Improving File Transfer Service — FTS3

September 2015

Author:
Hamza Zafar
Email: hamza.zafar@cern.ch

Supervisor(s):
Oliver Keeble
Alejandro Alvarez

CERN openlab Summer Student Report 2015

Project Specification

The Experiments at CERN generate colossal amount of data. The data centre stores it and sends it around the world for analysis. In the first run of LHC, 30 petabytes of data was produced annually, larger amounts of data are expected to be produced during the second run of LHC [**1**]. To store and process this data, CERN relies on a grid infrastructure known as WLCG (Worldwide LHC Computing Grid), which consists of 170 collaborating computing centres in 42 countries. One of the major challenges at CERN is to globally replicate and distribute the data coming colliders across the WLCG infrastructure. To address this problem, a file transfer service (FTS3) is developed at CERN for bulk transfers of physics data. In this manner, real-time LHC data is not only distributed and replicated across WLCG, but also made available to a community of ~8000 physicist around the globe.

Improving the file transfer service projects is geared towards effectively utilizing the available networks resources as well as introducing new algorithms in FTS3 scheduler to make intelligent decisions for scheduling file transfers.

Abstract

This project deals with two aspects of improving the file transfer service – FTS3. The first one is the selection of best source site for file transfers. Since files are replicated at different sites, the selection of the best source site based on the networks throughput and success rate can have a major impact on FTS3. The second one is maximizing the file throughput across WLCG network by increasing the TCP buffer sizes. TCP is the only transport layer protocol used widely for data transfers; it was originally designed with focus on reliability and long-term fairness. In high bandwidth networks, the system administrators have to manually optimize/tune the TCP configurations. Some of these configurations have a major impact on throughput. TCP buffer size is one such setting, which sets a limit on TCP congestion window size. With the release of Linux Kernel 2.6, a new feature "Linux TCP Auto-Tuning" was introduced, which selects the optimal TCP buffer sizes based on system resource usage. Another way to increase the TCP buffer size is to use setsocketopts system call. Since FTS3 implements gridFTP protocol, it gives us the flexibility to set TCP buffer sizes manually. This project evaluates the pros and cons of different techniques for setting TCP buffer sizes.

Keywords: FTS3, TCP , Linux TCP Auto-Tuning

Table of Contents

Contents

List of Figures

1 Introduction

FTS3 [**2**] is one of the projects critical for the data management at CERN [**3**]. It is the major service for distributing the majority of LHC [**4**] data across WLCG [**5**] infrastructure. It provides reliable bulk transfers of files from one WLCG site to another, while allowing participating sites to control the network resource usage. FTS3 is a mature service, running for more than 2 years at CERN.

1.1 WLCG Architecture

WLCG stands for WorldWide LHC Computing Grid; it is a collaboration of more than 170 computing centres in 42 countries. The mission of WLCG is to store, analyse and replicate LHC data. Figure 1 shows the architecture of WLCG, it consist of three tiers 0, 1 and 2. These tiers are made up of several computing centres. Tier 0 is the CERN's datacentre, which is connected to 13 tier 1 sites with 10Gbps links. Tier 2 sites are connected using 1Gbps links. FTS3 service plays a vital role in moving data across this complex mesh of computing centres.

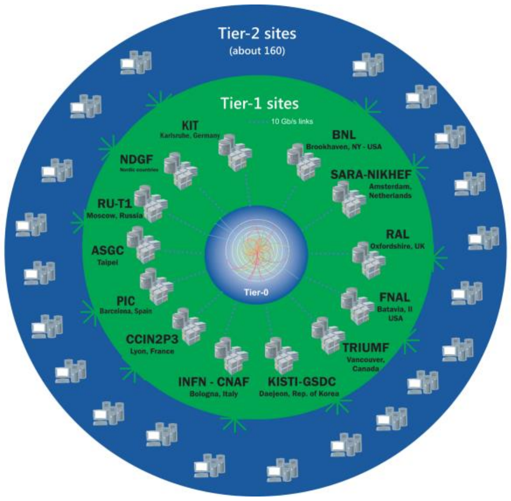


Figure 1: Tiers in WLCG

1.2 Use Cases of FTS3

- An individual or small team can access the web interface to FTS to schedule transfers between storage systems. They can browse the contents of the storage, invoke and manage transfers, and leave FTS to do the rest.
- A team's data manager can use the FTS command line interface to schedule bulk transfers between storage systems.
- A data manager can install an FTS service for local users. The service is equipped with advanced monitoring and debugging capabilities which enable her to give support to her users.
- Maintainers of frameworks which provide higher level functionality can delegate responsibility for transfer management to FTS by integrating it using the various programming interfaces available, including a REST API. The users thus continue to use a familiar interface while profiting from the power of FTS transfer management.

1.3 FTS3 @ CERN

Four major experiments at CERN are ATLAS, CMS, LHC(b) and ALICE. The first three experiments use FTS for file transfer purposes. The figure 2 shows the general flow for using FTS3 service. The clients --- CMS, ATLAS and LHC(b) --- send file transfer requests to FTS3, gridFTP [**6**] protocol is used by FTS3 to initiate third party transfers on storage endpoints. FTS3 supervises the transfers between the storage endpoints and finally archives the job status. *fts-transfer-status* command provided by FTS3 command line interface can be used by to inquire about the job status. Alternatively, clients can also use the FTS3 web interface for transfer management and monitoring. On average, FTS3 transfers 15 petabytes of data per month.

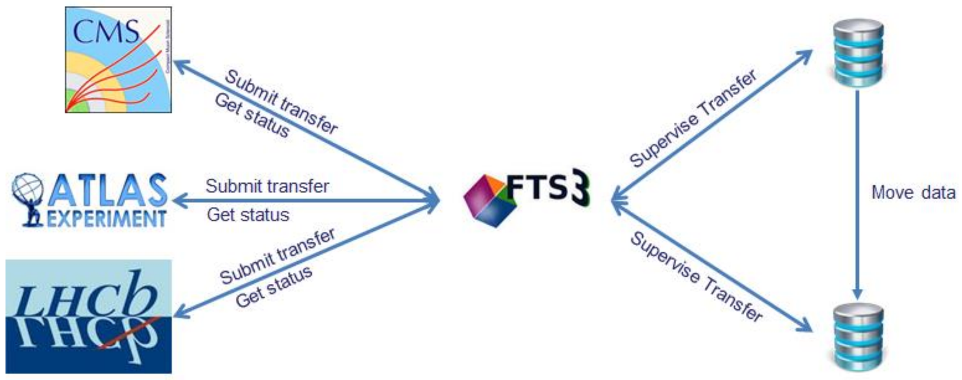
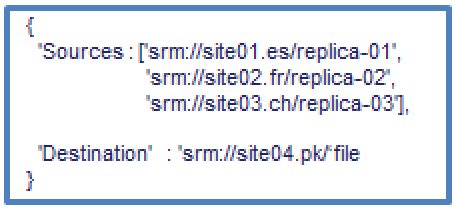


Figure 2: FTS3 workflow

2 Improving the FTS3 Scheduler

To handle the incoming transfer jobs, FTS3 maintains a separate queue for each link between two WLCG endpoints. Files are usually replicated across different sites in WLCG. If a client --- Atlas, CMS, LHC(b) --- submits request for transferring a file with multiple replicas, then FTS3 scheduler is responsible for selecting the best source site. The figure 3 shows a transfer request specifying source sites for multiple replicas.



```
{
  'Sources : ['srm://site01.es/replica-01',
              'srm://site02.fr/replica-02',
              'srm://site03.ch/replica-03'],

  'Destination'  : 'srm://site04.pk/'file
}
```

Figure 3: Transfer request specifying multiple replicas

2.1 Current Scheduling Behaviour for multiple replicas

For each site, FTS3 database maintains the count of pending files in the queue. It also contains information about the throughput and success rate from previous transfers. In order to choose the best replica, FTS3 scheduler queries the number of pending files. The site with the minimum number of pending files in the queue is chosen as a source site. This approach is not efficient because the factors like throughput and success rate are completely ignored.

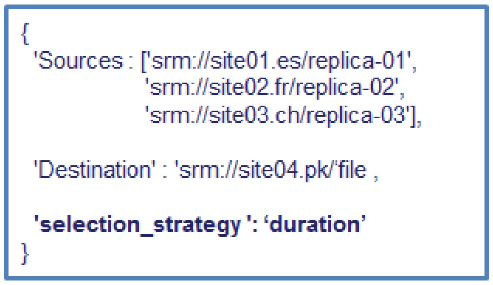
2.2 New Scheduling Behaviour for multiple replicas

We have developed a number of algorithms to address the short comings of FTS3 scheduling decisions for multiple replicas. In addition to the number of pending files in the queue, the new algorithms also consider the throughput and success rate when making a scheduling decision. The algorithms are listed in table below.

Algorithm	Selection Criteria
orderly	Selects the first site mentioned in transfer request
queue / auto	Selects the source site with the least number of pending files
success	Selects the source site with highest success rate
throughput	Selects the source site with highest total throughput
file-throughput	Selects the source site with highest per file throughput
pending-data	Selects the source site with the minimum amount of pending data
waiting-time	Selects the source site with the earliest waiting time
waiting-time-with-error	Selects the source site with the earliest waiting time with error
duration	Selects the source site with the earliest finish time

Figure 4: List of scheduling algorithms with their selection criteria

FTS3 clients can now mention their selection strategy in the transfer request. In this way clients can control the behaviour of FTS3 scheduler. The figure 5 shows a transfer request specifying the selection strategy.

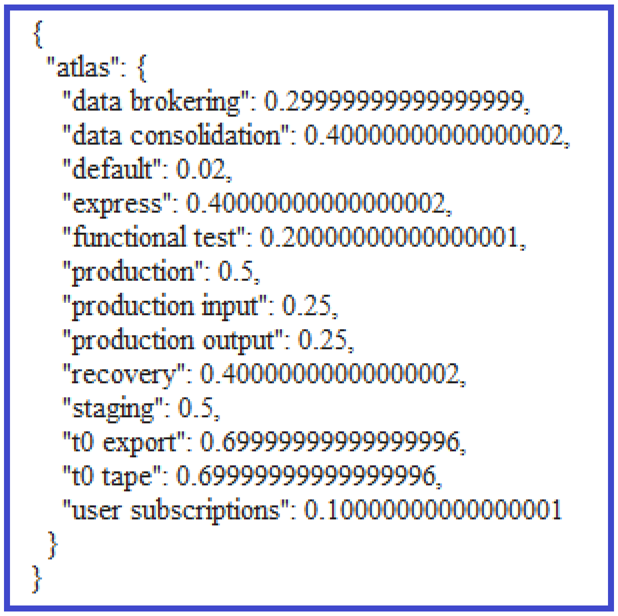


```
{
  'Sources : ['srm://site01.es/replica-01',
               'srm://site02.fr/replica-02',
               'srm://site03.ch/replica-03'],

  'Destination' : 'srm://site04.pk/'file ,

  'selection_strategy ': 'duration'
}
```

Figure 5: Transfer request specifying selection strategy

- **Calculation of Pending Data:**

  FTS3 maintains the priority configuration for client's activities. Figure 6 shows the configurations for Atlas. If a newly submitted job has higher priority than the jobs waiting in queue, then it takes precedence and the transfer is started immediately. Therefore, in order to calculate the amount of pending data in queue, we aggregate the amount of data for all jobs with priorities greater or equal the priority of newly submitted job.



```
{
 "atlas": {
  "data brokering": 0.29999999999999999,
  "data consolidation": 0.40000000000000002,
  "default": 0.02,
  "express": 0.40000000000000002,
  "functional test": 0.20000000000000001,
  "production": 0.5,
  "production input": 0.25,
  "production output": 0.25,
  "recovery": 0.40000000000000002,
  "staging": 0.5,
  "t0 export": 0.69999999999999996,
  "t0 tape": 0.69999999999999996,
  "user subscriptions": 0.10000000000000001
 }
}
```

Figure 6: Activity priorities for ATLAS

- **Calculation of waiting time:**
  Waiting time is defined as the time a transfer request spends in the FTS3 queue. The formula for calculating waiting time is as follow:

$$Waiting\ Time = \frac{Pending\ Data}{Throughput}$$

- **Calculation of waiting time with error:**
  Error is the predicted amount of data that should be resent in case of transfer failures. It is as follow:

$$Failure\ Rate = 100 - Success\ Rate$$

$$Error = \ Pending\ Data * \frac{Failure\ Rate}{100}$$

$$Waiting\ Time\ With\ Error = \frac{Pending\ Data + Error}{Throughput}$$

- **Calculation of Finish Time:**
  The algorithm named *"duration"* ranks the source sites based on total finish time. Finish time is defined as the time it takes to complete the file transfer; it also includes the waiting time in FTS3 queue.

$$Finish\ Time = Waiting\ Time\ With\ Error + \frac{Submitted\ File\ Size}{Per\ File\ Throughput}$$

2.3 Caching Database Queries

The new FTS3 scheduler depends on the number of pending files; throughput and success rate information stored in FTS3 database, so it makes a lot of queries and eventually increases the load on database. To overcome this problem, we have implemented a caching layer between FTS3 scheduler and database. In order to maximize the performance we maintain a separate cache for each thread. A cache entry expiration timer is configured for each cache entry. Default time for cache entry expiration is 5 minutes. We also ended up with situations when cache had a large number of expired entries. Therefore, to free the memory, we added a cache clean-up timer. When the cache clean-up timer expires, FTS3 caching module deletes all the expired entries from cache. It should be noted here that the maximum memory the cache could consume is estimated to be less than a megabyte. The default time for cache clean-up timer is 30 minutes. In the future, distributed memory caching e.g memcached [**7**] and Redis cache [**8**] can also be integrated in the caching layer.

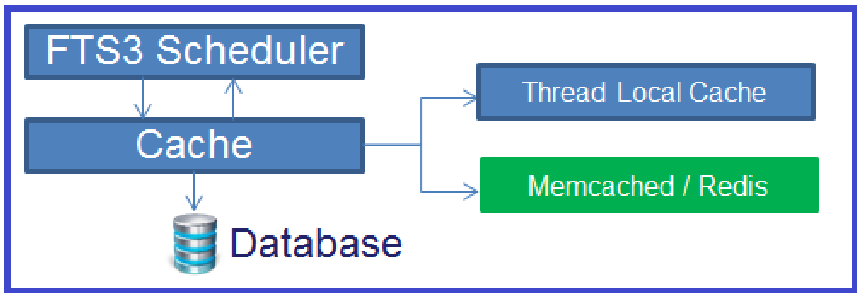


Figure 7: FTS3 caching layer

3 Effect of TCP configurations on throughput

TCP is the only transport layer protocol used widely for data transfers; it was originally designed with focus on reliability and long-term fairness. In high bandwidth networks, the system administrators have to manually optimize/tune the TCP configurations. Some of these configurations have a major impact on throughput. TCP buffer size is one such setting, which limits the size of TCP congestion window.

Since computer centres in WLCG infrastructure are linked with high bandwidth and high latency networks. We are focused on effectively utilizing the available network resources. Our end goal is to increase the throughput for FTS3 file transfers. In this section we compare and contrast impact of different TCP tuning methods on the throughput of FTS3 file transfers.

3.1 TCP Optimal Buffer Size

TCP maintains a *"congestion window"* to determine how many packets can be sent at one time. This implies that larger the size of congestion window, higher the throughput. The kernel enforces a limit on the maximum size of TCP congestion window. By default, on most Linux distributions, the maximum limit is 4MB, which is still very small for high bandwidth links. System admins can edit the /etc/sysctl.conf file change the default settings. The optimal TCP buffer size can be calculated using the following formula

$$Optimal\ Buffer\ Size = 2 * Bandwidth * delay$$

We can use the ping command to calculate the delay between two endpoints. Since ping command returns the Round Trip Time (RTT), the above formula can be reduced to:

$$Optimal\ Buffer\ Size = Bandwidth * RTT$$

For example, if the ping time is 200ms and the network bandwidth is 1Gbps, then the optimal buffer is 25.6MB which is way larger than the default settings.

$$Optimal\ Buffer\ Size = \frac{1\ Gbit}{8\ bits} * 0.200 sec = 25.6\ MB$$

System admins have to add/modify the following setting in /etc/sysctl.conf file:

```
# increase max memory for sockets to 32MB
net.core.rmem_max = 33554432
net.core.wmem_max = 33554432
# increase Linux autotuning TCP buffer limit to 32MB
net.ipv4.tcp_rmem = 4096 87380 33554432
net.ipv4.tcp_wmem = 4096 65536 33554432
```

Now there are two ways to make use of the increased TCP buffer sizes:

- Linux TCP Auto-Tuning:
  Linux TCP auto-tuning automatically adjusts the socket buffer sizes to balance the TCP performance and system's memory usage. TCP auto-tuning is enabled by default in Linux release after version 2.6.6 and 2.4.16. For Linux auto-tuning the maximum send and receive buffer limit is specified by *net.ipv4.tcp_wmem* and *net.ipv4.tcp_rmem* parameters respectively.

- Manually setting the socket buffer sizes:
  Application programmers can mention the socket buffer sizes using the setsocketopts system call. *net.core.wmem_max* and *net.core.rmem_max* parameters impose an upper limit on the amount of memory requested for send and receive buffers respectively. The Linux kernel allocates double the amount of memory requested. It should be noted here that setting the socket buffer sizes manually disables the Linux auto-tuning.

3.2 Performance Evaluation of FTS3 transfers with and without Linux auto-tuning

During our initial testing, we transferred files from CERN to Australia. Our calculations for the optimal buffer sizes suggested a 37 MB buffer, whereas the configured maximum TCP buffer size on our system was 4MB. Therefore, we increased the maximum buffer limit to 37 MB and calculated the throughput. The Figure 8 shows a plot of throughput with system configured with a 4MB and a 37 MB TCP buffer. It should be noted here that the buffer sizes are not passed to *fts-transfer-submit* (buffer sizes are not set using setsocketopt system call) command, in fact TCP auto-tuning is taking care of socket memory allocation. It is evident that increasing the default limits on maximum TCP buffer sizes, the congestion window can open more, which results in higher throughput.

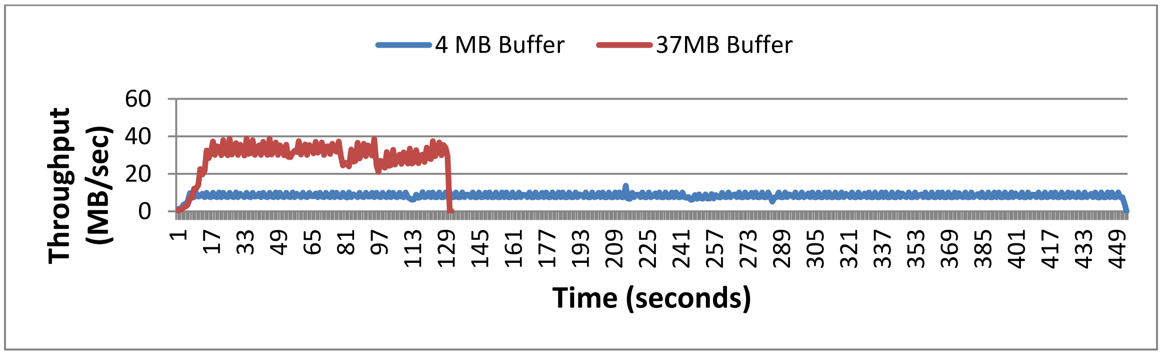


Figure 8: Effect of large buffer sizes on throughput

Now the question arises, should FTS3 rely on Linux auto-tuning or the users should pass the buffer size to *fts-transfer-submit* command?

To answer this question, we transferred files from CERN to Tokyo. The number of streams was set to 1. The endpoints at both sites were configured with 32 MB optimal buffer size. Receiver's advertised window and CNWD sizes were recorded from tcpdump and Linux ss utility respectively. Figure 9 shows the results when Linux auto-tuning is in action and Figure 10 shows the results by passing the buffer size to *fts-transfer-submit* command. When we manually set the buffer sizes, Linux allocates twice the amount requested. Therefore, for a fair comparison with auto-tuning, we request half the amount of memory available for Linux auto tuning i.e 32MB.

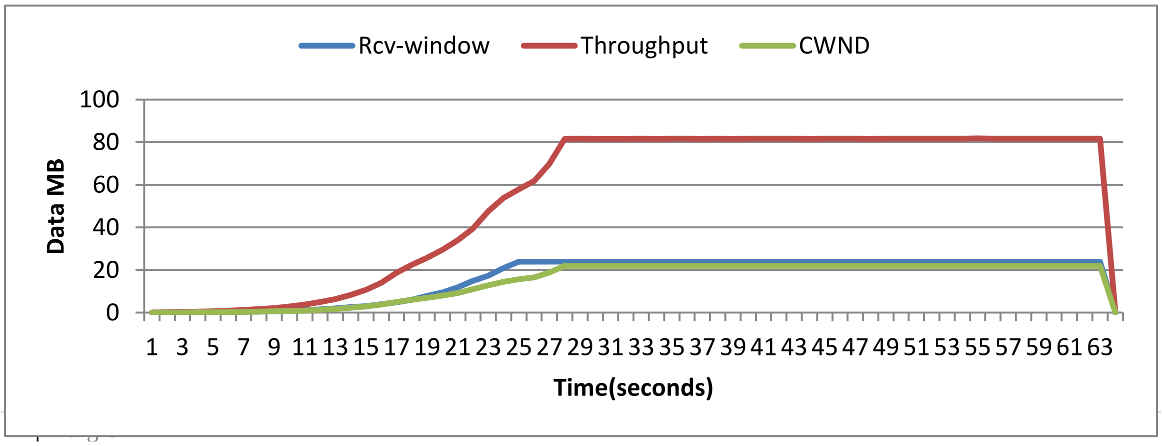


Figure 9: Graph for file transfer when Linux auto-tuning is in action

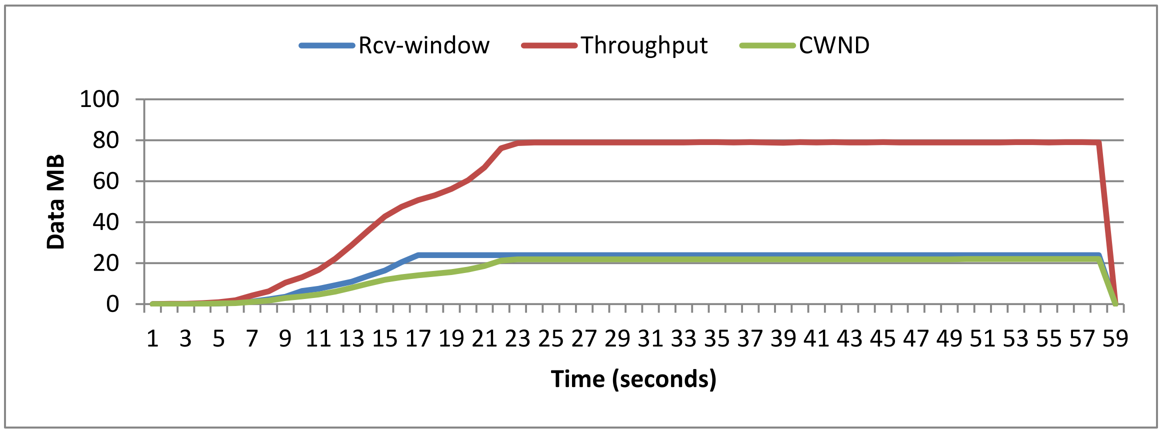


Figure 10: Graph for file transfer when manually setting a 16MB Buffer (32 MB allocated)

With Linux auto-tuning the transfer time is 63 seconds, whereas the transfer time by manually setting the buffer sizes is 59 seconds. Since both transfers reach the same maximum throughput, we can conclude that manually setting the buffer sizes has no advantage over Linux auto-tuning. Auto-Tuning is a much safer option to use as compared to manually setting the buffer sizes because it can dynamically resize the TCP buffers based on network performance and system resource usage.

We now shift our focus on comparing the effect of using multiple streams (with and without auto-tuning). We transferred files from CERN to Tokyo with multiple numbers of streams. Figure 11 shows a graph when Linux auto-tuning is in action. Figure 12 shows the graph of transfers when we are manually setting buffer sizes to 32MB. If we compare the file transfer with 1 stream (auto-tuning vs manually setting buffer), we achieve higher throughput for manually setting buffer. This is due to the fact that auto-tuning can increase the CNWD up to 32MB whereas when setting the buffer size to 32MB , kernel allocates 64MB, then CNWD can increase up to 64MB. For 2 and 4 number of streams the throughput is almost the same (with and without auto-tuning). It is also evident from the graphs that increasing the number of streams fills the pipe more quickly.

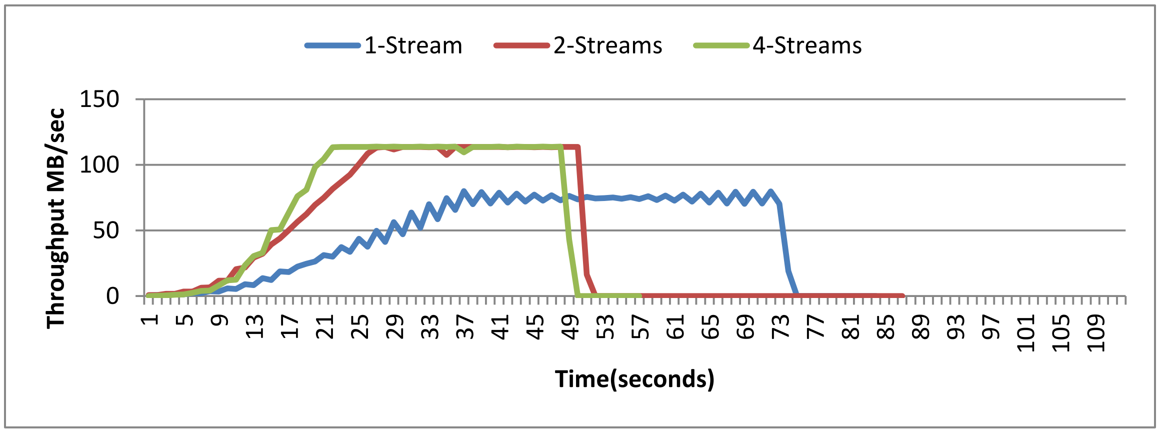


Figure 11: Graph for file transfers when Linux auto-tuning is in action

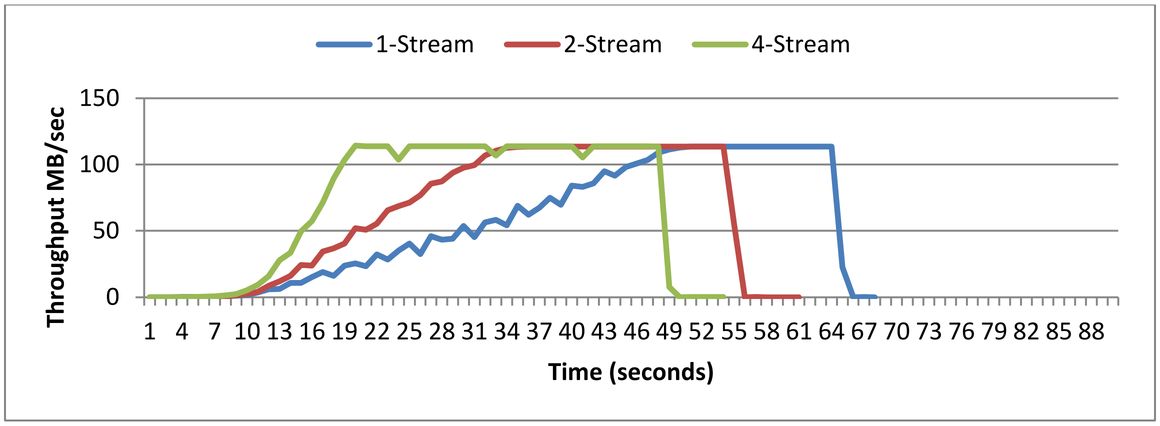


Figure 12: Graph for file transfers when manually setting a 32MB Buffer (64 MB allocated)

With this work we conclude that the operating system's configured maximum buffer sizes are too small for WLCG's high bandwidth network, the kernel enforced limits on TCP buffer sizes should be increased. Since, FTS3 supports 3rd party file transfers, there is currently no mechanism to get the RTTs by remotely pinging two storage endpoints, and hence there is no possibility of calculating the bandwidth delay product, unless we have the access to the storage endpoint. In the future, we would be able to get the RTT information from WLCG perfSONAR project [**9**]. We have also seen that there is no difference on throughput whether we use Linux auto-tuning or set the buffer sizes explicitly. All we have to do is to increase the maximum TCP buffer size on storage endpoints and let the Linux auto-tuning decide optimal buffer sizes.

Future work includes the support of distributed memory caching techniques for FTS3 caching layer and the performance evaluation of single file transfer with multiple streams vs. multiple file transfers with single stream.

4 Bibliography

[1] LHC Season 2. [Online]. http://home.web.cern.ch/about/updates/2015/06/lhc-season-2-first-physics-13-tev-start-tomorrow

[2] M Salichos, M K Simon, O Keeble A A Ayllon, "FTS - New Data Movement Service for WLCG".

[3] CERN. [Online]. http://home.web.cern.ch/

[4] Large Hadron Collider. [Online]. http://home.web.cern.ch/topics/large-hadron-collider

[5] WorldWide LHC Computing Grid. [Online]. wlcg.web.cern.ch

[6] gridFTP. [Online]. https://en.wikipedia.org/wiki/GridFTP

[7] Memchached. [Online]. http://memcached.org/

[8] Redis. [Online]. http://redis.io/

[9] WLCG perSONAR. [Online]. http://maddash.aglt2.org/maddash-webui/