Integration of Network Performance Monitoring Data at FTS3

July-August 2013

Author:
Rocío Rama Ballesteros

Supervisor(s):
Michail Salichos
Alejandro Álvarez

CERN openlab Summer Student Report 2013

Project Specification

The main goal of this project is to optimize the tcp buffer size to make more efficient the file transfers with FTS3. The library that has been implemented provides a way to calculate this providing a source and a destination. This way, whoever is transferring the files does not have to know anything about the logic of how calculate it.

In this project, I have done a library to make easy the access to PerfSONAR’s information between two hosts, calculating the optimized tcp buffer size and thereby to making more efficient the transfer of files.

As part of my work, I have also tested the library to check if it actually improved the transfer throughput with tools as GridFTP and Globus.

Acknowledgements

I would like to express my gratitude to my supervisor Michail Salichos, who was solving always my questions, although he was on holidays. Thanks also to Michal Kamil Simon, and special thanks to Alejandro Álvarez, who was taking care of my work during my period as OpenLab Summer Student like he was my supervisor as well. It was a pleasure working with the FTS team.

And of course, thanks to the entire section, because they were always helping me, during almost 2 months, when I needed it and they had always time to listen my presentations or to take a coffee and talk not just about work.

I hope one day they see the results of using this library.

Abstract

The File Transfer Service (FTS) manages the distribution of data coming mainly from the Large Hadron Collider (LHC) across the global computing grid. Recent developments in the monitoring of the infrastructure offer opportunities for optimising the efficiency of transfers based on an improved knowledge of network characteristics and state. The project is centred on integrating appropriate network monitoring data into FTS's transfer scheduling and parameterization.

Contents

1 Introduction

Recent developments in the monitoring of the infrastructure offer opportunities for optimizing the efficiency of transfers based on improved knowledge of network characteristics and state.

Nowadays, TCP is the main transfer protocol used, also in FTS3, and because of this, we need it to work well.

The maximum achievable throughput for a single TCP connection is determined by different factors. One trivial limitation is the maximum bandwidth of the slowest link in the path. But there are also other, less obvious limits for TCP throughput.

One of the causes of poor TCP performance is incorrect configuration parameters of the host TCP implementation on the data transfer system.

The appropriate configuration of TCP on data transfer nodes for very long round trip (RTT) (high latency) paths can be accomplished by competent system administrators with the help of public knowledge base sites.

It is critical to use the optimal TCP send and receive socket buffer sizes for the RTT of the path that the applications see end-to-end. The default maximum operative system TCP buffer sizes are too small for today’s high speed networks. Until around 8 years ago, default TCP send/receive buffers were typically 64 KB, however the buffer size needed to fill. [1]

2 Optimizing TCP buffer size

In this project we want to integrate appropriate network monitoring data into FTS’s transfer scheduling and parameterization. Giving an optimized tcp buffer size, in order to make more efficient the transfers files, is one of the goals.

Before getting into details, it is necessary to give an introduction about the project’s ecosystem.

2.1 FTS

FTS (File Transfer Service) is a transfer job scheduler for scientific experiments producing, analyzing and storing high amount of data, used mainly in CERN/LHC experiments. It is the services responsible for distributing the majority of LHC data across the WLCG infrastructure, transferring 25PB in 2012.

The users have the following properties:

- Produce and analyze lots of data, in petabyte scale.
- Replicate data between different sites.
- Data storage a network infrastructure is heterogeneous.[2]

2.2 PerfSONAR

PerfSONAR (Performance Service Oriented Network monitoring Architecture) is an infrastructure for network performance monitoring, making it easier to solve end-to-end performance problems on paths crossing several networks. It contains a set of services delivering performance measurements in a federated environment. These services act as an intermediate layer, between the performance measurement tools and the diagnostic or visualization applications. This layer is aimed at making and exchanging performance measurements between networks, using well-defined protocols. [3]

PerfSONAR makes possible to automate monitoring data exchange between networks, to simplify troubleshooting performance problems occurring between sites connected through several networks. It can collect both passive and active network measures, convert these to standard format and publish the data where it is publically accessible. Thanks to this, now is possible gather the information that is necessary to calculate the optimize tcp buffer size.

In addition, it is open, and any tool can take advantage of it.

2.3 How optimize the TCP buffer size

If there is no network congestion or packet loss, network throughput is directly related to TCP buffer size and the network latency. Network latency is the amount of time for a packet to traverse the network. [4]

Most networking experts agree that the optimal tcp buffer size is twice the bandwidth*delay product of the link. The *ping* program will give us the round trip time (RTT) for the network link, which is twice the delay, so the formula simplifies to:

$$BufferSize\ (KB) = \frac{Bandwidth(Mbs) * RTT(ms)}{8}$$

In PerfSonar, we have the information about the round trip time. Therefore we can use this formula instead of the previous one.

Example:

If the round trip time is 50ms, and the end-to-end network consists of all 2G or 20G Ethernet, the TCP buffers should be:

0.05 sec * (2Gbit/8) = 12.5 Mbytes

If you know the TCP window size and the round trip latency you can calculate the maximum possible throughput of a data transfer between two hosts, regardless of how much bandwidth you have.

3 Implementation

The library has been implemented following the next ideas:

- Need to access to *PerfSONAR* and parse the json's file that is obtained.
- Get the information from the json's file that is needed to calculate the optimize tcp buffer size.
- Consider the possibility to add new source of information that can impact the calculation of the tcp buffer size.

The json's format has the following structure:

```
{
    "summary": "PS_CHECK_THROUGHPUT OK - Average throughput is 0.3086419Gbps ",
    "id": "1648",
    "monitor": "lhcmon.bnl.gov",
    "source": "lhcmon.bnl.gov",
    "status": 0,
    "lastCheckTime": "2013-07-16 11:12:17.0",
    "parameters": {
        "Throughput Avg": {
            "unit": "Mb/s",
            "description": "Throughput Avg",
            "value": 0.308642
        },
        "Throughput Max": {
            "unit": "Mb/s",
            "description": "Throughput Max",
            "value": 0.738979
        },
        "Throughput Min": {
            "unit": "Mb/s",
            "description": "Throughput Min",
            "value": 0.0840497
        },
        "Sigma": {
            "unit": "Mb/s",
            "description": "Sigma",
            "value": 0
        }
    },
    "destination": "psmsu02.aglt2.org"
}
```

To calculate the tcp buffer size, it is important check the summary. If it is: *"PS_CHECK_THROUGHPUT OK"*, we need to save the information. Other messages like "*PS_CHECK_THROUGHPUT WARNING*", we do not take into consideration because we just need the success throughput between the two hosts.

PerfSONAR offers us a lot of information about the connection between two hosts: summary, id, monitor, source, status, lastCheckTime, destination and parameters (throughput average, maximum, minimum and sigma), but the information that we need to calculate depends on the throughput and the source and destination.

Additionally, is interesting to distinguish between "*pull files*" and "*push files*":

- Pull, is when the destination gets data from the source.
- Push, is when the source sends data to the destination.

In the json's file, if the monitor and the source are the same, is pushing data; if the monitor and the source are different, is pulling data.

Example pushing data:

```
"monitor":"psmsu02.aglt2.org",
"source": "lhcmon.bnl.gov",
```

Example pulling data:

```
"monitor":"lhcmon.bnl.gov",
"source": "lhcmon.bnl.gov",
```

The next diagram shows an overview of the library.

- *ParserInfoPerfSonar* and *PerfSonarJson* are classes implemented to parse the json's file that is retrieved from PerfSONAR.

The information is read using the library *boost*, and is stored in a structure so we can later access to this data. An example is:

```
boost::property_tree::read_json(fileToRead, pt);
BOOST_FOREACH(boost::property_tree::ptree::value_type &v, pt.get_child("root")){
///…….}
```

Moreover, functions to set and to get the data, as: average throughput value, or maximum throughput value, and so on, are available.

In the class *ParserInfoPerfSonar* is possible to get the pushing value or the pulling value, the user will decide what is better for the buffer.

- *INetworkLink* contains the information of the json's file, and store the data needed to calculate the tcp buffer size.

Also, it acts like an interface, letting to add in a future, other classes that can affect the calculation of the tcp buffer size.

To get all the information that PerfSONAR provides us, is enough to know the source and the destination, calling the function *getInfo(source, destination).*

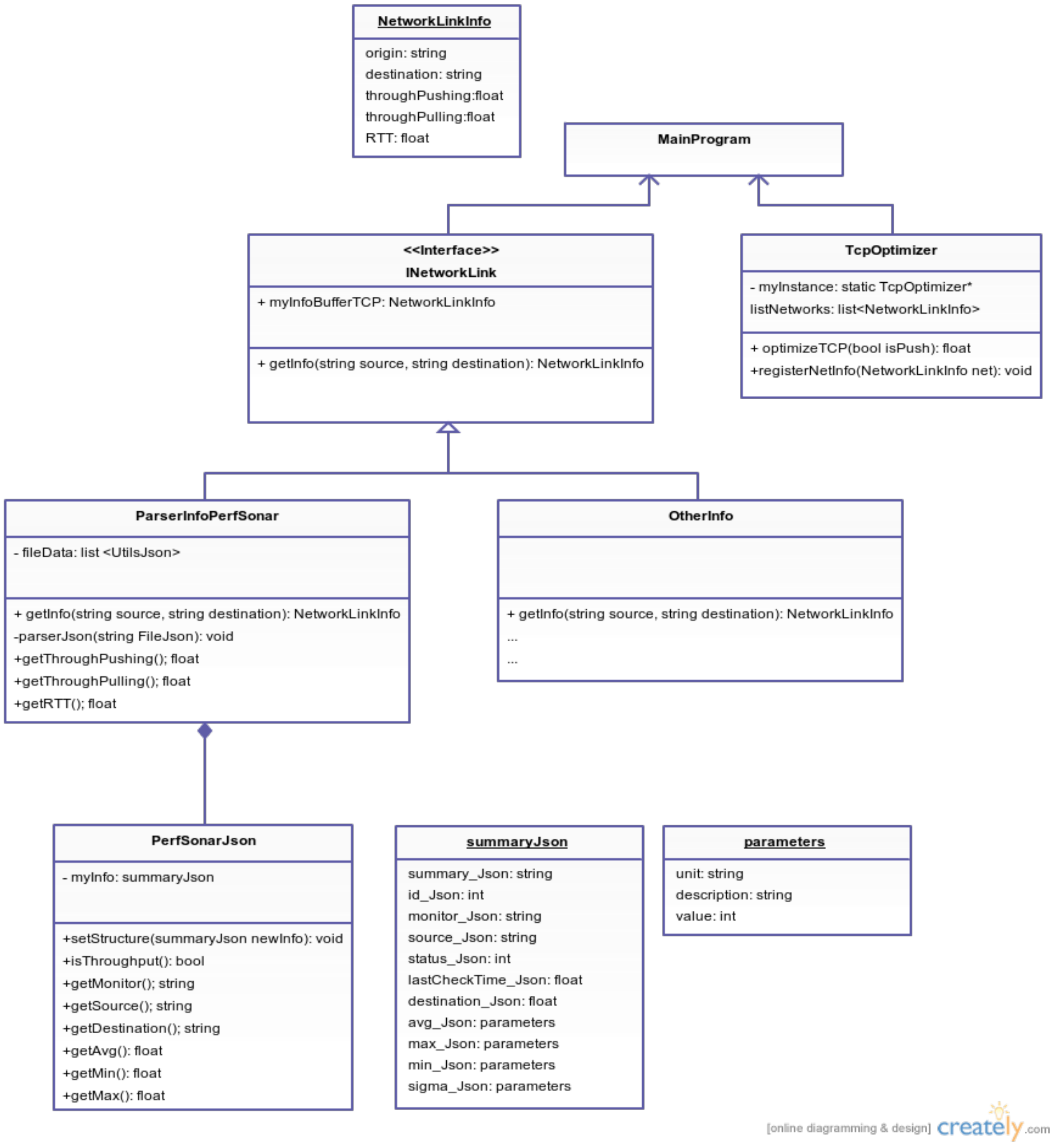
```
getInfo("lhcmon.bnl.gov", "psmsu02.aglt2.org");
```

- *TcpOptimizer* is the class that the main program has to call to calculate the optimized tcp buffer size. Also, it follows a singleton pattern design. In this way, it is registering the different links, and later can calculate the total optimize tcp buffer size.

An example about how to use the functions is:

TcpOptimizer* tcpBuffer; *//Create the variable*
tcpBuffer = TcpOptimizer::getInstance(); *//Get the instance (is a singleton)*
ParserInfoPerfSonar *fileData = new ParserInfoPerfSonar(); *//Create the structure to*
// store the data

struct INetworkLink::NetworkLinkInfo info = fileData->getInfo("lhcmon.bnl.gov", "psmsu02.aglt2.org"); *//Get the data from Perfsonar between two points*
tcpBuffer->registerNetInfo(fileData); *//Register the information*
tcpBuffer->optimizeTCP(false); *//Calculate the tcp buffer size. False if is pulling,*
//True if is pushing


NetworkLinkInfo
origin: string
destination: string
throughPushing:float
throughPulling:float
RTT: float
MainProgram
<<Interface>>
INetworkLink
+ myInfoBufferTCP: NetworkLinkInfo
+ getInfo(string source, string destination): NetworkLinkInfo
TcpOptimizer
- myInstance: static TcpOptimizer*
listNetworks: list<NetworkLinkInfo>
+ optimizeTCP(bool isPush): float
+registerNetInfo(NetworkLinkInfo net): void
ParserInfoPerfSonar
- fileData: list <UtilsJson>
+ getInfo(string source, string destination): NetworkLinkInfo
-parserJson(string FileJson): void
+getThroughPushing(); float
+getThroughPulling(); float
+getRTT(); float
OtherInfo
+ getInfo(string source, string destination): NetworkLinkInfo
...
...
PerfSonarJson
- myInfo: summaryJson
+setStructure(summaryJson newInfo): void
+isThroughput(): bool
+getMonitor(); string
+getSource(); string
+getDestination(); string
+getAvg(): float
+getMin(): float
+getMax(): float
summaryJson
summary_Json: string
id_Json: int
monitor_Json: string
source_Json: string
status_Json: int
lastCheckTime_Json: float
destination_Json: float
avg_Json: parameters
max_Json: parameters
min_Json: parameters
sigma_Json: parameters
parameters
unit: string
description: string
value: int
[online diagramming & design] creately.com

To implement the library it has been considered a singleton pattern to ensure that one and only one object is instantiate for the main class and also, to give us a global point of access.

Other necessary libraries:

- Curl

- Boost: is a set of libraries for the C++ programming language that provide support for tasks and structures such as linear algebra, pseudorandom number generation, multithreading, image processing, regular expressions, and unit testing. It is used to parse the json's file and save the data in a specific structure.

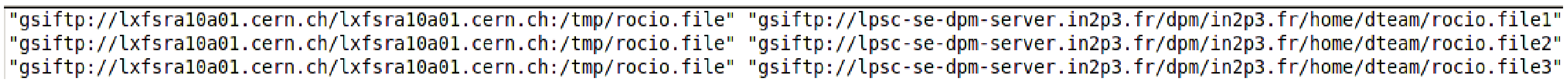
4 Test

The library is not still integrated in FTS3, thus to test the library and check if all the calculations improve the throughput and following the transfer files, we have been testing it using Globus tools and GridFTP protocol (that it is what FTS3 itself uses).

Therefore, the round trip time (RTT) was calculated doing a *ping* from the source to the destination.

We have been doing different tests, transferring several files concurrently (3, 4, 5, 6, 8 and 10 files).

We have created a file, *bulk.txt*, with the same source and different destinations:



```
"gsiftp://lxfsra10a01.cern.ch/lxfsra10a01.cern.ch:/tmp/rocio.file" "gsiftp://lpsc-se-dpm-server.in2p3.fr/dpm/in2p3.fr/home/dteam/rocio.file1"
"gsiftp://lxfsra10a01.cern.ch/lxfsra10a01.cern.ch:/tmp/rocio.file" "gsiftp://lpsc-se-dpm-server.in2p3.fr/dpm/in2p3.fr/home/dteam/rocio.file2"
"gsiftp://lxfsra10a01.cern.ch/lxfsra10a01.cern.ch:/tmp/rocio.file" "gsiftp://lpsc-se-dpm-server.in2p3.fr/dpm/in2p3.fr/home/dteam/rocio.file3"
```

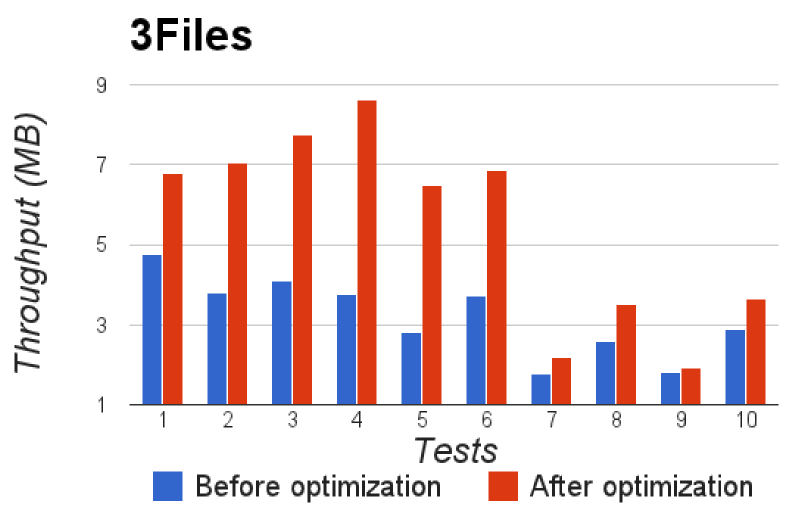
This way, we can have a better overview about how affects the optimized TCP buffer size in the transfer of files.

The tests have been done at the same time, first with the default OS tcp buffer size, using the next command line:

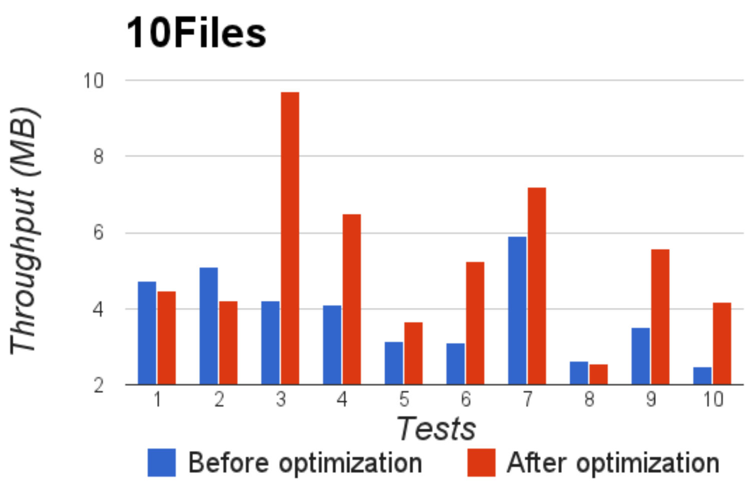
"globus-url-copy -nodcau -p 10 -fast -vb -concurrency 10 -f ./bulk.txt"

And just after get the throughput, we tuned the parameters, with the optimized tcp buffer size, using the next command line, to get the throughput:

"globus-url-copy -nodcau -p 10 -fast -vb -concurrency 10 -tcp-bs ***tcpsize(bytes)*** *-f ./bulk.txt"*



RTT (ms)	Before optimization (MB)	After optimization (MB)	TCP_buffer (bytes)
	4.76	6.8	7486832
	3.82	7.04	5958270
	4.1	7.75	7523532
	3.77	8.62	6720323
	2.81	6.49	4640735
	3.72	6.85	5802295
	1.79	2.18	3120431
	2.58	3.52	4937220
	1.81	1.93	3107848
12.7	2.89	3.65	4810735



RTT (ms)	Before optimization (MB)	After optimization (MB)	TCP_buffer (bytes)
	4.72	4.46	7362051
	5.08	4.2	9055502
	4.2	9.71	6550978
	4.1	6.49	6771179
	3.15	3.66	4913233
	3.1	5.26	6338641
	5.92	7.19	4671406
	2.61	2.56	4344643
[illegible]	3.51	5.59	6210846
12	2.48	4.18	3900702

The graphics show the throughput obtained. The blue color means before optimization, and the orange color means after optimization. Higher throughput means better results (vertical axis). The horizontal axis means the different tests done at different times.

We can see that much of the times we get better throughput. We can even double the throughput, if we optimize the tcp buffer size.

Also, other times, the difference between before and after the optimization is not so clear. That is because it depends on the network state: available bandwidth, people using the network...

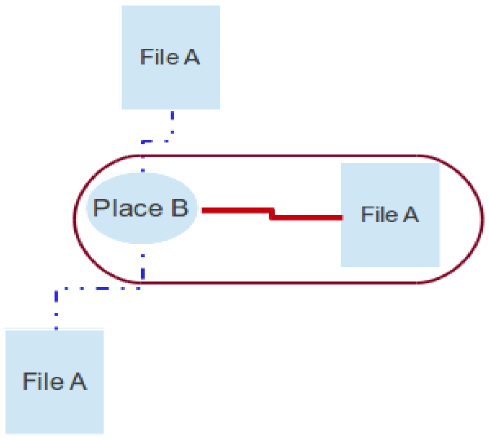
5 Conclusions and future work

We achieve a better throughput in most of the tests done. That means that we get a more efficient transfer.

Still it is needed to integrate the library in FTS3 and evaluate the throughput and the transfer, and this will be the next step in this project. But seeing the results from the initial tests, we are going in the good way.

Moreover, perfSONAR information can be used in a future to select replicas.

That means that, for example, we want to copy a *file*, with multiples copies in the world, to place B. If we know the network performance between site B and the copies of the file, we can choose the best source.



6 References

[1] Enabling high throughput in widely distributed data management and analysis systems: Lessons from the LHC.

[2] FTS3, https://svnweb.cern.ch/trac/fts3

[3] http://www.perfsonar.net/

[4] LAMP article: TCP Tuning and Network Troubleshooting

[5] "High Performance Bulk Data Transfer"- Brian Tierney and Joe Metger, ESnet http://fasterdata.es.net/assets/fasterdata/JT-201010.pdf

[6] https://github.com/carioka88/fts3-perfsonar