UNIVERSITY OF AMSTERDAM

MSc Security and Network Engineering

Track: Research Project 1

Master Research Thesis

Capability analyses of mdtmFTP

by

Kees de Jong

11853719

February 11, 2018

Supervisor:
dhr. dr. ing. Leon Gommans

Assessor:
dhr. prof. dr. ir. Cees de Laat

Contents

Abstract

A decade ago it was theorized that if enough parallel processing power was thrown at an I/O problem, the bottleneck would shift towards the NIC's link speed [1]. However, today NUMA architecture changes this paradigm, it also matters on which cores these threads are located and how they are managed [2]. Transport protocols are also critical in high latency networks, TCP's congestion control algorithm is window based and every time a congestion event is detected, the window size is reduced to half. This congestion control algorithm has significant limitations on HBDP (High Bandwidth Delay Product) network environments [3].

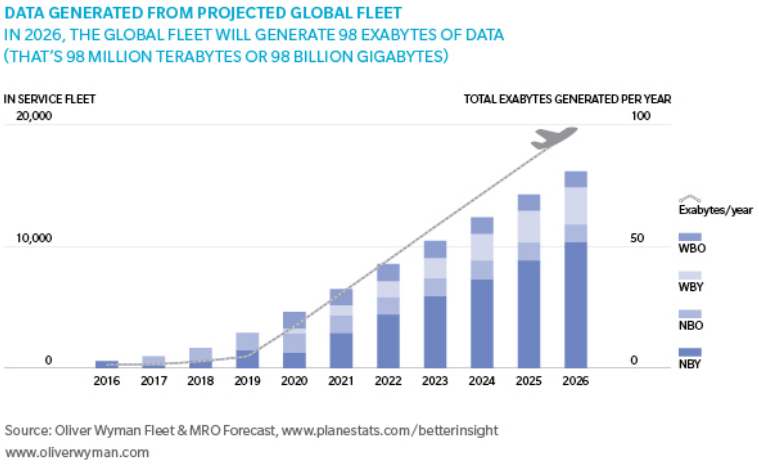
This research project report delves into the capabilities of mdtmFTP (Multicore-Aware Data Transfer Middleware File Transfer Protocol) [4] and compares it to Globus GridFTP [5] using transatlantic data transfer experiments on a 100 Gbit/s light path between Amsterdam and Chicago with a RTT (Round Trip Time) of 95 ms. mdtmFTP aims to accelerate data movement toolkits on multicore systems by using an application level resource scheduler which is aware of the underlying hardware topology. Globus GridFTP will be configured with both TCP and UDT [6] to evaluate UDT's effectiveness on HBDP network environments. UDT is an application-level data transport protocol that uses UDP to transfer bulk data, while implementing its own reliability and congestion control mechanisms and can be enabled as a Globus XIO driver in Globus GridFTP [1]. A variety of experiments will conclude which tool optimally utilizes the 100 Gbit/s transatlantic light path.

1 Introduction

1.1 Background

Airplanes not only transport passengers and cargo, but also data, recorded during the flight, which can be used for post-flight maintenance analyses. As seen in figure 1, it is estimated that the accumulated global fleet flight data could reach 98 exabytes by 2026 [7]. It therefore becomes critical to facilitate an infrastructure capable of handling this increase in data volume. At present day an airplane could collect several terabytes of flight data. The collected flight data includes e.g. sensor readings, engine data and more. For KLM flights, the post flight data analyses is performed in Amsterdam, with airplanes flying all over the world. If data is needed to avoid a flight delay and therefore quickly needed diagnose a problem when at the gate of a remote airport, it becomes a challenge to transfer this volume of data fast enough back to Amsterdam within the time frame between arrival and subsequent departure. Transferring these terabytes of data back to Amsterdam from e.g. Chicago over a public internet connection could easily take 30 hours [1]. This is not fast nor private enough for this use case, therefore KLM is interested in using private transatlantic 100 GBit/s light paths provided by Netherlight [8]. To better understand an emerging data transfer tool, expected to provide improved high performance data transfer across a light path, we will compare the capabilities of a new tool called mdtmFTP, and compare it with the traditional Globus GridFTP tool. DTN's (Data Transfer Node) will be the hardware platform used in order to conduct transatlantic data transfers between Amsterdam and Chicago. A DTN is a purpose-built server, running Linux, which are dedicated to the function of wide area data transfers. These machines are equipped with fast network and storage capabilities.

Figure 1: Estimated accumulated global flight data



1.2 Structure

The structure of this report is as follows. A brief technical overview of mdtmFTP, Globus GridFTP and UDT are discussed in section 3. Related work (section 2) will be discussed after the research question and scope are discussed in section 1.3 and 1.4. The methodology used to conduct experimentation is discussed in section 4. The results are outlined and discussed in section 5 followed by a conclusion (section 6) and future work (section 7).

[1] Assuming continuous availability of 100 Mbit/s bandwidth

1.3 Research question

The main research question is: "**What are the capabilities of mdtmFTP compared to Globus GridFTP on a 100 Gbit/s light path between Amsterdam and Chicago (RTT 95 ms)?**". This translates into the following sub questions:

1. Which features and/or design allows optimum throughput?
2. How do these data transfer tools behave with various sets of different file sizes and quantity?
3. Is the conclusion still valid that Globus GridFTP over UDT outperforms TCP on a high latency network [9] (this premise will be discussed in more detail in section 2)? And is it enough to beat mdtmFTP?

1.4 Scope

To answer the research questions; mdtmFTP and Globus GridFTP will undergo several capability experiments. This will include several transfer experiments to observe their capabilities; 1) a large contiguous file transfer experiment of sufficient size to fill up the available bandwidth. 2) A 500MB directory transfer experiment; containing files of various sizes to expose the LOSF (Lots Of Small Files) problem [10]. 3) An experiment with KLM flight data to observe the usability for practical use cases.

2 Related work

2.1 Scheduling techniques

Various transfer experiments were done by L. Zhang et al. [11, 12] using node-to-node data transfers on a shared network and simulated WAN path. The WAN path was realized by creating a loop between NERSC (Oakland) and the StarLight network in Chicago. The WAN path had a bandwidth of 100 Gbit/s and had a RTT of 95 ms. With a 100GB file node-to-node transfer mdtmFTP outperformed Globus GridFTP by about 20%. An mdtmFTP LOSF transfer experiment transferring 550MB (containing 50.000 files) completed 60% faster than Globus GridFTP. All experiments were done using the same I/O block size and number of parallel streams. It concluded the hardware topology awareness of mdtmFTP was the prime factor to its success compared to GridFTP.

2.2 Transport protocols

Work done by J. Bresnahan et al. [9] compared the performance of Globus GridFTP using UDT and TCP on various test beds. The highest latency network that was used during their experiments was between Argonne National Laboratory (ANL) and the University of Auckland (New Zealand) with a RTT of 204 ms. A few experiments showed that TCP performed equally or better compared to UDT. There was no mentioning of efforts trying to duplicate the results with subsequent experiments. However it was concluded that UDT outperformed TCP in most cases, as a trade off UDT did show a significant deeper resource footprint during the experiments in terms of CPU and RAM.

3 Technical overview

This section is a literary research which maps out the relevant features of Globus GridFTP and mdtmFTP, this is to understand the performance features and their application. The software releases used were Globus GridFTP 6.0 and mdtmFTP 1.0.2.

3.1 GridFTP protocol

Both mdtmFTP and Globus GridFTP use the GridFTP protocol which is an extension of the FTP protocol and adds reliability, performance and security features. Just like FTP, GridFTP also uses control and data channels. The control channels are used to orchestrate the data channels, the data channels are used to send the actual data. This separation allows certain optimizations such as the third-party data transfer option in both mdtmFTP and Globus GridFTP, as will be discussed in this technical review.

3.2 Globus GridFTP

Globus GridFTP is recognized for its secure, reliable, high-performance data transfers by e.g. CERN. The Globus GridFTP tool is part of the Globus Toolkit [13] and is a modular framework which allows the coordination of multiple data streams. Globus GridFTP offers many features, the following features are relevant within the scope of this research.

3.2.1 Parallelism

This feature allows the transfer a file over multiple transport streams by dividing up the file into blocks, optimizing throughput. This feature can be used for both LOSF and large single contiguous files.

3.2.2 Concurrency

When transferring multiple small files, with a short time to transfer, an extra overhead is introduced in orchestrating such a data transfer, i.e. OCRW (Open, Close, Read and Write) operations. To balance this overhead, concurrent FTP sessions can be established to speed up and balance the orchestration of these transfers. Therefore, concurrency in general doesn't add a performance gain for single file transfers.

3.2.3 Pipelining

With connection oriented data transfers such as with TCP, acknowledgments need be received successfully before the next data transfer can initiate. On HBDP (High Bandwidth Delay Product) network environments this latency can greatly affect performance. By applying pipelining, the client is allowed to have many outstanding unacknowledged transfers and its not forced to wait for the "226 Transfer Successful" message [10]. This feature makes the latency transparent and can improve performance, especially on high RTT network paths.

3.2.4 Third-party transfer

Three parties are involved in a third-party transfer; the client, which will only take on a data orchestration role and two servers, one of which will be sending data and the other will receive data. By offloading the initiation, monitoring and control of a data transfer to a third party, resources are freed from the servers that send/receive the data. This speeds up data processing capabilities, especially with concurrent data transfers.

3.2.5 UDT

By default, Globus GridFTP uses the TCP transport protocol. However, it is known that the TCP protocol is less suitable for fast transmission on paths with a high RTT. Even if TCP is used in various concurrent and parallel data transfers with pipelining applied, it requires a relative long time to adapt to the available network bandwidth. This is also known as BDP (Bandwidth Delay Product) which is mainly a product of TCP's AIMD characteristic [3, 14]. The additive-increase/multiplicative-decrease (AIMD) algorithm is a feedback control algorithm that combines linear growth of the congestion window with an exponential reduction when congestion takes place. The window size is then increased at most one segment each round-trip time, this congestion control algorithm has significant limitations on fast, long-distance networks. To address the limitations of the above-mentioned TCP's AIMD-based congestion control mechanism, numerous alternatives have been developed.

One of the alternatives that can be used is UDP, compared to TCP it transmits data more quickly due to its light control overhead, but it cannot guarantee reliability. As a response to this shortcoming a number of UDP-based adaptations have emerged such as UDT and other transmission protocols. UDT is well known to excel in HBDP network environments [15] due to its congestion control and reliability mechanisms. UDT is built on top of UDP and can be enabled as a driver within the Globus XIO module [16, 9]. All the error, reliability and congestion control mechanisms reside on the application-level [6].

3.2.6 Threading

When a user initiates a data transfer from/to a Globus GridFTP server, a thread/process will be forked. This thread/process will handle disk I/O, network I/O, and all other processing. In NUMA systems, this design is inefficient (as will be discussed in more detail in section 3.3.3). Also, when the number of users increase, too many threads could be forked, as a result the server will run out of resources. However, work done by W. Allcock et al. [16] demonstrated that a server could support 1800 concurrent clients without excessive load. Exposing this problem is beyond the scope of this research report.

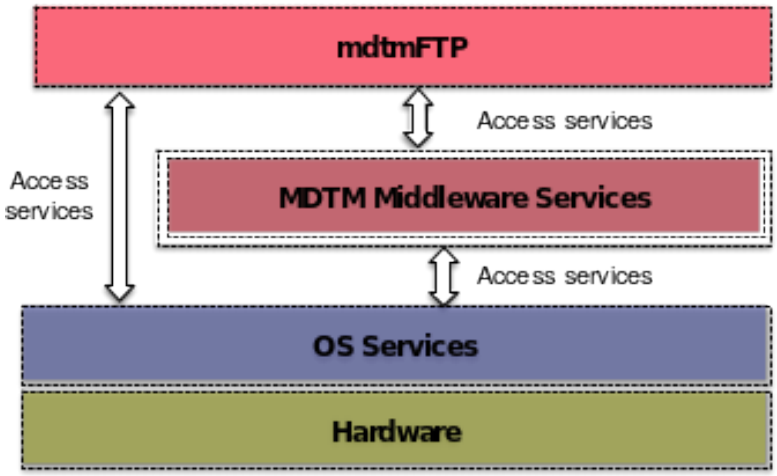
3.3 mdtmFTP

mdtmFTP uses several Globus modules viz. the GridFTP protocol, XIO and user interface modules. mdtmFTP is a response to the increase demand in fast data movement tooling. Existing tooling was falling short on a few levels which resulted in less bandwidth utilization. This section will discuss the features that set mdtmFTP apart from Globus GridFTP, within the scope of the research. First, mdtmFTP has its own user-space I/O scheduler to optimize data transfer tasks. Second, it also utilizes the underlying hardware topology more efficiently through optimized thread scheduling. And third, it implements a large virtual file mechanism to address the lots of small files (LOSF) problem. mdtmFTP also has a few other optimization mechanisms such as zero copy, asynchronous I/O, pipelining, batch processing, and pre-allocated buffer pools to enhance performance [12].

3.3.1 I/O centric middleware

MDTM (Multicore-Aware Data Transfer Middleware File Transfer Protocol) is a middleware which functions as a user-space resource scheduler implemented as a system daemon and does not totally depend on OS I/O thread scheduling, unlike Globus GridFTP. Instead mdtmFTP has its own I/O user-space scheduler. MDTM's purpose is to accelerate data movement toolkits on multicore systems. Figure 2 shows the layered approach of MDTM. In essence it consists of two large logically components; the data transfer applications/tools, in this case mdtmFTP, and the generic middleware MDTM. The underlying hardware topology is being used to enable a generic middleware to perform at extreme-scale data movement, especially on NUMA systems. These features enable efficient network and disk I/O on multicore systems [4].

Figure 2: MDTM system architecture



3.3.2 Thread scheduling

When in operation, the MDTM middleware daemon is launched which queries the system configuration and status on behalf of the MDTM application, i.e. mdtmFTP. Based on the configuration and hardware topology viz. I/O locality, NUMA and other factors, the MDTM middleware schedules and assigns system resources, according to the need of the application (hence pipelined I/O centric scheduling). During this operation, three types of threads are spawned; management threads, disk/storage I/O threads and network I/O threads. Respectively the first thread is used to execute and implement user requests. The last two threads execute the function of data access and transmission. These threads facilitate the I/O centric data transfer architecture. Threads are created to read and write data from disks or storage systems as well as sender and receiver threads to send and receive data via the NIC. The MDTM middleware uses cost functions for each resource (e.g., NIC or disk/storage) to select the nearest core. In abstraction, this cost function is comparable to the shortest path problem in graph theory. The Dijkstra's algorithm is used for this [17]. Once the most optimized paths between resources have been chosen it enforces I/O locality and avoids thread migrations. Thread migration would otherwise happen by OS scheduling, to load balance a thread over multiple processors (more about this in section 3.3.3).

3.3.3 NUMA

Existing data movement tools are limited by major inefficiencies when run on multicore systems. This is specially true for NUMA (Non-Uniform Memory Access) systems [2]. In such a design it becomes profitable to ensure data locality by scheduling threads on a certain NUMA node that has certain devices such as disks, NIC's and memory locally attached. Pinning threads to a NUMA node is also known as NUMA affinity. An operating system is often designed to load balance the threads over the multiple cores, this will create NUMA effects. These effects are a consequence when NUMA nodes use remote resources, i.e. not their locally attached resources. This affects performance since traveling between these NUMA nodes involves a cost in terms of latency. Thus running threads on remote I/O devices and memory banks, as well as remote CPU I/O access to I/O devices [18] should be avoided. This is the reason why mdtmFTP identifies itself as a pipelined I/O centric application rather than a monolithic one, i.e. Globus GridFTP. mdtmFTP makes use of the underlying hardware topology in a more efficient way by having each I/O thread pinned on the same NUMA node that has the relevant I/O device locally attached.

Also, two thread zones are created to prevent the interference of non-MDTM applications. mdtmFTP runs in its respective MDTM-zone while other applications are confined in the non-MDTM-zone. The performance requirements for the management threads are minimal and thus resides in the non-MDTM-zone, this thread is used to execute and implement user requests (section 3.3.2). This design reduces interference and results in optimum data transfer performance [12].

3.3.4 Handling lots of small files

Existing data movement tooling not only neglect the full utilization of the underlying hardware topology, but are also unable to effectively address the lots of small files (LOSF) problem [12]. Handling a data transfer with lots of small files generates a lot of file management overhead, i.e. OCRW (Open, Close, Read and Write) operations. Many data movement tooling use pipelining, concurrency or memory-hungry tar-based solutions for both small and large files. These solutions only improve the throughput by a fraction. mdtmFTP features a large virtual file mechanism which logically groups small files (regular files, folders, and symbolic links) and treats it as a segment of a single large virtual file. The segments of this larger virtual file are transferred together with a metadata file containing the file name, path, type, sequential start and end position information of the individual files within the segment. This speeds up the process because it eliminates the small file processing overhead and also the lots of per-file basis network transfers. These segments can then in turn be send by the MDTM scheduler in batches over multiple parallel TCP streams. This makes much faster data movement possible for LOSF situations [18]. The receiver does the reverse; when a data block is received, it reconstructs the data based on the metadata file and stores it to disk.

3.3.5 Direct I/O

Direct I/O is a system-wide feature that supports direct reads/writes from/to a storage device to/from user memory space, bypassing system page cache. Buffered I/O is usually the default I/O mode enabled by most operating systems. With buffered I/O, the data is copied twice between storage and memory because of the page cache as the proxy between the two. If data is reused, the introduction of page cache could achieve better performance. But for bulk data transfer, data is seldom reused. Therefore Direct I/O would be a better choice for such a scenario.

3.3.6 Splice

Splice avoids data copies within the kernel. It is a feature that moves data directly from storage to the NIC (sender side), or from NIC to storage (receiver side) using PIPE [19] within the kernel.

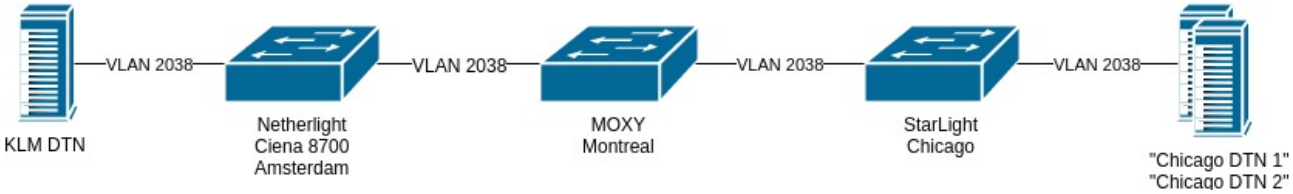
3.3.7 Pipelining, concurrency, parallelism and third-party transfers

Since mdtmFTP uses the Globus GridFTP module it shares some common features like pipelining, parallelism and third-party transfers. These are discussed in more detail in section 3.2. As an additional feature, mdtmFTP also supports a third-party option for recursive folder transfers, Globus GridFTP does not. Furthermore, the way mdtmFTP handles pipelining, parallelism and concurrency is different. As already mentioned in section 3.3.1, mdtmFTP has an I/O centric design. A data transfer job involves network and disk I/O operations, which are executed in one or multiple network I/O threads and disk I/O threads, in a parallel and pipelined manner. The number of I/O threads allocated to each I/O device (disk, NIC) is based on the capacity of the device. For example; 1 or 2 network I/O threads are assigned for a 10GE NIC, 4 network I/O threads for a 40GE NIC. For a mdtmFTP server, no matter how many users run data transfer from/to the server, the number of I/O threads are fixed. This design scales well with multiple users.

3.4 Network

Figure 3 illustrates the direct layer 2 connection between Amsterdam and Chicago. For the duration of the experiments an exclusive reservation was arranged for the full 100 Gbit/s bandwidth. Access to performance metrics could only be measured on the DTN's, no access was provided to the switches. All DTN's had CentOS 7 installed.

Figure 3: Network diagram



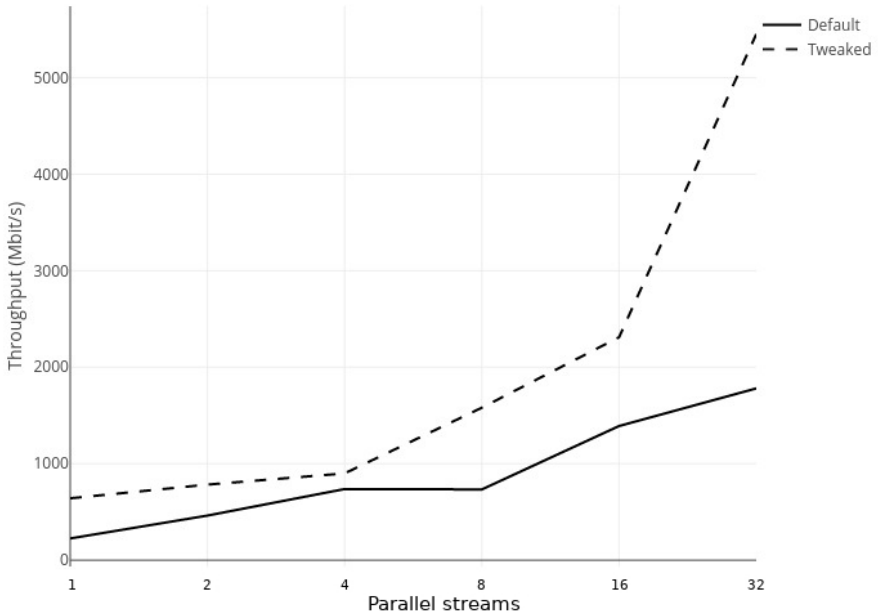
4 Methodology

This section describes how the results were gathered.

4.1 Capabilities of the test environment

In order to make an informed evaluation of the experiment results, the capabilities of the network and disks need to be known. For the network an iPerf [20] test was used as a baseline for the network performance. In figure 4 the iPerf results are graphed with incremental parallel streams (indicated on the X axis). The thick line indicates the default CentOS 7 network settings which includes the Cubic TCP congestion algorithm and the default TCP buffer sizes. The dotted line represents the tweaked parameters which were taken from the Chicago DTN's. These tweaks include using the HTCP TCP congestion control algorithm, TCP window scaling, fair queuing and increased TCP buffer sizes. Since network tuning was not part of the scope and the Chicago DTN's were already configured to do transatlantic experiments with CERN, the Chicago DTN network tweaks were also applied to the KLM DTN. The sanity of these settings were verified with the recommendations of ESnet [21]. With 32 parallel streams on the dotted line, representing the tweaked configuration, a throughput of 5500Mbit/s is realized. Note that this is just a baseline to show the performance impact of the network tweaks on the KLM DTN, the throughput of iPerf doesn't guarantee an absolute minimum or maximum of the bandwidth. Different applications and data transfers can influence the throughput.

Figure 4: iPerf baseline results



With a baseline of the network performance known it then becomes important to verify if disk performance is able to theoretically saturate the link. Table 1 lists the collected performance metrics obtained with bonnie++ [22]. The KLM DTN displays the highest read performance, it therefore took on the server role because it's able to provide twice the speed that a single Chicago DTN's disk can write. These roles will also scale better with the third-party data transfers (section 3.2.4).

Table 1: Disk performance of DTN's

	KLM DTN	**Chicago DTN 1**	**Chicago DTN 2**
Max read speed	~1500 MB/s	~1000 MB/s	~1200 MB/s
Max write speed	~800 MB/s	~700 MB/s	~700 MB/s
# disks	2	6	6

4.2 Expose behavior of the data transfer tools

As stated in the scope (section 1.4), several data transfers are needed of different sizes and quantity to measure the capabilities of mdtmFTP and Globus GridFTP. A file size of 100GB contiguous data (generated with `dd` with `/dev/zero` as a source) cannot be transmitted within a single TCP window (with TCP Window Scaling enabled the maximum size is 1GB). It therefore needs to send multiple data packets, this exposes BDP (Bandwidth Delay Product) effects [23]. This data set will expose the efficient utilization of the bandwidth available.

For the LOSF experiment the GIMP git repository [24] was used which has a total size of 501MB. This git repository contained 8577 files, 258 directories with an average file size of 100KB. These data properties expose the overhead of lots of small files management (as explained in section 3.3.4).

The KLM flight data consists of 30 encrypted zip files with an average file size of 15MB, with a total directory size of roughly 500MB. These data properties also expose the overhead of lots of small files management, but since the average file size is larger the overhead should be less. The experiment results should reflect this by having a faster TTC.

4.3 Performance measurement

Performance metrics were collected on both the network level and application level. This was done because having a high throughput on the network level doesn't always mean that the application is also capable of processing the data at the same pace. This processing time needs to be taken into account since it affects usability. The TTC was measured in seconds from the point the command was issued to the point an exit 0 was returned.

Network performance was measured on the KLM DTN with collectd [25], with a metric collection interval of 5 seconds and a retention of 90 days. The network send and receive metrics were collected from the network interface as octets (bytes). Other metrics that were collected include packet loss, RAM and CPU usage of the KLM DTN. This data was collected to analyze the throughput realized and evaluate the TTC metrics in more detail.

4.4 Experimentation and expected results

The experiments were repeated 10 times to observe any deviations in the results. Before each data transfer experiment, the buffers and caches were dropped on the involved DTN's. This was done to prevent caching and hence create inconsistent performance results. As discussed in section 4.1, the KLM DTN took on the server role due to its superior read speed. For all experiments 4 parallel streams (section 3.2.1) were used for both Globus GridFTP and mdtmFTP [26]. Because the parallel streams are enabled, it enables *Mode E* which stands for extended block mode. This mode is used to send data as blocks over the data channels. Both mdtmFTP and Globus GridFTP have support for *Mode E*.

100GB file transfer + third-party data transfer experiment

For large contiguous file transfers there are a few additional features that can be enabled to improve performance for mdtmFTP; Splice and Direct I/O (section 3.3.5 and 3.3.6). These features will also be tested in third-party mode. It is expected that these features will shorten the TTC for mdtmFTP. It's not expected to encounter thread scaling issues, although mdtmFTP is better designed to handle multiple users (section 3.2.6 and 3.3.7), the experiments are too small to expose these effects.

LOSF and KLM flight data transfer experiment

Concurrency is a feature that could greatly benefit performance with lots of OCRW (Open, Close, Read and Write) operations by spreading the tasks over multiple concurrent processes (section 3.2.2). Globus GridFTP can set several concurrency levels. Based on previous work by E. Yildirim et al. [27] and initial experimentation, a concurrency level of 2 and 4 will be used. Adding too many concurrent processes wastes resources without getting additional performance in return. By testing two different concurrency levels, a trend could be visible, indicating the effect on the TTC. Pipelining (section 3.2.3) can assist in making latency transparent and is well suited for data transfers involving lots of files. It is expected that by adding extra concurrency and pipelining, the TTC shortens. If pipelining

doesn't shorten the TTC then latency is most probably not a negative key performance factor in this experiment.

mdtmFTP also shares these above mentioned features albeit improved, in the sense it is designed to be aware of the hardware topology (section 3.3.7), an additional feature that could greatly improve performance is the large virtual file mechanism (section 3.3.4). This feature is expected to create a major performance boost compared to handling files on a one-by-one basis. Also the hardware topology awareness of mdtmFTP should give a performance boost, as discussed in section 3.3.3. Direct I/O is expected to improve performance since files will not be reused (section 3.3.5). Splice can speed up the process by putting files from disk directly onto the NIC (section 3.3.6).

5 Results and discussion

This section shows and discusses the results. The graphs are included as attachments in section 8. The TTC metrics are displayed in boxplots to illustrate the TTC distribution of the repeated experiments. The TTC is displayed in seconds where a lower TTC is considered better. The throughput is displayed in line graphs and is only used for the third-party data transfers to show the throughput behavior and the average and maximum throughput. The throughput in the line graphs are displayed in bytes where more is better. The Splice functionality in mdtmFTP crashed and is therefore not included in the graphs.

5.1 100GB experiment

Figure 5 shows the TTC of a single 100GB node-to-node data transfer. This transfer was done from the KLM DTN to the Chicago DTN 2. The boxplot shows that Globus GridFTP performs better, with an average of 6 seconds faster TTC. With Direct I/O enabled, mdtmFTP is able to come close to Globus GridFTP's faster performance. This is interesting since Globus GridFTP doesn't have the benefit of bypassing the operating system's page cache, which Direct I/O allows.

5.2 100GB third-party experiment

The following experiments were done in third-party mode, where the Chicago DTN 1 was taking on the client role and the KLM DTN and Chicago DTN 2 took on the server role. From the 2 KLM DTN disks, 6 100GB files were transferred (each drive transferred 3 of the 6 files) concurrently to the 6 disks of the Chicago DTN 2.

Table 2 gives a brief overview of the TTC metrics per third-party data transfer. With UDT enabled, Globus GridFTP performs roughly 40% better than with TCP enabled. In figure 6 the TCP slow start is visible and the throttling back and forth in throughput, which is typical for TCP. An average of 1 GBps of additional throughput is realized with UDT enabled (figure 7). This confirms earlier findings of J. Bresnahan et al. [9]. However, an increase in CPU/RAM usage was not observed in contrast to TCP. Other UDT characteristics are visible, such as the ability to obtain and sustain a faster throughput. After 2 minutes there is a noticeable drop in throughput visible in figure 7. No packet loss or increase/decrease in system resources was recorded. It's therefore possible that the switches on the light path experienced a brief performance drop related to buffers. Repeated experiments displayed minor deviations in the TTC of these third-party data transfers.

mdtmFTP appears not to scale very well with third-party data transfers, this is in contrary of work done by L. Zhang et al. [11, 12]. Direct I/O does improve performance by about 80%. Without Direct I/O performance starts out fairly well but then collapses after roughly 30 seconds. CPU/RAM did not display a sudden increase in usage around that time. Having Direct I/O disabled seems to have a negative performance impact. This is surprising since mdtmFTP is designed to scale better with multiple clients in contrast to Globus GridFTP. Furthermore, Globus GridFTP lacks the Direct I/O feature, this does not seem to affect performance. The Direct I/O feature, or other features might be experiencing bugs which are exposed in the research test environment.

Table 2: Third-party performance overview

Experiment	**TTC**	**Max throughput**	**Average throughput**
Globus GridFTP with TCP	195	5.517 GBps	3.589 GBps
Globus GridFTP with UDT	161	6.571 GBps	4.486 GBps
mdtmFTP without Direct I/O	520	3.989 GBps	1.367 GBps
mdtmFTP with Direct I/O	215	4.566 GBps	3.262 GBps

5.3 LOSF experiment

The following experiments expose the tools ability to handle the LOSF problem. In figure 10 the performance with a concurrency level of 2 is illustrated. It can be concluded that TCP performs

slightly better than UDT. Adding pipelining doesn't lower the TTC significantly, from that observation it can be concluded that network latency isn't affecting performance in this experiment, but rather the overhead involved with handling LOSF. This becomes more apparent in figure 11, which uses a concurrency level of 4. This increased level of concurrency allows Globus GridFTP to handle the overhead of LOSF more efficiently.

mdtmFTP also shows an improvement in performance with Direct I/O enabled for LOSF as illustrated in figure 12. It also becomes apparent that the large virtual file mechanism of mdtmFTP shortens the TTC almost by a factor of 10. This verifies earlier work done by L. Zhang et al. [11, 12] which concluded that this mechanism greatly increases performance by removing much of the overhead involved with LOSF.

5.4 KLM flight data experiment

As seen in figure 13, with a concurrency level of 2 and 4 the performance differs on average a few seconds for both UDT and TCP. However, again it can be concluded that UDT performs poorly with LOSF compared to TCP. The aggressive nature of UDT to increase throughput doesn't benefit the LOSF transfer use cases. However, when parallelism is added in figure 14, the performance metric landscape changes dramatically. On average the TTC increases for all experiments, except for UDT with a concurrency level of 2 and pipelining enabled. This is interesting since this puts Globus GridFTP on equal terms with mdtmFTP, as can be seen in figure 15. Taking into account the performance benefits and trade-offs of concurrency and pipelining, it could be concluded that a concurrency level 4 creates congestion in processing these transfers and could be considered wasteful. Furthermore, pipelining hides the latency of the light path for UDT and improves performance. mdtmFTP has on average a TTC of 5 seconds, just like Globus GridFTP with a concurrency level of 2 and pipelining enabled. With larger files (roughly 10MB) in an LOSF experiment, Globus GridFTP is able to perform much better since this lowers the overhead of handling short period data transfers, as was expected in section 4.2.

6 Conclusion

In this research report the performance capabilities were identified of mdtmFTP and Globus GridFTP over both TCP and UDT. Based on the gathered results, the main following three conclusions can be drawn. 1) For 100GB GB large contiguous file and third-party transfers it can be concluded that Globus GridFTP over UDT is performing best. Globus GridFTP is on average 75% faster in third-party mode than mdtmFTP fastest configuration (Direct I/O enabled). And 2) mdtmFTP is clearly benefiting performance from its large virtual file mechanism for LOSF and KLM flight data transfers. However, 3) Globus GridFTP with a concurrency level of 2 and pipelining enabled performs equally well with KLM flight data transfers.

Furthermore, mdtmFTP is a very promising project in both features as in performance. Related work concluded better performance with mdtmFTP compared to Globus GridFTP on all levels. In the research test environment of this report, performance findings only concluded mdtmFTP as superior with LOSF experiments. It should be noted that a more modern Linux distribution (especially with a more recent kernel) could improve performance for mdtmFTP significantly. The KLM DTN node had the default CentOS 7 Linux 3.10 kernel installed. With a limited reservation time on the light path and difficult physical access to the data center, upgrading the kernel was considered too risky due to the possibility of permanent connection loss after a reboot. A recent kernel could benefit the stability of mdtmFTP, in particular the Splice functionality and performance as a whole. Further testing of mdtmFTP is needed to pinpoint the exact cause of performance differences compared to related work (section 2).

As a final conclusion is that the use of mdtmFTP in a production environment is not ready, based on a test environment consisting of CentOS 7 DTN's. Globus GridFTP over UDT provides higher throughput and a more mature user experience in terms of documentation and usability for large file transfers and KLM flight data.

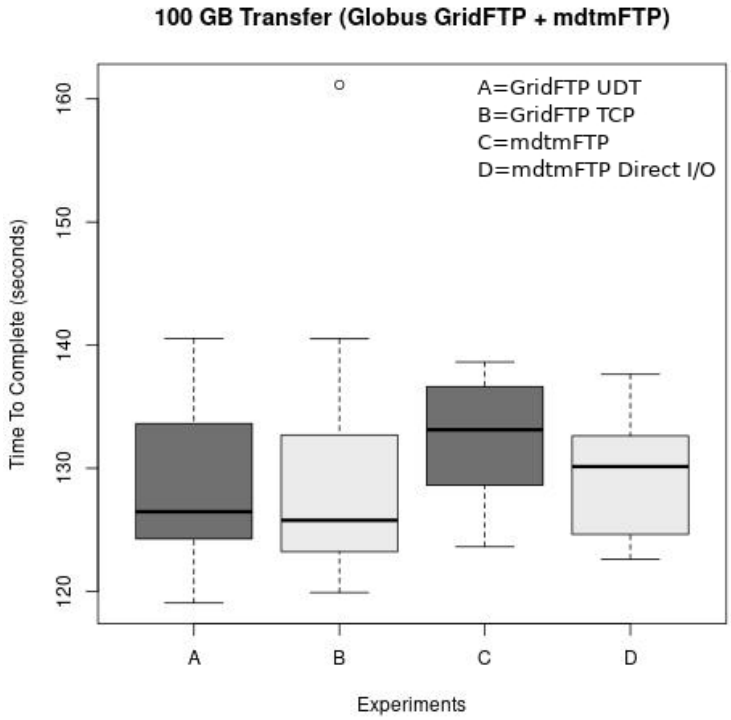
7 Future work

Further testing of mdtmFTP is needed, preferably on more recent Linux distributions, relative to CentOS 7, in order to establish a firm conclusion on its performance compared to Globus GridFTP. For a better comparison between TCP's abilities in contrast to UDT, further testing with TCP BBR [28, 29] (by default not available on CentOS 7) could provide a more balanced result. Furthermore, although strong efforts were taken to have a full reservation of the 100 Gbit/s bandwidth, there was no technically enforced reservation. Testing these findings on a true enforced reservation is needed to verify the results of this report.

8 Attachments

8.1 100 GB transfer graph

Figure 5: 100GB experiment



8.2 Third-party data transfer graphs

Figure 6: 100GB experiment (Globus GridFTP with TCP)

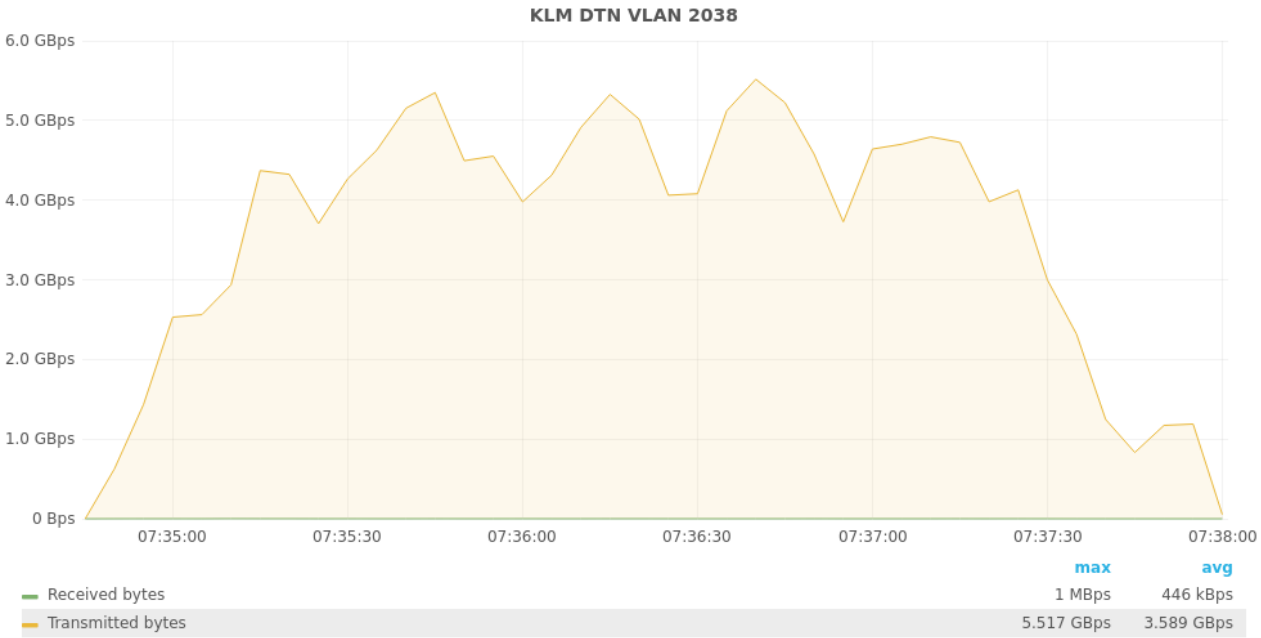


Figure 7: 100GB experiment (Globus GridFTP with UDT)

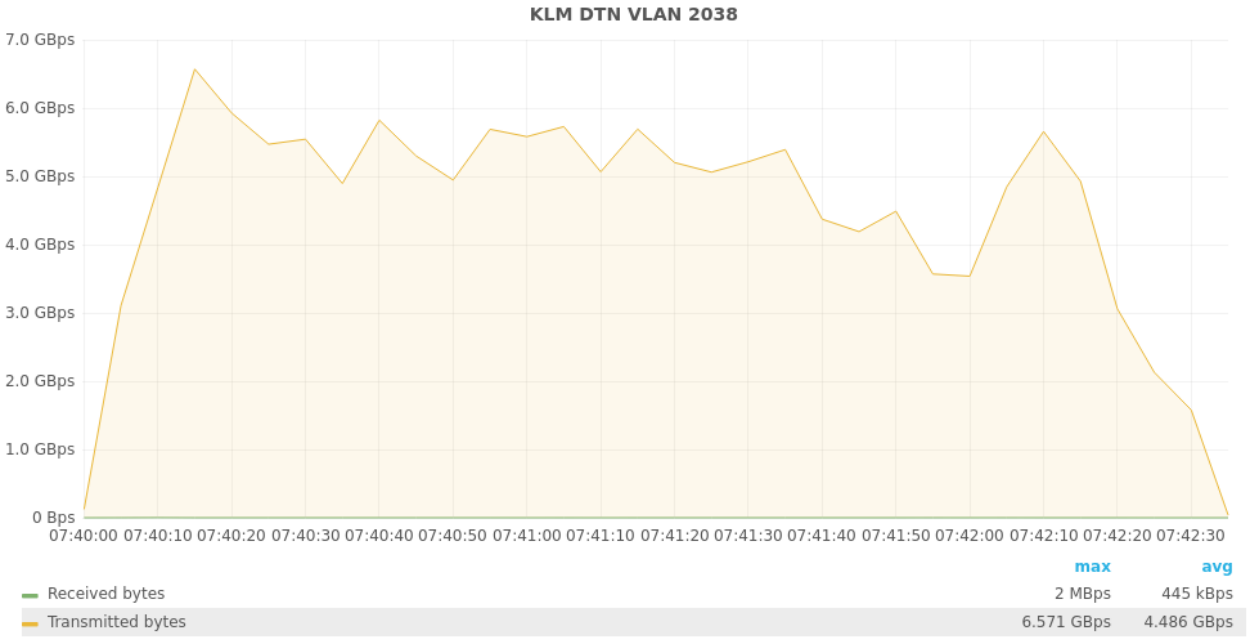


Figure 8: 100GB experiment (mdtmFTP without Direct I/O)

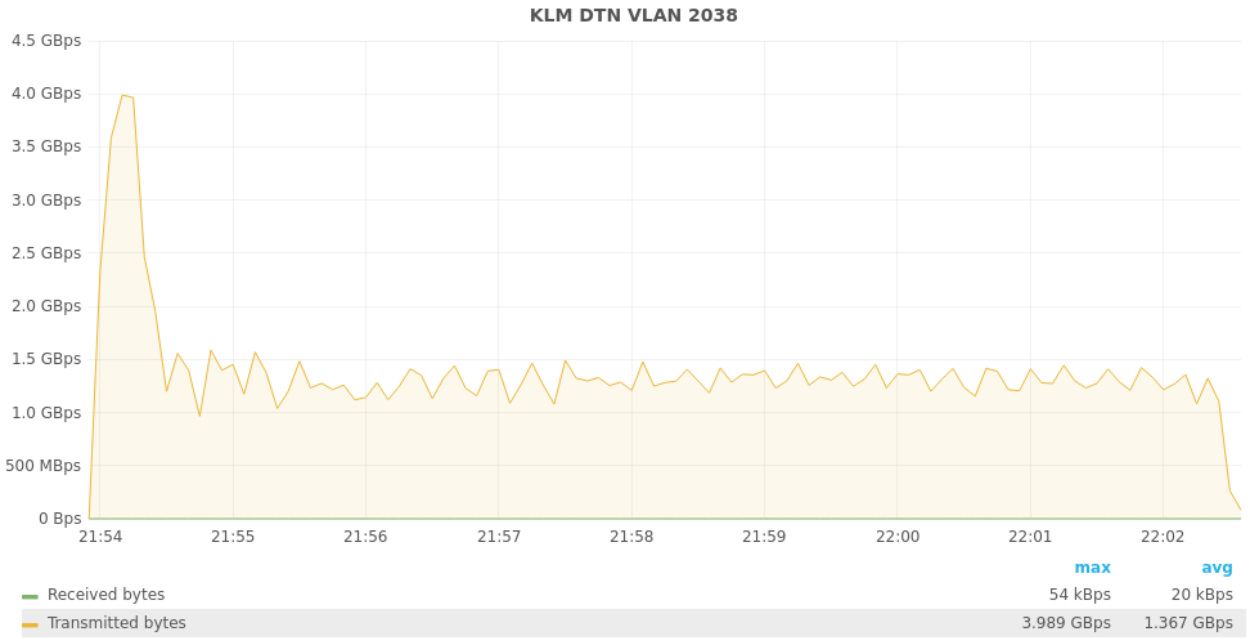


Figure 9: 100GB experiment (mdtmFTP with Direct I/O)

8.3 LOSF data transfer graphs

Figure 10: LOSF experiment (Globus GridFTP with concurrency level 2)

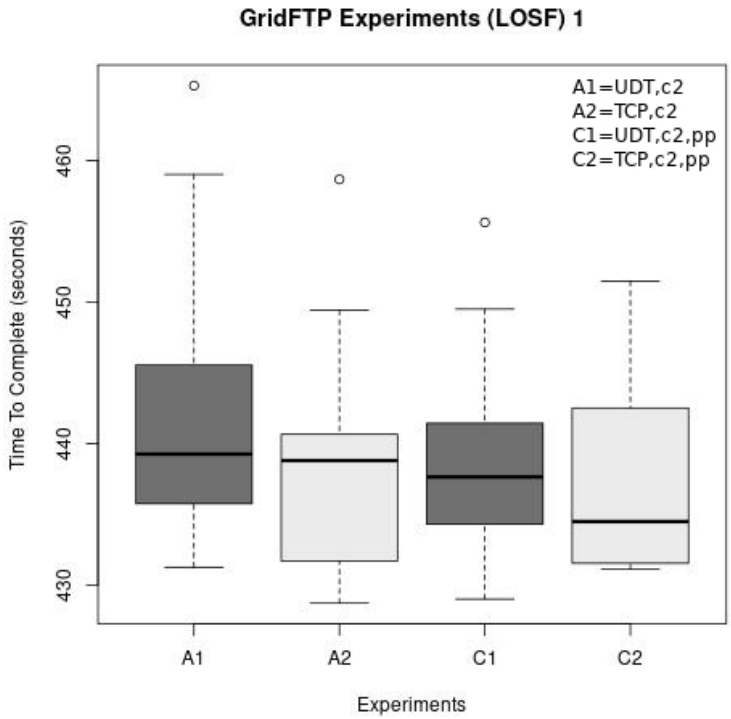


Figure 11: LOSF experiment (Globus GridFTP with concurrency level 4)

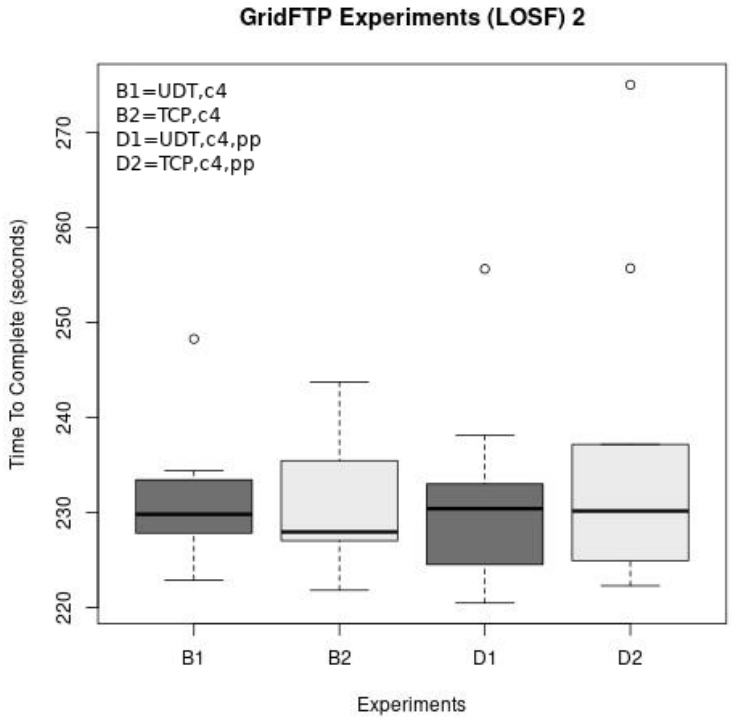
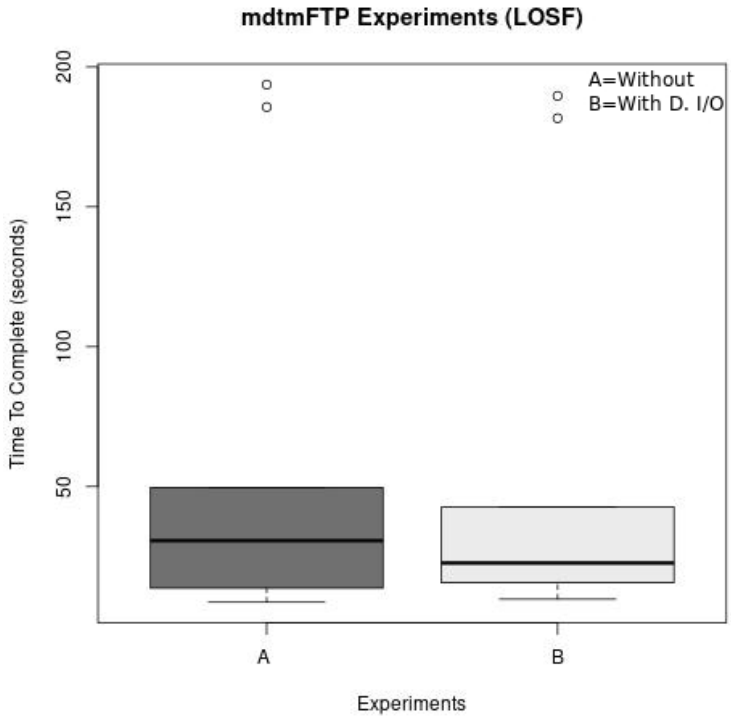


Figure 12: LOSF experiment (mdtmFTP with and without Direct I/O)



8.4 KLM data transfer graphs

Figure 13: KLM experiment (Globus GridFTP with concurrency level 2 and 4)

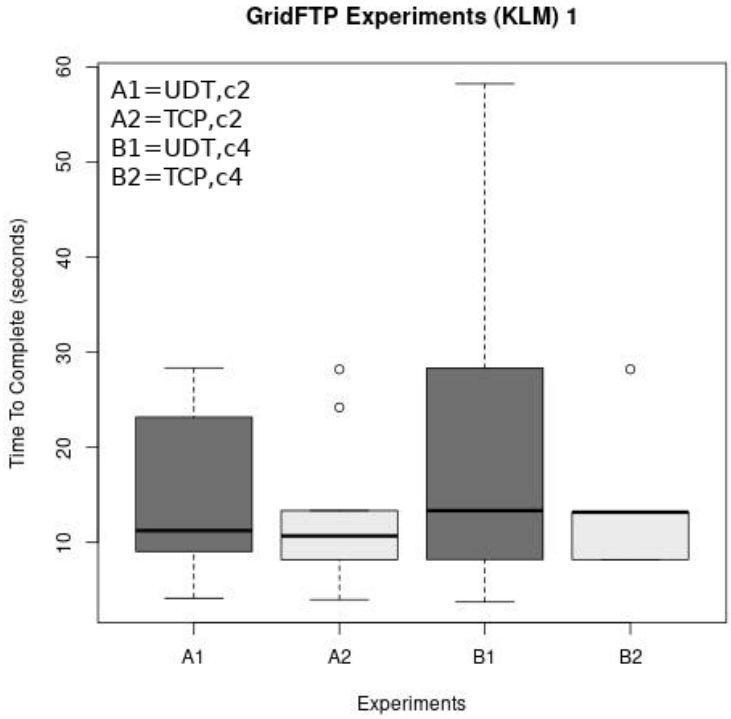


Figure 14: KLM experiment (Globus GridFTP with pipelining)

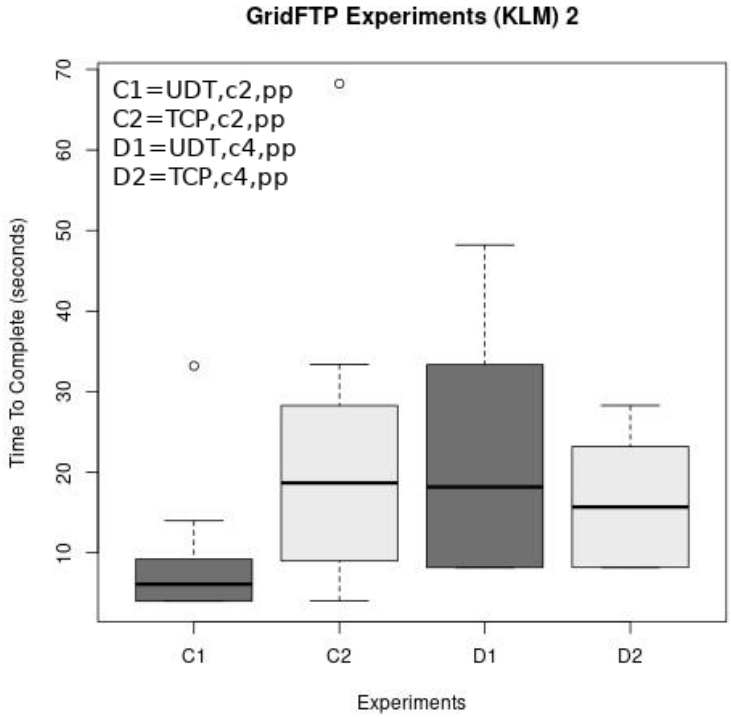
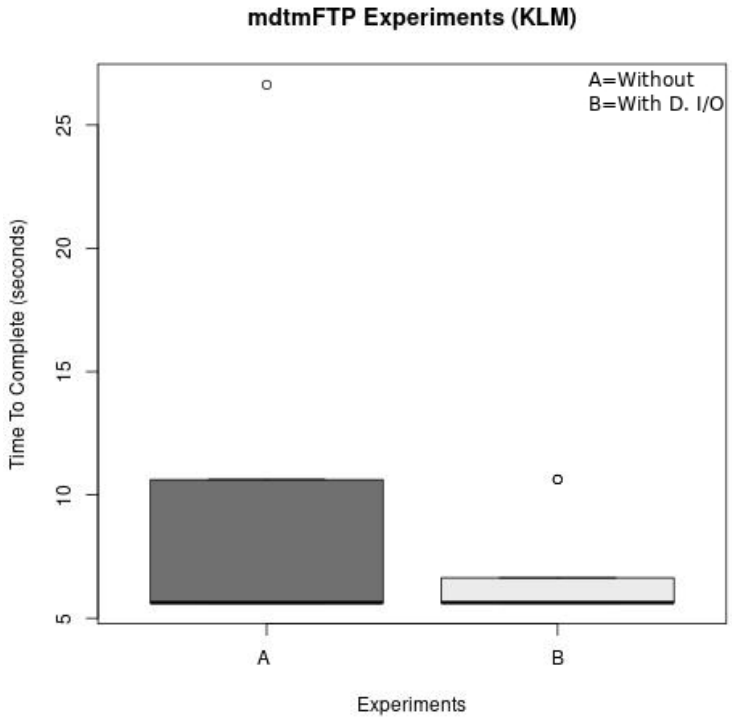


Figure 15: KLM experiment (mdtmFTP with and without Direct I/O)



References

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

[2] N. Manchanda and K. Anand, "Non-uniform memory access (numa)," *New York University*, vol. 4, 2010.

[3] D. An, J. Park, G. Wang, and G. Cho, "An adaptive UDT congestion control method with reflecting of the network status," in *Information Networking (ICOIN), 2012 International Conference on*. IEEE, 2012, pp. 492 – 496.

[4] (2018) What is mdtmFTP. Date visited; February 7th 2018. [Online]. Available: http://mdtm.fnal.gov/

[5] GridFTP key concepts. Date visited; February 7th 2018. [Online]. Available: http://toolkit.globus.org/toolkit/docs/latest-stable/gridftp/key/

[6] R. L. Grossman. (2010) UDT: A Transport Protocol for Data Intensive Applications. Date visited; February 7th 2018. [Online]. Available: https://tools.ietf.org/html/draft-gg-udt-03

[7] (2016) MRO Survey 2016. Date visited; January 30th 2018. [Online]. Available: http://www.oliverwyman.com/our-expertise/insights/2016/apr/mro-survey-2016.html#.VwOuAU3fPDd

[8] Netherlight. Date visited; February 11th 2018. [Online]. Available: https://www.surf.nl/en/services-and-products/netherlight/index.html

[9] J. Bresnahan, M. Link, R. Kettimuthu, and I. Foster, "UDT as an alternative transport protocol for GridFTP," in *International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT)*, 2009, pp. 21 – 22.

[10] J. Bresnahan, M. Link, R. Kettimuthu, D. Fraser, I. Foster *et al.*, "Gridftp pipelining," in *Proceedings of the 2007 TeraGrid Conference*, 2007.

[11] L. Zhang, W. Wu, P. DeMar, D. Katramatos, and D. Yu, "mdtmFTP: a High-performance Data Transfer Tool in Big Data Era," 2005.

[12] L. Zhang, W. Wu, P. DeMar, and E. Pouyoul, "mdtmFTP and its evaluation on ESNET SDN testbed," *Future Generation Computer Systems*, vol. 79, pp. 199 – 204, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17306489

[13] Globus Toolkit homepage. Date visited; February 7th 2018. [Online]. Available: http://toolkit.globus.org/toolkit/

[14] Y. Gu, X. Hong, and R. Grossman, "An analysis of AIMD algorithms with decreasing increases," *Gridnets 2004*, 2004.

[15] Y. Gu and R. Grossman, "UDTv4: Improvements in performance and usability," in *International Conference on Networks for Grid Applications*. Springer, 2008, pp. 9 – 23.

[16] 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*. IEEE Computer Society, 2005, p. 54.

[17] S. Skiena, "Dijkstra's algorithm," *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, Reading, MA: Addison-Wesley*, pp. 225–227, 1990.

[18] L. Zhang, P. Demar, W. Wu, and B. Kim, "MDTM: Optimizing Data Transfer using Multicore-Aware I/O Scheduling," in *IEEE Local Computer Networks*, no. FERMILAB-PUB-17-151-CD. Fermi National Accelerator Lab.(FNAL), Batavia, IL (United States), 2017.

[19] "Pipe - overview of pipes and fifos," Date visited; February 10th 2018. [Online]. Available: http://man7.org/linux/man-pages/man7/pipe.7.html

[20] iPerf homepage. Date visited; February 8th, 2018. [Online]. Available: https://iperf.fr/

[21] (2018) 40G/100G Tuning. Date visited; February 8th 2018. [Online]. Available: https://fasterdata.es.net/host-tuning/100g-tuning/

[22] Bonnie++ homepage. Date visited; February 8th, 2018. [Online]. Available: https://www.coker.com.au/bonnie++/

[23] M. Jain, R. S. Prasad, and C. Dovrolis, "The tcp bandwidth-delay product revisited: network buffering, cross traffic, and socket buffer auto-sizing," Georgia Institute of Technology, Tech. Rep., 2003.

[24] GIMP git repository. Check out commit used (February 8th 2018): 580ed29fd2262722c9e749c0755566db0de346f1. [Online]. Available: https://git.gnome.org/browse/gimp

[25] Collectd homepage. Date visited; February 8th 2018. [Online]. Available: https://collectd.org/

[26] Effects of parallel streams. Date visited; February 9th 2018. [Online]. Available: http://toolkit.globus.org/toolkit/docs/5.2/5.2.4/gridftp/user/#globus-url-copy-parallelismvalue

[27] E. Yildirim, J. Kim, and T. Kosar, "How gridftp pipelining, parallelism and concurrency work: A guide for optimizing large dataset transfers," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, Nov 2012, pp. 506–515.

[28] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Queue*, vol. 14, no. 5, p. 50, 2016.

[29] BBR congestion control. Date visited; February 10th 2018. [Online]. Available: https://lwn.net/Articles/701165/