicore system physical layout (e.g., NUMA topology), configurations, and system loads. Using this information, MDTM middleware will provide query and scheduling services to the data transfer tool, mdtmFTP.

Today, MDMT middleware supports the following features:

- Computer system layout profiling.
- Real-time system status monitoring: (a) CPU usage of each core, and (b) memory load latency of each NUMA node. This feature allows mdtmFTP to use system resources (cores and data buffers) intelligently to avoid overloading particular cores or NUMA nodes.
- NUMA topology-based core scheduling, which supports I/O locality (see section 2.1).
- Supporting core affinity on I/Os
- System zoning, which partitions system cores into two zones – MDTM zone and non-MDTM zone. mdtmFTP runs in the MDTM-zone while other applications are confined to run in the non-MDTM-zone.
- Data buffer allocation and pinning capability

MDTM middleware was designed to support mdtmFTP. However, it can be readily extended to support other types of applications. Or, it can be used to study advanced scheduling algorithms and policies on NUMA systems.

2.3 A large virtual file mechanism to address the LOSF problem

Existing data transfer tools are unable to effectively address the LOSF problem. GridFTP uses pipelining and concurrency to address the inefficiency in LOSF. However, GridFTP's data transfer performance is not satisfying [see section 3.3]. Some data transfer applications such as BBCP [10] make a tar ball of the dataset and then transfer the tar ball as one file. The problem is the "tar" process might involve significant amount of disk/memory operations, which is normally costly and slow. FDT adopts a data stream mechanism to stream a dataset (list of files) continuously, using a managed pool of buffers through one or more TCP sockets [18]. However, FDT failed in an experiment we designed to evaluate its capability in addressing the LOSF problem (see section 3.3).

mdtmFTP implements a large virtual file mechanism to address the LOSF problem. The mechanism works as shown in Figure 3.

1. A mdtmFTP sender receives a request to transfer a dataset to a mdtmFTP receiver. The sender quickly traverses the dataset, and creates a large "virtual" file for the dataset. Logically, each file in the dataset, which include regular files, folders, and symbolic links, is treated as a file segment in the virtual file. Each file in the dataset is sequentially "added" to the virtual file with start position and end position. The virtual file is not physically created. Instead, a content index table is created to maintain metadata for the virtual file. The content index table consists of entries. Each entry corresponds to a file in the dataset. It records the file's metadata, such as file name, path, type, and its start and end positons in the virtual file.
2. The sender serializes the content index table and transmits it to the receiver.
3. The receiver deserialize and reconstructs the content index table, and then asks for data transfer.
4. Using the content index created earlier, the sender continuously reads data blocks from disk and sends them out to networks. The whole dataset is transferred continuously and seamlessly as a single virtual file in one or multiple TCP streams.
5. When receiving a data block from the sender, the receiver first looks up the content index table to determine which file the data block belongs to and its positon with the file, and then store the data block into disk/storage.

Our large virtual file mechanism has two benefits: (1) it eliminates protocol processing between the sender and receiver on a per-file basis. And (2) it allows for batch processing small files in the sender and receiver. Therefore, I/O performance can be optimized.

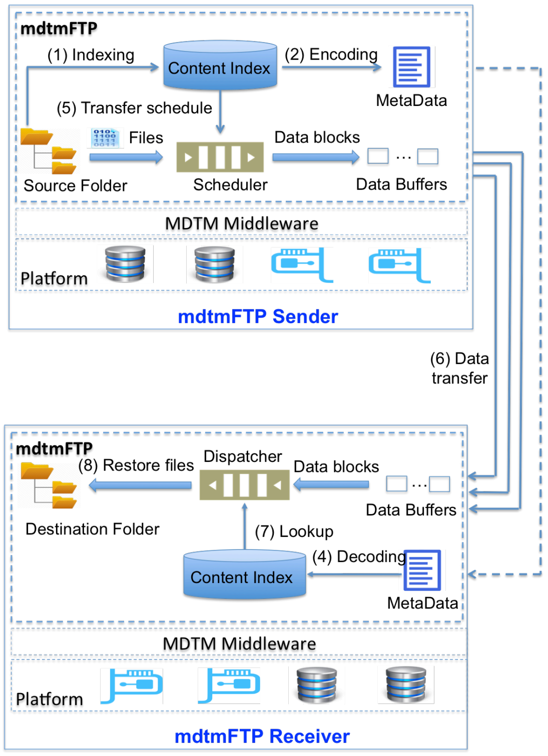

Figure 3 Large virtual file transfer mechanism

3. mdtmFTP evaluation @ ESNET testbed

3.1 ESnet testbed

We evaluated mdtmFTP within the ESNET 100GE testbed, using high performance systems at NERSC (Oakland, CA) [17]. This testbed focuses on high performance data plane experiments, providing sufficient computing/IO resources. The topology of the testbed is shown in Figure 4.

To emulate a wide area network (WAN) path, our tests utilized a path with a latency of 95ms RTT. The path was created using a loop on a 100GE circuit between NERSC and the StarLight network exchange (Chicago).

Our tests were run on the testbed performance hosts, which were designed to support experimentations that require very fast networking or disk operations. Those hosts, nersc-tbn-1 and nersc-tbn-2 are located in Oakland, California and are typically used as source and sink of data between themselves using the 100GE dedicated 95ms loop.

Those hosts are running Proxmox hypervisors. Users are given a dedicated Linux container with root access. Network interfaces are attached to the Linux containers. Containers are similar to “bare metal” access on a host, and have similar performance. The container technology that we used on the testbed is called Proxmox, and the underlying Operating system was Ubuntu 12. This means that no matter what OS is running in the container, the experiment will be using the Ubuntu TCP stack.

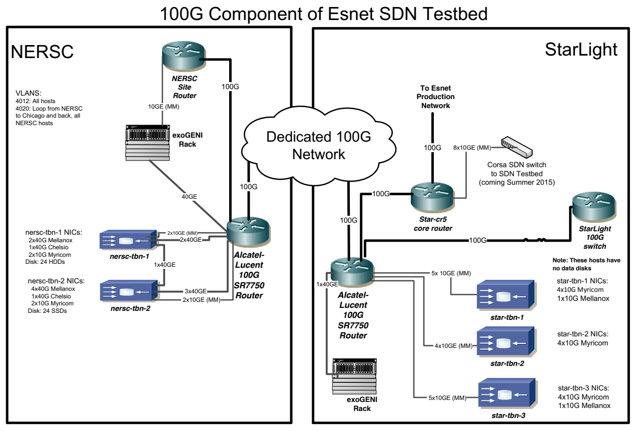

Figure 4 ESnet Testbed

3.2 Evaluation methodology

We ran data transfer from “nersc-tbn-2” to “nersc-tbn-1”. In our evaluation, mdtmFTP was compared with FDT, GridFTP, and BBCP. For fair comparisons, all the tools were configured with the same parameters—I/O block size and the number of parallel streams. The detailed configurations are listed in Table 1.

Tools	Streams	Pipelining	Concurrency	TCP/IP Parameter
FDT	4	N/A	N/A	Default*
GridFTP	4	-PP	-CC 8	Default*
BBCP	4	N/A	N/A	Default*
mdtmFTP	4	N/A	2 I/O Threads	Default*

*System configuration.

Table 1 Testing Configuration

We were investigating two transfer modes: client-server and third-party. In the client-server mode, the client starts the transfer task and also takes the role as either data source or data destination. In the third-party mode, the client starts the transfer task but the data is exchanged between two other servers.

Three data transfer scenarios were evaluated: (1) Large file trafer, which transfers a 100GByte large file from nersc-tbn-2 to nersc-tbn-1. (2) Folder transfer 1, which transfers a folder that has 30 10G files from nersc-tbn-2 to nersc-tbn-1. It aims to evaluate a tool’s capability in transferring folders with large files. And (3) Folder transfer 2, which tranfers a Linux folder that is ~554 MBytes in total, and contains ~50,000 files of various sizes and types. It aims to evaluate a tool’s capability in addressing the LOSF problem.

Time-To-Completion (TTC) is used as the performance metric. For better comparion, we used GridFTP as base and calculated the Relative Performace Improvement (RPI), which is defined as:

$$RPI = \frac{GirdFTP's\ TTC}{Other\ tool's\ TTC}$$

3.3 Evaluation results

a. Large Single File Transfer: Client-Server Mode

The first comparison is to transfer a single large file with the size of 100GB from nersc-tbn-2 to nersc-tbn-1. The results are listed in Table 1. The relative performance comparison is depicted in Figure 5. It can be seen that mdtmFTP is approximately 14% faster than FDT, and ~20% faster than GridFTP as shown in Figure 4. BBCP takes much longer time to finish the transfer and therefore is not included in the figure.

	mdtmFTP	FDT	GridFTP	BBCP
TTC (second)	74	80	91	Poor

Table 1 TTC - Large file data transfer (smaller is better)

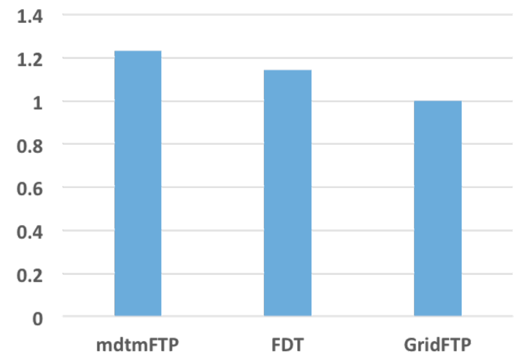

Figure 5 RPI - Large File Transfer (larger is better)

b. Folder Transfer: Client-Server Mode

For folder transfers, the results are shown in Table 2 and 3. The relative performance comparisons are illustrated in Figure 6 and 7.

In both folder transfer scenarios, mdtmFTP performs better than existing data transfer tools. Especially with folder transfer #2, mdtmFTP is hundreds of times faster than GridFTP and BBCP. For folder transfer #2, the Linux folder is ~554 MBytes in total, and contains ~50,000 files of various sizes and types. This is a tpical LOSF scenario. The experiment clearly show that mdtmFTP is able to address the LOSP problem effectively.

To our surprise, FDT crashed in folder transfer #2. It seems like that FDT has bugs in handling folder data transfer.

	mdtmFTP	FDT	GridFTP	BBCP
TTC (second)	192	217	320	Poor

Table 2 TTC Folder transfer 1 (smaller is better)

	mdtmFTP	FDT	GridFTP	BBCP
TTC (second)	11	N/A	1006	6274

Table 3 TTC - Folder transfer 2 (smaller is better)

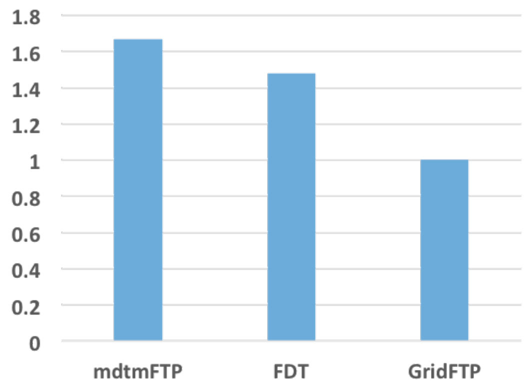

Figure 6 RPI – Folder data transfer 1 (larger is better)

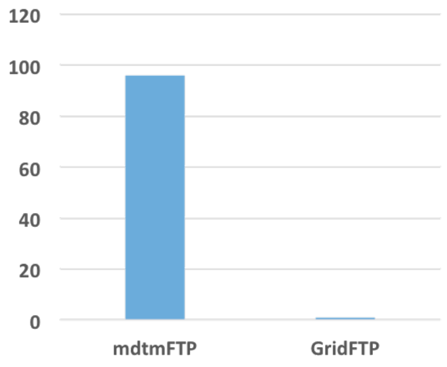

Figure 7 RPI – Folder data transfer 2 (larger is better)

c. Large File Transfer: Third Party Mode

Third party mode enables a client to control file transfer between two remote servers. It is very useful in a number of user cases. All these data transfer tools (mdtmFTP, BBCP, FDT, and GridFTP) all support 3rd party data transfer. However, only mdtmFTP and GridFTP can run 3rd party data transfer in ESNET testbed. Note: there is not a third system in the WAN loop, except nersc-tbn-1 and nersc-tbn-2. GridFTP and mdtmFTP can start data transfer on a designated data interface. However, FDT and BBCP do not have such a feature.

Table 4 and Figure 8 show the resutls for the single large file transfer in third party mode. mdtmFTP is about three times faster than GridFTP.

	mdtmFTP	FDT	GridFTP	BBCP
TTC (second)	35	N/A	107	N/A

Table 4 TTC – Large file transfer (smaller is better)

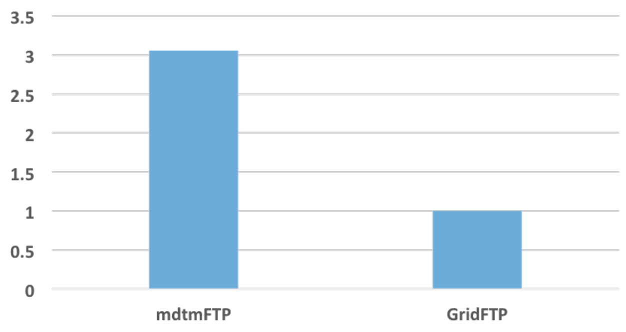

Figure 8 RPI - Large File Transfer (larger is better)

d. Folder Transfer: Third Party Mode

Table 5 and 6 show the results for two folder transfer secnarios in the third party mode: one contains 30 files with size of 10GB; the other is the standard Linux source tree. Only mdtmFTP supports folder transfer in the 3rd party mode. GridFTP does not support this feature.

	mdtmFTP	FDT	GridFTP	BBCP
TTC (second)	96	N/A	Not Working	N/A

Table 5 TTC - Folder transfer 1

	mdtmFTP	FDT	GridFTP	BBCP
TTC (second)	10	N/A	Not Working	N/A

Table 6 TTC - Folder transfer 2

4. Conclusion

To address the high-performance challenges of data transfer in the big data era, we are, developing and implementing mdtmFTP: a high-performance data transfer tool for big data. mdtmFTP has several salient features. First, it adopts an I/O centric architecture to execute data transfer tasks. Second, it can fully utilize the underlying multicore system. Third, it implements a large virtual file mechanism to address the lots-of-small-files (LOSF) problem. Finally, mdtmFTP integrates multiple optimization mechanisms—zero copy, asynchronous I/O, pipelining, batch processing, and pre-allocated buffer pools—to optimize performance. mdtmFTP has been extensively tested and evaluated at ESNET 100G testbed. The evaluation shows that mdtmFTP achieves better performance than existing data transfer tools, such as GridFTP, FDT, and BBCP.

Acknowledgement

We would like to thank Brian Tienery who contributed to mdtmFTP evaluation @ ESNET testbed.

5 REFERENCES

[1] "Synergistic Challenges in Data-Intensive Science and Exascale Computing", DOE ASCR Data Subcommittee Report 2013.

[2] Eli Dart, Mary Hester, Jason Zurawski, "Basic Energy Sciences Network Requirements Review - Final Report 2014", ESnet Network Requirements Review, September 2014, LBNL 6998E

[3] Eli Dart, Mary Hester, Jason Zurawski, "Fusion Energy Sciences Network Requirements Review - Final Report 2014", ESnet Network Requirements Review, August 2014, LBNL 6975E

[4] Eli Dart, Mary Hester, Jason Zurawski, Editors, "High Energy Physics and Nuclear Physics Network Requirements - Final Report", ESnet Network Requirements Workshop, August 2013, LBNL 6642E

[5] Eli Dart, Brian Tierney, Editors, "Biological and Environmental Research Network Requirements Workshop, November 2012 - Final Report"", November 29, 2012, LBNL LBNL-6395E

[6] David Asner, Eli Dart, and Takanori Hara, "Belle-II Experiment Network Requirements", October 2012, LBNL LBNL-6268E

[7] Eli Dart, Brian Tierney, editors, "Advanced Scientific Computing Research Network Requirements Review, October 2012 - Final Report", ESnet Network Requirements Review, October 4, 2012, LBNL LBNL-6109E

[8] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke, "GridFTP: Protocol Extension to FTP for the Grid," Grid Forum Internet-Draft, Mar. 2001.

[9] B. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu and I. Foster, "The Globus Striped GridFTP Framework and Server," SC'2005, 2005.

[10] BBCP, http://www.slac.stanford.edu/~abh/bbcp/

[11] http://mdtm.fnal.gov

[12] Han, Huaizhong, et al. "Multi-path tcp: a joint congestion control and routing scheme to exploit path diversity in the internet." IEEE/ACM Transactions on Networking (TON) 14.6 (2006): 1260-1271.

[13] Wang, Bing, et al. "Application-layer multipath data transfer via TCP: schemes and performance tradeoffs." Performance Evaluation 64.9 (2007): 965-977.

[14] Iyengar, Janardhan R., Paul D. Amer, and Randall Stewart. "Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths." Networking, IEEE/ACM Transactions on 14.5 (2006): 951-964.

[15] Gunter, Dan, et al. "Exploiting network parallelism for improving data transfer performance." High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:. IEEE, 2012.

[16] Bresnahan, John, et al. "Gridftp pipelining." Proceedings of the 2007 TeraGrid Conference. 2007.

[17] https://www.es.net/network-r-and-d/experimental-network-testbeds/100g-sd

[18] http://monalisa.cern.ch/FDT/

[19] S. Akram, M. Marazkis, and A. Bilas, "NUMA Implications for Storage I/O Throughput in Modern Servers," In 3rd Workshop on Computer Architecture and Operating System co- design (CAOS'12), Paris, France, January 2012.

[20] S. Moreaud, B. Goglin, "Impact of NUMA Effects on High-Speed Networking with Multi-Opteron Machines," In PDCS 2007, Cambridge, Massachussetts (2007).